Desarrollo de un catálogo multimedia unificado

Memoria de Proyecto Final de Máster

Máster Universitario en Aplicaciones multimedia

Itinerario profesional

Autor: Jonathan Patricio Yajamín Yajamín

Consultor: Sergio Schvarstein Liuboschetz
Profesor: Sergio Schvarstein Liuboschetz

Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de un catálogo multimedia unificado*
Nombre del autor:	*Jonathan Patricio Yajamín Yajamín*
Nombre del consultor/a:	*Sergio Schvarstein Liuboschetz*
Nombre del PRA:	*Laura Porta Simó*
Fecha de entrega (mm/aaaa):	*06/2018*
Titulación:	*Máster Universitario en Aplicaciones multimedia*
Área del Trabajo Final:	*El nombre de la asignatura de TF*
Idioma del trabajo:	*Español*
Palabras clave	*Web, catálogo, multimedia*

Resumen del Trabajo (máximo 250 palabras):

En los últimos años, se ha visto como la adquisición y consumo de contenido multimedia (música, películas, series de televisión, etc.) ha aumentado.

Debido a este hecho, con el paso del tiempo es necesario guardar un historial que nos permita mantener un historial del contenido que se ha adquirido.

En la actualidad, hay múltiples aplicaciones que permiten llevar a cabo un seguimiento, pero la gran mayoría se centra en un solo ámbito, ya sea contenido audiovisual, música, libros o comics.

En el presente documento se propone la construcción de una aplicación web que unifique la funcionalidad que se encuentra dispersa en diversas aplicaciones mediante un catálogo multimedia, el cual permita realizar el seguimiento del contenido multimedia adquirido.

Además, debido a que gran parte de la adquisición multimedia es en formato digital, se quiere dotar a la aplicación de la funcionalidad de conexión a un dispositivo de almacenamiento remoto para poder descargar el contenido que este almacena.

Abstract (in English, 250 words or less):

In recent years, the acquisition and consumption of multimedia content (music, films, tv series…) has increased.

Due to this, it is necessary to register the multimedia content that a user has acquired over time.

Today, we can find several applications for this purpose. However, most of them are only focused on one aspect at a time such as films, music, books or comics.

The following document proposes the creation of a web application which unifies the scattered capabilities into one multimedia catalog. The resulting catalog would allow the users to track all the multimedia content they have acquired.

Furthermore, since much of the multimedia acquisition is in digital format, a connection to a remote device, where the content is stored, would be implemented to allow it to be downloaded.

Dedicatoria

Dedico este trabajo a la persona que ha estado siempre apoyándome, a mi madre Elsa Martha Yajamín Yajamín.

Agradecimientos

Quiero dar las gracias a mi madre, Elsa Martha Yajamín Yajamín, por todo su apoyo y por haber creído siempre en mí. Por todos los sacrificios que ha hecho durante estos años para que tenga una educación.

También quiero agradecer a mis amigos y familiares por haberme animado a seguir adelante y no rendirme cuando las cosas se torcían.

Sin vosotros, no habría llegado hasta aquí. Gracias.

Resumen

En los últimos años, se ha visto como la adquisición y consumo de contenido multimedia (música, películas, series de televisión, etc.) ha aumentado.

Debido a este hecho, con el paso del tiempo es necesario guardar un historial que nos permita mantener un historial del contenido que se ha adquirido.

En la actualidad, hay múltiples aplicaciones que permiten llevar a cabo un seguimiento, pero la gran mayoría se centra en un solo ámbito, ya sea contenido audiovisual, música, libros o comics.

En el presente documento se propone la construcción de una aplicación web que unifique la funcionalidad que se encuentra dispersa en diversas aplicaciones mediante un catálogo multimedia, el cual permita realizar el seguimiento del contenido multimedia adquirido.

Además, debido a que gran parte de la adquisición multimedia es en formato digital, se quiere dotar a la aplicación de la funcionalidad de conexión a un dispositivo de almacenamiento remoto para poder descargar el contenido que este almacena.

Palabras clave

Web, catálogo, multimedia, unificar, conexión, remoto.

Abstract

In recent years, the acquisition and consumption of multimedia content (music, films, tv series…) has increased.

Due to this, it is necessary to register the multimedia content that a user has acquired over time.

Today, we can find several applications for this purpose. However, most of them are only focused on one aspect at a time such as films, music, books or comics.

The following document proposes the creation of a web application which unifies the scattered capabilities into one multimedia catalog. The resulting catalog would allow the users to track all the multimedia content they have acquired.

Furthermore, since much of the multimedia acquisition is in digital format, a connection to a remote device, where the content is stored, would be implemented to allow it to be downloaded.

Keywords

Web, catalog, multimedia, unify, connection, remote.

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.Introducción

Durante los últimos años se ha podido observar un aumento en la adquisición y consumo de contenido multimedia, tales como música, películas, series de televisión, etc. (El Economista, 2017; Marketing Directo, 2017; Observatorio Nacional de Telecomunicaciones y Sociedad de la Información , 2017)

Teniendo en cuenta la adquisición de este tipo de contenido, muchas veces las personas se encuentran en la situación en la que no sabe si ya ha comprado o no cierto artículo, ya sea una película o un libro. Por este motivo, es importante guardar un registro de todo este contenido para evitar realizar una compra duplicada.

En la actualidad se dispone de múltiples aplicaciones que sirven para este propósito, pero estas tienen el defecto de que se centran en un solo sector: películas y series, libros o comics; por lo que el usuario se ve obligador a usar varias aplicaciones, lo cuál muchas veces puede ser caótico y pesado.

Partiendo de esta premisa, en el presente trabajo de fin de máster se pretende realizar el desarrollo de una aplicación web que unifique el seguimiento para películas, series de televisión, libros y comics, formando de esta forma un catálogo unificado de contenido multimedia.

Además, partiendo del aumento de consumo en *streaming* y descargas (Sweney, 2017), en caso de que haya tiempo suficiente para dicho desarrollo, se pretende añadir la posibilidad de conectar dicha aplicación a un dispositivo de almacenamiento remoto, como podría ser un **NAS** (Almacenamiento conectado en red, o del inglés, *Network Attached Storage*), para poder descargar el contenido adquirido en el dispositivo en el que se está usando la aplicación.

2. Descripción

La idea para este proyecto parte del uso de distintas aplicaciones para mantener el registro del contenido multimedia que se ha consumido o adquirido, tales como **SeriesGuide**, **Movie Mate** y **Whakoom**. Sin embargo, estas aplicaciones solo ayudan en un ámbito del problema que se busca resolver.

Como se puede apreciar de estos ejemplos, todas estas ayudan para mantener el registro del contenido multimedia, pero seccionándolo, haciendo de esta tarea algo muy pesado para el usuario, que se ve obligado a utilizar múltiples aplicaciones.

Con el presente proyecto, se busca unificar cierta funcionalidad de estas aplicaciones, ofreciendo al usuario una única aplicación donde pueda ver los catálogos (de series de televisión, películas, libros y comics) que este posee.

Además, se pretende que esta aplicación sea sencilla de utilizar y que resuelva el problema que se plantea, para más adelante poder seguir su desarrollo y añadir más funcionalidad.

Como resultado de este proyecto, se entregará dos desarrollos: una **aplicación cliente** que se ejecutará en el navegador y otra destinada a ser ejecutada en un servidor, y que proveerá a la primera de la funcionalidad necesaria para resolver el problema, exponiendo una **API Rest**.

3. Objetivos generales

3.1 Objetivos principales

Para la realización del presente trabajo de fin de máster se han planteado los siguientes objetivos a cumplir:

1. Realizar un prototipo de la aplicación.
2. Implementación de una aplicación que permita realizar el seguimiento del contenido de ocio que posee un usuario, tales como series de televisión, películas, libros y comics.
3. Uso de **APIs** de terceros para la obtención de la información del contenido de ocio.

Con este desarrollo se pretende que el usuario sea capaz de realizar las siguientes acciones:

- Registrarse en la aplicación.
- Iniciar sesión en la aplicación.
- Cerrar sesión en la aplicación.
- Eliminar su cuenta de usuario y los datos asociados.
- Listar las series de televisión que posee.
- Listar las películas que posee.
- Listar los libros que posee.
- Listar los comics que posee.
- Buscar por contenido multimedia.
- Añadir un ítem al catálogo.
- Eliminar un ítem del catálogo.
- Modificar el ítem del catálogo: especificar si se posee este contenido en formato físico (DVD, Blu-ray, etc.) o en formato digital o añadir una descripción de la localización del mismo para cada uno de los formatos.

Por último, se listarán los objetivos personales a la hora de elección de este proyecto y su desarrollo:

- Como usuario de las aplicaciones que se han nombrado en el anterior apartado, disponer de una única aplicación que permita realizar el seguimiento del contenido nombrado.
- Aprender a utilizar las tecnologías web que se utilizan actualmente, tales como el *framework* JavaScript **Vue.js**, el entorno de ejecución de JavaScript **Node.js** junto con el *framework* **Express** y el sistema de bases de datos NoSQL **MongoDB**.
- Realizar la implementación de la aplicación utilizando las tecnologías mencionadas en el anterior punto para aprender como se integran las unas con las otras.

3.2 Objetivos secundarios

En caso de disponer del tiempo suficiente, se pretende añadir la siguiente funcionalidad a la aplicación:

- Conectar la aplicación a un dispositivo de almacenamiento remoto.
- Descargar contenido almacenado en dicho dispositivo remoto.

4. Metodología y proceso de trabajo

En primer lugar, se hablará de la metodología de trabajo que se ha escogido para la realización de este proyecto.

A pesar de haber distintas aplicaciones segmentadas que se utilizan para realizar el registro del contenido multimedia, estas no eran de código abierto y si lo eran, estas estaban implementadas para una plataforma específica, como es el caso de **SeriesGuide**, la cual esta disponible solo para el **Sistema Operativo** móvil **Android**.

Debido a esto, se ha optado por realizar un desarrollo desde cero, haciendo uso de **APIs** de terceros para la obtención de la información necesaria para el funcionamiento de la aplicación.

Con relación a la metodología de desarrollo, en primer lugar, se ha optado por la creación de un prototipo de alta fidelidad, para luego continuar con el estudio de las **APIs**, concluyendo con la implementación del proyecto.

Durante el desarrollo del proyecto, se han ido haciendo entregas intermedias, para que el tutor vea el avance del mismo.

5. Planificación

Antes de empezar con el desarrollo del proyecto, se realizó una planificación del mismo que tenía en cuenta las entregas intermedias de la asignatura y los hitos que se marcaban para cada una. A continuación, se muestra dicha planificación.

	Duración (días)	Inicio	Final
PEC 3	**28**	**27/03/2018**	**23/04/2018**
Prototipo	9	27/03/2018	04/04/2018
Formación en Vue.js	9	05/04/2018	13/04/2018
Estudio de las APIs seleccionadas	3	14/04/2018	16/04/2018
Diseño de la base de datos	7	17/04/2018	23/04/2018
PEC 4	**28 días**	**24/04/2018**	**21/05/2018**
Conexión con las APIs	7	24/04/2018	30/04/2018
Implementación de las vistas del catálogo	9	01/05/2018	09/05/2018
Preparación de la Raspberry Pi como NAS	3	10/05/2018	12/05/2018
Conexión con el NAS	9	13/05/2018	21/05/2018
PEC 5	**21 días**	**22/05/2018**	**11/06/2018**
Mapeo de datos	7	22/05/2018	28/05/2018

Tabla 1: Planificación inicial del proyecto

Esta primera planificación fue bastante optimista y se incluyo el desarrollo de los objetivos secundarios, expuestos en el apartado 3.2 Objetivos secundarios.

Es importante destacar que durante el desarrollo se sufrieron ciertos retrasos y las fechas de los hitos fueron desplazados en el tiempo. En la siguiente tabla se pueden ver dichos desplazamientos y la planificación final que se siguió en el proyecto.

	Duración (días)	Inicio	Final
PEC 3	**28**	**27/03/2018**	**23/04/2018**
Prototipo	9	27/03/2018	04/04/2018
Formación en Vue.js y Webpack	12	05/04/2018	16/04/2018
Estudio de las APIs seleccionadas	3	17/04/2018	19/04/2018
Diseño de la base de datos	4	20/04/2018	23/04/2018
PEC 4	**28**	**24/04/2018**	**21/05/2018**
Conexión con las APIs	3	08/05/2018	10/05/2018
Implementación de la base de datos	2	11/05/2018	12/05/2018
Implementación de la autenticación del usuario	5	13/05/2018	17/05/2018
Implementación de las vistas del catálogo y búsqueda.	5	18/05/2018	22/05/2018
PEC 5	**21**	**22/05/2018**	**11/06/2018**
Implementación de la aplicación servidor	16	22/05/2018	06/06/2016
Implementación de las vistas de detalle de ítem y modificación.	17	22/05/2018	07/06/2018
Comunicación entre la aplicación cliente y la aplicación servidor	16	22/05/2018	06/06/2018

Tabla 2: Planificación final del proyecto

Como se puede ver, hay mucha diferencia entre la planificación inicial y la final debido a que no se tuvo en cuenta funcionalidad importante como la autenticación y la implementación de la aplicación servidor y su comunicación con la aplicación cliente.

La implementación de la aplicación servidor y la comunicación con la aplicación cliente se englobo en la implementación de las vistas y se subestimó en gran medida su complejidad.

Otro punto a remarcar es la eliminación de la conexión y comunicación con un dispositivo remoto debido al retraso de tareas y al deseo de poder dejar la aplicación resultante en un estado estable.

6. Presupuesto

El desarrollo de este proyecto se puede dividir en dos: la creación del prototipo y la implementación del proyecto.

Para la creación del prototipo se ha utilizado la herramienta **Adobe XD**, la cual se puede obtener de forma gratuita, pero con limitaciones, o pagando **12,09€** para un solo proyecto. Además de estas opciones, se dispone de una versión sin restricciones que cuesta 60,49€. Para el desarrollo de este proyecto con las dos primeras versiones sería suficiente.

Para la implementación del proyecto se ha intentado utilizar herramientas gratuitas y/o de código abierto. Entre las herramientas que se han usado están **Visual Studio Code**, **Postman**, **MongoDB Compass**.

También se ha seguido el mismo ejemplo para los *frameworks* utilizados: **Vue.js, Bootstrap**, **Node.js**, **MongoDB**.

Teniendo en cuenta esto, el presupuesto se calculará sobre el trabajo realizado y las horas dedicadas al proyecto.

Para la realización de este proyecto se ha necesita sobre todo dos perfiles: un diseñador y un desarrollador.

De la aplicación **Indeed**, obtenemos que el salario medio de un desarrollador es 1129€/mes (Indeed, 2018) y el de un desarrollador junior es 1283€/mes (Indeed, 2018).

Partiendo de estos datos y teniendo en cuenta la planificación, el diseñador gráfico sería necesario para la implementación del prototipo como mínimo, lo que conllevo 9 días en la planificación. A esto hay que añadir que se le podría necesitar para le diseño de las vistas en la aplicación cliente, por lo que se supondrá que se le contrata por un mes.

En el caso del programador, es necesario para casi todo el desarrollo del proyecto, por lo que se asume que se le contrata por dos meses y medio.

De estas suposiciones, nos sale un total de 4336,5€. Pero no hay que olvidar que estos sueldos son para trabajadores a tiempo completo, unas 400 horas aproximadamente.

Teniendo en cuenta que se ha dedicado al proyecto unas 300 horas aproximadamente, el presupuesto para el equipo técnico sería **3252,38€**.

A este presupuesto habría que añadirle el material que se ha usado: un ordenador ASUS valorado en **700,00€**, un monitor valorado en **129,00€** y un ratón valorado en **13,80€**.

Haciendo la suma de estas cantidades obtenemos un presupuesto final de **4107,27€**.

7. Organización del documento

Por último, en este apartado se expone la estructura de este documento, el cual este compuesto por los siguientes capítulos, sin contar el presente:

- **Estado del arte**. Este capítulo se centra en platear el contexto del presente proyecto y recoger información de trabajos similares y en los que se ha inspirado la idea para el mismo.
- **Diseño**. En este capítulo se mostrará la estructura de la aplicación y sus dependencias, así como el flujo de navegación y vistas que la componen.
- **Implementación**. En este capítulo se detallará el proceso de desarrollo que se ha llevado a cabo.
- **Despliegue de la aplicación**. En este capitulo se detallará el proceso de despliegue que se ha seguido para realizar la subida de las aplicaciones a un servidor.
- **Conclusiones y líneas de futuro**. En este último capítulo se expondrán las conclusiones una vez se ha finalizado el proyecto y posibles líneas de trabajo futuro que se puede hacer para dar continuidad al proyecto.

Capítulo 2: Estado del arte

En el anterior capítulo, se han nombrado algunas aplicaciones que han servido como inspiración para el desarrollo de este Trabajo de Fin de Máster. Dos de ellas son utilizadas personalmente, SeriesGuide y Whakoom, mientras que las otras son resultado de búsquedas de aplicaciones para realizar la tarea principal que se busca solucionar (González, Xataka Android, 2015).

1. SeriesGuide

Esta aplicación sirve principalmente para realizar un seguimiento de las series de televisión y películas que se han visualizado, permitiendo también guardar un registro de si se posee dicho material (Trottmann, s.f.). Sin embargo, en el caso de las series, el catálogo no puede ser consultado fácilmente, obligando al usuario a mirar siempre los detalles de las series de televisión para comprobar si se tiene o no.

A pesar de ser una excelente aplicación, tiene el defecto de estar disponible solo para la plataforma móvil **Android**.

La aplicación se nutre de la información que obtiene de las **APIs The TVDb** y **The Movie DB** y además permite la conexión con la plataforma **Trakt.tv**, que sirve para mantener un registro del contenido audiovisual que el usuario consume, entre otras cosas (Trakt, n.d.).

2. Movie Mate

Otra aplicación que existe en el mercado para mantener un registro del contenido audiovisual es **Movie Mate**, la cual se centra principalmente en las películas.

Al centrarse solo en películas, esta aplicación es capaz de mostrar más información que, por ejemplo, **SeriesGuide**. Además, la aplicación incluye un sistema de recomendación y notifica de nuevos lanzamientos de películas en DVD (González, Xataka Android, 2016).

Esta es otra buena aplicación, pero, al igual que **SeriesGuide**, solo esta disponible para dispositivos móviles **Android**.

Esta aplicación obtiene los datos de **Track.tv**, **The Movie DB**, **Rotten Tomatoes**, **IMDB** y **Metacritic** (Play, 2018).

3. Whakoom

Whakoom es una aplicación que sirve para catalogar los comics de un usuario. Es una aplicación muy potente que, además notifica al usuario de los lanzamientos de nuevos ejemplares en una colección que sigue, mantener una lista de deseos y dar recomendaciones al usuario en base a lo que este ha leído entre otras funcionalidades (Whakoom, s.f.).

Esta aplicación se encuentra disponible para dispositivos móviles **Android** e **iOS**, además de estar disponible en la web.

Esta herramienta no hace uso de ninguna API externa para la obtención de datos, sino que se apoyan en sus usuarios para que estos introduzcan los datos en la aplicación. Estos datos son de dominio público y la empresa renuncia a los derechos de atribución (Whakoom, s.f.).

4. Libib

Por último, tenemos la aplicación **Libib** que es la que más se asemeja a la aplicación que se implementará en este proyecto.

Esta aplicación permite crear diferentes catálogos para guardar información de libros (en los cuales se incluyen algunos comics), películas (que engloba también las series de televisión), videojuegos y música (Libib, n.d.).

Es una aplicación bastante completa, que permite además introducir la información de los artículos mediante el escaneo del código de barras, búsqueda por texto e introducción manual.

Sin embargo, desde un punto de vista personal, la separación que se hace de los artículos es demasiado global y puede dar lugar a confusión por los literales que utilizan.

Otro inconveniente que se ha encontrado es que la búsqueda de artículos puede llegar a ser confusa debido a que por defecto realiza la búsqueda por ISBN en caso de los libros y el UPC (*Universal Product Code*) en caso de las películas.

Esta aplicación, se encuentra disponible para dispositivos móviles **Android** e **iOS** y también mediante una aplicación web.

Capítulo 3: Diseño

1. Arquitectura general de la aplicación

En este apartado se describirá la arquitectura de la aplicación. Como se ha comentado antes, esta está formada por una aplicación cliente y una aplicación servidor. Además, la aplicación cliente también se conecta con tres **APIs** de terceros para la obtención de la información, la cual es guardada posteriormente en el sistema.

A continuación, se muestra un diagrama de la arquitectura.

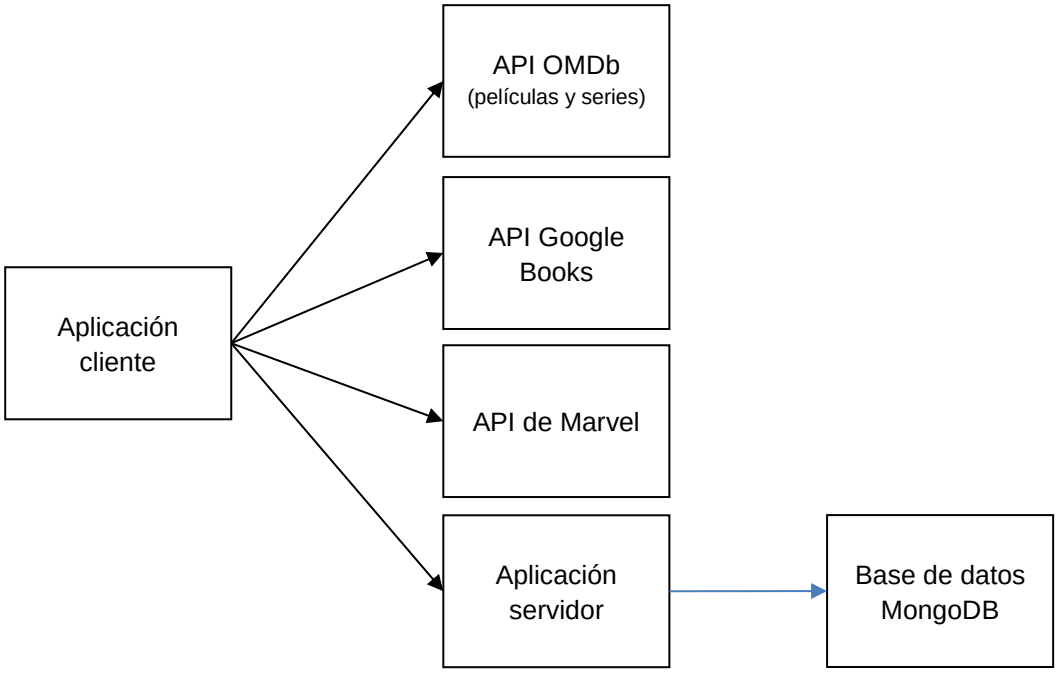

Figura 1: Arquitectura de la aplicación

En la imagen se puede ver que la aplicación cliente depende de tres **APIs**:

- **API OMDb**. Esta **API** se utiliza para obtener información tanto de películas como de series de televisión.
- **API Google Books**. Esta API se utiliza para la obtención de la información de los libros que se van a almacenar en la aplicación.
- **API de Marvel**. Esta aplicación se utiliza para la obtención de la información de los comics de la editorial Marvel.

Se han seleccionado estas **APIs** por su simplicidad de uso y por la cantidad de información que proveen.

La información que proveen es almacenada en la aplicación servidor cuando el usuario añade dicho contenido a su catálogo. Esto se hace por dos motivos:

1. Evitar hacer sucesivas llamadas a las APIs para no superar el límite impuesto de llamadas.
2. Disminuir el tiempo de carga de la información en la aplicación cliente al realizar una sola llamada al servidor.

Junto a esta información, también se almacena la información referente al formato del ítem que posee el usuario.

Además, la aplicación del servidor se utiliza para la autenticación del usuario y la autorización al contenido de sus catálogos.

2. Arquitectura de la información y diagramas de navegación

En esta subsección se mostrarán los diagramas correspondientes al diseño de la base de datos que utiliza la aplicación servidor, el árbol de navegación que sigue la aplicación cliente y la estructura de la **API Rest** que provee de funcionalidad a la aplicación cliente.

2.1 Base de datos

A continuación, se mostrará el diseño de la base de datos que se ha usado para el desarrollo de la aplicación.

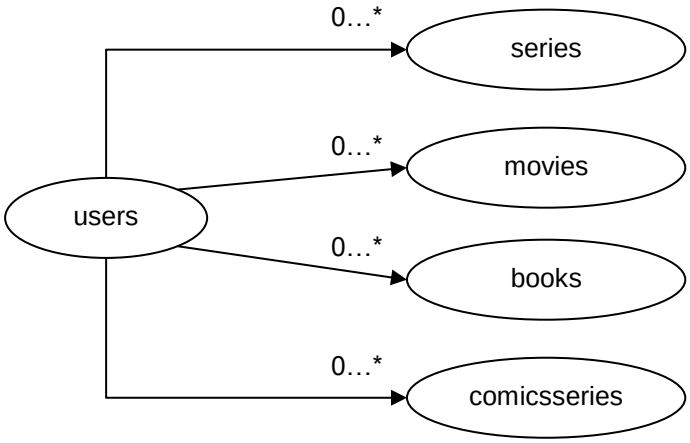

Figura 2: Diagrama de la base de datos de la aplicación.

La base de datos está formada por cinco colecciones:

- **Users**. Colección que almacena la información del usuario, como sus credenciales y sus catálogos, los cuales incluyen in listado con los formatos y su localización.
- **Series**. Colección que almacena la información de las series, sus temporadas y capítulos.
- **Movies**. Colección que almacena la información obtenida de las películas.
- **Books**. Colección que almacena la información obtenida de los libros.
- **ComicsSeries**. Colección que almacena la información de las series de comics y los comics que las forman.

Las cuatro últimas colecciones se utilizan para guardar los datos completos de los artículos, evitando de esta forma realizar múltiples peticiones a las **APIs** para obtener la misma información.

Una vez se ha mostrado el diagrama de la base de datos, se mostrarán breves fragmentos de código que se han utilizado para la creación de la base de datos y sus esquemas.

En primer lugar, se va a hablar de la definición usada para el registro y autenticación de un usuario. Esta está formada por un nombre de usuario, un email y una contraseña. Como se puede apreciar en el fragmento, se ha hecho uso de indices únicos sobre el nombre del usuario y el email para prevenir la inserción de usuarios repetidos. Además, también se le ha añadido el atributo *required* para hacer todos los campos obligatorios.

```
const UserSchema = new mongoose.Schema({
  local: {
    username: { type: String, required: true, index: { unique: true } },
    email: { type: String, required: true, index: { unique: true } },
    password: { type: String, required: true }
  },
  ...
});
```

Figura 3: Definición del esquema de usuario para la autenticación.

Después de exponer la definición usada para la autenticación, se va a exponer un fragmento de código que se utiliza para la definición del catalogo de las series de televisión.

```
const UserSchema = new mongoose.Schema({
  ...,
  series: [{
    id: { type: String, required: true },
    title: { type: String, required: true },
    year: Number,
    cover: String,
    formats: [{
      type: { type: String, required: true },
      location: String
    }],
    seasons: [{
      number: { type: Number, required: true },
      formats: [{
        type: { type: String, required: true },
        location: String
      }],
      episodes: [{
        number: { type: Number, required: true },
        formats: [{
          type: { type: String, required: true },
          location: String
        }]
      }]
    }]
  }],
  ...
});
```

Figura 4: Definición del esquema de usuario para el catálogo de series

En el anterior fragmento de código se puede observar que se ha definido un *array* de formatos para cada nivel. Además, se ha definido los campos indispensables que son necesarios cuando se muestra el catálogo de series.

Los dos fragmentos anteriores son parte del esquema de base de datos de usuario. La definición entera del esquema se puede ver el archivo ***models/user.js*** en el proyecto de la aplicación servidor.

Por último, para ejemplificar un solo tipo de articulo se mostrará el esquema utilizado para las series de televisión.

```
const SeriesSchema = new mongoose.Schema({
  id: { type: String, required: true, index: { unique: true } },
  title: { type: String, required: true },
  plot: { type: String, required: true },
  year: Number,
  cover: String,
  totalseasons: { type: Number, required: true },
  seasons: [{
    number: { type: Number, required: true },
    year: Number,
    totalepisodes: { type: Number, required: true },
    episodes: [{
      number: { type: Number, required: true }
    }]
  }]
});
```

Figura 5: Definición del esquema de Series

En este esquema solo se ha incluido la información necesaria para el funcionamiento de la aplicación. Sin embargo, cuando se decida añadir más funcionalidad este esquema puede ser cambiado para que almacene más datos.

Al igual que en el caso de los usuarios, se usa in índice único sobre el campo **id** del esquema para evitar almacenar dos veces el mismo artículo. Es importante destacar que este campo se corresponde al identificador que se usa en la **API** de **OMDb**.

Los esquemas de los artículos restantes siguen el mismo patrón utilizado para las series. Su definición se puede ver en los ficheros que se encuentran en el directorio ***models*** del proyecto que engloba la aplicación servidor.

2.2 Árbol de navegación

Después de definir brevemente el esquema de la base de datos, se va a mostrar el árbol de navegación de la aplicación.

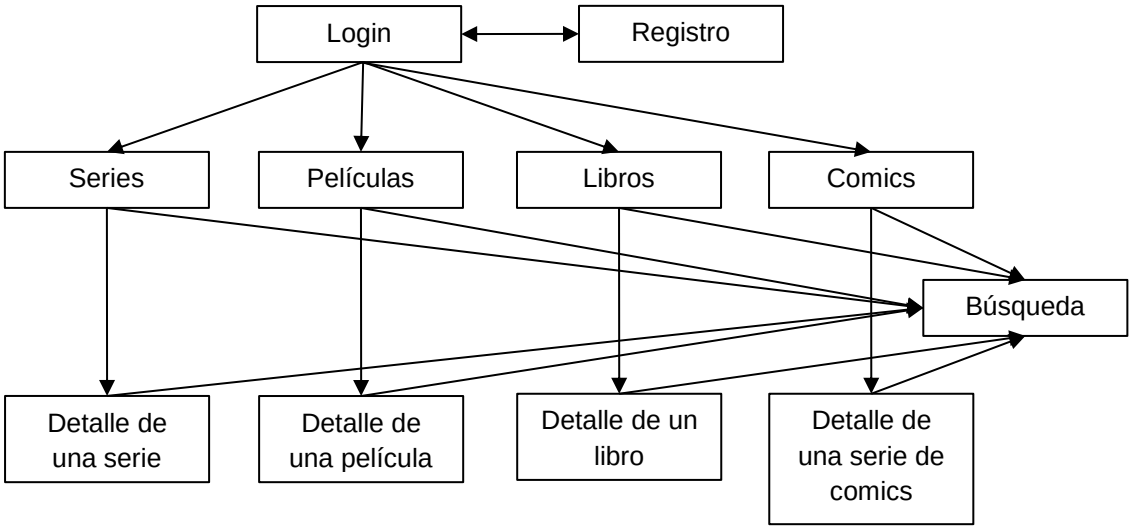

Figura 6: Árbol de navegación de la aplicación cliente

Como se puede ver en la anterior figura, la aplicación cuenta con una página para el inicio de sesión en la aplicación y otra para registrar un usuario nuevo.

Una vez el usuario haya accedido a la aplicación, puede navegar entre los cuatro catálogos definidos: series, películas, libros, comics y la página de búsqueda. Los resultados mostrados en esta última dependerán de donde se inició la búsqueda. Por ejemplo, si al iniciar la búsqueda se estaba en la página que muestra el catálogo de las series, los resultados de la búsqueda incluirán series que cuyo nombre coincida con el introducido en el campo de búsqueda.

Dentro de cada catálogo, se puede acceder a la vista del detalle del artículo, en la cual se podrá realizar modificaciones sobre la información de los formatos del mismo.

2.3 API Rest

Para el funcionamiento de la aplicación web, ha sido necesario la implementación de una **API Rest** que expone servicios para la autenticación (gestión de los usuarios), la gestión de series, de películas, de libros y de comics.

A continuación, se mostrarán unas tablas que contienen un listado de todos los *endpoints* implementados en la **API.**

Autenticación	
Registrar	**POST /api/auth/register** Registra un nuevo usuario en la aplicación y devuelve un token para la autenticación/autorización de las consecutivas llamadas.
Autenticar	**POST /api/auth/token** Comprueba las credenciales del usuario y devuelve un token para la autenticación/autorización de las consecutivas llamadas.
Eliminar	**DELETE /api/auth/unregister/{userId}** Elimina la cuenta del usuario que se especifica en la petición.

Tabla 3: *Endpoints* de autenticación y gestión de usuarios

Series	
Obtener catálogo	**GET /api/users/{userId}/series** Devuelve el catálogo de series del usuario especificado en la petición.
Obtener detalles	**GET /api/users/{userId}/series/{id}** Devuelve los detalles de la serie dentro del catálogo de usuario especificado en la petición.
Crear	**POST /api/users/{userId}/series** Añade la serie descrita en el cuerpo de la petición al usuario especificado en la misma.
Actualizar serie	**PATCH /api/users/{userId}/series/{id}** Actualiza los formatos que el usuario ha especificado para una serie en su catálogo.
Actualizar temporada	**PATCH /api/users/{userId}/series/{id}/seasons/{season}** Actualiza los formatos que el usuario ha especificado para una temporada de una serie dentro del catálogo del usuario.
Actualizar capítulo	**PATCH /api/users/{userId}/series/{id}/seasons/{season}/episodes/{episode}** Actualiza los formatos que el usuario ha especificado para un capítulo de una temporada de una serie dentro del catálogo del usuario.
Eliminar	**DELETE /api/users/{userId}/series/{id}** Elimina la serie del catálogo del usuario especificado en la petición.

Tabla 4: *Endpoints* para la gestión de series

Películas	
Obtener catálogo	**GET /api/users/{userId}/movies** Devuelve el catálogo de películas del usuario especificado en la petición.
Obtener detalles	**GET /api/users/{userId}/movies/{id}** Devuelve los detalles de la película dentro del catálogo de usuario especificado en la petición.
Crear	**POST /api/users/{userId}/movies** Añade la película descrita en el cuerpo de la petición al usuario especificado en la misma.
Actualizar	**PATCH /api/users/{userId}/movies/{id}** Actualiza los formatos que el usuario ha especificado para una película en su catálogo.
Eliminar	**DELETE /api/users/{userId}/movies/{id}** Elimina la película del catálogo del usuario especificado en la petición.

Tabla 5: *Endpoints* para la gestión de películas

Libros	
Obtener catálogo	**GET /api/users/{userId}/books** Devuelve el catálogo de libros del usuario especificado en la petición.
Obtener detalles	**GET /api/users/{userId}/books/{id}** Devuelve los detalles del libro dentro del catálogo de usuario especificado en la petición.
Crear	**POST /api/users/{userId}/books** Añade el libro descrito en el cuerpo de la petición al usuario especificado en la misma.
Actualizar	**PATCH /api/users/{userId}/books/{id}** Actualiza los formatos que el usuario ha especificado para un libro en su catálogo.
Eliminar	**DELETE /api/users/{userId}/books/{id}** Elimina el libro del catálogo del usuario especificado en la petición.

Tabla 6: *Endpoints* para la gestión de libros

Series	
Obtener catálogo	**GET /api/users/{userId}/comicsseries** Devuelve el catálogo de las series de comics del usuario especificado en la petición.
Obtener detalles	**GET /api/users/{userId}/comicsseries/{id}** Devuelve los detalles de la serie de comics dentro del catálogo de usuario especificado en la petición.
Crear	**POST /api/users/{userId}/comicsseries** Añade la serie de comics descrita en el cuerpo de la petición al usuario especificado en la misma.
Actualizar serie de comics	**PATCH /api/users/{userId}/comicsseries/{id}** Actualiza los formatos que el usuario ha especificado para una serie de comics en su catálogo.
Actualizar comic	**PATCH /api/users/{userId}/comicsseries/{id}/comics/{comic}** Actualiza los formatos que el usuario ha especificado para un comic perteneciente a una serie dentro del catálogo del usuario.
Eliminar	**DELETE /api/users/{userId}/comicsseries/{id}** Elimina la serie de comics del catálogo del usuario especificado en la petición.

Tabla 7: *Endpoints* para la gestión de las series de comics

Es importante remarcar que todas las peticiones que se realizan a estos servicios, salvo el de registro y el de autenticación, deben llevar en su cabecera el token de autenticación generado por los dos servicios de acceso anónimo.

3. Diseño gráfico e interfaces

Antes de empezar el desarrollo de la aplicación, se realizó un prototipo de alta fidelidad haciendo uso de la herramienta **Adobe XD**.

Esta herramienta escogió por diversos motivos, siendo uno de ellos la licencia provista por la Universitat Oberta de Catalunya.

Se generaron dos prototipos distintos para la aplicación: uno para dispositivos móviles y otro para dispositivos medianos (*tablets*) y de escritorio. Estos prototipos serán incluidos en la entrega del presente proyecto.

3.1 Estilos

Para el diseño de la interfaz grafica de la aplicación se han utilizado los siguientes recursos:

- Fuente: Roboto (Google, Google Fonts, s.f.)
- Iconos: FontAwesome (Fontawesome, s.f.)
- *Framework* CSS: Bootstrap (Bootstrap, s.f.)

Es importante remarcar que, a pesar de haber usado Bootstrap, se han modificado los estilos del *framework* para que tenga ciertas semejanzas con la guía de diseño **Material Design**.

4. Lenguajes de programación y APIs utilizadas

Para el desarrollo de las aplicaciones se ha decidido utilizar un *stack* puramente JavaScript debido a que el uso de un solo lenguaje tanto para la aplicación cliente como para la aplicación servidor es más cómoda que el uso de lenguajes distintos.

Otro motivo de la elección de este *stack* es el planteamiento a futuro, para que cuando haya más de una persona en el desarrollo de la aplicación, cualquiera pueda trabajar tanto en la aplicación cliente como en la aplicación servidor.

4.1 Aplicación cliente

Para el desarrollo de la aplicación cliente se ha escogido el *framework* de desarrollo *front-end* **Vue.js** (You, 2018).

La elección de este *framework* se debe a que no presenta una curva elevada de aprendizaje además de ser ligero en comparación a otros *frameworks* de desarrollo como **Angular** o **React** (Vynogradenko, 2017), además de ser progresivo. Con esto último se quiere decir que el uso de este no obliga a utilizarlo en toda la aplicación, ya que se puede utilizar en puntos puntuales de la aplicación en caso de ser necesario.

Se ha descartado el uso de **CMS** debido que se busca construir una aplicación sencilla. En caso de usar un **CMS** como **WordPress** o **Drupal**, supondría una curva de aprendizaje elevada, además de una complejidad innecesaria en la aplicación.

Junto a **Vue.js**, se ha utilizado el *router* **Vue Router** (Vue Router, s.f.) para el control de las rutas de la aplicación web.

Por último, para la implementación de la comunicación entre la aplicación cliente y la aplicación servidor se ha usado el paquete **Axios** (Nick Uraltsev, s.f.).

4.2 Aplicación servidor

Para el desarrollo de la aplicación servidor, se ha utilizado el entorno de ejecución para JavaScript **Node.js** (Node.js, s.f.).

Node.js está construido sobre el motor de JavaScript de Google Chrome, **V8**. Este entorno está orientado a eventos asíncronos lo que contrasta con el paradigma de hilos que usa un sistema operativo, por ejemplo.

Debido a esta orientación, **Node.js** permite realizar más tareas debido a que no está limitado por el número máximo de hilos, gestionando mejor los recursos.

Se ha escogido este entorno de desarrollo y no otros como **Laravel** con PHP o **Web API** de .NET debido al afán de aprender como se utiliza este entorno ya que conforme pasa el tiempo, va ganando más y más popularidad.

Por encima de **Node.js**, se utiliza el *framework* **Express**, el cual presenta una infraestructura para construir aplicaciones web sobre **Node.js** (Express, s.f.).

Para el almacenamiento de datos, se ha escogido el sistema de bases de datos NoSQL **MongoDB**, el cual está orientado a documentos y es de código abierto (Wikipedia, 2018).

Este tipo de base de datos almacena la información en colecciones, utilizando el formato **BSON**. Este formato es una variante de **JSON** y es una estructura binaria de estructuras y mapas (Wikipedia, 2013).

Este sistema de base de datos esta ideado para ser escalable, de altor rendimiento y disponibilidad. A diferencia de bases de datos SQL, permite el crecimiento horizontal de forma sencilla.

Se ha escogido este sistema de bases de datos debido a que se busca gestionar contenidos y que la aplicación este disponible para dispositivos móviles, los cuales entran dentro los casos de uso del sistema.

4.3 APIs de terceros

En este apartado se va a hablar de las **APIS** de terceros que se han usado para la obtención de la información que nutre la aplicación.

En el apartado 1. Arquitectura general de la aplicación, se han nombrado brevemente dichas **APIs**. A continuación, se detallará un poco más sobre las mismas y los recursos que se han utilizado.

La primera **API** de la que se va a hablar es la que se utiliza para la obtención de la información de las series de televisión y las películas, **OMDb**.

El uso de esta aplicación es simple y expone la información realizando peticiones **GET** a un único *endpoint*: https://www.omdbapi.com.

A partir de este *endpoint* y utilizando parámetros en la *query string*, se obtienen datos de búsqueda, de una serie, sus temporadas o episodios y de una película.

Los parámetros que se han utilizado son los siguientes:

Parámetro	Descripción
s	Texto de búsqueda
i	Identificador IMDB de la serie o la película
type	Tipo de búsqueda: series o películas
apikey	Clave de acceso a la **API**
season	Indica el número de temporada de una serie
episode	Indica el número de episodio de una temporada de una serie.

Tabla 8: Parámetros de la **API OMDb**

Un factor importante que remarcar es la autenticación que utiliza la API para dar acceso a la información, la cual se realiza mediante un parámetro de la *query string* que contiene la clave que se genera en el momento de registro, y que se dispone de un límite máximo de 1000 peticiones al día.

La segunda **API** utilizada es la de **Google Books**, que se utiliza para la búsqueda de libros. Esta, a diferencia de la anterior, no necesita método de autenticación y es accesible a través de los siguientes *endpoints*.

Endpoint	Descripción
https://www.googleapis.com/books/v1/volumes	Se utiliza para realizar la búsqueda de libros por texto y admite parámetros en *query string*.
https://www.googleapis.com/books/v1/volumes/{id}	Se utiliza para obtener la información del libro cuyo identificador es igual al que se pasa en la URL de la petición.

Tabla 9: *Endpoints* de la **API Google Books**

Parámetro	Descripción
q	Texto de búsqueda
startIndex	Valor numérico que indica la posición del elemento en el que se empieza la búsqueda. Se usa principalmente para la paginación de contenido.

Tabla 10: Parámetros de la **API Google Books**

Por último, se ha usado la **API de Marvel** para la obtención de la información de los cómics. Se ha decidido empezar con la integración de una única fuente de información para los comics, para posteriormente ir aumentando el número de fuentes.

Se han utilizado los siguientes *endpoints*:

Endpoint	Descripción
https://gateway.marvel.com/v1/public/series	Se utiliza para la búsqueda de series de comics y admite parámetros en la *query string*.
https://gateway.marvel.com/v1/public/series/{id}/comics	Se utiliza para obtener el listado de comics que forma la serie. También admite parámetros en la *query string*.

Tabla 11: *Endpoints* de la **API de Marvel**

Los dos *endpoints*, comparten los siguientes parámetros:

Parámetro	Descripción
apikey	Clave pública de acceso a la **API**.
ts	Marca de tiempo del momento en el que se realiza la petición.
hash	Hash de la concatenación del timestamp, la clave privada y la clave pública.
limit	Límite de elementos que se mostrarán en la respuesta de la petición.
offset	Se utiliza para realizar la paginación de los resultados. Se salta tantos elementos como se indique en este campo.

Tabla 12: Parámetros comunes a los *endpoints* de la **API de Marvel**

Para el primer *endpoint* de la Tabla 11: *Endpoints* de la **API de Marvel**, también se hace uso de el parámetro **title** para la búsqueda.

Es importante señalar que estos son solo los parámetros que se han utilizado en el desarrollo, pero la **API** acepta muchos más.

Para finalizar, se hablará de la autenticación que se utiliza en la obtención de la información de la **API de Marvel**. Esta se hace de dos formas, dependiendo de si la petición viene de la aplicación cliente o de la aplicación servidor.

En el caso de la aplicación cliente, con enviar la clave pública generada por la **API** en el momento del registro es suficiente.

Sin embargo, en las peticiones que ser realizan desde la aplicación servidor hay que especificar, además de la clave pública, el *timestamp* del momento en el que se realiza la petición y el **hash md5** resultante de aplicarlo sobre la concatenación del *timestamp*, la clave privada y la clave pública.

El nombre de los parámetros que se utilizan para almacenar estos valores se puede ver en la Tabla 12: Parámetros comunes a los *endpoints* de la **API de Marvel**.

Capítulo 4: Implementación

1. Requisitos de instalación e implementación

1.1 Aplicación cliente

Como se ha especificado en el apartado 4. Lenguajes de programación y APIs utilizadas, se ha hecho uso de **Node.js** para el desarrollo de las aplicaciones cliente y servidor.

Para realizar la instalación de **Node.js** es necesario descargar el instalador de la página oficial. Al instalar **Node.js**, también se realiza la instalación del gestor de paquetes **NPM**, el cual se ha utilizado para instalar las dependencias de los proyectos *front-end* y *back-end*.

Antes de instalar paquetes con **NPM**, hay que configurar la herramienta para evitar que durante el desarrollo del producto se utilicen distintas versiones de los mismos paquetes. Para evitar esto, hay que añadir el fichero ***.npmrc*** que debe contener las siguientes líneas.

```
save=true
save-exact=true
```

Figura 7: Contenido del fichero ***.npmrc***

Con **Node.js** instalado en la máquina de desarrollo y configurado en el proyecto, se puede proceder a la creación del proyecto *front-end*, el cual se hecho usando el **CLI** (*Command-line Interface*) de **Vue.js**. Para instalar el programa hay que ejecutar el siguiente comando.

```
> npm i -g @vue/cli
```

Figura 8: Comando para la instalación del **CLI de Vue.js**

Una vez se ha instalado el componente, se puede crear la estructura del proyecto ejecutando uno de los comandos del **CLI**.

```
> vue create project-name
```

Figura 9: Comando para la creación de la estructura del proyecto con **Vue.js**

Con el proyecto creado, se pueden instalar los paquetes que se van a utilizar en el proyecto: **Vue Router** y **Axios**. Esto se hace con el siguiente comando.

```
> npm i -S vue-router axios
```

Figura 10: Comando para la instalación de las dependencias del proyecto *front-end*

1.2 Aplicación servidor

Como se ha especificado en el apartado 4. Lenguajes de programación y APIs utilizadas, se ha hecho uso de **Node.js** y **MongoDB**, por lo que es necesario instalar estos programas en la máquina de desarrollo. En el apartado anterior ya se ha descrito como hay que realizar la instalación de **Node.js**, así que a continuación solo se detallará la instalación del sistema de base de datos **MongoDB**.

Para su instalación, hay que ir a la página oficial y bajar el instalador. Una vez descargado, hay que ejecutarlo y seguir las indicaciones que se muestran en pantalla.

Una vez se haya finalizado la instalación de **MongoDB**, se puede acceder al entorno sin ningún tipo de credenciales.

Para facilitar el despliegue de la aplicación y que la formación de la *connection string* sea uniforme y se realice de la misma forma tanto en el entorno de producción como en el de desarrollo, se tuvo que configurar el sistema de **MongoDB** para la creación de un usuario administrador y los permisos de este.

Para realizar esta tarea, hay que ejecutar los siguientes comandos en una terminal del sistema.

```
> mongod  // Para levantar el servicio de base de datos
> mongo   // Para conectarse al sistema de base de datos
> use admin  // Para usar la base de datos admin
> db.createUser(  // Creación de usuario administrador
    {
      user: "myUserAdmin",
      pwd: "abc123",
      roles: ["userAdminAnyDatabase", "dbAdminAnyDatabase", "readWriteAnyDatabase"]
    }
  )
```

Figura 11: Comandos para la creación de un usuario administrador en **MongoDB**

Una vez creado el usuario y dado permisos, se puede proceder a la creación de la *connection string*, la cual tiene que tener el siguiente formato:

```
mongodb://<username>:<password>@<host>:<port>/<database>
```

Figura 12: Estructura de la *connection string* de **MongoDB**

Es importante destacar que para que esta configuración funcione en el entorno de desarrollo, hay que concatenar la *connection string* con las opciones de conexión.

```
?authSource=admin
```

Figura 13: Opciones de configuración de la conexión a una base de datos con **MongoDB**

Para que toda esta configuración funcione correctamente, es necesario indicar que el sistema de base de datos tiene que utilizar autenticación en el momento de iniciar el servicio. Esto puede hacerse modificando el primer comando que se muestra en la Figura 11: Comandos para la creación de un usuario administrador en **MongoDB**.

```
> mongod --auth
```

Figura 14: Comando para la inicialización del servicio de **MongoDB** usando autenticación

Dejando de lado la configuración del entorno de desarrollo, a falta de una herramienta que genere la estructura del proyecto para la implementación de una **API Rest**, se hizo una investigación sobre como estructurar el proyecto. Como resultado, se estructuró la aplicación de siguiente manera:

- **config**. Directorio donde se almacenan los ficheros de configuración de la base de datos, autenticación, logs y de **Express**.
- **controllers**. Directorio que almacena la implementación de los servicios expuestos.
- **middleware**. Directorio que almacena la implementación de las funciones middleware que se usan con **Express**.
- **models**. Directorio que almacena las definiciones de los esquemas que forman la base de datos.
- **.env**. Fichero que almacena la configuración de la aplicación.
- **server.js**. Fichero que contiene el programa principal que se encarga del arranque de la **API Rest**.

Para la implementación de la aplicación servidor se han utilizado los siguientes paquetes:

- **Axios**. Paquete utilizado para hacer llamadas a las **APIs de terceros** desde el servidor.
- **Bcryptjs**. Paquete utilizado para calcular el *hash* de la contraseña almacenada en la base de datos.
- **Body parser**. Paquete utilizado para *parsear* el cuerpo de la petición a formato JSON.
- **CORS**. Paquete utilizado para habilitar las llamadas desde dominios distintos al dominio donde está desplegada la aplicación servidor.
- **Dotenv**. Paquete que se utiliza para cargar la configuración del fichero ***.env***.
- **Express**. Paquete que contiene el framework **Express**.
- **HTTP Status Codes**. Paquete que contiene las definiciones de los códigos HTTP utilizados para las respuestas.
- **JSON Web Token**. Paquete utilizado para generar los tokens de autenticación/autorización.
- **MD5**. Paquete que se utiliza para calcula el *hash* necesario para las peticiones desde el servidor a la **API de Marvel**.
- **Mongoose**. Paquete utilizado para crear las definiciones de los modelos de la base de datos, así como la conexión a la misma.
- **Morgan**. Paquete que se utiliza para generar los *logs* de acceso.
- **Winston**. Paquete que se utiliza para generar los *logs* de la aplicación.

Adicionalmente, se utilizan otros paquetes que son exclusivos para el proceso de desarrollo:

- **Jest**. Es una librería de **Facebook** que permite añadir pruebas unitarias y de integración a un proyecto.
- **Node Mocks HTTP**. Paquete que se encarga de crear peticiones y respuestas falsas (*mock data*) para ser utilizadas en las pruebas de integración con **Express**.

- **Supertest**. Paquete que permite realizar pruebas de integración a los *endpoints* de la **API Rest.**

Las últimas tres dependencias se han instalado en el proyecto para habilitar la creación y comprobación de la funcionalidad de la implementación de los servicios y del middleware.

La definición de estos test se puede encontrar a cada fichero con extensión ***test.js*** que acompaña a cada fichero dentro de los directorios ***controllers*** y ***middleware***.

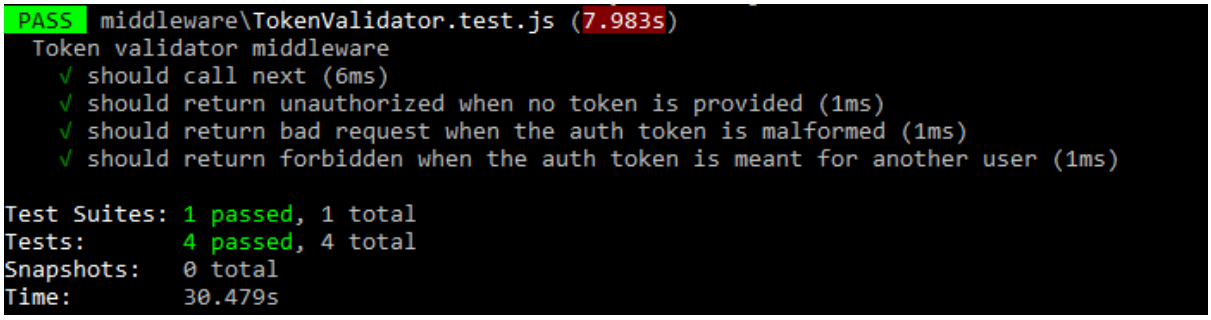

Figura 15: Resultado de la ejecución de los *tests* del middleware de validación de tokens.

2. Autenticación

Un aspecto importante del proyecto es la autenticación que se debe realizar en la aplicación cliente para poder usar los servicios de la aplicación servidor.

En un principio se realizó un desarrollo de una aplicación demo que hacía uso de la autenticación de **Google** para ganar acceso a la aplicación.

A pesar de que en esta aplicación se consiguió realizar correctamente la implementación de este método de autenticación, esta implementación no se consiguió trasladar a la implementación de la aplicación cliente. El factor más importante que imposibilitó la traslación fue la diferencia de estructura entre la aplicación demo y la aplicación cliente final.

La aplicación demo era una aplicación donde tanto el *back-end* como el *front-end* convivían en el mismo proyecto y como resultado de las peticiones se devolvía la página entera. Sin embargo, la aplicación cliente que se ha implementado se trata de una **SPA** (*Single Page Application*) y la cual hace uso del *back-end* para la obtención de la información en formato **JSON** y el flujo de la misma dependerá de los datos que reciba como respuesta.

Para corregir esta imposibilidad, se ha implementado un proceso de autenticación local, donde las credenciales se guardan en la base de datos. Como se ha comentado en el apartado anterior, se utiliza un paquete para almacenar el *hash* de la contraseña para evitar guardar la misma en formato texto plano.

En este modelo de autenticación, se utilizan tokens que se envían en las cabeceras de las peticiones para comprobar la identidad y los permisos de acceso. Para la obtención de los tokens, se han expuesto dos servicios: uno para el registro del usuario y otro para su autenticación.

Una vez se ha obtenido el token de acceso, este se guarda en la aplicación cliente la cual desde ese momento en adelante lo utiliza para realizar peticiones a la aplicación servidor.

Es importante destacar que, por razones de seguridad, en caso de robo de credenciales, la duración de validez del token es de 24 horas, que es un tiempo ni muy largo, en caso de que se haya producido un robo, ni muy corto para el uso por parte del usuario.

Este cambio en el método de autenticación se tuvo en cuenta debido a que aún quedaba funcionalidad por ser implementada y no parecía sensato invertir más tiempo del que ya se había invertido.

Capítulo 5: Despliegue

Una vez se ha finalizado el proceso de desarrollo, se procedió con el despliegue de ambas aplicaciones para que estén disponibles para su corrección.

1. Aplicación cliente

Para el despliegue de la aplicación cliente, se ha escogido la plataforma **Firebase**, de **Google**. Se ha escogido esta plataforma debido a que se había utilizado con anterioridad y es sencilla de utilizar.

Para el despliegue de la aplicación, es necesario tener instalado el **CLI de Firebase**, el cual se realiza con el siguiente comando.

```
> npm i -g firebase-tools
```

Figura 16: Comando de instalación del **CLI de Firebase**

Una vez finalizada la instalación, hay que iniciar **Firebase** en la aplicación y configurarla para utilizarlo como método de *hosting*. La configuración que se debe escoger se puede ver en la Figura 17: Configuración de **Firebase** para el despliegue de la aplicación cliente.

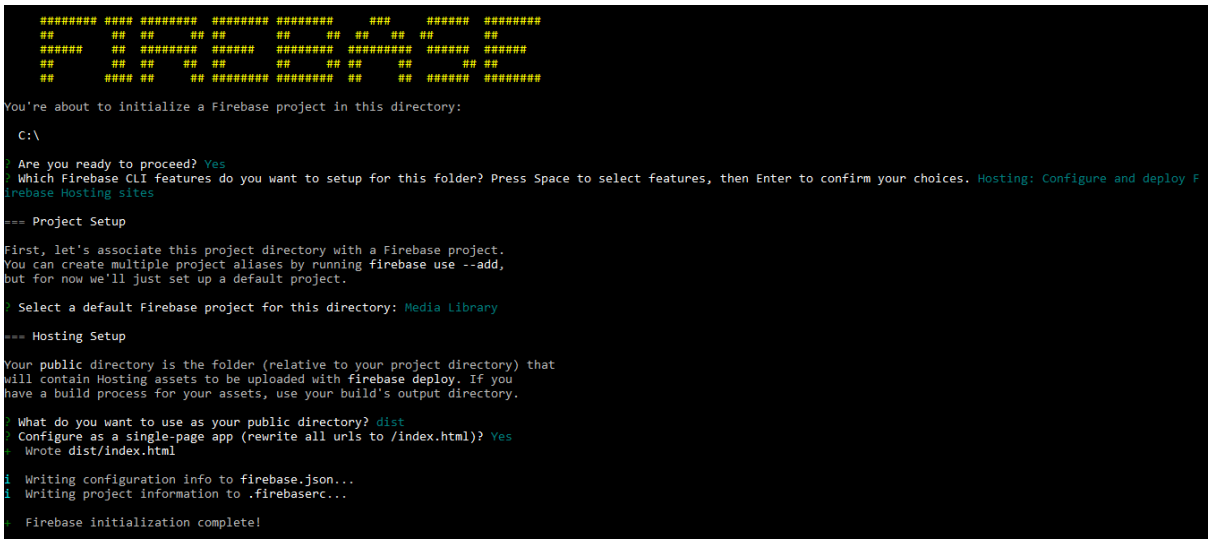

Figura 17: Configuración de **Firebase** para el despliegue de la aplicación cliente

Es importante utilizar como directorio público el directorio ***dist*****,** debido a que es el directorio que genera el **CLI de Vue.js** cuando genera el código que se sube a producción, y configurar la aplicación como una **SPA** para la reescritura de todas las URLs.

Los comandos que se tiene que ejecutar en una terminal del sistema para realizar el despliegue, son los siguientes:

```
> firebase init  // Inicia Firebase en el proyecto
> npm run build  // Genera los ficheros de producción
> firebase deploy  // Despliega los ficheros generados en la plataforma Firebase
```

Figura 18: Comandos para el despliegue de la aplicación cliente

La aplicación cliente se encuentra disponible en el siguiente enlace.

2. Aplicación servidor

Para el despliegue de la aplicación servidor se optaron por dos plataformas: **Google Cloud** y **Heroku**. Se buscón información sobre cómo realizar el despliegue en ambas plataformas y se escogió **Heroku** por tener un proceso más sencillo. Además, junto a **Heroku** se ha utilizado el servicio **mLab** para el *hosting* de la base de datos.

Para el despliegue de la aplicación, primero se tuvo que crear una cuenta y el proyecto en la plataforma e instalar el **CLI de Heroku** en la máquina de desarrollo.

Antes de desplegar la aplicación, es necesario definir las variables de entorno de esta en la configuración del proyecto en el panel de configuración de la plataforma.

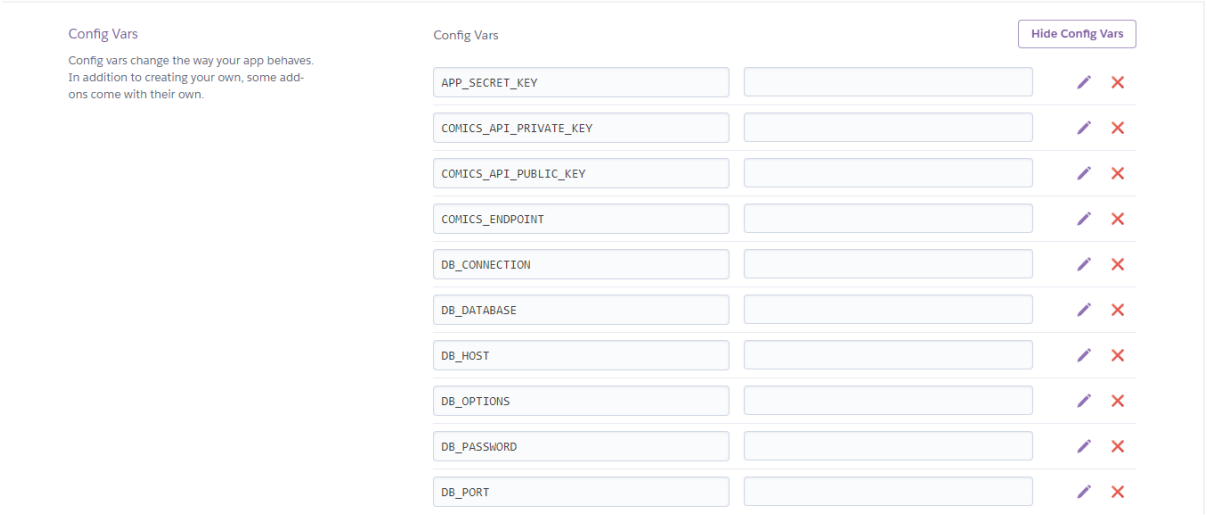

Figura 19. Pantalla de configuración de las variables de entorno de la aplicación servidor

Una vez se haya establecido la configuración, hay que ejecutar la siguiente secuencia de comandos para realizar el despliegue.

```
> heroku login  // Inicia sesión en la plataforma
> heroku git:remote -a medialibrary-jy  // Añade un origen a git para el despliegue
> git push heroku master  // Despliega la aplicación en Heroku
```

Figura 20: Comandos para el despliegue de la aplicación servidor

La aplicación servidor se encuentra disponible en el siguiente enlace.

Capítulo 6: Conclusiones y líneas de futuro

1. Conclusiones

Del desarrollo del proyecto se ha aprendido lo vital que resulta hacer una buena planificación, pensar bien todos los aspectos que incluye, sin menospreciar el trabajo que puede conllevar y respetarla.

El producto que se entrega cumple con las funcionalidades básicas que se quería entregar en un principio, pero no con la funcionalidad adicional que se especificó. Esto se debió principalmente a retrasos en los desarrollos y el desplazamiento en el tiempo de estos y porque no se añadió a la planificación el desarrollo de funcionalidades que tenían un factor decisivo en la aplicación.

A pesar de los inconvenientes que se han encontrado por el camino, termino contento con el resultado debido a que, en este tiempo, y gracias a la elaboración de este proyecto, he aprendido e interiorizado mucho conocimiento sobre el desarrollo web que hace uso de las últimas novedades en tecnología.

Como he comentado al principio, he aprendido la importancia de una buena planificación. A pesar de haber cometido errores en esta y de haber sido muy optimista en algunas tareas, me ha ayudado a pensar en una planificación más realista, al igual que en los objetivos marcados.

Por último, he contemplado como una idea inicial puede ir cambiando y evolucionando teniendo en cuenta los conflictos que se encuentra en el desarrollo. En este caso, se ha tenido que introducir ciertos cambios, por ejemplo en la autenticación, para garantizar que el proyecto llegara a buen puerto y con las funcionalidades que se querían ofrecer.

2. Líneas de futuro

Al presente proyecto se le puede dar continuidad de diversas maneras, siendo una de ellas añadir la funcionalidad que se marcó en los objetivos secundarios, el cual quedó fuera del desarrollo por falta de tiempo.

Otras mejoras que se pueden realizar es la introducción de más puntos de obtención de datos, para nutrir la aplicación de datos que las fuentes actuales puedan no tener o la introducción de catálogos de música y videojuegos.

Actualmente, la inteligencia artificial está siendo muy relevante para dotar de más personalización a una aplicación, por lo que podría ser otro punto de partida.

Como se ha dicho al principio, las opciones son muy diversas y las opciones que se han nombrado aquí solo son unas pocas.

Bibliografía

Bootstrap. (s.f.). Obtenido de Bootstrap: https://getbootstrap.com/docs/4.1/getting-started/introduction/

Cadenhead, T. (s.f.). *tysoncadenhead*. Obtenido de tysoncadenhead: http://www.tysoncadenhead.com/blog/where-should-i-put-javascript-unit-tests/

Ceddia, D. (19 de 9 de 2017). *daveceddia*. Obtenido de daveceddia: https://daveceddia.com/snapshot-testing-apis-with-jest/

Deutsch, D. (20 de 5 de 2017). *Medium*. Obtenido de Medium: https://medium.com/of-all-things-tech-progress/starting-with-authentication-a-tutorial-with-node-js-and-mongodb-25d524ca0359

El Economista. (4 de Septiembre de 2017). *El Economista*. Obtenido de El Economista: http://www.eleconomista.es/tecnologia-internet/noticias/8585192/09/17/Un-tercio-de-los-espanoles-compra-contenidos-digitales.html

Express. (s.f.). Obtenido de Express: http://expressjs.com/es/

Fontawesome. (s.f.). Obtenido de Fontawesome: https://fontawesome.com/icons

González, J. C. (16 de 12 de 2015). *Xataka Android*. Obtenido de Xataka Android: https://www.xatakandroid.com/aplicaciones-android/android-seriefilos-estas-son-las-cinco-mejores-aplicaciones-para-seguir-vuestras-series

González, J. C. (14 de 1 de 2016). *Xataka Android*. Obtenido de Xataka Android: https://www.xatakandroid.com/aplicaciones-android/movie-mate-una-app-muy-completa-para-gestionar-las-peliculas-que-vemos-y-descubrir-nuevas

Google. (s.f.). *Google Books APIs*. Obtenido de Google Books APIs: https://developers.google.com/books/docs/overview?hl=es-419

Google. (s.f.). *Google Fonts*. Obtenido de Google Fonts: https://fonts.google.com/?selection.family=Roboto

Holland, B. (8 de 3 de 2018). *Medium*. Obtenido de Medium: https://medium.freecodecamp.org/heres-how-you-can-actually-use-node-environment-variables-8fdf98f53a0a

Indeed. (2018). Obtenido de Indeed: https://www.indeed.es/salaries/Desarrollador/a-junior-Salaries?period=monthly

Indeed. (2018). Obtenido de Indeed: https://www.indeed.es/salaries/Dise%C3%B1ador/a-gr%C3%A1fico/a-Salaries

Kariuki, J. (24 de 7 de 2017). *scotch.io*. Obtenido de scotch.io: https://scotch.io/tutorials/nodejs-tests-mocking-http-requests

Kundel, D. (11 de 8 de 2017). *twilio Blog*. Obtenido de twilio Blog: https://www.twilio.com/blog/2017/08/working-with-environment-variables-in-node-js.html

Libib. (n.d.). *Libib*. Retrieved from Libib: https://www.libib.com/

Lombard, J. (6 de 5 de 2017). *Medium*. Obtenido de Medium: https://medium.com/@JeffLombardJr/organizing-tests-in-jest-17fc431ff850

Marketing Directo. (4 de Septiembre de 2017). *Marketing Directo*. Obtenido de Marketing Directo: https://www.marketingdirecto.com/digital-general/digital/tercio-los-espanoles-compra-contenidos-digitales

Marvel. (s.f.). *Marvel*. Obtenido de Marvel: https://developer.marvel.com/

MongoDB. (s.f.). Obtenido de MongoDB: https://docs.mongodb.com/manual/

MongoDB. (s.f.). *MongoDB*. Obtenido de MongoDB: https://docs.mongodb.com/manual/reference/connection-string/

MongoDB. (s.f.). *MongoDB*. Obtenido de MongoDB: https://docs.mongodb.com/manual/reference/method/db.updateUser/

mongoose. (s.f.). Obtenido de mongoose: http://mongoosejs.com/

Nick Uraltsev, M. Z. (s.f.). *NPM*. Obtenido de NPM: https://www.npmjs.com/package/axios

Node.js. (s.f.). Obtenido de Node.js: https://nodejs.org/es/

NPM. (s.f.). Obtenido de NPM: https://www.npmjs.com/

NPM. (2018). Obtenido de NPM: https://www.npmjs.com/package/node-mocks-http

Observatorio Nacional de Telecomunicaciones y Sociedad de la Información . (Diciembre de 2017). *Observatorio Nacional de Telecomunicaciones y Sociedad de la Información* . Obtenido de Observatorio Nacional de Telecomunicaciones y Sociedad de la Información : http://www.ontsi.red.es/ontsi/es/content/informe-anual-del-sector-de-los-contenidos-digitales-en-españa-edición-2017

OMDb API. (s.f.). Obtenido de OMDb API: http://www.omdbapi.com/

Play, G. (8 de 4 de 2018). *Google Play*. Obtenido de Google Play: https://play.google.com/store/apps/details?id=com.moviematelite&hl=es_419

Quora. (s.f.). Obtenido de Quora: https://www.quora.com/How-can-I-integrate-data-from-an-API-into-an-SQL-database

Raboy, N. (6 de 11 de 2017). *thepolyglotdeveloper*. Obtenido de thepolyglotdeveloper: https://www.thepolyglotdeveloper.com/2017/11/handling-cors-express-framework-nodejs-application/

Rahić, A. (5 de 3 de 2017). *Medium*. Obtenido de Medium: https://hackernoon.com/restful-api-design-with-node-js-26ccf66eab09

Rahić, A. (4 de 9 de 2017). *Medium*. Obtenido de Medium: https://medium.freecodecamp.org/securing-node-js-restful-apis-with-json-web-tokens-9f811a92bb52

Sergent, H. T. (1 de 2 de 2017). *Runscope Blog*. Obtenido de Runscope Blog: https://blog.runscope.com/posts/6-common-api-errors

Stackoverflow. (25 de 8 de 2017). Obtenido de Stackoverflow: https://stackoverflow.com/questions/34499544/storing-amazon-api-data-in-local-database

Sweney, M. (5 de 1 de 2017). *The Guardian*. Obtenido de The Guardian: https://www.theguardian.com/media/2017/jan/05/film-and-tv-streaming-and-downloads-overtake-dvd-sales-for-first-time-netflix-amazon-uk

Thibaud. (18 de 1 de 2018). *sqreen*. Obtenido de sqreen: https://blog.sqreen.io/authentication-best-practices-vue/

Trakt. (n.d.). *Trakt*. Retrieved from Trakt: https://trakt.tv/about

Trottmann, U. (s.f.). *SeriesGuide*. Obtenido de SeriesGuide: https://seriesgui.de/

tutorialspoint. (s.f.). Obtenido de tutorialspoint: https://www.tutorialspoint.com/mongodb/index.htm

Vue Router. (s.f.). Obtenido de Vue Router: https://router.vuejs.org/

Vynogradenko, A. (27 de 12 de 2017). *GithubGist*. Obtenido de GithubGist: https://gist.github.com/Restuta/cda69e50a853aa64912d

Whakoom. (s.f.). *Whakoom*. Obtenido de Whakoom: https://www.whakoom.com/features

Wikipedia. (9 de 3 de 2013). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/BSON

Wikipedia. (15 de 4 de 2018). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/MongoDB

Xplenty. (28 de 9 de 2017). *Medium*. Obtenido de Medium: https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2

You, E. (2018). *Vue.js*. Obtenido de Vue.js: https://vuejs.org/

Anexos

Anexo A: Glosario

- **NAS**: *Network Attached Storage*
- **API**: *Application Programming Interface*
- **Endpoint**: URL de un servicio expuesto.
- **CLI**: *Command-line Interface*
- **SPA**: *Single Page Application*

Anexo B: Entregables del proyecto

Junto a este documento, se entregan los siguientes recursos:

- Prototipo para dispositivos móviles
- Prototipo para dispositivos de escritorio
- Código de la aplicación cliente
- Código de la aplicación servidor
- Presentación académica
- Presentación publica

Anexo C: Capturas de pantalla

En este anexo se incluirán capturas de pantalla de la aplicación que se ha desarrollado.

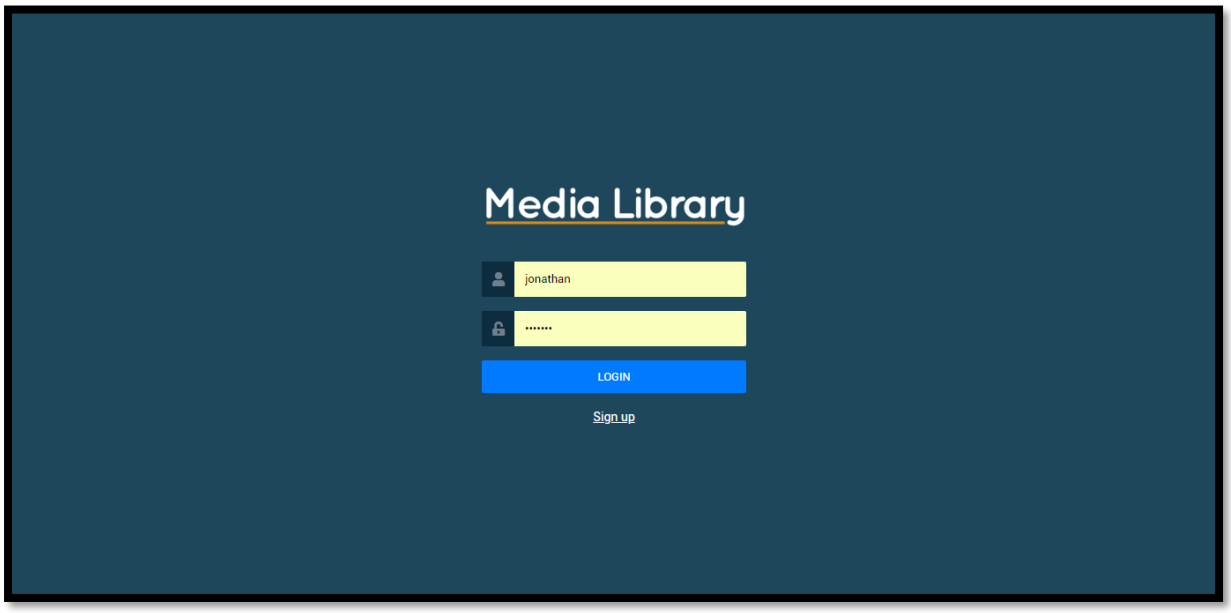

Figura 21: Página de inicio de sesión en un ordenador de escritorio

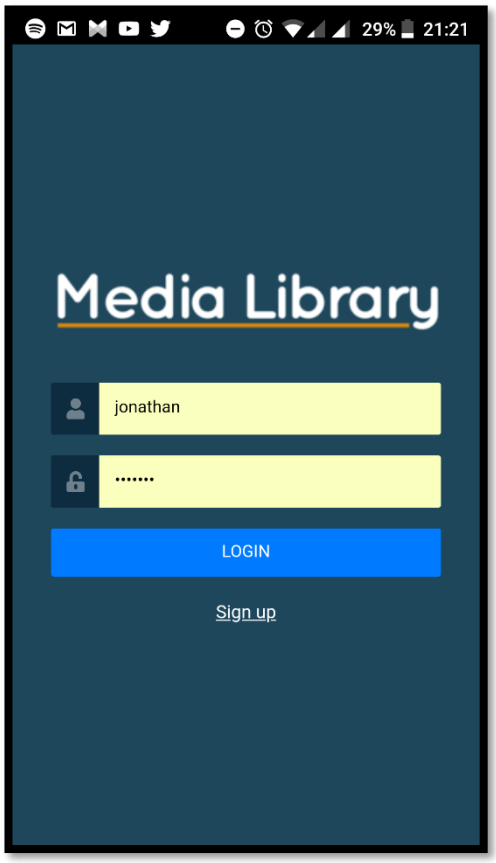

Figura 22: Página de inicio de sesión en un dispositivo móvil

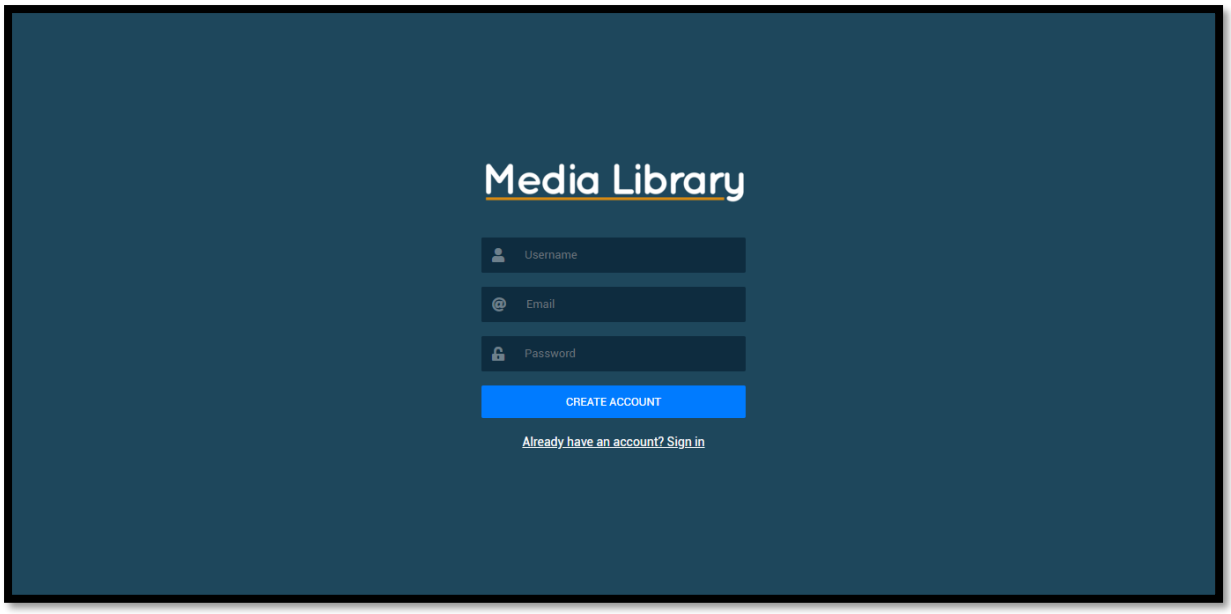

Figura 23: Página de registro en un ordenador de escritorio

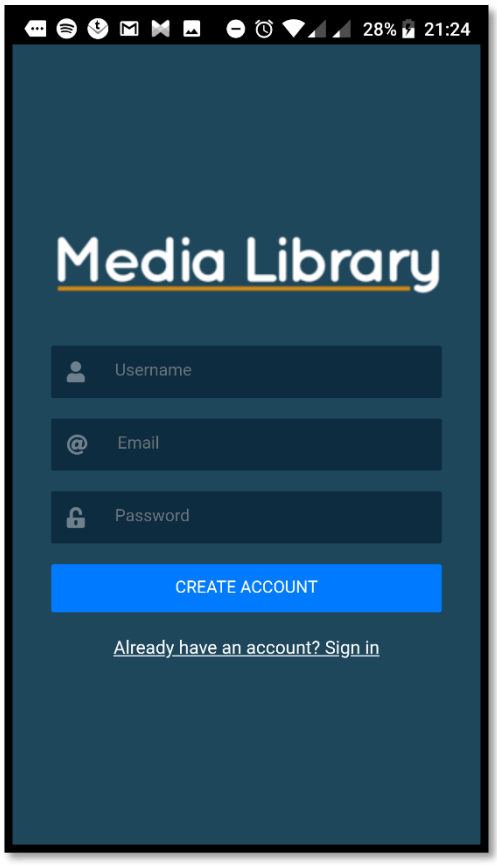

Figura 24: Página de registro en un dispositivo móvil

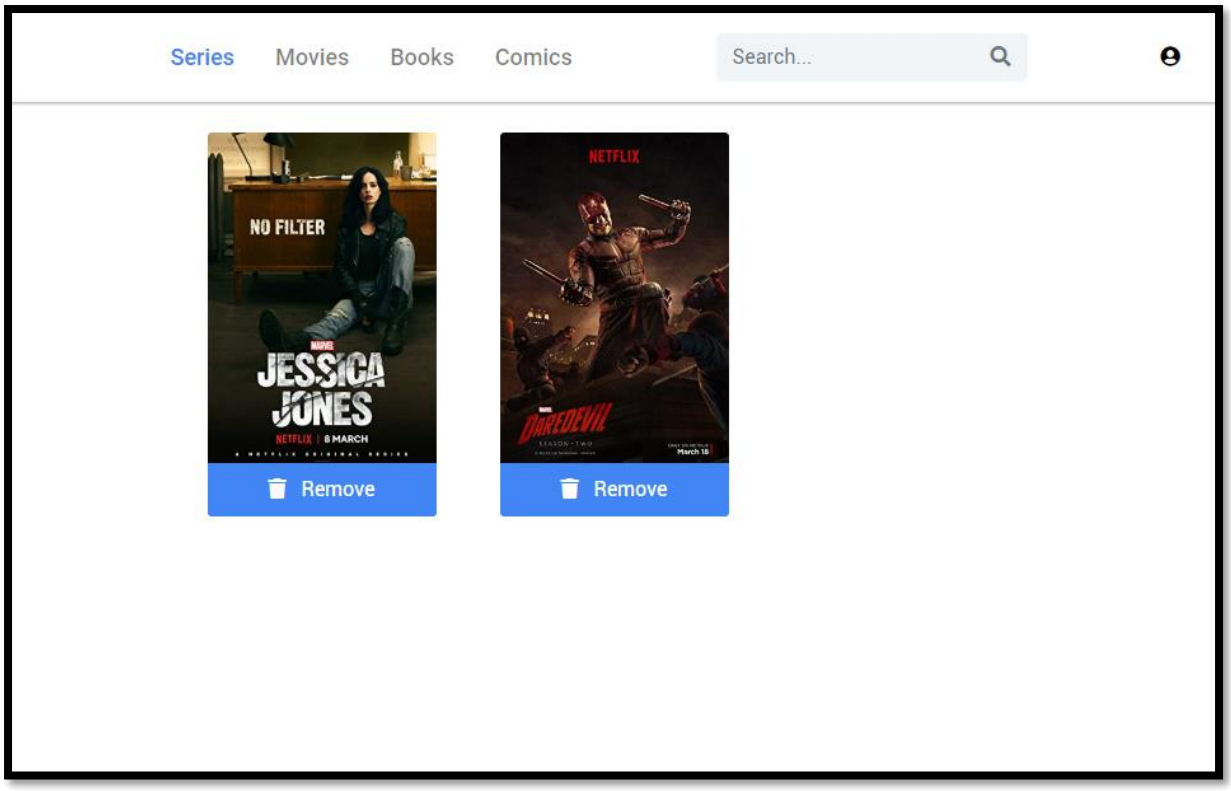

Figura 25: Página principal en un ordenador de escritorio

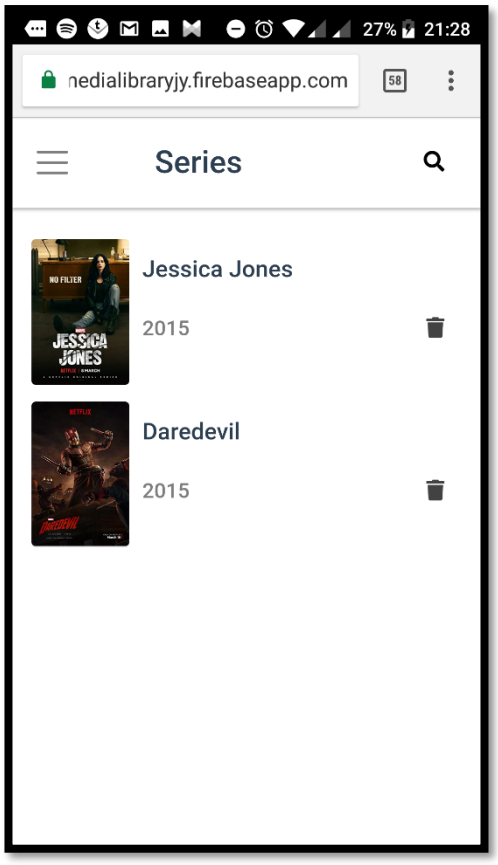

Figura 26: Página principal en un dispositivo móvil

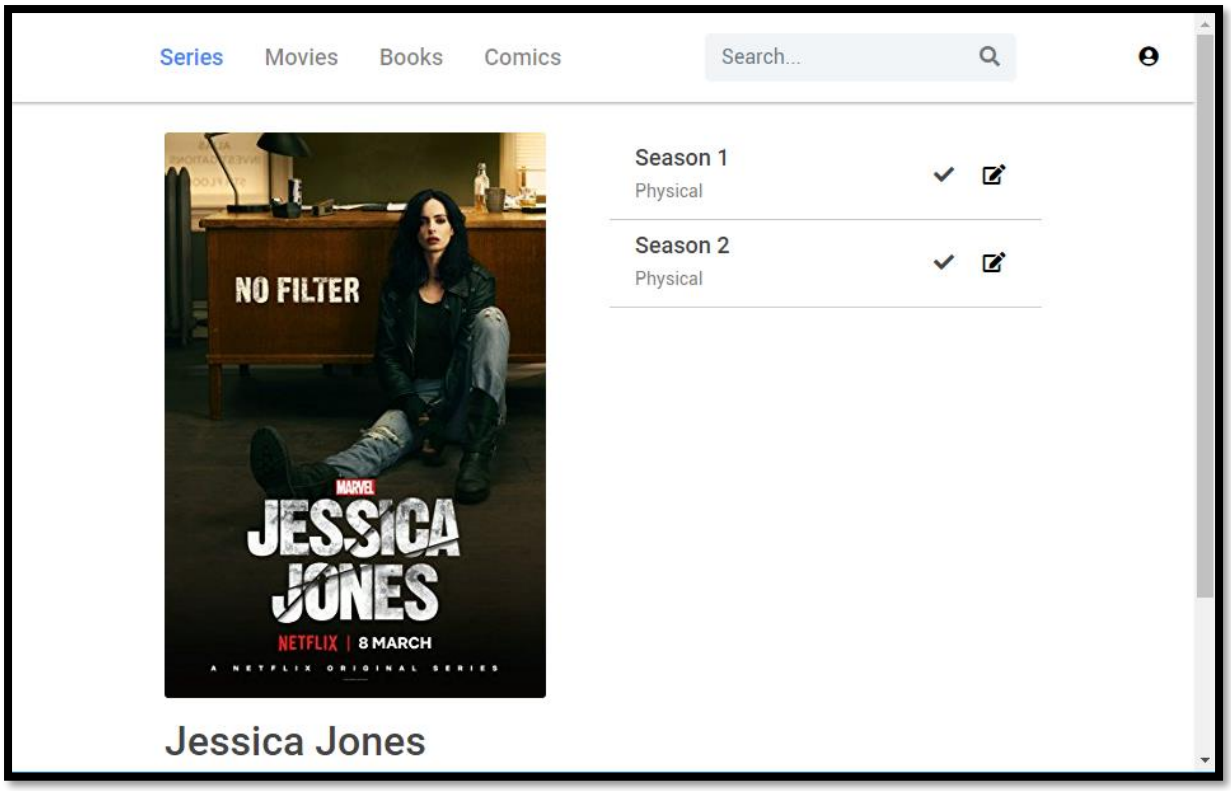

Figura 27: Página de información de una serie en un ordenador de escritorio

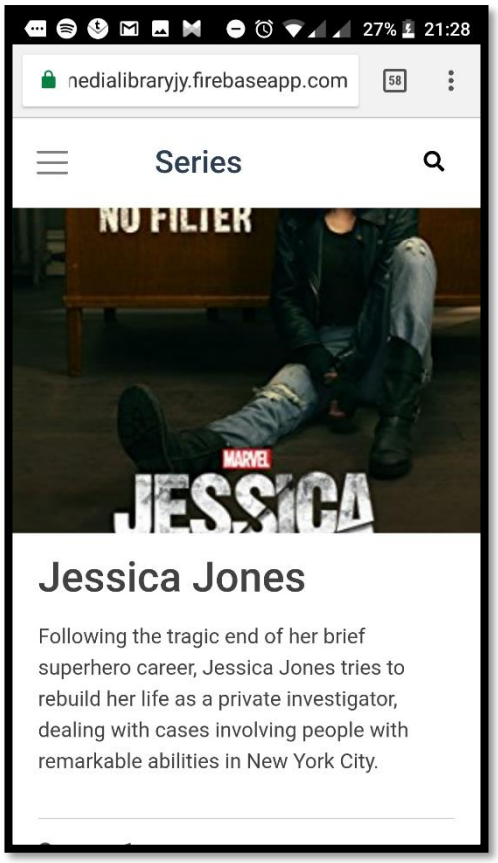

Figura 28: Página de información de una serie en un dispositivo móvil

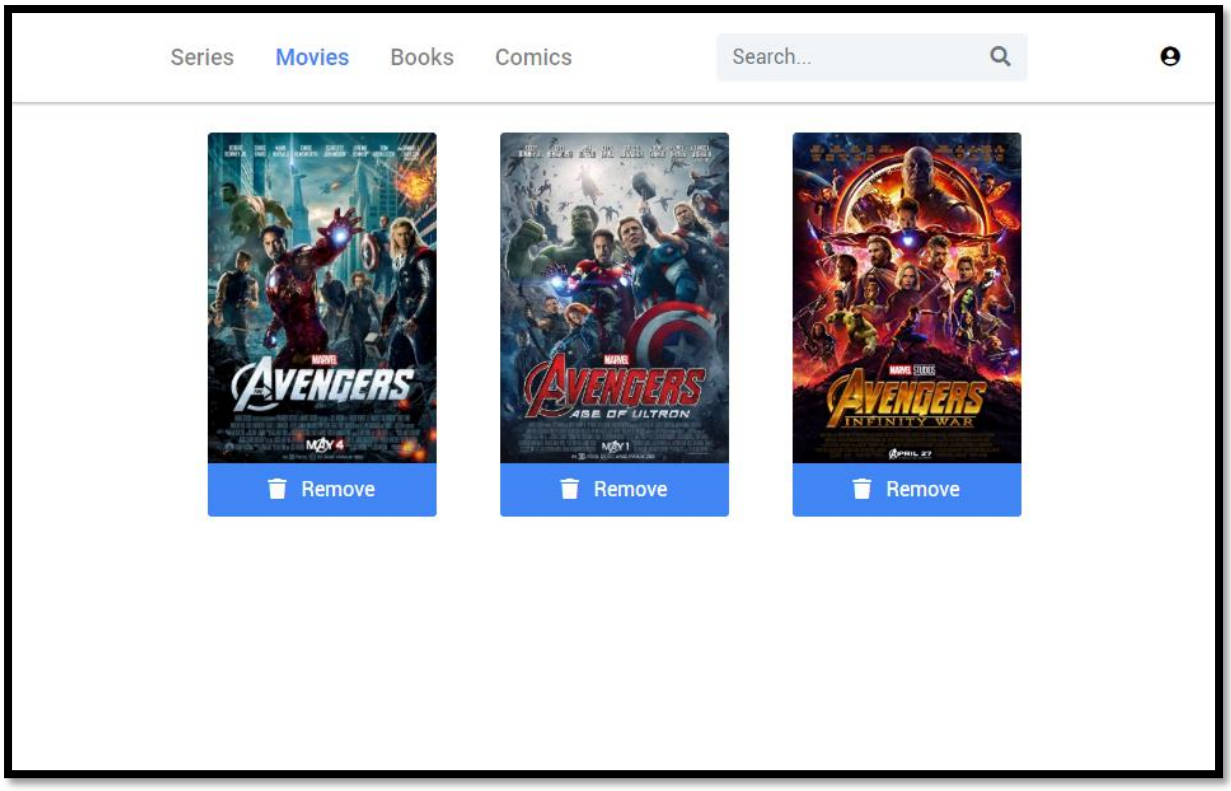

Figura 29: Página del catálogo de películas en un ordenador de escritorio

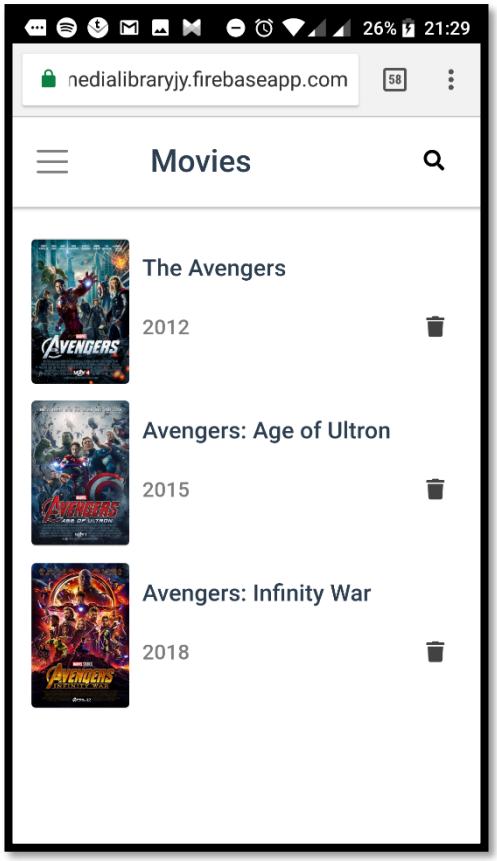

Figura 30: Página del catálogo de películas en un dispositivo móvil

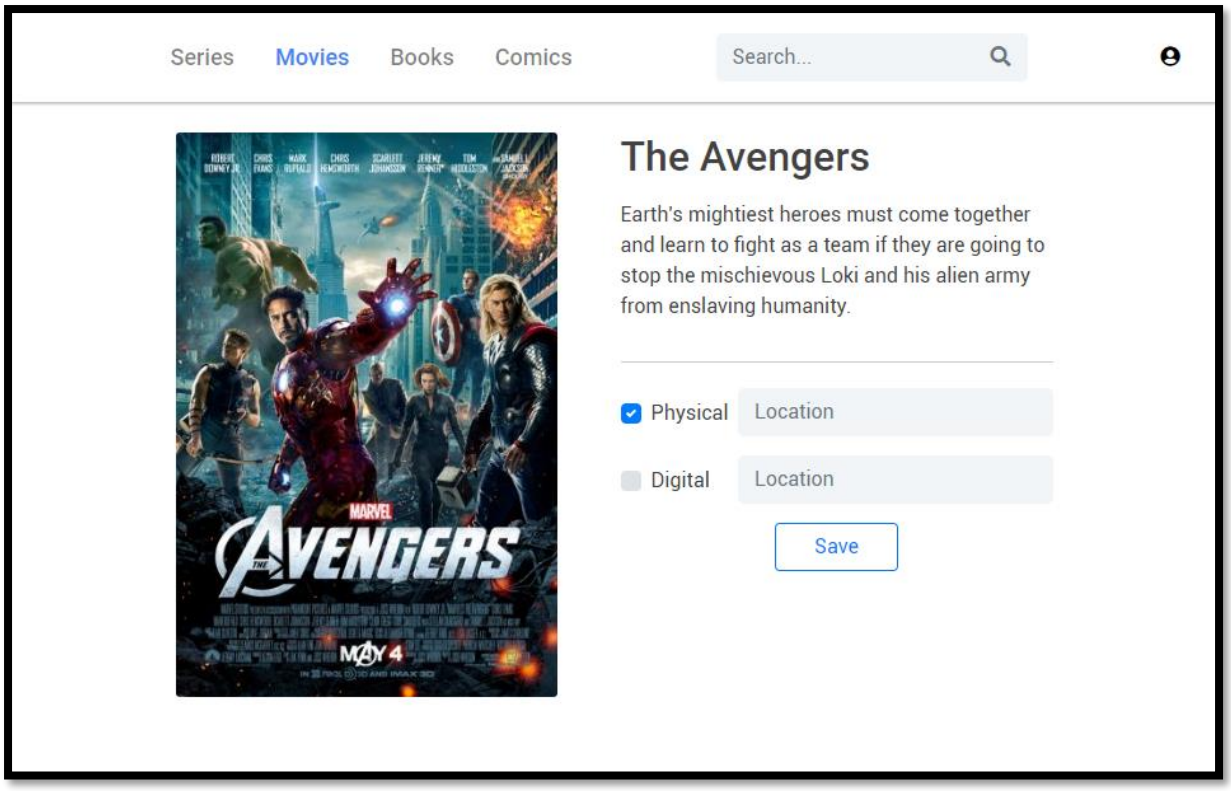

Figura 31: Página de información de una película en un ordenador de escritorio

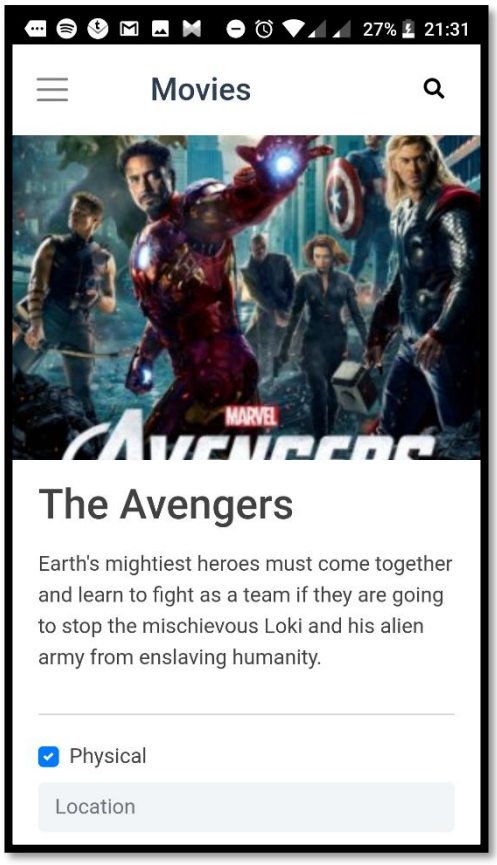

Figura 32: Página de información de una película en un dispositivo móvil