PROYECTO INSTAFILMS

Carlos Andrés García,
Douglas Abraham Rodríguez,
Germán Gutiérrez
& Carlos Pérez

Enero 2019.

Universidad Europea del Atlántico.
Ingeniería Informática.
Programación Web I

Copyright © 2019 por Carlos Andrés Gutiérrez, Douglas Abraham Rodríguez, Germán Gutiérrez y Carlos Pérez. Todos los derechos reservados.

Abstract

Los grandes avances en el ámbito de las Redes Sociales, así como la radical disminución del número de portales web dedicados al ámbito cinematográfico hacen que surja la idea de Instafilms, una red social de películas. Portales web como Netflix o Amazon Prime Video son punteros en este sector, sin embargo, Instafilms va más allá. Permite al usuario interactuar con otros dentro de la misma plataforma, lo cual supone un gran avance en cuanto a la calidad del servicio prestado. Instafilms reúne las características más novedosas de las mejores Redes Sociales de la actualidad, lo cual hace que sea un sistema intuitivo y fácil de manipular, dos de las características que más ansía un usuario a la hora de buscar un portal web.

Tabla de Contenidos

Capítulo 1
Diseño de la BD. ER. SQL de creación **4**
Almacenamiento de datos 4
Obtención de información 4

Capítulo 2
Arquitectura: Instancias o máquinas y comunicación entre ellas **8**

Capítulo 3
Diseño de la API: Métodos, parámetros de entrada y salida **11**

Capítulo 4
Previo de Mocks **23**

Capítulo 1

Diseño de la BD. ER. SQL de creación

Almacenamiento de datos

Para conformar su sólida base de datos, InstaFilms recoge información de películas proveniente de la API de 'The Movie Database'. Una vez extraída y utilizando IMDB para obtener la información deseada de cada película, estos datos son parseados e importados en una base de datos ,previamente creada y estructurada, en phpMyAdmin**.** Dicha base de datos contiene todas las películas que ofrece la plataforma, además de información de usuarios y gestión de los mismos.

Obtención de información

Como hemos mencionado anteriormente, obtendremos la información necesaria para rellenar la tabla de películas de nuestra base de datos a través de un script.

```
<script>

    var urlbase = "https://api.themoviedb.org/3/discover/
        movie?api_key=7dd6e501f6ab8c86a83fb262dddb6195&primary_release_year=";
    var urlbase2 = "http://www.omdbapi.com/?t=";
    var apikey = "&apikey=54d82eef";


    const populateArray = async () => {
        let filmlist = [];

        for (var i = 2000; i < 2020; i++) {
            var url = urlbase + i;

            //esperamos a que la funcion getJSON nos devuelva todos los results
            await $.getJSON(url, function (data) {
                $.each(data.results, function (index, film) {
                    let title = film.title;
                    title = title.replace(/ /g, "+");
                    filmlist.push(title);
                });
            });
        }

        console.log("Longitud de list en getJSON:", filmlist.length)
        return filmlist;
    };
```

Se define la variable ‘urlbase’, que contiene la dirección de la API de ‘The Movie Database’. Además, también definimos las variables ‘urlbase2’ y ‘apikey’, que serán necesarias para acceder correctamente a IMDB.

Con el método asíncrono ‘populateArray()’, definimos un array llamado ‘filmlist[]’, en donde almacenaremos los nombres de todas las películas. Con un bucle ‘for’, especificando en cada iteración el año deseado con la variable ‘i’, vamos rellenando dicho array, sustituyendo para cada nombre los espacios en blanco por ‘+’, lo cual es necesario para poder buscar las películas por el título en IMDB. Cabe destacar el uso de la función ‘getJSON’ junto con la propiedad ‘await’, que hace que el script espere a que terminemos de rellenar el array con los datos que nos da la API antes de continuar con su ejecución. Finalmente, el método nos devolverá el array ‘filmlist[]’ con todos los títulos de las películas.

```
const getInfo = async () => {
  const fullMovies = await populateArray();
  let infoMovies = [];

  for (var i=0; i<fullMovies.length; i++) {
    var url2 = urlbase2 + fullMovies[i] + apikey;

    // imdbID, titulo, ano, duracion, genero, director, escritor, actores, trama, poster
    await $.getJSON(url2, function (data) {
      var movie = new Object();
      movie.imdbid = data.imdbID;
      movie.title = data.Title;
      movie.year = data.Year;
      movie.runtime = data.Runtime;
      movie.genre = data.Genre;
      movie.director = data.Director;
      movie.writer = data.Writer;
      movie.actors = data.Actors;
      movie.plot = data.Plot;
      movie.poster = data.Poster;
      infoMovies.push(movie);
    });
  }
  return infoMovies;
};

getInfo();

</script>
```

El siguiente método a realizar, también asíncrono, es el método ‘getInfo()’. En él, se declara la constante ‘fullMovies’ llamando a la función ‘populateArray()’ con un array. Ésto guardará en dicha constante los nombres preparados para introducirlos en la url de IMDB junto con su API key y así volcar la información requerida de las películas. También declaramos un nuevo array bidimensional ‘infoMovies[][]’, el cual nos servirá para estructurar el JSON que importamos más adelante en nuestra base de datos. Con otra función ‘getJSON’ precedida de un ‘await’ vamos creando los campos que necesitamos, hasta hacer un ‘push’ de la película y así rellenar todo.

Finalmente, nuestro script llamará a la función ‘getInfo()’.

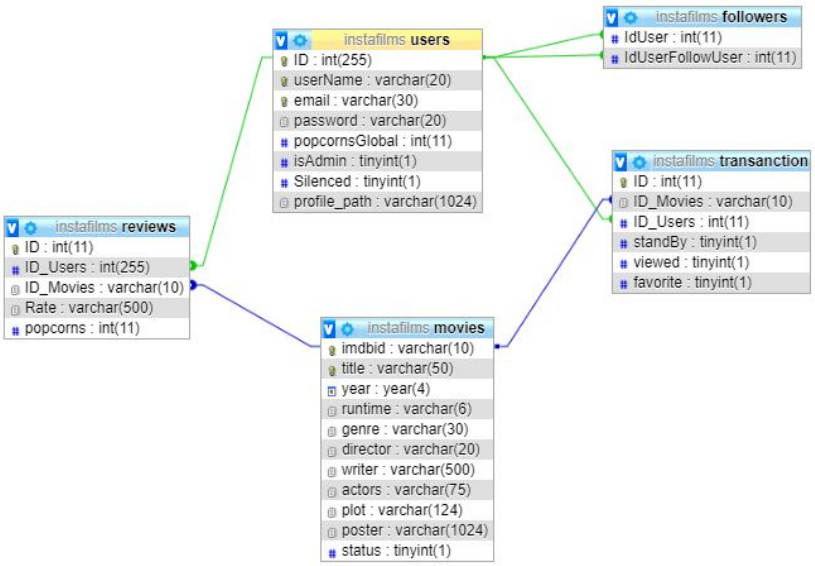
Una vez obtenidos los datos en formato JSON, utilizamos el ‘JSON Formatter’ para obtener un archivo físico que poder importar a la base de datos. phpMyAdmin sólo admite formato CSV, por lo que es necesario convertir nuestro ‘.json’ file en un ‘.csv’ file, para lo cual utilizaremos un conversor.

```
+[
+   {
+      "imdbid":"tt0217869",
+      "title":"Unbreakable",
+      "year":"2000",
+      "runtime":"106 min",
+      "genre":"Drama, Mystery, Sci-Fi, Thriller",
+      "director":"M. Night Shyamalan",
+      "writer":"M. Night Shyamalan",
+      "actors":"Bruce Willis, Samuel L. Jackson, Robin Wright, Spencer Treat Clark",
+      "plot":"A man learns something extraordinary about himself after a devastating accident."
+      "poster":"https://m.media-amazon.com/images/M/MV5BMDIwMjAxNzktNmEzYS00ZDY5LWEyZjktM2Y0MmU
+   },
+   {
+      "imdbid":"tt0209144",
+      "title":"Memento",
+      "year":"2000",
+      "runtime":"113 min",
+      "genre":"Mystery, Thriller",
+      "director":"Christopher Nolan",
+      "writer":"Christopher Nolan (screenplay), Jonathan Nolan (short story \"Memento Mori\")",
+      "actors":"Guy Pearce, Carrie-Anne Moss, Joe Pantoliano, Mark Boone Junior",
+      "plot":"A man with short-term memory loss attempts to track down his wife's murderer.",
+      "poster":"https://m.media-amazon.com/images/M/MV5BZTcyNjk1MjgtOWI3Mi00YzQwLWI5MTktMzY4ZmI
+   },
```


Éste es el aspecto que tendría nuestro archivo JSON tras pasar por el ‘JSON Formatter’.

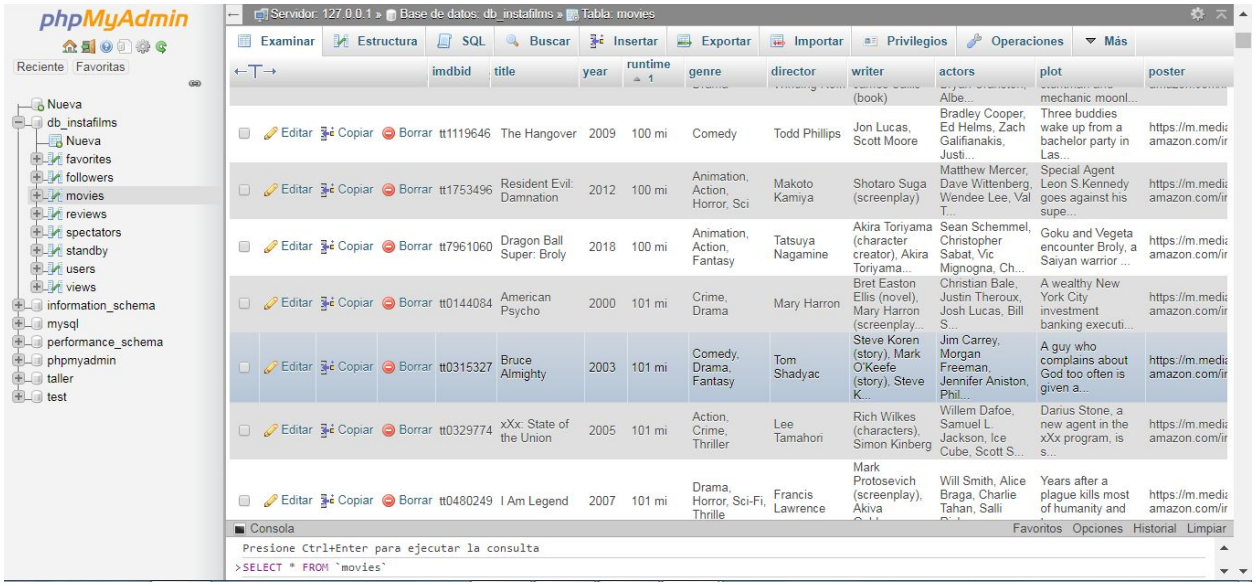
Base de datos

Como hemos mencionado con anterioridad, nuestra base de datos está estructurada en phpMyAdmin, presentando la siguiente vista global.



Cada usuario está identificado por su ‘id’, además de tener atributos como ‘username’, ‘email’, ‘password’, ‘popcornsGlobal’, ‘isAdmin’ y ‘silenced’. Cada usuario tendrá un conjunto de ‘followers’ y ‘spectators’, que serán todos los usuarios a los que siguen, y los que le siguen respectivamente. Además, cada usuario tiene un conjunto de ‘views’, que son las películas que ha visto en la plataforma, en las que podrá escribir ‘reviews’ para futuros usuarios, o añadirlas a ‘favorites’.

Cada película está identificada por el id de IMDB ‘imdbid’, además del título de la película, el año de producción, la duración, el género, el director, los escritores, los actores, la sinopsis y una imágen.

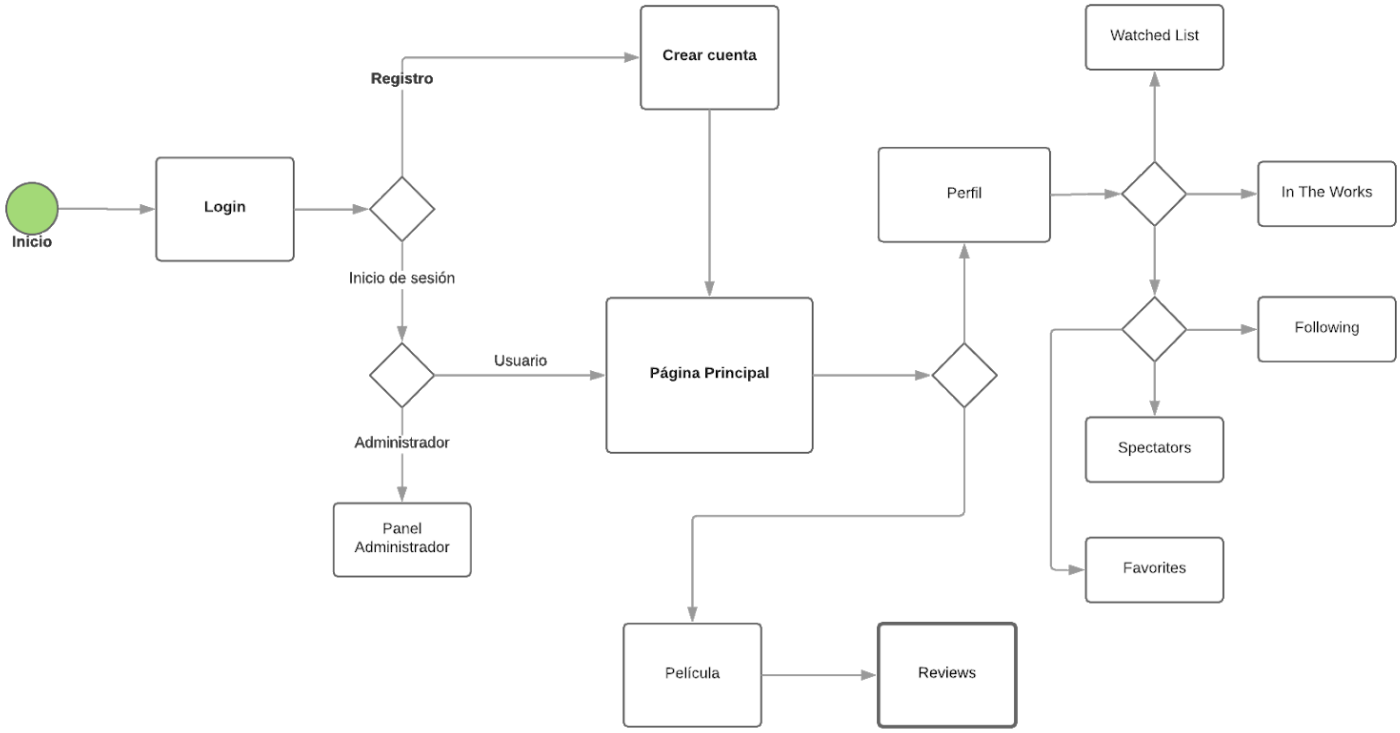


Capítulo 2

Arquitectura: Instancias o máquinas y comunicación entre ellas

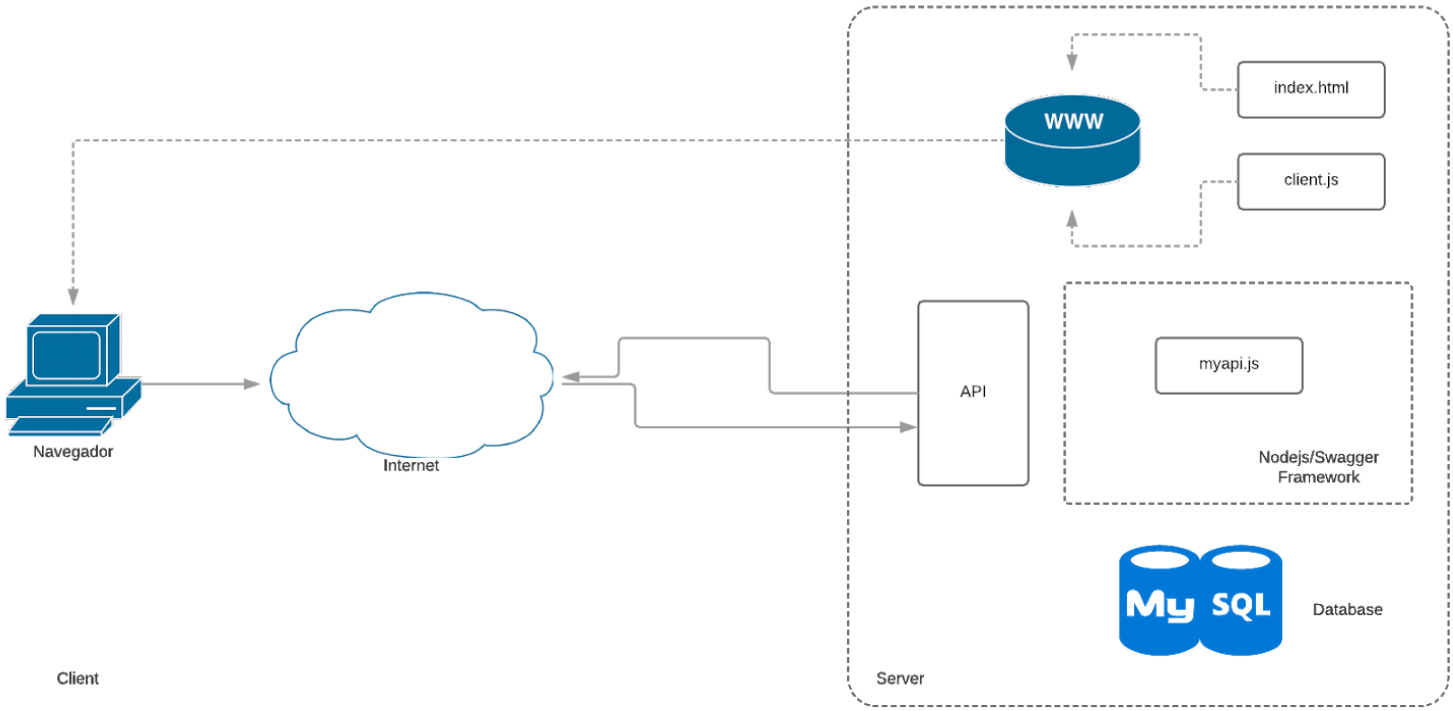
Diagrama de flujo

El diagrama de flujo representa gráficamente el flujo de rutinas simples dentro de nuestra plataforma web.



Arquitectura funcional

El diagrama de la arquitectura representa cómo se estructuran los diferentes elementos de la aplicación.



Capítulo 3

Diseño de la API: Métodos, parámetros de entrada y salida

En nuestro diseño de API contemplamos 4 elementos los cuales serán el centro de la API. ***Movie, User, Transaction y Review.***

Movie: el registro de cada película que se encuentre en nuestra DB.
Los métodos que tendemos serán:

- addMovie,el que se agrega una nueva película a la db.

POST /movie

Description

Add movie to db

Parameters

Name	Located in	Description	Required	Schema
movie	body	add movie	No	⇄ ▾Movie { IDimdb: ▸ string * title: ▸ string year: ▸ string runtime: ▸ string genre: ▸ string director: ▸ string writer: ▸ string actors: ▸ string plot: ▸ string poster: ▸ string status: ▸ boolean }

Responses

Code	Description	Schema
201	Success	⇄ ▾GetMovieResponse { IDimdb: ▸ string * title: ▸ string * year: ▸ string * runtime: ▸ string * genre: ▸ string * director: ▸ string * writer: ▸ string * actors: ▸ string * plot: ▸ string * poster: ▸ string * status: ▸ boolean * }

- getMovieById, retorna una película individualmente por su ID que en este caso es el de IMDB.

GET /movie/{id}

Description

Returns individual movie by id

Parameters

Name	Located in	Description	Required	Schema
id	path	Movie to be returned	Yes	string

Responses

Code	Description	Schema
200	Success	GetMovieResponse { IDimdb: string * title: string * year: string * runtime: string * genre: string * director: string * writer: string * actors: string * plot: string * poster: string * status: boolean * }

- updateMovieById, actualizar un campo de la película.

PUT /movie/{id}

Description

Updates movie by securityDefinitions

Parameters

Name	Located in	Description	Required	Schema
id	path	Movie id to update	Yes	string
movie	body	The movie to update.	No	Movie { }

Responses

Code	Description	Schema
200	Success	GetMovieResponse { }
default	Error	ErrorResponse { message: string * }

- getMovieLanding, nos devolverá el poster de la película correspondiente al ID de IMDB.

GET /movie/landing

Description

get the movie poster

Parameters

Name	Located in	Description	Required	Schema
id	query	Movie id	Yes	string

Responses

Code	Description	Schema
200	Success	GetMoviePosterResponse { IDimdb: string * poster: string * }

- getMovieTitle, nos devolverá el título de la película correspondiente al ID de IMDB.

GET /movie/title

Description

Returns titles (LIKE +"OCEANS"+)

Parameters

Name	Located in	Description	Required	Schema
id	query	Movie id	Yes	string

Responses

Code	Description	Schema
200	Success	GetMovieTitleResponse { title: string * }

Ahora pasamos a usar **Transactions** los cuales están asociadas tanto con el usuario y las películas, el usuario podrá agregarla a su lista de favoritas, vista o pendientes.

- addmovieFavorite, el usuario agrega la película a favoritas.

POST /movie/transactions

Description

AddTransaction

Parameters

Name	Located in	Description	Required	Schema
id	query	Id of the movie	Yes	string
user	query	Id of user	Yes	integer
transaction	query	initialize standBy, viewed or favorite	Yes	string

Responses

Code	Description	Schema
200	Success	Transanction { ID: integer, IDMovie: string, IDUser: integer *, standBy: integer, viewed: integer, favorite: integer }
default	Error	ErrorResponse { message: string * }

- getTransaction, se obtendrá el estado de la transacción del usuario con respecto a una película (Favoritos, Vista o Pendiente).

GET /movie/transactions

Description

get transaction of a user, (pending, watched, favorite)

Parameters

Name	Located in	Description	Required	Schema
id	query	Id of user	Yes	string

Responses

Code	Description	Schema
200	Success	GetMovieListResponse { Movies: [] }
default	Error	ErrorResponse { message: string * }

- updateTransaction, el usuario actualiza el estado de una película en su lista ya sea pasar a vista o favorita.

PUT /movie/transactions

Description

update a state of Transaction (viewed, fav, pending)

Parameters

Name	Located in	Description	Required	Schema
id	query	Id of the movie	Yes	string
user	query	Id of user	Yes	integer

Responses

Code	Description	Schema
200	Success	GetMovieListResponse { Movies: [] }
default	Error	ErrorResponse { message: string * }

- deleteTransaction, borra el registro.

DELETE /movie/transactions

Description

Delete a transaction

Parameters

Name	Located in	Description	Required	Schema
id	query	Id of the movie	Yes	string
user	query	Id of user	Yes	integer

Responses

Code	Description	Schema
200	Success	▾ SuccessResponse { message: string * }

Ahora los métodos para los usuarios.

- addUser, cuando se crea un usuario nuevo.

POST /user

Description

Add user to db

Parameters

Name	Located in	Description	Required	Schema
user	body	add user	No	⇄

```
▾User {
  ID:              ▸ integer *
  userName:        ▸ string *
  email:           ▸ string *
  password:        ▸ string *
  popcornsGlobal:  ▸ integer
  isAdmin:         ▸ integer
  silenced:        ▸ integer
  profilePath:     ▸ string
}
```

Responses

Code	Description	Schema
201	Success	⇄

```
▾GetUserResponse {
  ID:              ▸ integer *
  userName:        ▸ string *
  email:           ▸ string *
  password:        ▸ string *
  popcornsGlobal:  ▸ integer *
  isAdmin:         ▸ integer *
  silenced:        ▸ integer *
  profilePath:     ▸ string *
```

- getUserById, busca un usuario por su id.

GET /user/{id}

Description

user by id

Parameters

Name	Located in	Description	Required	Schema
id	path	Returns user	Yes	⇄ integer

Responses

Code	Description	Schema
200	Success	⇄

```
▾GetUserResponse {
  ID:              ▸ integer *
  userName:        ▸ string *
  email:           ▸ string *
  password:        ▸ string *
  popcornsGlobal:  ▸ integer *
  isAdmin:         ▸ integer *
  silenced:        ▸ integer *
  profilePath:     ▸ string *
}
```

- updateUser, el usuario puede actualizar su contraseña o su foto de perfil.

PUT /user/{id}

Description

Update user

Parameters

Name	Located in	Description	Required	Schema
id	path	user id	Yes	integer
user	body	The user to update.	No	User { ID: integer *, userName: string *, email: string *, password: string *, popcornsGlobal: integer, isAdmin: integer, silenced: integer, profilePath: string }

Responses

Code	Description	Schema
200	Success	SuccessResponse { message: string * }

Ahora entran las review (Críticas), las cuales realiza el usuario a una película.

- getReview, obtenemos todas las reviews de un usuario o de una película.

GET /review

Description

get all reviews

Parameters

Name	Located in	Description	Required	Schema
user	query	id User	Yes	integer
movie	query	id imdb movie	Yes	string

Responses

Code	Description	Schema
200	Success	GetReviewResponse { ID: integer * IDMovie: string * IDUser: integer * rate: string * popcorns: integer * }
default	Error	ErrorResponse { message: string * }

- addReview, se agrega una review.

POST /review

Description

add review of movie

Parameters

Name	Located in	Description	Required	Schema
id	query	Review Id	Yes	integer
user	query	id User	Yes	integer
movie	query	id imdb movie	Yes	string

Responses

Code	Description	Schema
201	Success	GetReviewResponse { ID: integer * IDMovie: string * IDUser: integer * rate: string * popcorns: integer * }

- updateReview, el método se llama para cambiar el número de popcorns de esa review (La puntuación).

PUT /review

Description

update a popcorns of reviwe

Parameters

Name	Located in	Description	Required	Schema
id	query	Review Id	Yes	integer
popcorns	query	numer of popcorn of review	Yes	integer

Responses

Code	Description	Schema
200	Success	GetReviewResponse { ID: integer * IDMovie: string * IDUser: integer * rate: string * popcorns: integer * }

- getReviewById, obtener todas la review de un usuario por su id.

GET /review/{id}/{type}

Description

get all reviews

Parameters

Name	Located in	Description	Required	Schema
id	path	id User	Yes	integer
type	path	id User	Yes	integer

Responses

Code	Description	Schema
200	Success	GetReviewListResponse { Reviews: [] }
default	Error	ErrorResponse { message: string * }

Ahora los usuario pueden seguir a otros y así tener seguidores.

- addFollowing, seguir un usuario.

POST /follow

Description

follow someone

Parameters

Name	Located in	Description	Required	Schema
idFollower	query	user	Yes	integer
idToFollow	query	user to follow	Yes	integer

Responses

Code	Description	Schema
200	Success	GetUserResponse { ID: integer * userName: string * email: string * password: string * popcornsGlobal: integer * isAdmin: integer * silenced: integer * profilePath: string * }
default	Error	ErrorResponse { message: string * }

- deleteFollow, cuando dejas de seguir al usuario.

DELETE /follow

Description

unfollow a user

Parameters

Name	Located in	Description	Required	Schema
id	query	id User to unfollow	Yes	integer
user	query	id of follower (Spectateurs)	Yes	integer

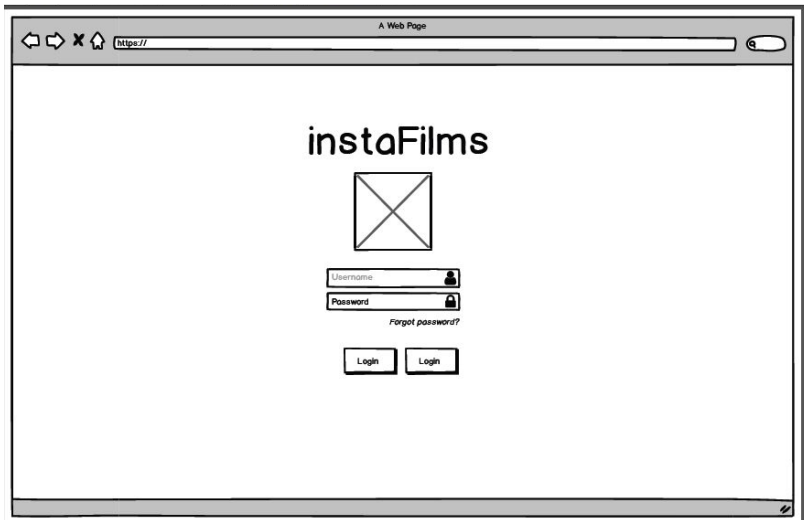
Responses

Code	Description	Schema
200	Success	SuccessResponse { message: string * }
default	Error	ErrorResponse { message: string * }

Capítulo 4

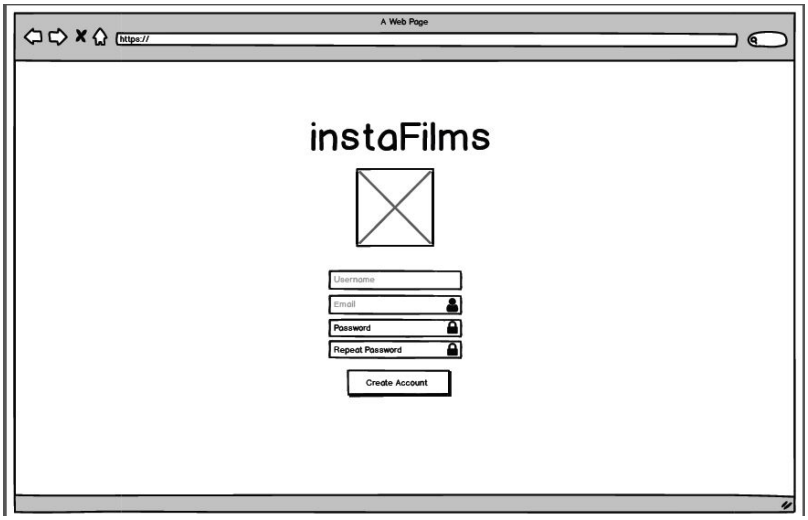
Previo de Mocks

Es necesaria la visualización y una pequeña descripción de los Mocks más relevantes de nuestra plataforma, de cara al conocimiento de la interfaz de la misma, así como su manejo. Hemos centrado el diseño en algo sencillo e intuitivo, tratando de otorgar al usuario de una mayor comodidad, evitando así procesos de aprendizaje y minimizando los fallos ocasionados por una inexperiencia en el uso.

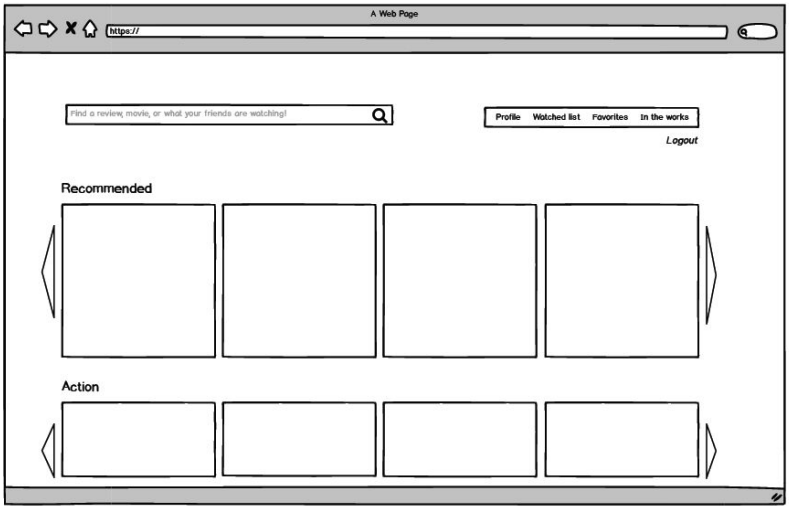


Éste es el Mock del Login. Consta de un formulario sencillo, en el que, para acceder a su cuenta, el usuario deberá introducir su username y la contraseña. En el caso de que la contraseña haya sido olvidada, se enviará inmediatamente un correo con las nuevas claves una vez pulsado el botón 'Forgot password?'. Finalmente, consta de dos botones, uno para loguearse una vez están

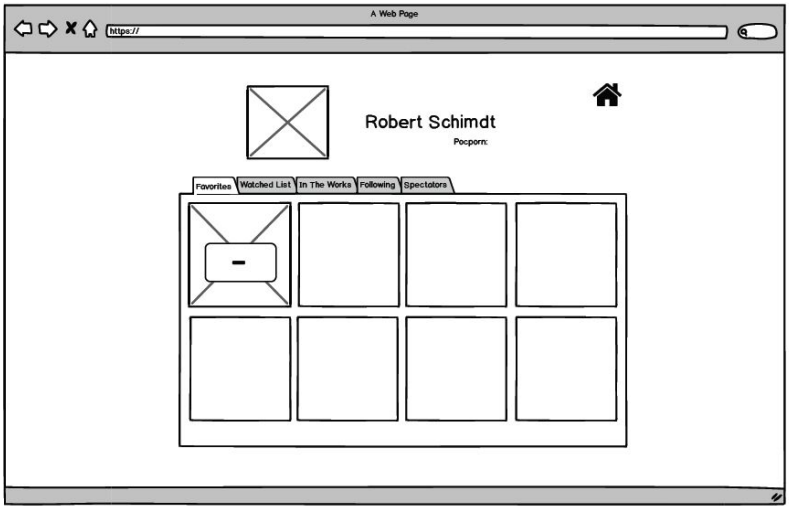
introducidos los datos correctamente, y otro para crear una cuenta en el caso de que el usuario no la haya creado antes.



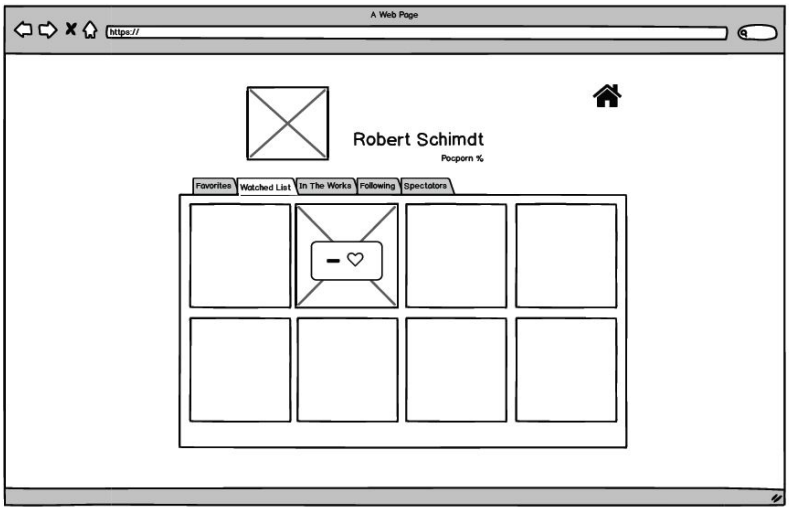
Si hacemos click en el botón de ‘New Account’, directamente nos envía a un formulario para crearla. En él, el usuario deberá introducir el username deseado, su email, así como su contraseña. Después se procede al envío del formulario con el botón ‘Create Account’.



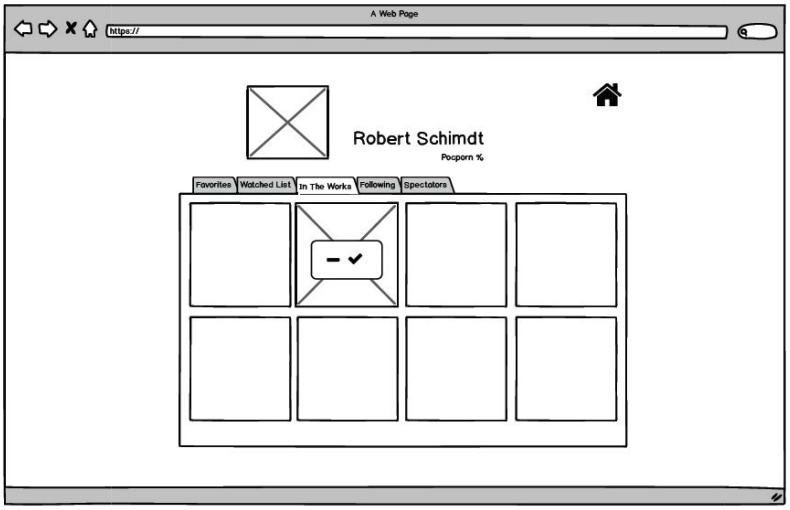
Vista del menú principal de la plataforma web una vez el usuario esté correctamente logueado. En ella se pueden buscar películas o personas en el buscador. Además de visitar las listas del propio usuario. La plataforma ofrece también una lista de películas recomendadas en función a las vistas anteriormente, además de ordenarlas por género.



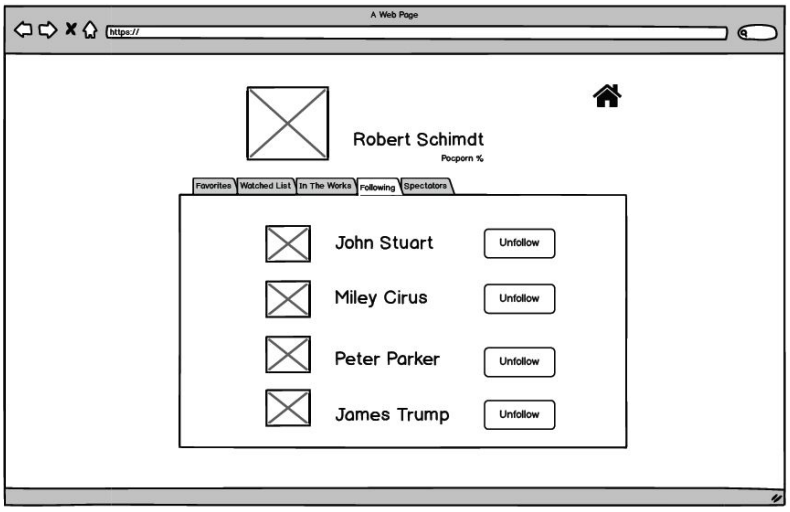
El usuario podrá acceder a su perfil pulsando el botón ‘Profile’, en la página principal. En él, podrá visualizar sus listas de películas favoritas, películas vistas, películas en curso, seguidores y espectadores, e interactuar con ellas.



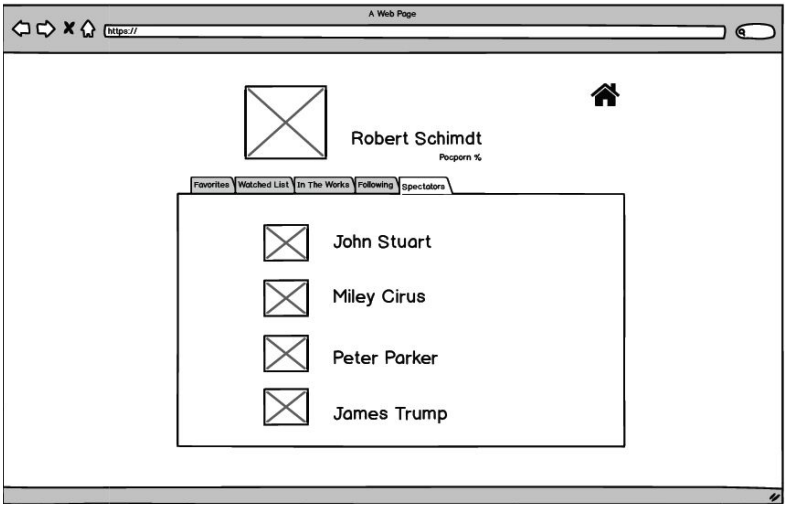
Vista de la lista de películas vistas. El usuario puede eliminar la película de la lista o enviarla directamente a la lista de películas favoritas.



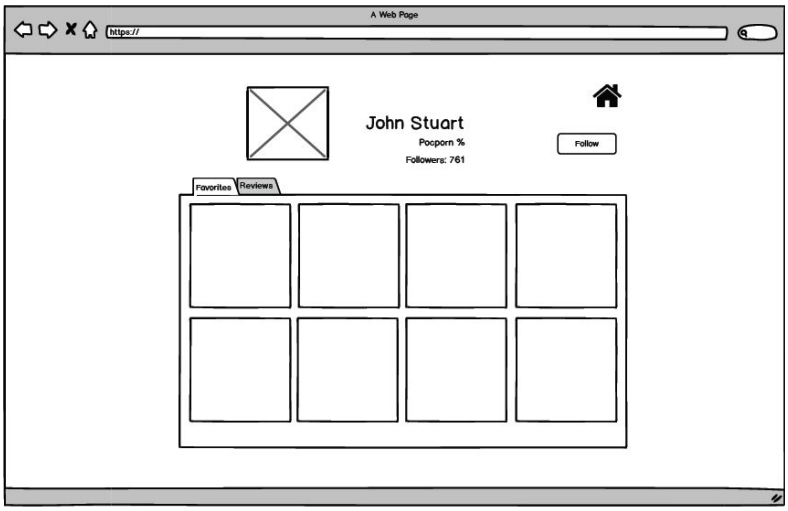
Vista de la lista de películas en curso. El usuario tiene la opción de eliminar la película y no terminar de verla, o seguir viéndola.



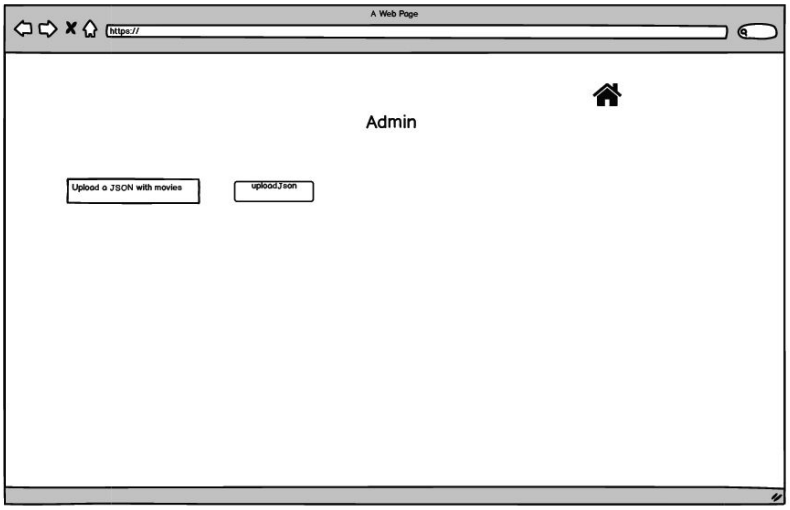
Vista de la lista de ‘following’ de un usuario. En ella, el propio usuario puede ver a quién sigue y dejar de hacerlo si así lo desea.



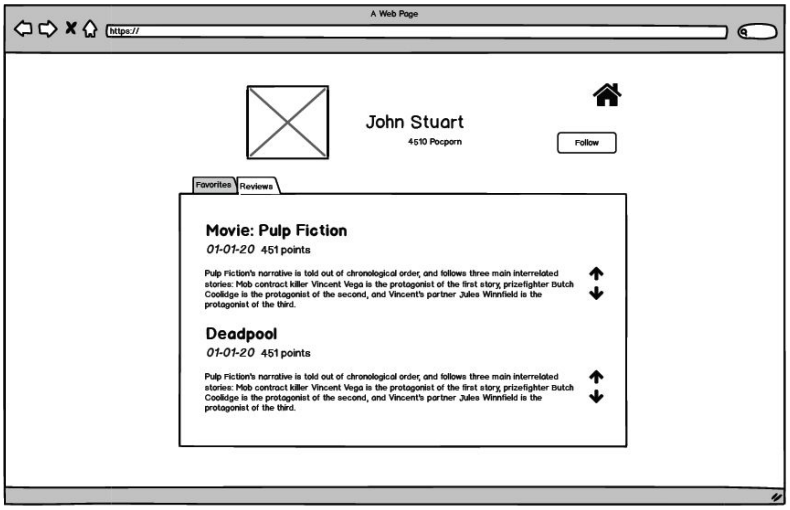
Vista de la lista de espectadores de un usuario. El propio usuario puede ver quién le sigue.



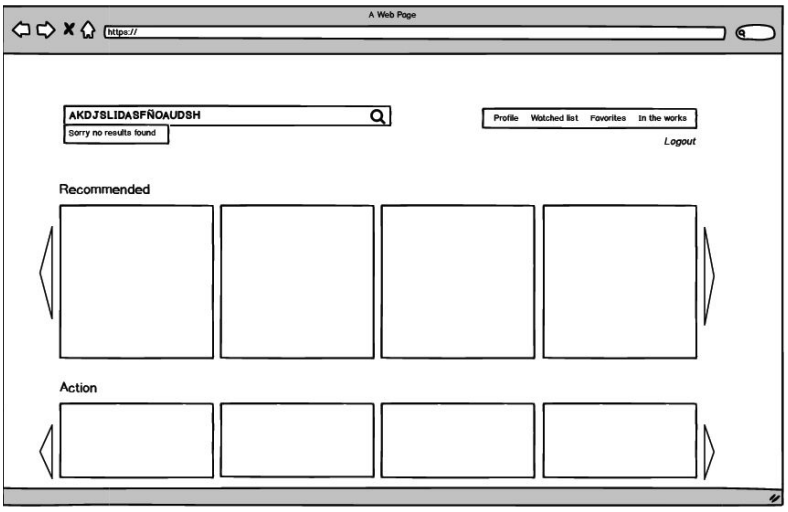
El propio usuario podrá visualizar los perfiles de otros usuarios, ver su lista de películas favoritas y las reviews de las mismas que han hecho. Si les gusta lo que ven y son afines a los gustos del usuario en cuestión, tendrán la opción de seguirlo. Así, podrá ser notificado de las películas que le gustan y sus opiniones al respecto. Esta vista muestra la lista de películas favoritas del perfil de un usuario.



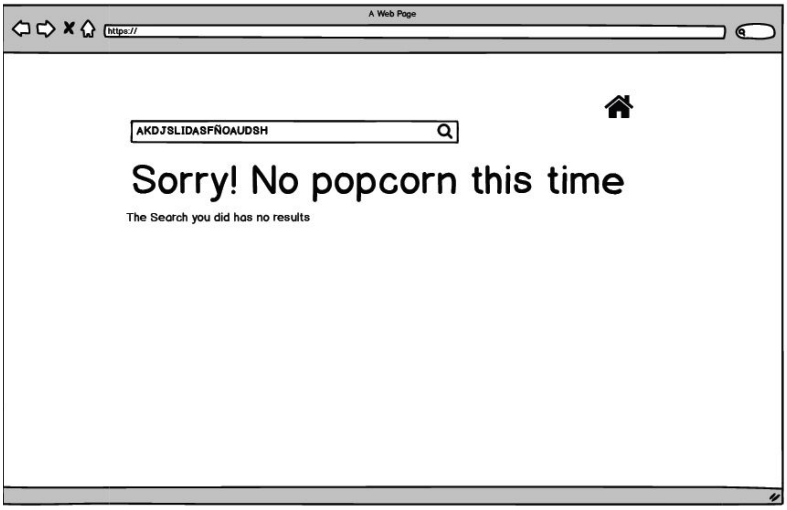
Ésta es una vista del usuario con permisos de administrador. Tendrá la potestad de subir películas a la plataforma a través del proceso explicado en el capítulo uno, así como de silenciar a los usuarios que no cumplan con la política de comportamiento y privacidad de la web.



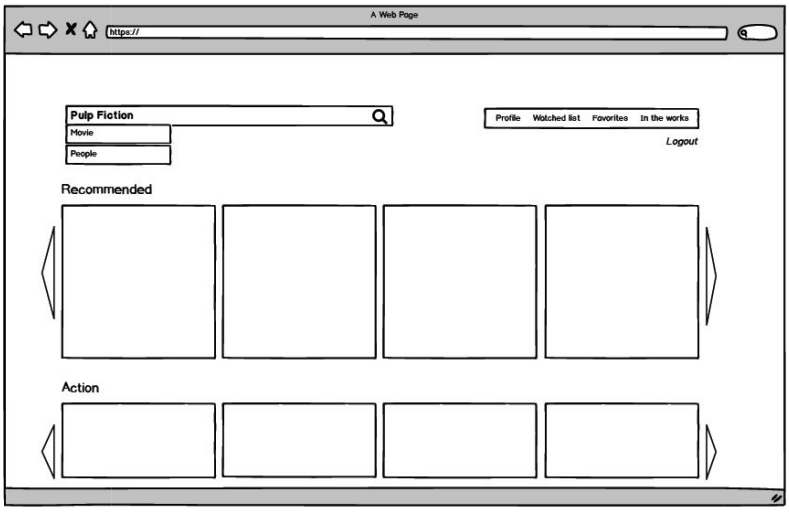
Este Mock muestra la vista del perfil de un usuario cualquiera, en la sección de reviews. En ella se visualizan todas las reviews de las películas que ha hecho, así como las puntuaciones que tiene cada una. El usuario tiene la opción de valorar positiva o negativamente una review determinada.



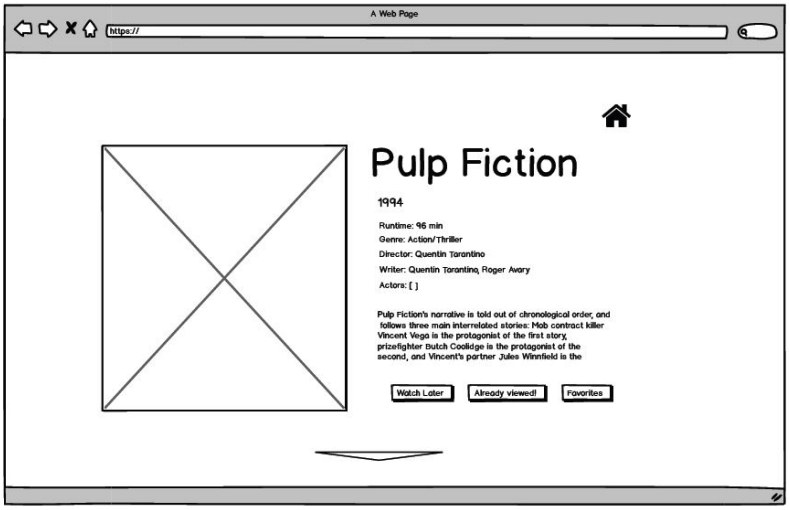
Si el buscador no encuentra resultados a la consulta realizada, se notificará de esta manera.



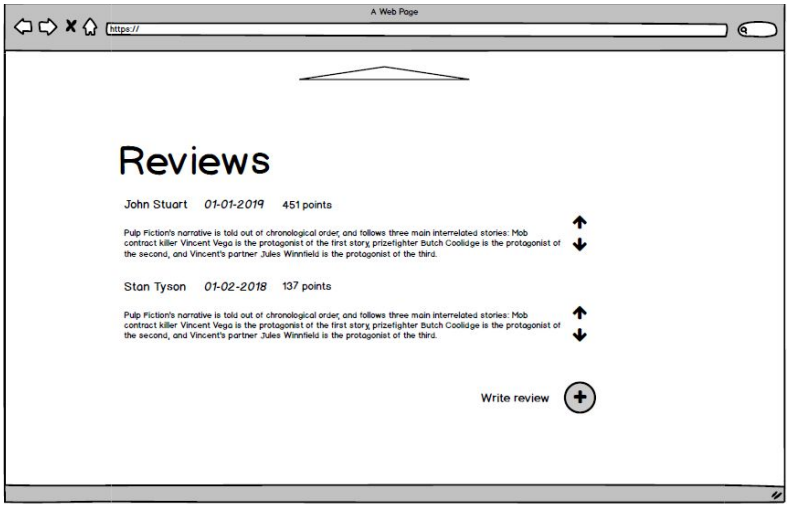
Si aún sabiendo que no existen resultados, el usuario continúa con la búsqueda, se le conducirá a esta vista, donde se le informará que no existen los resultados que busca.



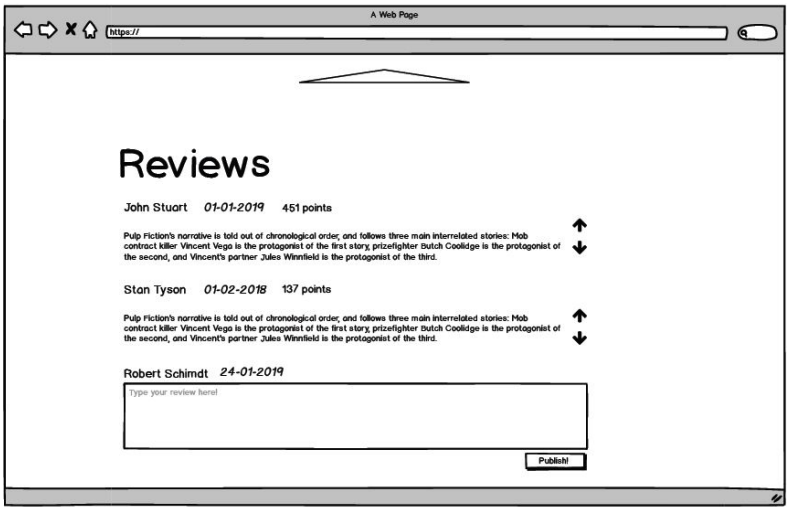
Por el contrario, si la búsqueda ha sido exitosa, devolverá los resultados de la misma organizados en películas o personas, para que el propio usuario elija la opción que esté buscando.



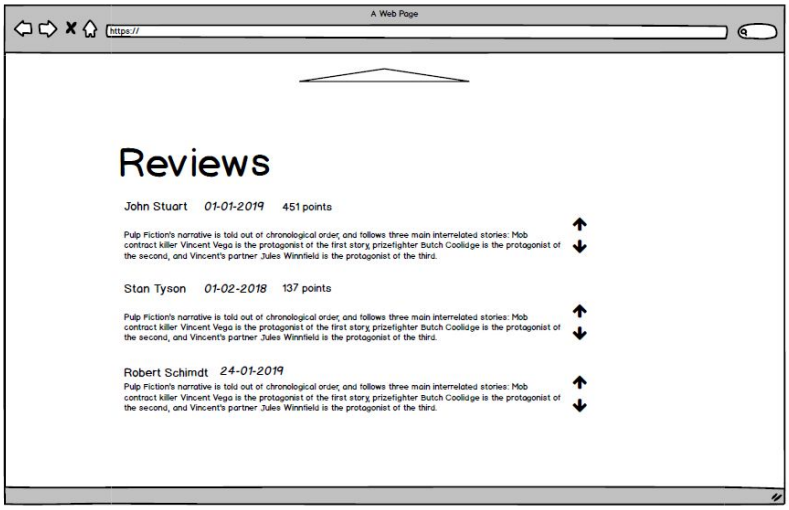
Vista de una película seleccionada. El navegador muestra la imagen de la misma y el título como elementos principales. Además, informa al usuario del año de producción, la duración de la película en minutos, el género, los escritores y el director, los actores y la sinopsis de la misma. Además, el usuario podrá guardar la película en la lista de ‘Watch Later para verla en otro momento, pulsar el botón ‘Already viewed!’ para introducir la película en la lista de vistas si no lo está o guardarla en la lista de favoritos interactuando con el botón ‘Favorites’.



Si el usuario pulsa la flecha inferior del anterior Mock, accederá a la lista de reviews de esa película. En ella, podrá ver quién puso esa review, que día la puso y los votos que tiene. Además, el mismo podrá valorar la review.



También, tiene la posibilidad de añadir una review propia pulsando la opción ‘Write review’.



Una vez publicada, aparecerá junto a las demás reviews, donde podrá ser valorada positiva o negativamente.