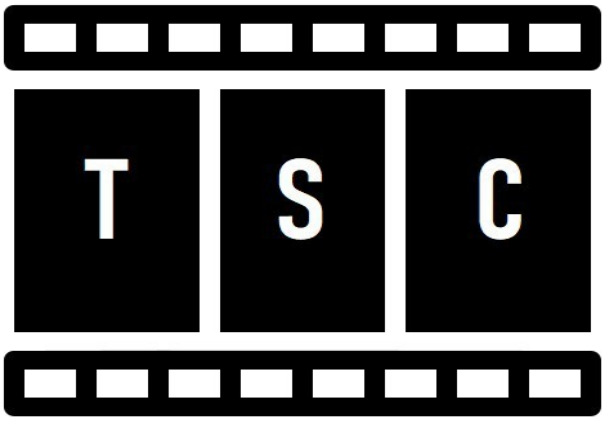
junio 2023

# TU SALA CINE




Sandra Sánchez Calzado

Desarrollo de Aplicaciones Web
IES Palomeras-Vallecas

# INDICE

# Introducción y justificación

**Tu Sala Cine** es una biblioteca de películas cuyo propósito principal es ayudar a encontrar una opción aleatoria o según la categoría seleccionada por el usuario con toda la información necesaria, incluyendo además una lista de todas las películas vistas o no vistas de esa categoría. Pero además de eso, si no encuentras una película también puedes solicitar incluir una a través de nuestro discord o por email, para que otros puedan ver también esas películas antiguas o que no todos conocen. Utilizando la tecnología de React y la librería de PrimeReact para la interfaz de usuario(UI)  y la API TheMovieDB para los datos junto a MongoDB para almacenar los datos, creando así una página web accesible y adaptable, que se ajuste a las necesidades de los usuarios y dispositivos.

He decidido realizar mi proyecto sobre este tema debido a que cuando quiero ver una película con más gente es muy complicado decidirse sobre lo que queremos ver, ya que normalmente estamos mucho tiempo buscando una película que no hayan visto los demás. De esta forma decidí crear una filmoteca personalizada y un generador de sugerencias de películas con React con MongoDB.

# Abstract

**Tu Sala Cine** is a movie library whose main purpose is to help you find a random option or according to the category selected by the user with all the necessary information, including a list of all movies seen or not seen in that category. But besides that if you don't find a movie you can also request to include a movie through a form so that others can also see those old movies or movies that not everyone knows. Using React technology and the PrimeReact library for the user interface and TheMovieDB API for the data along with MongoDB to store the data, creating an accessible and adaptable web page that fits the needs of users and devices.

# Antecedentes

En la actualidad hay muchas páginas de reviews de películas y series como Metacritic , IMDb o Filmaffinity , pero una de los sitios web primerizos que ha servido como inspiración para muchos más sitios web ha sido Rotten Tomatoes lanzada en 1998 creada con Bootstrap v3 y jQuery como librería.

Cuando decidí crear una aplicación web sobre películas me quise inspirar en Filmaffinity, la información que aporta básica de cada película y una parte del diseño de la información que ofrece me parece muy atractivo y me inspiro para crear las diferentes tarjetas con información básica utilizando React y prime React para que se vieran más responsive y atractivas, debido a que una de los inconvenientes que encontré de Filmaffinity fueron que no era responsive ni SPA.

Las páginas principales de Filmaffinity, usando jQuery y php, Rotten Tomatoes y sus respectivos apartados que muestran cada una de las películas me parece curioso y bastante útil por lo que me pareció interesante.

Según he estado investigando, me he dado cuenta que todas tienen una página de inicio muy sobrecargada por lo que mi propuesta es ofrecer una página con la misma cantidad de información pero más simple y atractiva. De hecho ninguna, excepto IMDb que utiliza las tecnologías React y Next.js, tienen una página responsive ya que son más antiguas.

Además, muchas aplicaciones de biblioteca de películas ofrecen características adicionales, como recomendaciones personalizadas de películas y programas de televisión basados en las preferencias de los usuarios, listas de seguimiento y descargas para ver películas sin conexión a internet. Pero ninguna de ellas he visto que ofrezca una funcionalidad que ayude al usuario a elegir una película de forma rápida, por lo que mi propuesta para la aplicación fue centrarme en quitar esa pesadez al usuario y ofrecerle la rapidez que en esta generación está más solicitada.

# Objetivos

- ➔ Ofrecer un gran catálogo de películas además de las distintas plataformas donde puedes ver las películas mostradas.
- ➔ Implementar una API para la obtención de las plataformas de cada películas.
- ➔ Aprender a utilizar el framework React con Node.js y MongoDB.
- ➔ Crear una página visiblemente atractiva.
- ➔ Crear una página responsive y accesible.
- ➔ Permitir que los usuarios personalicen su experiencia en la página web.
- ➔ Crear un motor de búsqueda funcional con filtros.
- ➔ Crear una página SPA.

# Desarrollo práctico

## 1. Especificación de requisitos funcionales

- ➔ Crear una página web con una lista completa de películas antiguas y nuevas.
- ➔ Crear una aplicación web para obtener una sola película a partir de las categorías seleccionadas por el usuario.
- ➔ Crear una página con un acceso de usuario atractiva.
- ➔ Poder acceder a la lista de películas vistas de los usuarios.

## 2. Especificación de requisitos técnicos

- ➔ Ser una página responsive y accesible.
- ➔ Crear una aplicación SPA.
- ➔ Crear una aplicación con el framework de React.
- ➔ Crear una base de datos de películas completa.
- ➔ Añadir animaciones bonitas y complejas con css y js.

## 3. Tecnologías y herramientas

- ➔ Tecnologías FrontEnd:
  - ✔ React
  - ✔ PrimeReact
  - ✔ Axios
- ➔ Tecnologías BackEnd:
  - ✔ Express
  - ✔ NodeJS
  - ✔ Mongoose
  - ✔ Nodemon
- ➔ Otros
  - ✔ Google Fonts
  - ✔ Pexels para las fotos con creative commons
  - ✔ TMDB, o una API parecida para obtener la información de las películas

# 4. Planificación(Diagrama de Gantt)

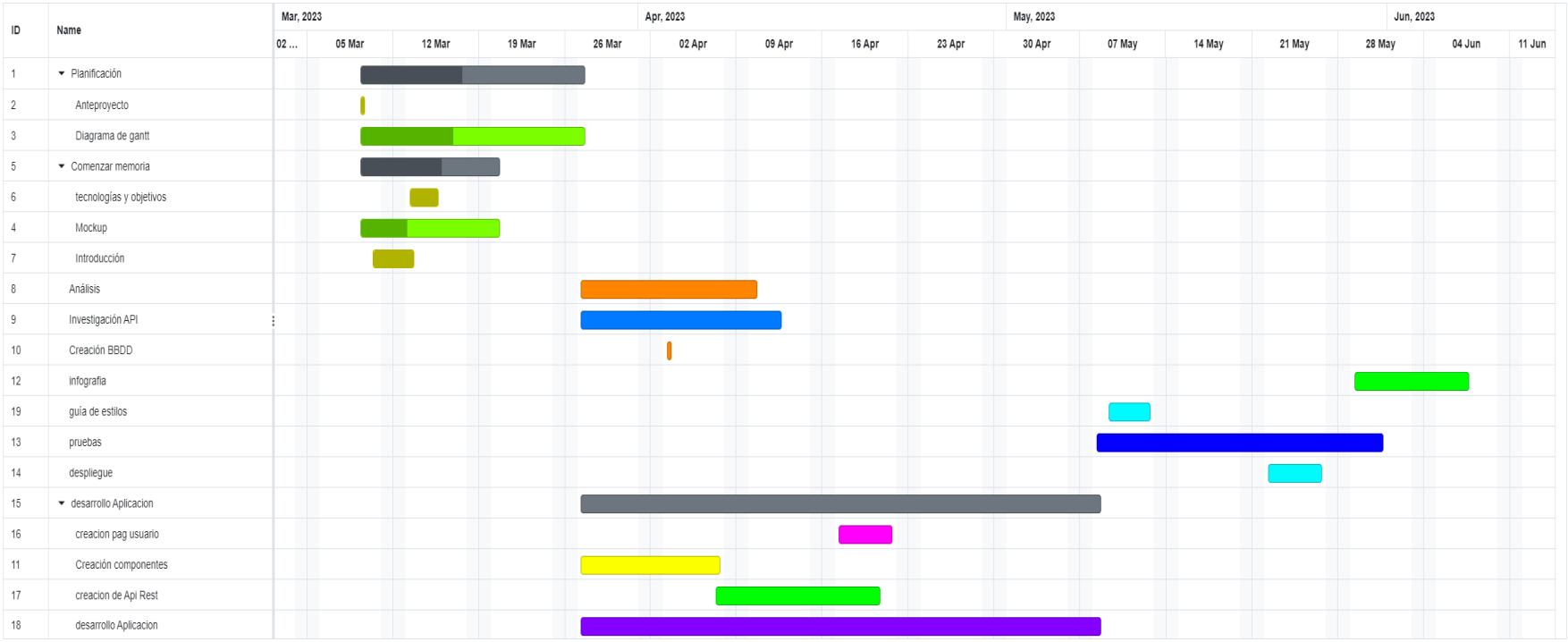



*Figura 1: Diagrama de gantt*

# 5. Análisis(casos de uso, diagrama entidad/relación y diagrama de colecciones)

- ***Casos de uso***

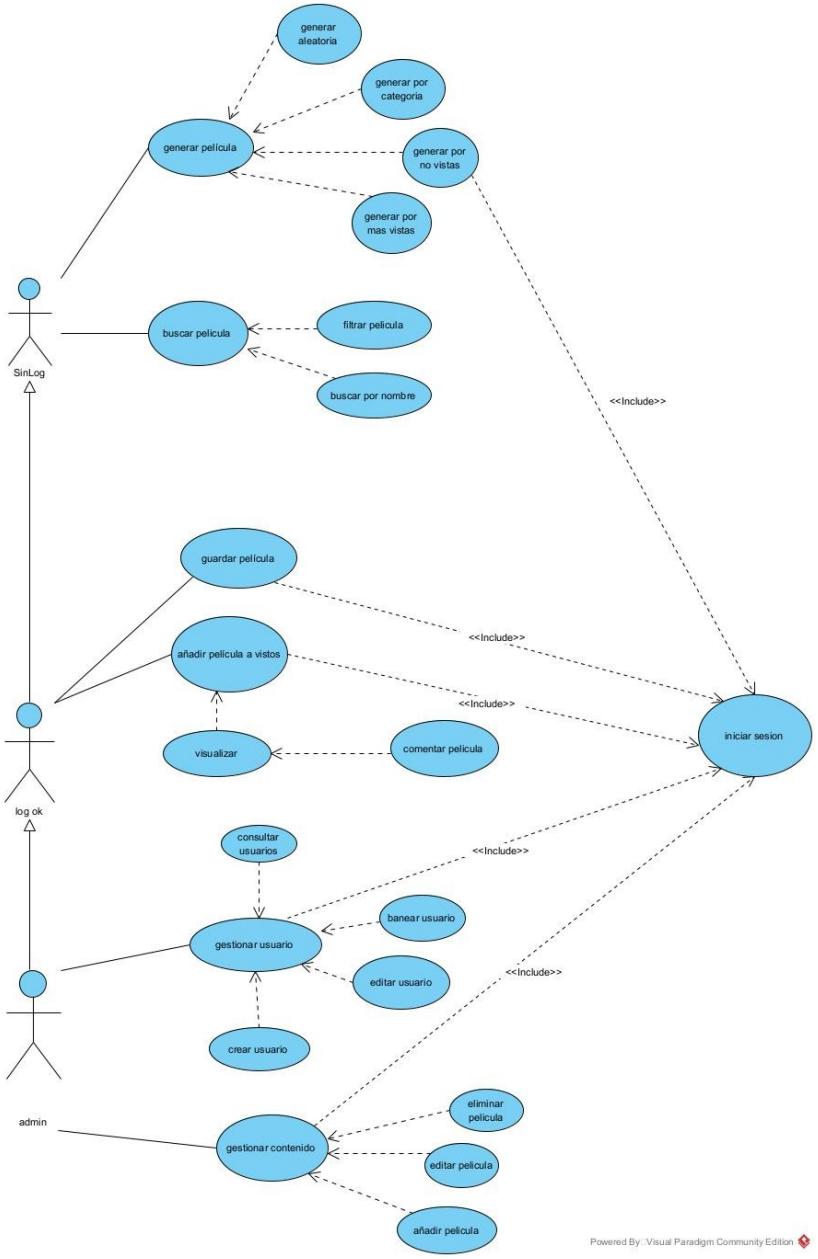



*Figura 2: Casos de uso*

- ***Diagrama de colecciones***

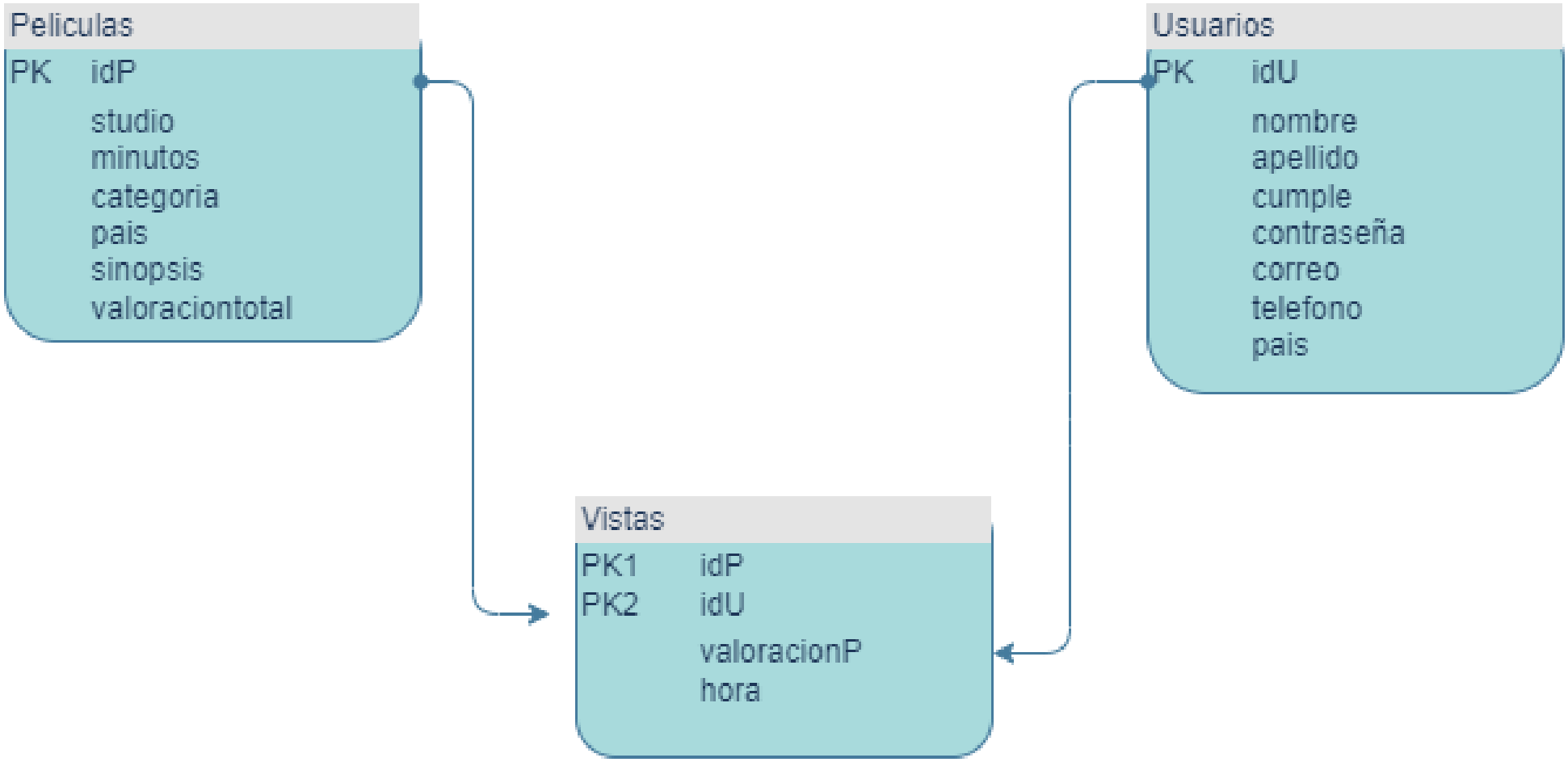



*Figura 3: Diagrama de colecciones*

- ***Paso a tablas***

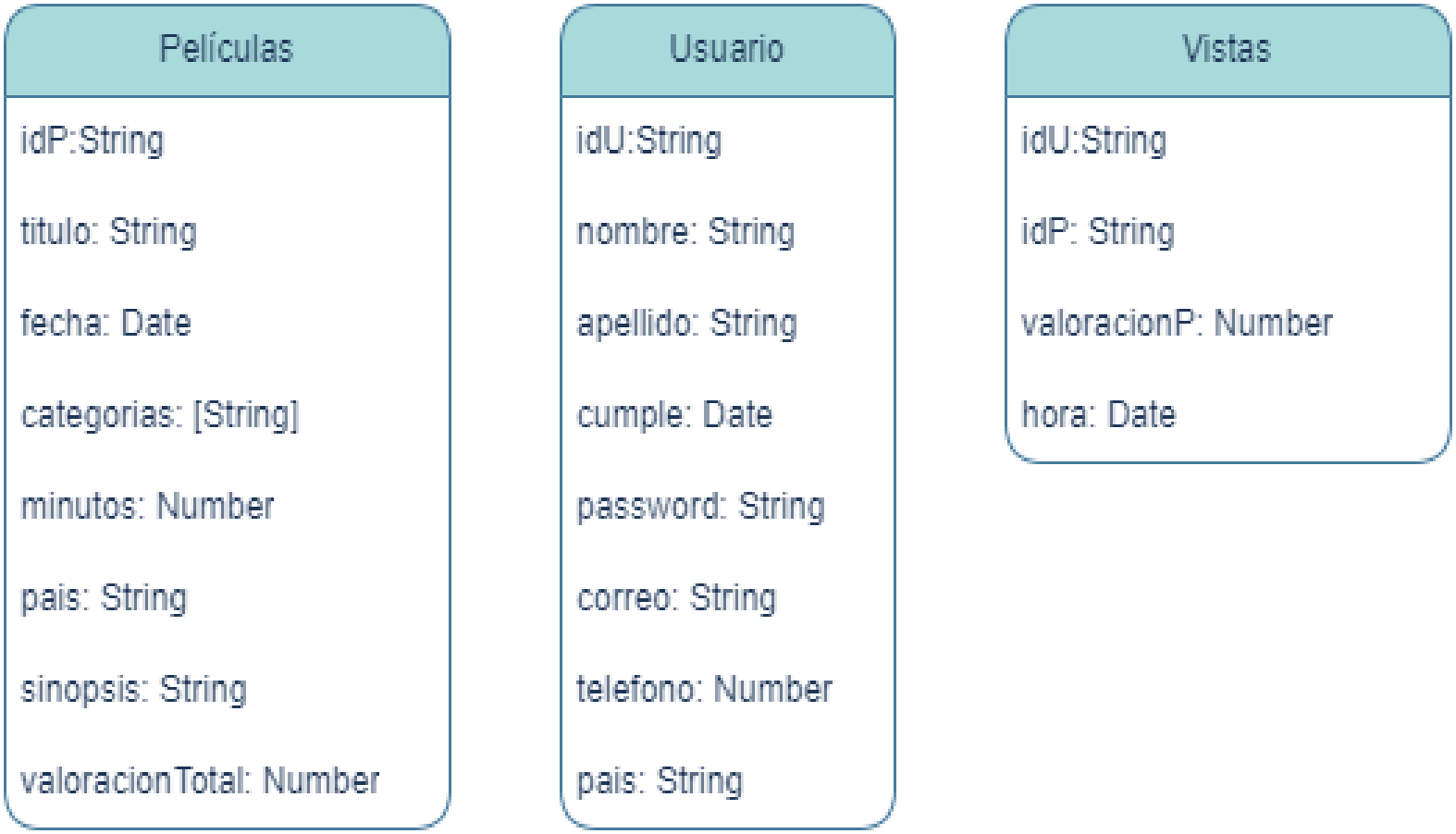



*Figura 4: Paso a tablas del diagrama*

# 6. Diseño

- ***Prototipos(Mockup)***

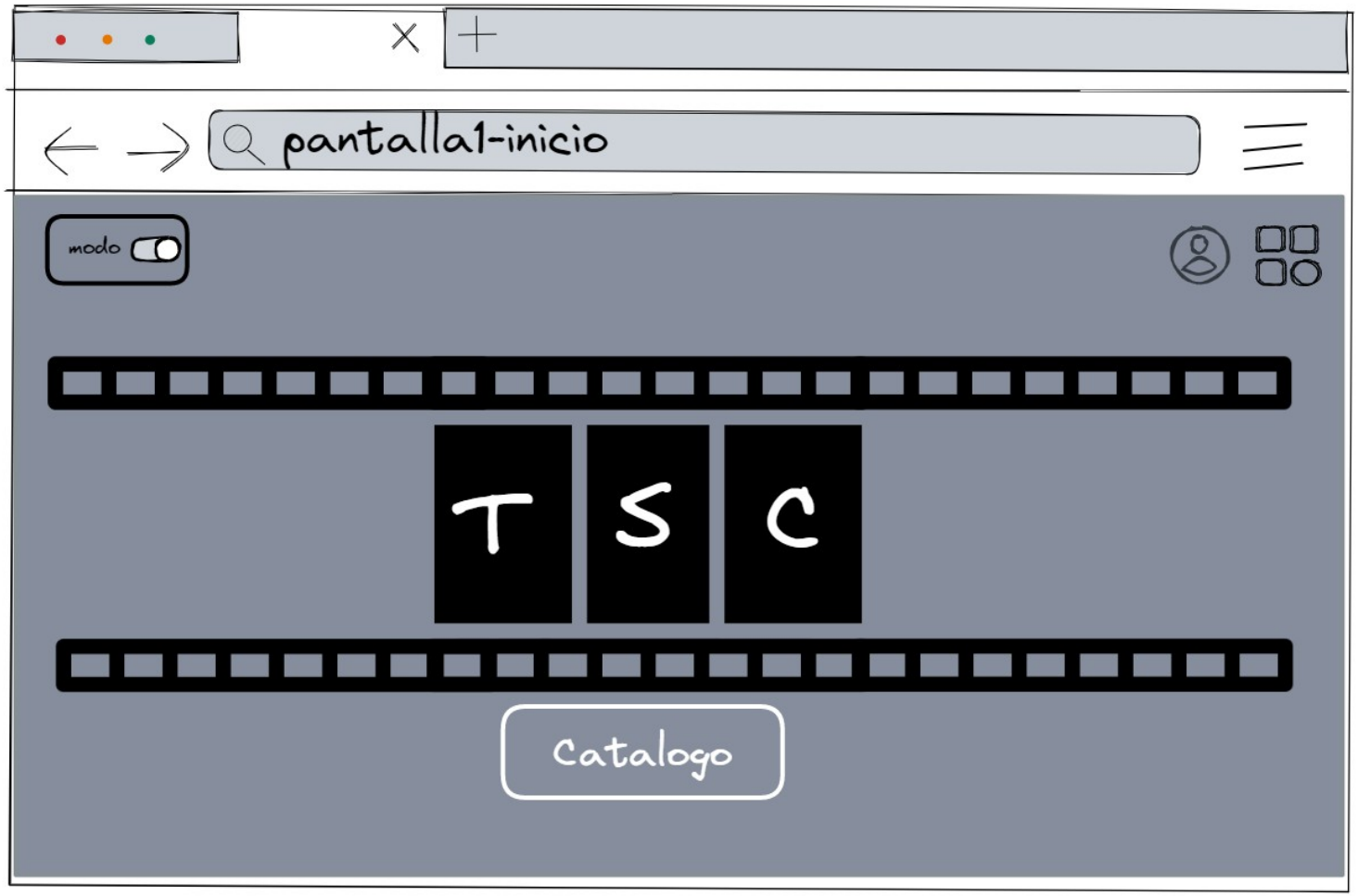



*Figura 5: Mockup  página inicio*

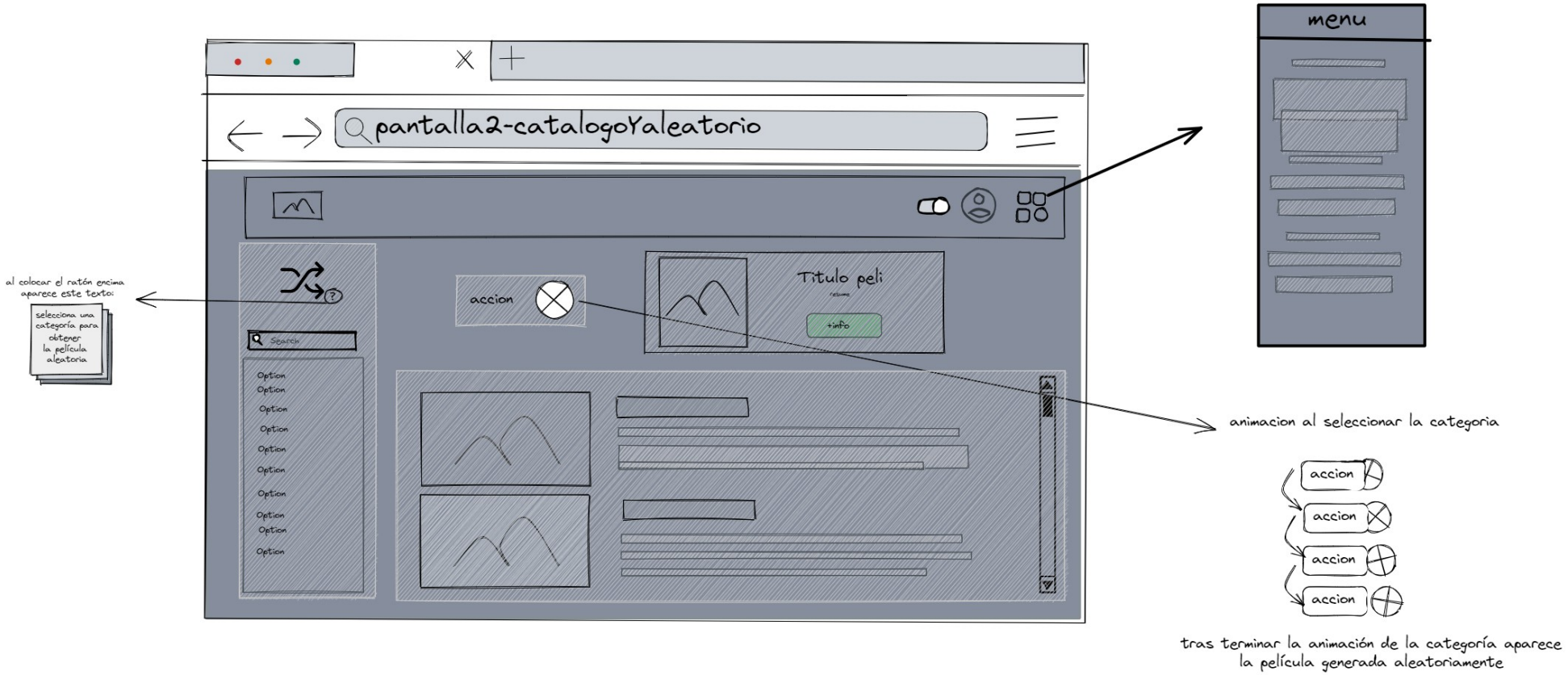



*Figura 6 : Mockup página categorías*


*Figura 7: Mockup página de usuario*

- ***Estructuras del proyecto***
  - **Back**: Carpeta con todas las carpetas y archivos del back-end:
    - index.js*:* Fichero que contiene lo necesario para iniciar el back-end, realiza la conexión a la base de datos de mongoDB y accede a las rutas.
    - Src*:* Carpeta con el controller, los modelos de tablas y las rutas.
      - *Controller*: Carpeta con los métodos que responden a las peticiones de routes.
      - *Model:* Carpeta con las estructuras de las tablas de películas, usuarios y vistas utilizadas en controller.
      - *Routes:* Carpeta con el fichero que enruta las solicitudes al controlador.
  - **tusalacine**: Carpeta con todas las carpetas y archivos del front-end:
    - public: Carpeta creada por React con el HTML y el favicon del logo.
    - src: Carpeta con todos los archivos necesarios para iniciar el front.
      - *assets:* Carpeta con dos carpetas con archivos estáticos.
      - *Context:* Carpeta con los objetos que se pasarán como
      - *Global.js:* Fichero que almacena la configuración de acceso a la back-end
      - *Router.js:* Fichero con las rutas de los componentes.

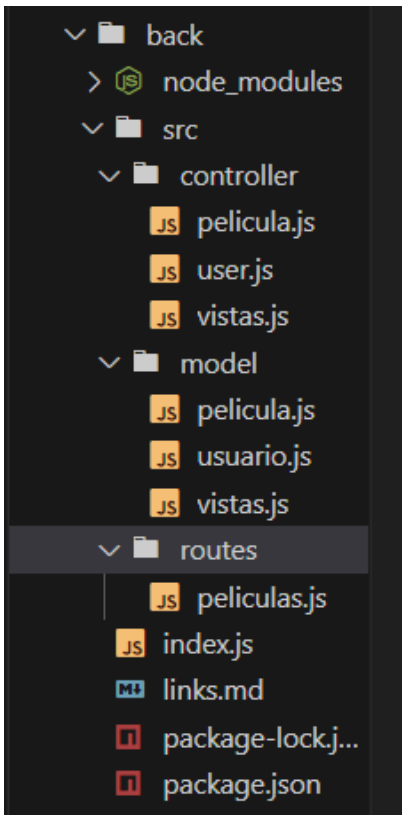



*Figura 9: Estructura back-end*

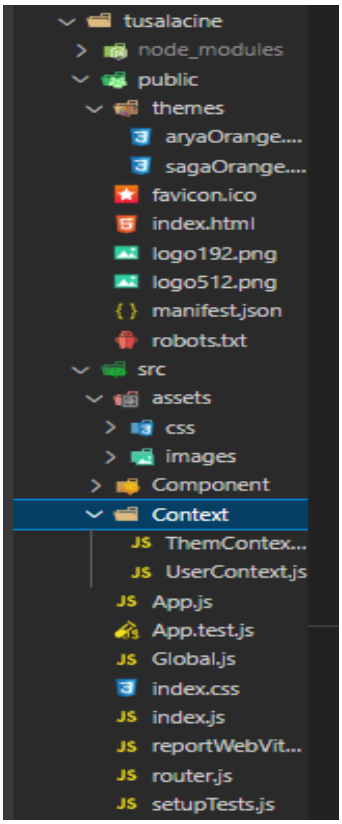



*Figura 8 : Estructura front-end*

- *Component:* Carpeta con todos los componentes.

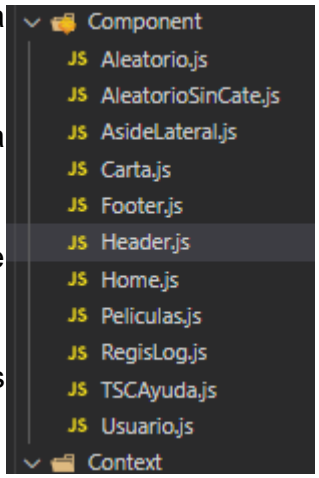


*Figura 10: Estructura componentes front-end*

➔ Aleatorio: Ficha de una película aleatoria filtrada por categoría

➔ AleatorioSinCategoria: Ficha de una película aleatoria recibida por props del componente.

➔ AsideLateral: Lista con las categorías y filtrado de las mismas.

➔ Carta: Ficha de la película para el listado de todas las películas.

➔ Footer: Pie de página para todo el sitio web.

➔ Header: Cabecera para todas las páginas

➔ Home: Pagina de inicio con botón prinicpal que llama a aleatorio sin categoría.

➔ Películas: Contenedor que llama a AsideLateral y contiene el div con el botón que llama a Aleatorio o AleatorioSinCategoria y crea listado de películas.

➔ RegisLog: Registro e inicio de sesión

➔ TSCAyuda: Página con la información sobre la aplicación web.

➔ Usuario: Página con el perfil de usuario y las películas que ha visto el usuario

# 7. Implementación

Para empezar a explicar el proceso de la generación de una película por categoría tendría que explicar primero como obtiene la categoría desde el front-end y como realiza la llamada al back-end.

- El sistema muestra un botón que llama al componente *Aleatorio* si el useState '*categoría*' no está vacío (*Figura12)*, por lo que el componente *AsideLateral* ha pasado correctamente la categoría mediante la función pasada por props (*Figura11, Figura13*).

```
<h1 className='logo'>Pelicula Random</h1>
{
    categoria
        ?
        <Aleatorio categoria={categoria} />
        :
        <div>
            {peliSinCate}
            <p>Seleccione una categoría</p>
        </div>
}
```

*Figura 12: Comprobar que hay categoría*

```
onSubmit: (data) => {
    data.item && show(data);
    props.getCategoria(data.item.name);
    formik.resetForm();
}
```

*Figura 11: Envío de la categoría*

```
return (
    <main >
        <AsideLateral getCategoria={recibirCategoria} />

        <div className='card contenedor backBlack'>
```

*Figura 13 : Recibimos la categoría*

- El usuario pulsa en el botón que muestra el sistema (*Figura14*) y se ejecuta la función '*getPeliCate' (Figura16)* que realiza una petición *get* a la base de datos accediendo desde la constante creada en el archivo *Global.js* (*Figura15*) y mediante el método '*getoneCate'* junto a la categoría recibida mediante *axios*.

```
return (
  <article className="bg-yellow-200 card">
    <div className="bg-yellow-200 card-container flex flex-co
      <StyleClass
        nodeRef={openBtnRef}
        selector=".box"
        enterClassName="hidden"
        enterActiveClassName="fadein"
      >
        <Button
          ref={openBtnRef}
          label="Pulsa aquí"
          icon="pi pi-refresh"
          size="large"
          onClick={getPeliCate}
        />
      </StyleClass>
```

*Figura 14: Botón pulsado*

Peliculas.js | Global.js M | AsideLateral.js

TSC > tusalacine > src > Global.js > Global

```
1  var Global = {
2      url: "https://backtsc.onrender.com/"
3  }
4  export default Global;
```

*Figura 15 : Enlace al back-end*

```
const Aleatorio = (categoria) => {
  const url = Global.url;
  const [random, setpeli] = useState([]);
  const openBtnRef = useRef(null);

  const {
    imbd_id,
    titulo,
    fecha,
    minutos,
    pais,
    sinopsis,
    valoracionTotal,
    poster
  } = random || {};
  function getPeliCate() {
    axios.get(url + "getoneCate/" + categoria.categoria).then((res) => {
      setpeli(res.data.PeliRandom);
    });
  }
```

*Figura16 : Función getPeliCate*

Tras explicar como se realiza la petición desde React al back-end de Node.js ahora procederé a explicar como devuelve la película generada aleatoriamente.

- El sistema accede a la ruta establecida en el archivo *peliculas.js* de la carpeta *routes* (*Figura17*) para poder acceder al método del archivo *película* de la carpeta *controller* (*Figura18*)

```
TSC > back > src > routes > peliculas.js > ...
 7   var route = express.Router();
 8
 9   //rutas para ejecutar los metodos de pelicula(controller)
10   route.post('/add', Peli.save);
11   route.post('/addone', Peli.saveone);
12   route.get('/getall', Peli.getPelis);
13   route.get('/getone', Peli.getRandom);
14   route.get('/getCate/:cate',  Peli.getCate);
15   route.get('/getoneCate/:cate', Peli.getRandomCate);
16   route.get('/getid/:id', Peli.getPelisId);
17   route.delete('/delete/:id', Peli.delete);
18
```

*Figura 17: Rutas de las Peli*

```
TSC > back > src > routes > peliculas.js > ...
1   'use strict'
2
3   const express = require('express');
4   const Peli = require('../controller/pelicula');
5   const User = require('../controller/user');
6   const Vistas = require('../controller/vistas');
7   const route = express.Router();
8
```

*Figura18 : Creación de la constante para acceder al controller*

- El sistema ejecuta el código del método '*getRandomCate*' del controllador guardando en constantes la categoría recibida por parámetro de entrada, las películas de la categoría recibida usando la función '*where*' de Mongoose para filtrar por esta y la función '*find*' para obtener todas las películas correspondientes, y la PeliRandom generada por la función *random* del objeto *Math* y todas las películas que hemos comentado anteriormente (*Figura19*).

```
//Metodo get una categoría
getRandomCate: async (req, res) => {
    try {
        const categoria = req.params.cate;
        const peliculas = await Peli.where({categorias: categoria}).find();
        const PeliRandom = peliculas[Math.floor(Math.random() * peliculas.length)];

        if (!peliculas) {
```

*Figura 19: Creación de Película Random filtrada por categoría*

- Tras generar la película que devolveremos al front-end realizamos una pequeña comprobación para controlar los errores como el de *error de compilación* y el de *Not Found(Figura20)*

```
    if (peliculas.length === 0 ) {
        return res.status(404).send({
            message: 'No hay películas con ese id'
        });
    } else {
        return res.status(200).send({
            PeliRandom
        });
    };
} catch (error) {
    return res.status(500).send({
        message: 'Ha habido un error y no se han encontrado las películas
    });
}
```

*Figura20: Control de errores y envio de datos*

- El sistema devuelve la película con el estado 200

Para finalizar el proceso debemos volver al front-end donde explicaré como muestra recoge y muestra la película generada en el back-end.

- El sistema recoge la película y la añade en el useState '*random' (Figura21)* creado al inicio del componente como vacío (*Figura22*).

```
function getPeliCate() {
  axios.get(url + "getoneCate/" + cate
    setpeli(res.data.PeliRandom);
  }).
```

*Figura 21: Cambiar valor del useState random*

```
const Aleatorio = (categoria) => {
  const url = Global.url;
  const [random, setpeli] = useState([]);
```

*Figura 22: Creación del useState random*

- Por defecto el sistema realiza el destructuring del array del *useState* '*random'* pero hasta que a la constate no se le asigne el array con los datos de la película se creará un objeto vacío (*Figura23*).

```
const {
  imbd_id,
  titulo,
  fecha,
  minutos,
  pais,
  sinopsis,
  valoracionTotal,
  poster
} = random || {};
```

*Figura 23: Destructuring de película random por categoría*

- Por último el sistema muestra mediante un *panel* de la librería de *PrimeReact* una carta con la carátula y toda la información obtenida ocultando el botón que se mostraba anteriormente gracias al *useRef* '*openBtnRef* ' y la etiqueta '*StyleClass'* también de *PrimeReact (Figura24, Figura25)*

```
return (
  <article className="bg-yellow-200 card">
    <div className="bg-yellow-200 card-container flex flex-column align-content-around">
      <StyleClass
        nodeRef={openBtnRef}
        selector=".box"
        enterClassName="hidden"
        enterActiveClassName="fadein"
      >
        <Button
          ref={openBtnRef}
          label="Pulsa aquí"
          icon="pi pi-refresh"
          size="large"
          onClick={getPeliCate}
        />
      </StyleClass>
      <Panel
        header={titulo}
        toggleable
        className="hidden animatior
      >
```

*Figura 24: Código jsx correspondiente al botón*

```
        className="hidden animation-duration-500 box"
      >
        <div className="m-0 flex flex-column justify-content-center align-items-center flex-wrap card-containe gap-3">
          <div className="flex justify-content-center flex-wrap card-containe gap-3">
            {header}
            <div id="subtitlePeli">
              <p>
                <Chip className="pl-0 pr-3" template={time} />
              </p>
              <p>
                <Chip className="pl-0 pr-3" template={world} />
              </p>
              <p>
                <Chip className="pl-0 pr-3" template={star} />
              </p>
              <p>
                <Chip className="pl-0 pr-3" template={minute} />
              </p>
            </div>
          </div>
          <Chip className="pl-0 pr-3" template={resumen} />
        </div>
      </Panel>
    </div>
  </article>
);
;
xport default Aleatorio;
```

*Figura25: Código jsx correspondiente al panel de la película por categoría*

# 8. Pruebas

*Tabla 1: Pruebas generar película categoría*

| Número | 1 | Tipo | **Pruebas** |
|---|---|---|---|
| Componente | Generar película por categoría | | |
| Descripción | Comprobamos que tras elegir una categoría muestra una película aleatoria de dicha categoría | | |

*Tabla 2: Datos de prueba*

| Versión | Fecha | Autora | Comentarios |
|---|---|---|---|
| 1.0 | 31/05/2023 | Sandra Sánchez Calzado | Funciona correctamente |

*Tabla 3: Datos de entrada para pruebas*

## ENTRADAS

1. Búsqueda de categoría existente por buscador

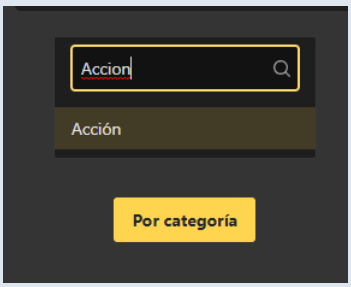



*Figura26 : Prueba 1*

2. Generación de una película por una categoría existente

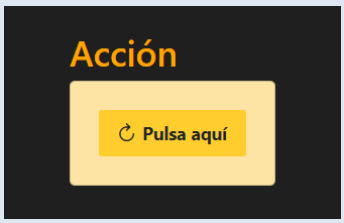



*Figura 27: Prueba 2*

3. Búsqueda de categoría no existente por buscador

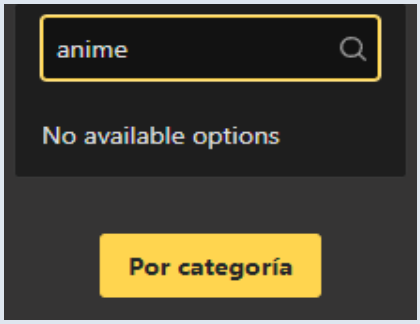



*Figura 28: Prueba 3*

4. Búsqueda de película por selección directamente en el panel

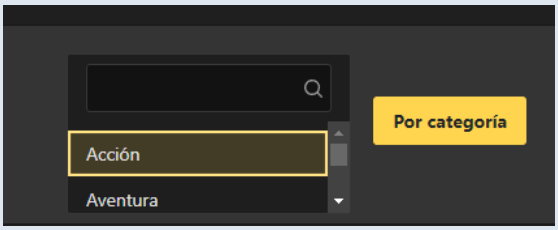



*Figura 29: Prueba 4*

5. Mandamos una categoría al back que no exista

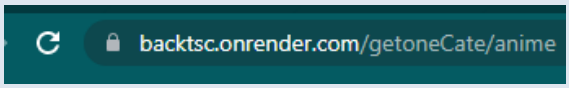



*Figura 30: Prueba 5*

6. Mandamos una categoría al back que exista

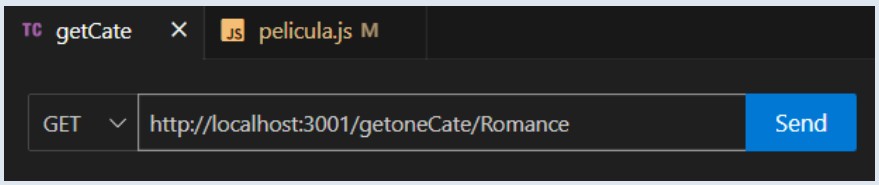



*Figura 31: Prueba 6*

## SALIDAS

### 1. Prueba válida

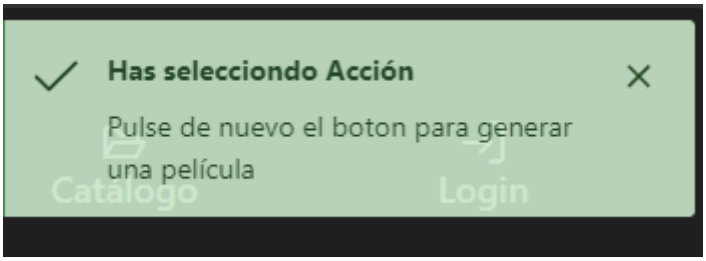



*Figura 32: Salida 1*

### 2. Prueba válida

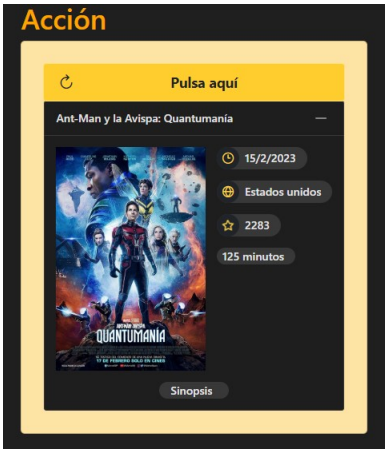



*Figura 33: Salida 2*

### 3. Prueba válida

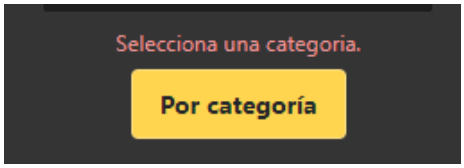



*Figura 34: Salida 3*

### 4. Prueba válida

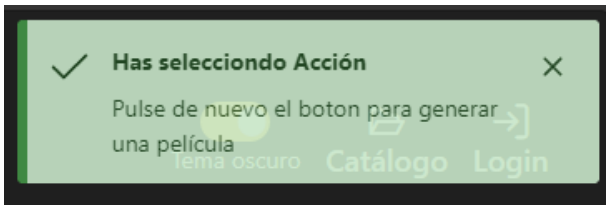



*Figura 35: Salida 4*

## 5. Prueba válida

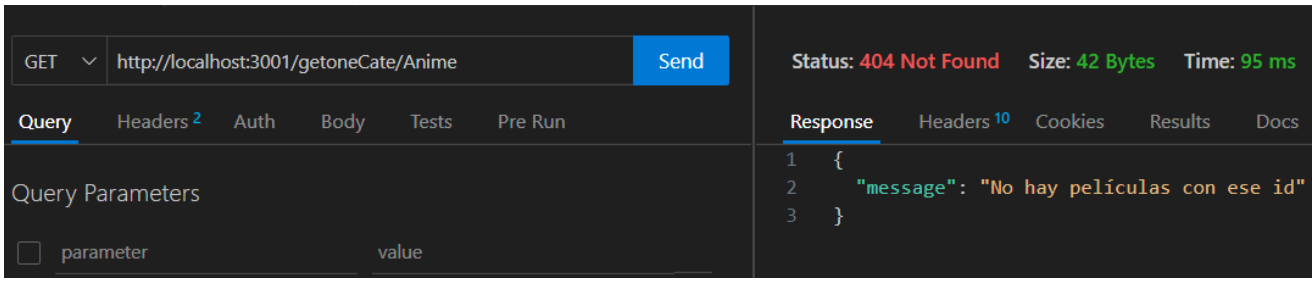



*Figura 36: Salida 5*

## 6. Prueba válida


*Figura 37: Salida 6*

# 9. Despliegue

Para desplegar la aplicación tendremos que dividirla en back y front y subirla a Render

## Back-End

1. Nos creamos una cuenta en Render y seleccionamos *Web Service* tras presionar el botón de *New*

2. Como hemos iniciado sesión en Render con GitHub nos conectamos a nuestro repositorio donde tenemos nuestro proyecto

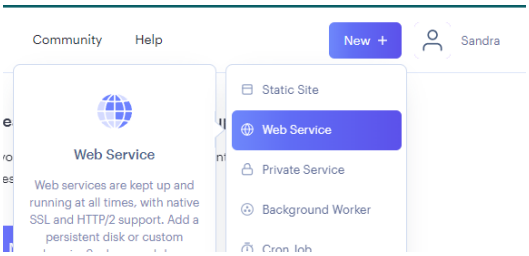



*Figura 38: Despliegue de la parte del back-end*

3. Escribimos lo siguiente(*Figura 8.3*) en la pestaña que nos aparece teniendo en cuenta que la ruta que le indiquemos debe ser relativa.

   Recuerda tener el archivo index.js en la carpeta que hayas definido como directorio.

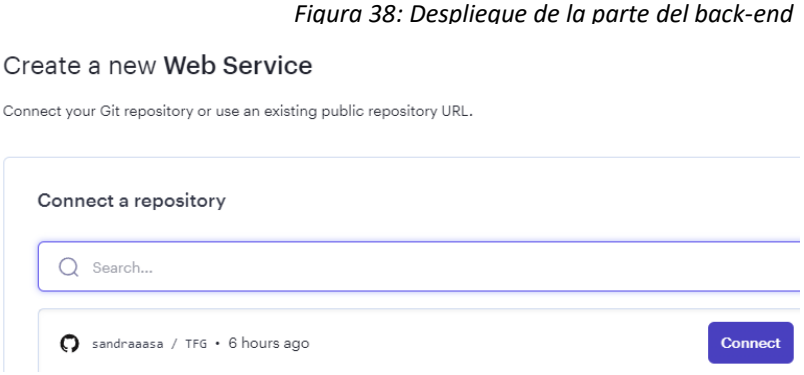



*Figura 39: Conectar con GitHub*

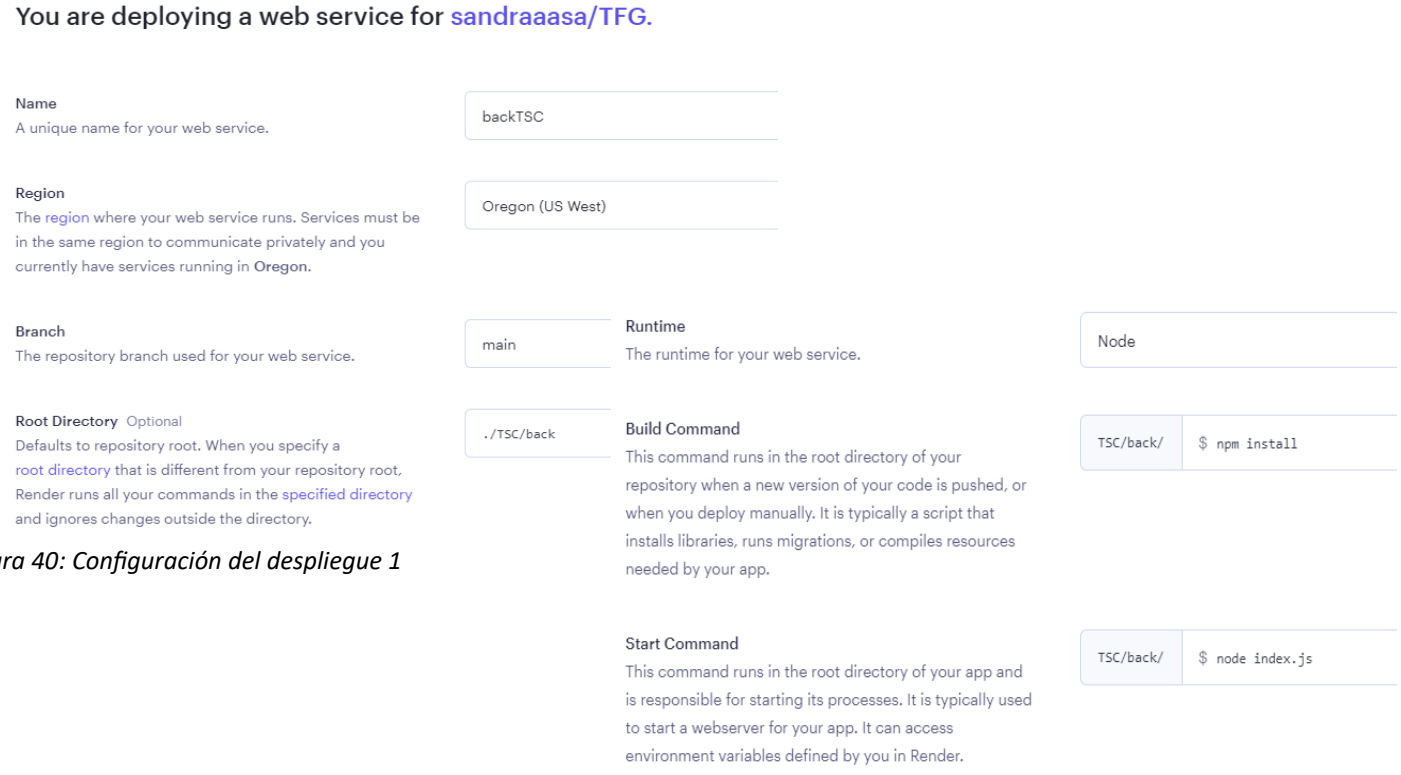



*Figura 40: Configuración del despliegue 1*

*Figura 41: Configuración del despliegue 2*

4. Para obtener el link y configurar después de desplegar la aplicación debes entrar en dashboard donde te aparecerán todas las aplicaciones desplegadas.

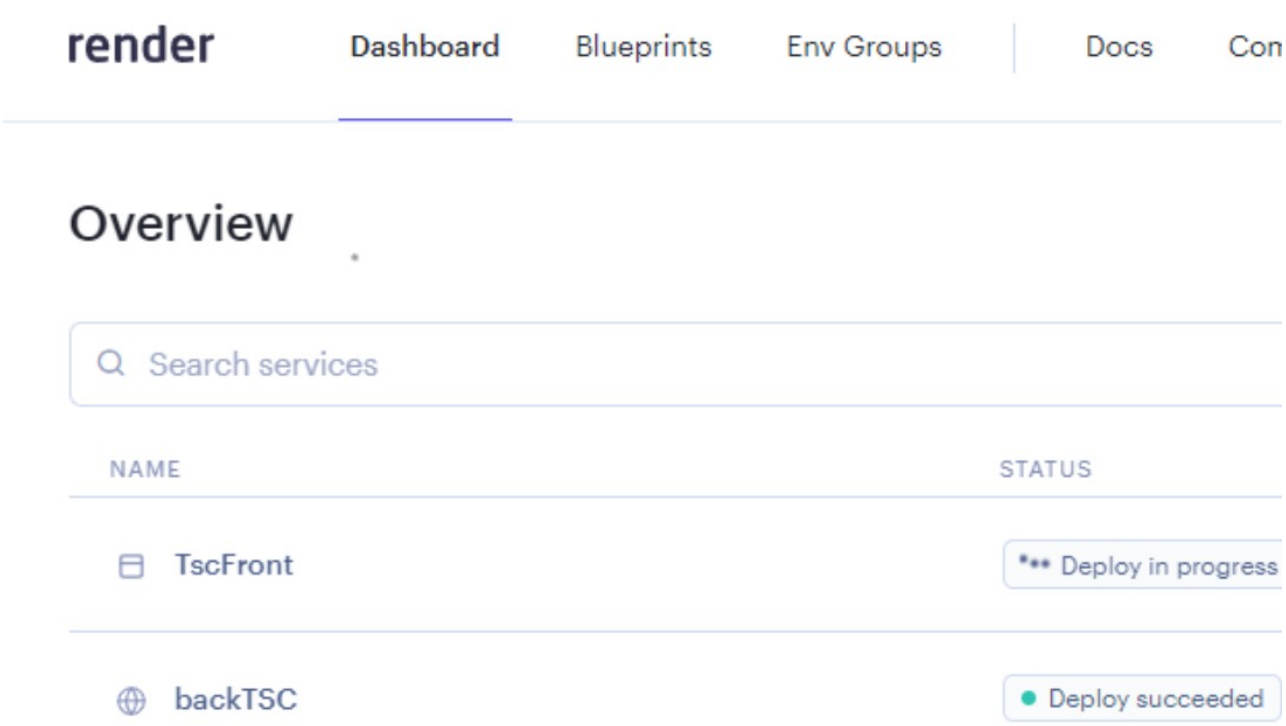



*Figura 42: Obtener link back-end*

5. Y ya tendríamos la página desplegada.

https://backtsc.onrender.com

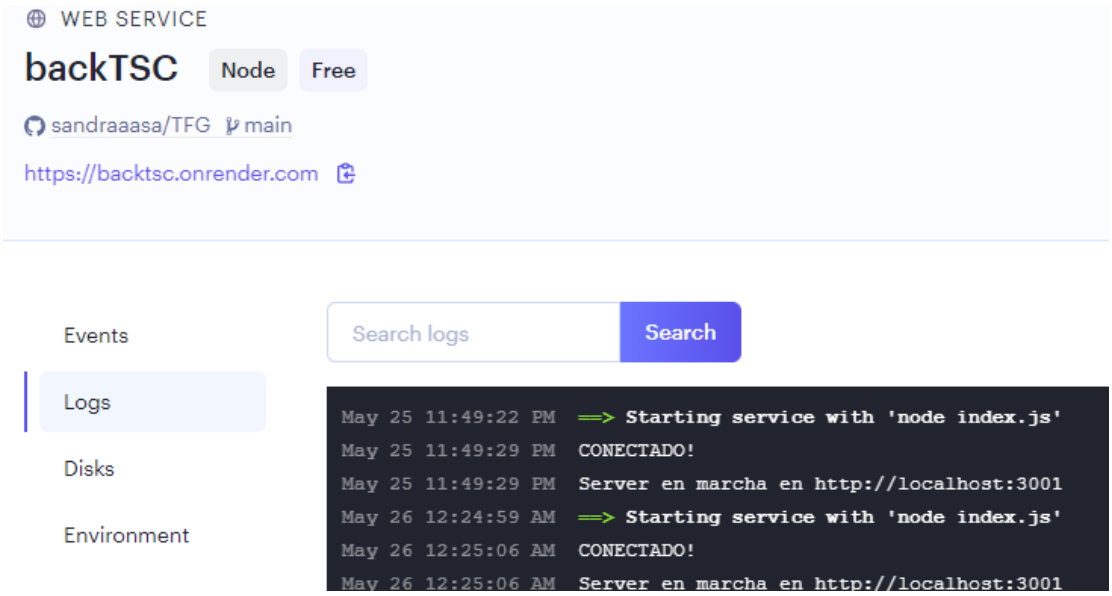



*Figura 43: Página desplegada*

**Front-End**

1. Seguimos los mismos pasos que en el back si aun no hemos subido nada. A diferencia del back-end seleccionamos *Static Site.*

2. Escribimos lo siguiente en la pestaña que nos aparece teniendo en cuenta que la ruta que le indiquemos debe ser relativa.

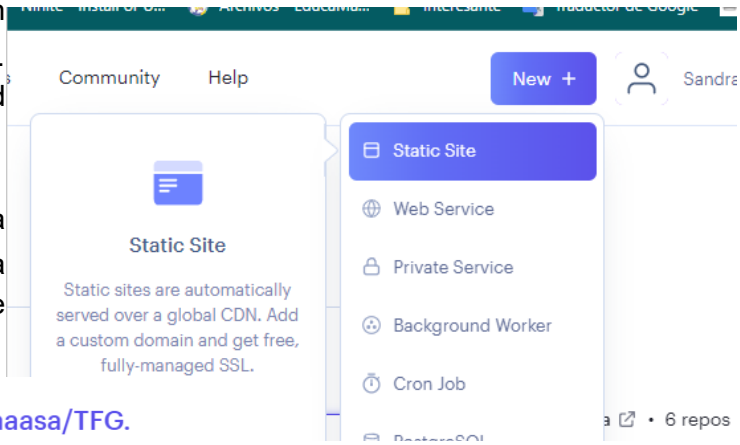



*Figura 44: Static Site*

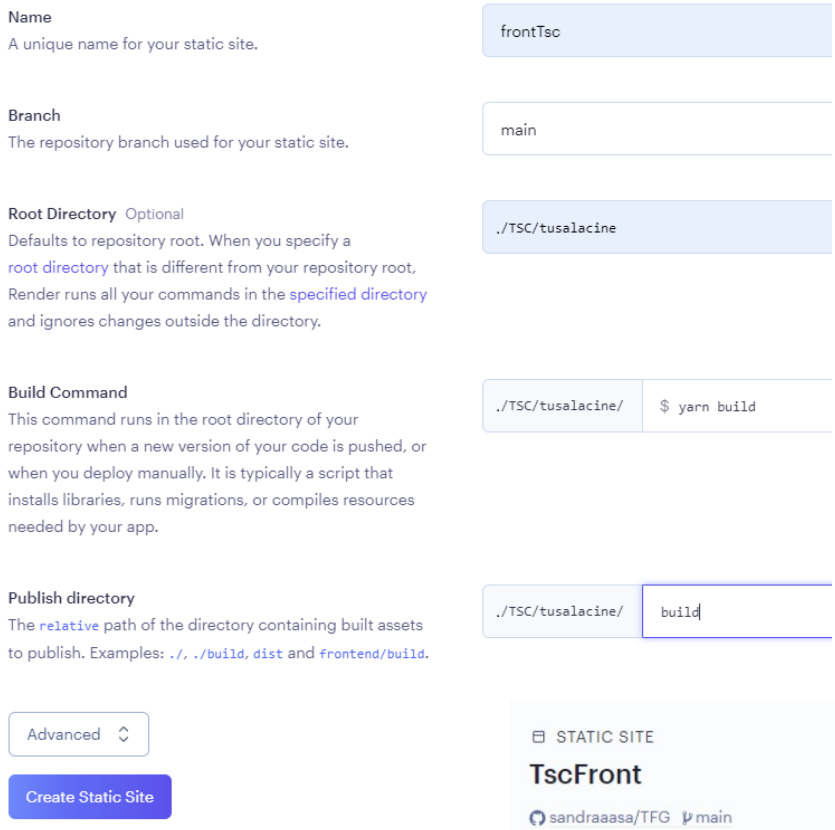



*Figura 45: Configuración despliegue front-end*

3. Esperamos a que el despliegue se efectúe correctamente

4. Obtenemos el enlace de la página

   https://tscfront.onrender.com

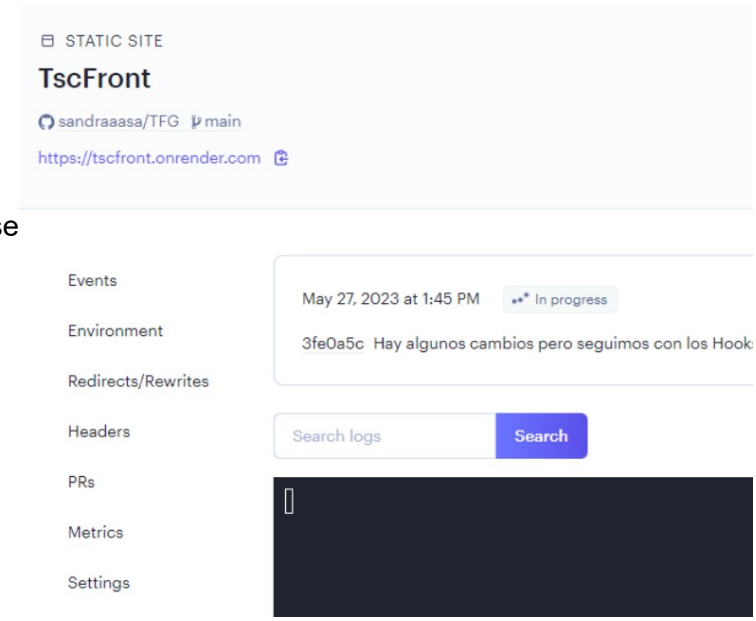



*Figura 46: Despliegue front-end*

# 10. Resultados

Se ha conseguido desarrollar de manera exitosa la funcionalidad principal, permitir a los usuarios disfrutar de películas aleatorias a partir de un pequeño filtro de categoría, además de mostrar un catálogo completo, filtrado y diverso de películas de una base de datos personal.

También se ha conseguido aplicar el tema oscuro y claro mediante métodos de PrimeReact.

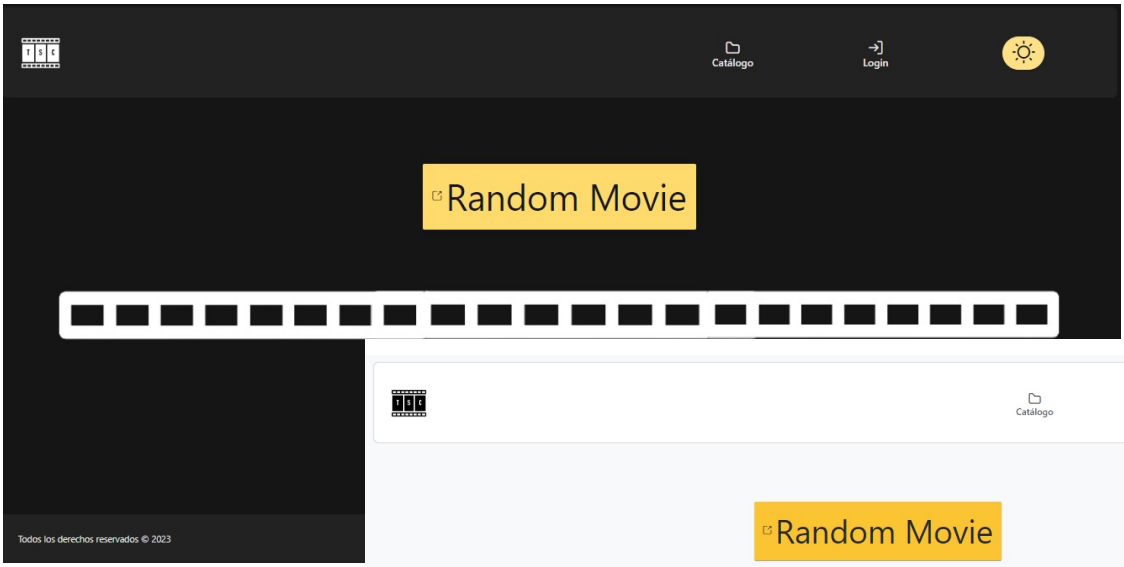



*Figura 47: Resultado 1*


*Figura 48: Resultado 2*

A partir de funciones de React y Node.js aprendidas en el desarrollo del proyecto, incluyendo estilos de PrimeReact, se ha conseguido crear un sistema de autenticación de usuarios funcional y atractivo al usuario.

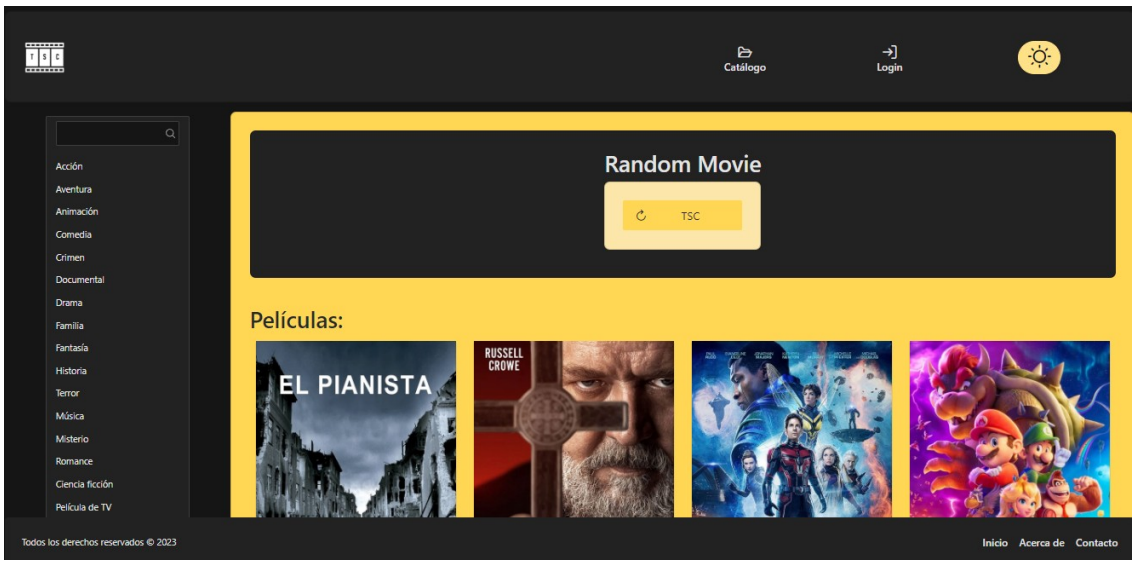



*Figura 49: Resultado 3*

Se ha aprendido bastante sobre las tecnologías de React, Node.js y Mongoose, como por ejemplo la creación de hooks y su modificación en los componentes hijos y la utilización de las diversas funciones de Mongoose.

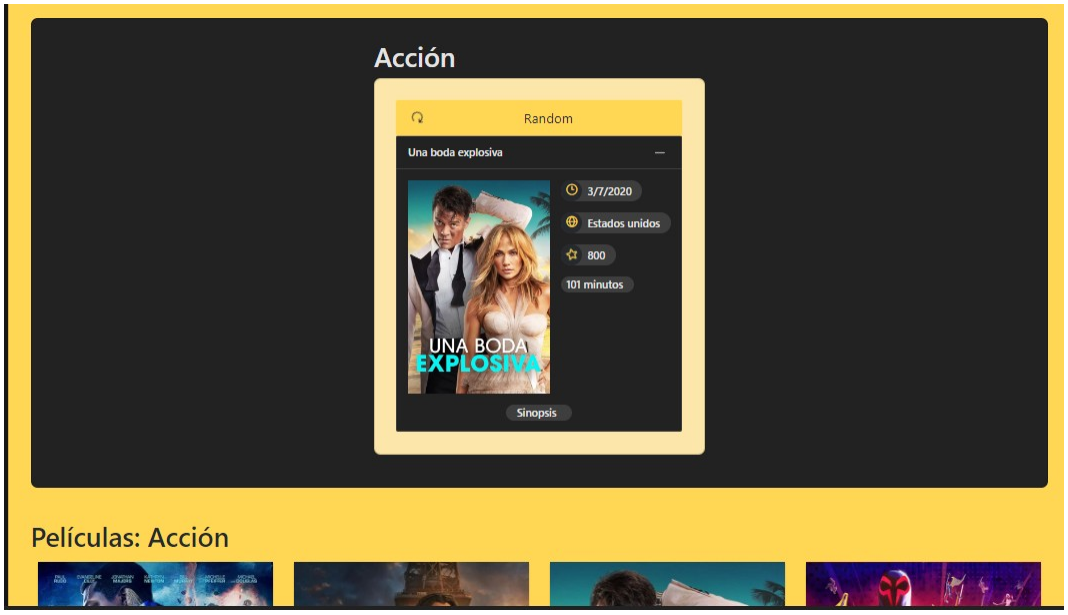



*Figura 50: Resultado 4*

# Conclusiones

Durante el desarrollo de este proyecto me he encontrado con algunos inconvenientes tales como la incorporación y el acceso a las películas desde una API externa, el manejo de los useState y useContext de React para la creación de los usuarios y su autenticación, y para su registro.

En cuanto al acceso a las películas, obtuve la información de cada una de ellas desde la API de TMDB, me dio algunas complicaciones para sacar una película en concreto sin saber el id, por lo que añadí a mi base de datos algunas mediante peticiones get a la API directamente.

A pesar de la complejidad de React y los useContext he logrado que los componentes se actualicen usando funciones, de manera que no solo se pasa la información del useContext sino que también se puede actualizar usando dichas funciones. De esta manera se puede iniciar sesión y cambiar el Header según el usuario logueado.

Debido a ciertas complicaciones, solo se ha conseguido incluir un catálogo limitado de películas en nuestra aplicación web.

# Nuevas propuestas

A partir de esta página se podría conseguir realizar una aplicación o una red social donde mostrar trailers de películas y su información como videos o publicaciones. También se podría crear un foro donde puedes mostrar de formas aleatoria películas y dejar tu opinión según las que se hayan visto.

# Anexos

## 1. Glosario de términos

- **API**: *Interfaz de programación de aplicaciones*, es una pieza de código que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades
- **SPA**: *Single-page aplication,* es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador.
- **Framework**: *Entorno de trabajo,* es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Filmoteca**: Conjunto o colección de filmes.
- **Reviews**: *Reseña*, es una forma de crítica que se escribe en medios impresos o digitales
- **Interfaz de usuario(UI):** Es el medio que permite la comunicación entre un usuario y una máquina, equipo, computadora o dispositivo, y comprende todos los puntos de contacto entre el usuario y el equipo.
- **Responsive**: *Diseño web adaptable*, es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas.
- **Props:** *Propiedades*, son un tipo de objeto donde se almacena el valor de los atributos de una etiqueta. Estos componentes de accesorios son componentes de solo lectura.
- **Hook:** Son funciones que te permiten “enganchar” el estado de React y el ciclo de vida desde componentes de función.
- **UseContext:** Es un hook de React que permite a los componentes acceder a un contexto específico, sin tener que pasar explícitamente los datos a través de props.

# 2. Webgrafía


- Diego Polo, Juan (2010): "*THEMOVIEDB – una base de datos abierta de películas"* Disponible en: https://wwwhatsnew.com/2010/04/27/themoviedb-una-base-de-datos-abierta-de-peliculas/
- The Movie DB: API TMDB puedes encontrar su página en: https://www.themoviedb.org/documentation/api/wrappers-libraries

  Y la información para usar la API en: https://developers.themoviedb.org/3/getting-started/introduction
- Filmaffinity- España: Página de referencia

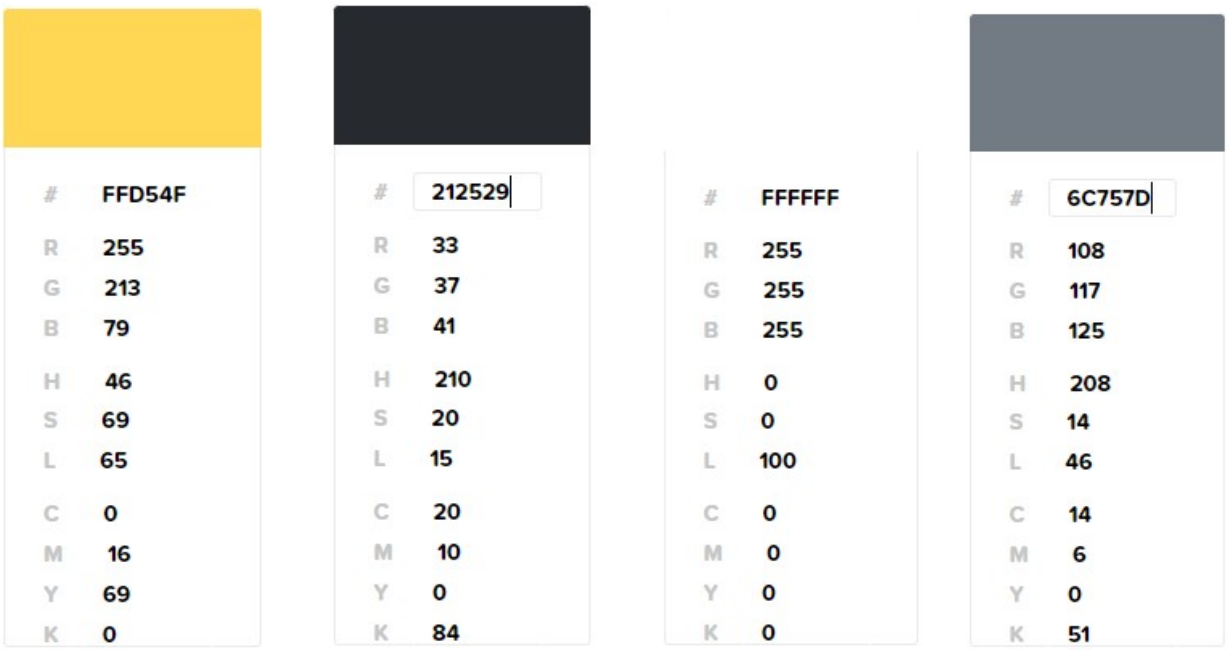
  Disponible en: https://www.filmaffinity.com/es/main.html
- Excalidraw: Página de ayuda. Disponible en https://excalidraw.com
- IMDb: Página de referencia disponible en :https://www.imdb.com/?ref_=nv_home
- Metacritic: Página de referencia disponible en: https://www.metacritic.com
- Rotten Tomatoes: Página de referencia disponible en: https://www.rottentomatoes.com
- Miguel Angel Alvarez( 29 de noviembre de 2016): "*Que es una SPA*" Disponible en:https://desarrolloweb.com/articulos/que-es-una-spa.html
- Jorge Pérez (1 de febrero de 2022): "*React-Notes"* Disponible en el repositorio de GitHub: https://github.com/CodenautaJorge/React-Notes.git
- Noah (30 de junio de 2013): "*Mongoose password hashing*" disponible en: https://stackoverflow.com/questions/14588032/mongoose-password-hashing
- Agustín Navarro Galdón (18 de diciembre de 2020): "*Usando el HOOK useContext de React JS*" disponible en: https://youtu.be/mnKHJDkpZos
- Luis Cabrera (3 de octubre de 2020): "*Context API - explicado al detalle - con ejemplo práctico y solución al problema de renders – React*" disponible en: https://youtu.be/b2psfRzk-r8
- ZeroCool (3 de septiembre de 2021): "*No puedo agregar y usar objetos en localStorage*" disponible en https://es.stackoverflow.com/questions/482113/no-puedo-agregar-y-usar-objetos-en-localstorage
- Verster, Rubén (16 de mayo de 2023): "*How can one have a theme switcher in primereact*" disponible en https://stackoverflow.com/questions/68327342/how-can-one-have-a-theme-switcher-in-primereact
- Gascón Arjol, Oscar (10 de enero de 2015): "*Validación de formularios con HTML5*" disponible en https://oscargascon.es/validacion-de-formularios-con-html5-y-expresiones-regulares-sin-uso-de-js-utilizacion-de-css-y-fontawesome-para-mostrar-campos-validos-en-formularios/

# 3. Guía de estilos

En este apartado hablaré sobre los colores, la tipografía, los iconos y la estructura básica de la página creada en este proyecto.

- ## Colores

Los colores principales de la web son el negro, blanco, gris y naranja.




- ## Tipografía

El tipo de letra que se utiliza en la página es Arial.

> *apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Helvetica, Arial, sans-serif, Apple Color Emoji, Segoe UI Emoji, Segoe UI Symbol*

- ## Iconos

Estos son los iconos usados:

- **Estructura**

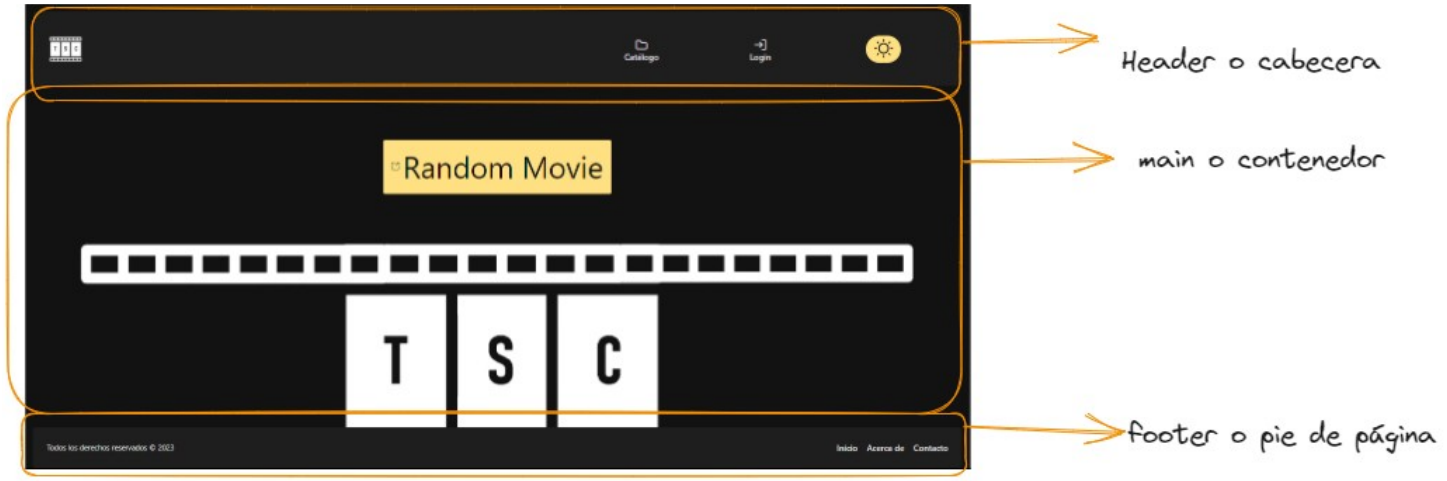



*Figura 51: Estructura final*

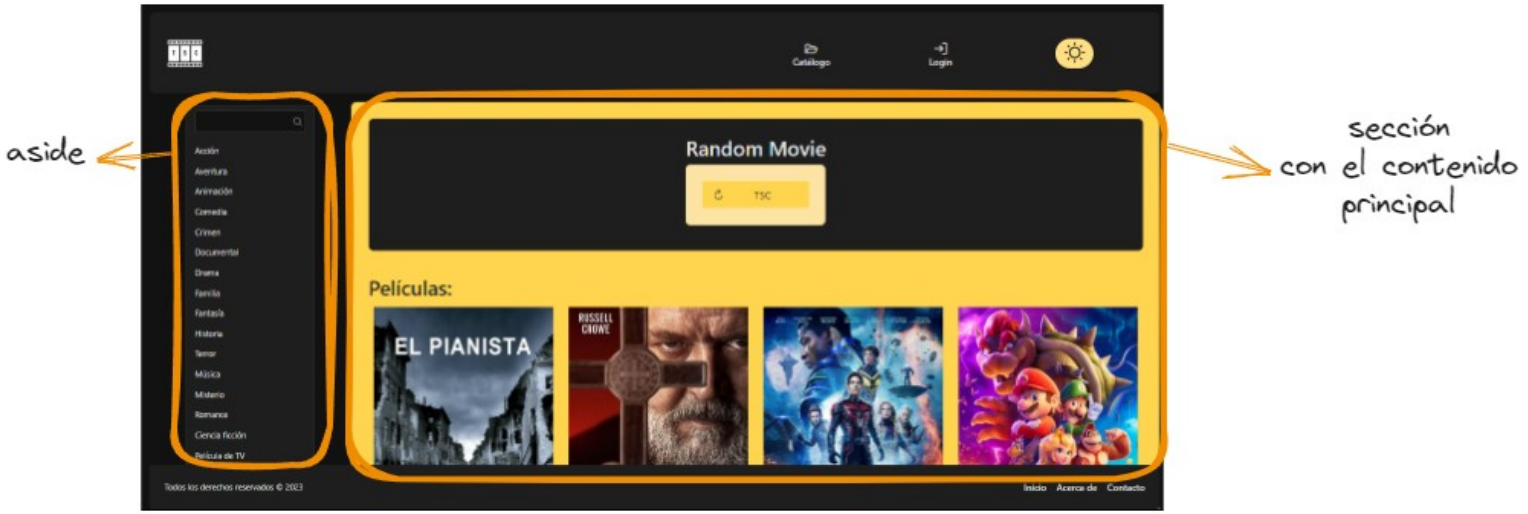



*Figura 52: Estructura final 2*

# 4. Indice de tablas y fotos

## Índice de figuras

# Índice de tablas