

Programación Multimedia y Dispositivos Móviles (UP4)

Autores: Antonio Calabuig Puigvert y Sebastián Villa Ponce

Centro: IES Salvador Gadea

Departamento: Informática

Ciclo: Desarrollo de Aplicaciones Multiplataforma

Fecha: 29/10/2025

1. Introducción y base teórica

2. MoviesApp

 2.1. Creación del proyecto

 2.2. Variables de entorno y API Key

 2.3. Cliente Axios - Primera petición

 2.4. Mapear respuestas

 2.5. Gestor de estado asíncrono

 2.6. Pantalla de carga

 2.7. Carrusel de imágenes

 2.8. FlatList horizontal

3. Actividad 4

1. Introducción y base teórica

- TanStack Query
- Axios
- FlatList
- Paquetes externos
- Reutilización de componentes y parametrización de props
- SafeArea manual

- Consideraciones de contenido
- Scroll dentro de otros Scrollable Elements
- Mapeo y tipado

2. MoviesApp

2.1. Creación del proyecto

1. Como hemos realizado en todos nuestros proyectos, creamos un proyecto nuevo con el comando `npx create-expo-app@latest MoviesApp` desde nuestra carpeta de proyectos.
2. Al igual que en nuestro anterior componente, instalamos NativeWind, para ello seguimos los pasos de la guía oficial:

```
npm install nativewind react-native-reanimated@~3.17.4 react-native-safe-area-context@5.4.0
npm install --dev tailwindcss@^3.4.17 prettier-plugin-tailwindcss@^0.5.11
```

3. Ejecutamos la instancia de NativeWind para crear el fichero `tailwind.config.js`:

```
npx tailwindcss init
```

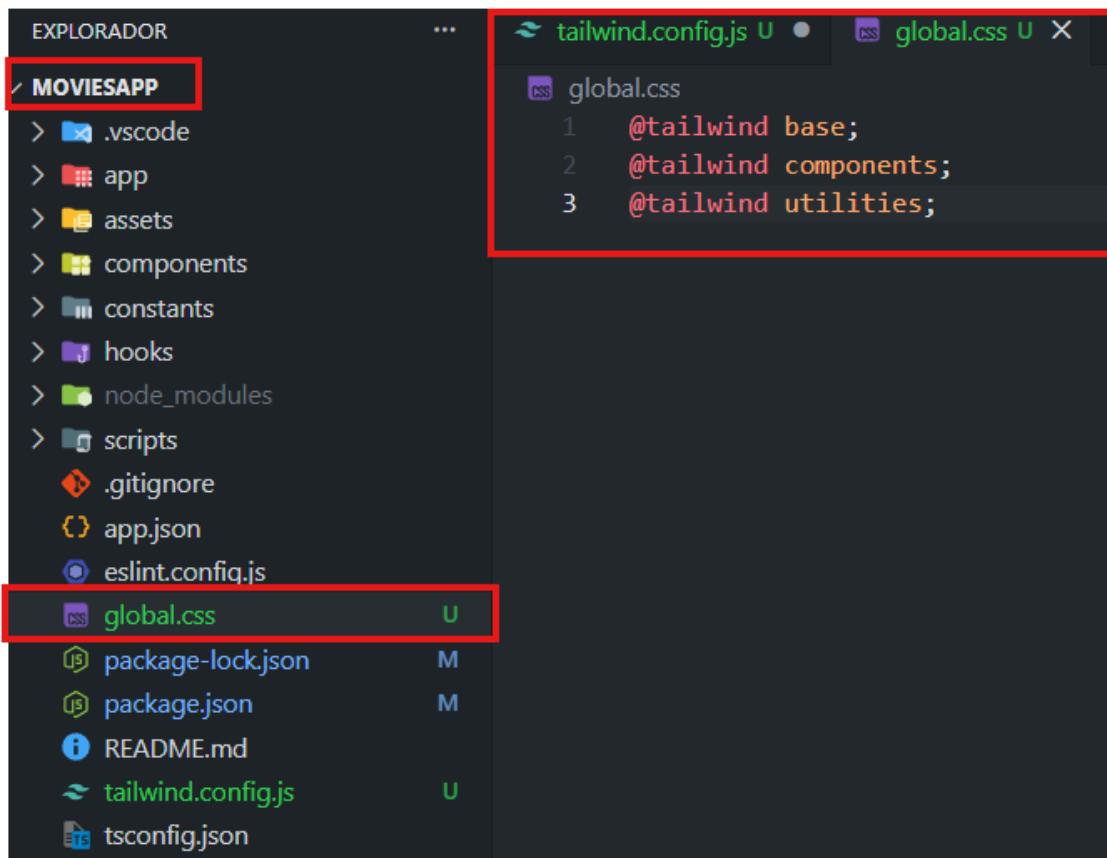
4. Sustituimos el contenido del fichero creado por el que debería tener para afectar solo a las carpetas que deseemos:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./App.tsx",
    "./app/**/*.{js,jsx,ts,tsx}",
    "./components/**/*.{js,jsx,ts,tsx}",
    "./presentation/**/*.{js,jsx,ts,tsx}"
  ],
  presets: [require("nativewind/preset")],
  theme: {
    extend: {},
  },
}
```

```
    plugins: [],
}
```

5. A diferencia del proyecto anterior vamos a respetar las buenas praxis que nos piden en la documentación de NativeWind y vamos a crear el fichero `global.css` en la raíz del proyecto:

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```



6. Creamos el fichero `babel.config.js` en la raíz:

```
module.exports = function (api) {
  api.cache(true);
  return {
    presets: [
      ["babel-preset-expo", { jsxImportSource: "nativewind" }],
      "nativewind/babel",
    ],
}
```

```
};  
};
```

7. Creamos el fichero `metro.config.js` en la raíz:

```
const { getDefaultConfig } = require("expo/metro-config");  
const { withNativeWind } = require('nativewind/metro');  
  
const config = getDefaultConfig(__dirname)  
  
module.exports = withNativeWind(config, { input: './global.css' })
```

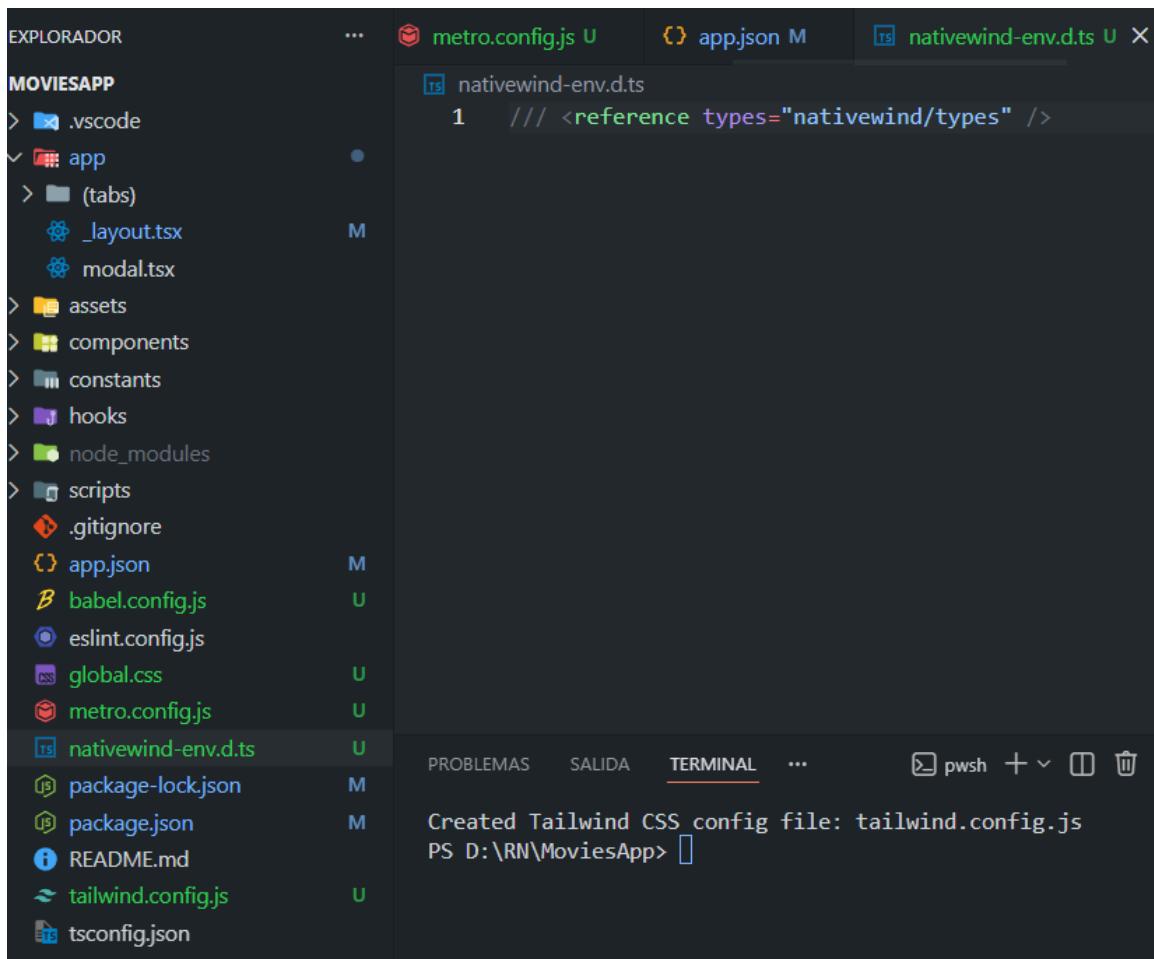
8. De los ficheros que vienen por defecto para mostrar los `tabs` que vienen de inicio vamos a usar por el momento la estructura base pero se irá modificando "a posteriori". Accedemos al fichero `_layout.tsx` de nuestra carpeta `(tabs)` e importamos nuestro fichero de estilos global:

```
// Recordamos poner ../ porque ahora nuestro estilo global está en raíz  
import "../global.css";
```

9. Modificamos nuestro fichero `app.json` para incorporar la supervisión del servidor "metro" de tal forma que nuestra línea de código esté en expo → web → `"bundler": "metro"`:

```
"web": {  
  "output": "static",  
  "favicon": "./assets/images/favicon.png",  
  "bundler": "metro"  
},
```

10. Creamos el fichero `nativewind-env.d.ts` en caso de no haber sido creado y comprobamos que su contenido es: `/// <reference types="nativewind/types" />`



11. Ya casi estamos listos para probar nuestro proyecto. Antes de eso vamos a borrar la estructura y contenido innecesario de nuestro proyecto. Borramos las carpetas: `scripts`, `hooks`, `constants`, de `components` borramos su contenido, la carpeta `(tabs)` la eliminamos, del fichero `_layout.tsx` cortamos nuestra importación de los estilos globales y sustituimos todo el código por un nuevo componente con `rnfe`. El fichero `modal.tsx` lo borramos y comprobamos que el código de nuestro `_layout.tsx` es:

```

import { Text, } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import "../global.css";

const RootLayout = () => {
  return (
    <SafeAreaView>
      <Text className="text-3xl">RootLayout</Text>
    </SafeAreaView>
  )
}
export default RootLayout;

```

12. Compilamos la aplicación y comprobamos que funciona:

50

RootLayout

2.2. Variables de entorno y API Key

1. Para poder acceder al servicio API de películas nos debemos registrar en la web:

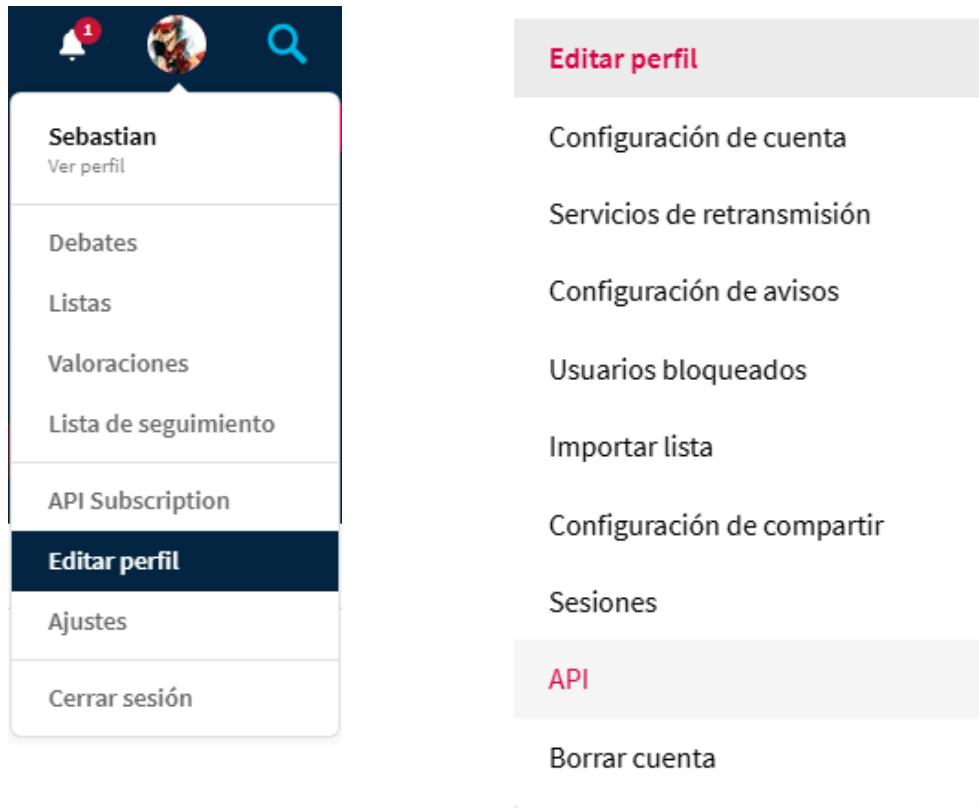
The Movie Database (TMDB)

The Movie Database (TMDB) is a popular, user editable database for movies and TV shows.



<https://www.themoviedb.org/>

2. **Se recomienda usar el correo corporativo.** Accedemos al apartado de **perfil** y posteriormente al apartado **API**:



3. Nos aparecerá un par de campos llamados **Token de acceso** y **Clave de la API**:

Token de acceso de lectura a la API

```
eyJhbGciOiJIUzI1NiJ9eyJhdWQiOiJmYjE1ZGI2YjAwZmJhYjMzMzNzIiYjYwMzM2MjFhNmI0YSIsIm5iZiI6MTcyMDgwODIwNi42NDIsInN1Yil6ljY2OTE3
MzMBIyMq4Njc5MWM0YmM0M2FIYlsInNjb3BlcyI6WyJhcGlfcmlVhZCJdLCJ2ZXJzaW9uljoxfQ.H5J12m7ilHK7fnSjPSmSE1k8hUYCz-
_eNkEs6EYdSbw
```

Clave de la API

```
fb15db6b00fbab3379bb6033621a6b4a
```

4. A partir de este momento vamos a crear nuestras variables de entorno que nos van a permitir el acceso de nuestra aplicación a la API. Para ello creamos un fichero **.env** en la raíz del proyecto y ponemos el siguiente código:

```
EXPO_PUBLIC_TMDB_BASE_URL=https://api.themoviedb.org/3/movie
EXPO_PUBLIC_TMDB_TOKEN=eyJhbGciOiJIUzI1NiJ9eyJhdWQiOiJmYjE1ZGI2Yj
AwZmJhYjMzMzNzIiYjYwMzM2MjFhNmI0YSIsIm5iZiI6MTcyMDgwODIwNi42NDIsI
nN1Yil6ljY2OTE3MzMBIyMq4Njc5MWM0YmM0M2FIYlsInNjb3BlcyI6WyJhcGlfc
mVhZCJdLCJ2ZXJzaW9uljoxfQ.H5J12m7ilHK7fnSjPSmSE1k8hUYCz-_eNkEs6E
YdSbw
EXPO_PUBLIC_TMDB_API_KEY=fb15db6b00fbab3379bb6033621a6b4a
```

El siguiente enlace será **IMPORTANTE** de recordar:

The screenshot shows the TMDB API Reference Getting Started page. On the left, there is a 'Getting Started' section with the URL <https://developer.themoviedb.org/reference/getting-started>. On the right, there is a preview of the page content.

5. Copiamos el fichero `.env` y le ponemos el nombre de `.env.template` para tener una copia de seguridad.
6. Para evitar subir ficheros que contienen credenciales al repositorio de Git donde todo el mundo lo tiene disponible, vamos a eliminar el fichero `.env` del escaneo de Git añadiendo `.env` en el fichero `.gitignore`.

```
# Expo
.expo/
dist/
web-build/
expo-env.d.ts
.env
```

7. En el enlace anterior podemos encontrar que nos pide seleccionar un lenguaje, elegimos `Node` y nos movemos al apartado que pone `Movie Lists`:

The screenshot shows the 'MOVIE LISTS' section of the TMDB API Reference. It lists four endpoints: 'Now Playing', 'Popular', 'Top Rated', and 'Upcoming', each with a 'GET' button.

Endpoint	Method
Now Playing	GET
Popular	GET
Top Rated	GET
Upcoming	GET

8. Con estos pasos ya tenemos hecha la primera preparación de la estructura para la petición de datos a la API.

2.3. Cliente Axios - Primera petición

1. Para separar la capa lógica de los componentes se va a crear una carpeta en raíz llamada `core` y dentro de ella tendremos otra llamada `actions` y a su vez esta contendrá otra dentro que hará referencia a `movies` dado que de momento vamos a llamar a esos métodos del API.

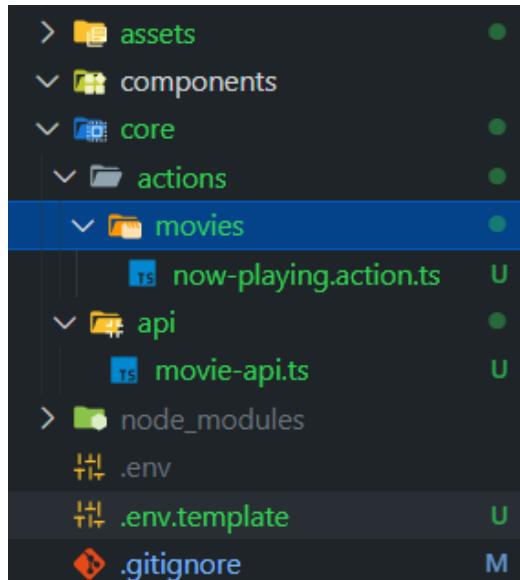
2. Dentro de la carpeta `movies` creamos el fichero `now-playing.action.ts` con el siguiente código a preparar:

```
export const nowPlayingAction = async() => {
  try{

  }catch(error){
    console.log(error);
    throw 'Cannot load now playing movies'
  }
}
```

3. Dentro de `core` creamos otra carpeta `api` que será la que contenga la lógica y la instancia de la aplicación para acceder a la API a través de un fichero llamado `movie-api.ts`

Quedará una estructura similar a:



4. Instalamos nuestro cliente de peticiones HTTP llamado Axios:

```
npm install axios --save
```

```
● PS D:\RN\MoviesApp> npm install axios
          added 7 packages, and audited 995 packages in 9s
          185 packages are looking for funding
            run `npm fund` for details
          found 0 vulnerabilities
○ PS D:\RN\MoviesApp>
```

5. En el fichero `movie-api.ts` importamos la instancia de Axios:

```
import axios from 'axios';

export const movieAPI = axios.create({
  baseURL: process.env.EXPO_PUBLIC_TMDB_BASE_URL,
  params: {
    language: 'es-ES',
    api_key: process.env.EXPO_PUBLIC_TMDB_API_KEY
  }
})
```

6. Volvemos a terminar el fichero `now-playing.action.ts` y lo dejamos con esta forma:

```
import { movieAPI } from '@/core/api/movie-api';

export const nowPlayingAction = async() => {
  try{
    const {data} = await movieAPI.get('/now_playing');
    console.log(JSON.stringify(data));
    return [];
  }catch(error){
    console.log(error);
    throw 'Cannot load now playing movies'
  }
}
```

7. En el `RootLayout` vamos a hacer la llamada a nuestra API para comprobar que funciona antes del retorno de componente:

```
nowPlayingAction();
```

8. Ahora tras recargar, veremos la información que nos devuelve la API en consola.

```
LOG {"dates": {"maximum": "2025-12-10", "minimum": "2025-10-29"}, "page": 1, "results": [{"adult": false, "backdrop_path": "/lZYMXXx74pWmbj5Q5jp1QaMvuuR.jpg", "genre_ids": [28, 14, 53], "id": 1180831, "original_language": "no", "original_title": "Troll 2", "overview": "Cuando un nuevo y peligroso trol amenaza con devastar su país, Nora, Andreas y el comandante Kris se embarcan en la misión más peligrosa de sus vidas.", "popularity": 530.1827, "poster_path": "/pbXxPTmUsrl7zoGc4M1YHMtzGLG.jpg", "release_date": "2025-11-30", "title": "Troll 2", "video": false, "vote_average": 6.8, "vote_count": 194}, {"adult": false, "backdrop_path": "/5h2EsPKNDdB3MAtOk9MB9Ycg9Rz.jpg", "genre_ids": [16, 35, 12, 10751, 9648], "id": 1084242, "original_language": "en", "original_title": "Zootopia 2", "overview": "Después de resolver el caso más importante en la historia de Zootrópolis, los policías novatos Judy Hopps y Nick Wilde descubren que su asociación no es tan sólida como pensaban cuando el Jefe Bogo les ordena unirse al programa de consejería \"Compañeros en Crisis\". Pero no pasa mucho tiempo para que su asociación se ponga a prueba al máximo, cuando la llegada de Gary, la serpiente, pone la ciudad patas arriba.", "popularity": 463.2796, "poster_path": "/oDaWFIrBuOaFrTmBpAlMfmsa81N.jpg", "release_date": "2025-11-26", "title": "Zootrópolis 2", "video": false, "vote_average": 7.707, "vote_count": 447}, {"adult": false, "backdrop_path": "/54BXpX2ieTXMDzHymdDMnUIzYG.jpg", "genre_ids": [27, 53, 10751], "id": 1228246, "original_language": "en", "original_title": "Five Nights at Freddy's 2", "overview": "Un año después de la pesadilla sobrenatural en Freddy Fazbear's Pizza, las historias sobre lo ocurrido allí se han convertido en una leyenda local extravagante, inspirando el primer Fazfest del pueblo. Con la verdad oculta, Abby se escapa para reencontrarse con Freddy, Bonnie, Chica y Foxy, desencadenando una serie de eventos aterradores que revelarán oscuros secretos sobre el verdadero origen de Freddy's y desatarán un horror oculto durante décadas.", "popularity": 312.7226, "poster_path": "/bYdANGEjbb3L1k1KpprAeZ6a5HN.jpg", "release_date": "2025-12-03", "title": "Five Nights at Freddy's 2", "video": false, "vote_average": 6.5, "vote_count": 97}, {"adult": false, "backdrop_path": "/otSXrUWOTX7tTigLBMFkov8n47r.jpg", "genre_ids": [27, 14, 10749], "id": 1246049, "original_language": "fr", "original_title": "Dracula", "overview": "Tras la muerte de su esposa, un príncipe del siglo XV renuncia a Dios y se convierte en vampiro. Siglos más tarde, en el Londres del siglo XIX, ve a una mujer parecida a su difunta esposa y la persigue, sellando así su propio destino.", "popularity": 211.55, "poster_path": "/
```

9. Al ver la información nos damos cuenta que el volumen es alto y la estructura del objeto resultado es profunda y densa y nosotros no tenemos como gestionar dicha estructura, para eso vamos a mapearla y guardarla en una interfaz propia. Para ello vamos a crear una carpeta en raíz con el nombre `infrastructure` y dentro otra llamada `interfaces` y finalmente, dentro tendremos nuestra interfaz llamada `moviedb-response.ts`.
10. Vamos a la API, hacemos la petición al método Now Playing y copiamos el objeto de la respuesta.

The screenshot shows the MovieDB API documentation interface. The 'Now Playing' endpoint is selected and highlighted with a red box. The interface includes a sidebar with various endpoints like Details, Remove Movie, MOVIE LISTS, MOVIES, and more. The main content area shows the endpoint details, recent requests, a note, and an equivalent curl command. The bottom section shows the JSON response with fields like original_language, original_title, overview, popularity, poster_path, release_date, title, video, vote_average, and vote_count. A red box highlights the JSON response area.

11. Con el código copiado, accedemos al fichero con la interfaz de respuesta y con el atajo Control + Shift + P abrimos la terminal de acciones, escribimos `Paste JSON as Code` y al confirmar nos pedirá que escribamos el título de la interfaz principal, en nuestro caso `MovieDBMoviesResponse`.
12. Con el nombre de la cabecera copiado vamos a nuestra acción de llamada y tipamos la respuesta del get:

```
import { MovieDBMoviesResponse } from '@/infrastructure/interfaces/moviedb-response';
...
const {data} = await movieAPI.get<MovieDBMoviesResponse>('/now_playing');
```

2.4. Mapear respuestas

1. Vamos a diferenciar entre la interfaz del objeto que nos llega del API y que no tiene una estructura propia respecto el que vamos a usar nosotros.

Esto sería el equivalente a modificar la estructura de los datos que nos llegan de la capa de persistencia en BD a través de llamadas con la capa DAO y su posterior modificación o transformación en la capa DTO.

2. Creamos una nueva interfaz llamada `movie.interface.ts` :

```
export interface Movie {  
    id: number;  
    title: string;  
    description: string;  
    releaseDate: Date;  
    rating: number;  
    poster: string;  
    backdrop: string;  
}
```

3. Para hacer la transformación vamos a necesitar un traductor de objetos que transforme de uno a otro, a este elemento de código se le llama **Mapper**. Creamos una carpeta hermana de `interfaces` llamada `mappers` y creamos el fichero `movie.mapper.ts`.

```
import { Movie } from '../interfaces/movie.interface';  
import { Result } from '../interfaces/moviedb-response';  
  
export class MovieMapper {  
    static fromTheMovieDBToMovie = (movie: Result): Movie => {  
        return {  
            id: movie.id,  
            title: movie.title,  
            description: movie.overview,  
            releaseDate: new Date(movie.release_date),  
            poster: `https://image.tmdb.org/t/p/w500${movie.poster_path}`,  
            backdrop: `https://image.tmdb.org/t/p/w500${movie.backdrop_path}`,  
            rating: movie.vote_average,  
        }  
    }  
}
```

4. Volvemos a nuestra acción e iniciamos la transformación de un objeto `MovieResponse` a un objeto `Movie`. Para ello modificamos nuestro código de esta forma:

```
const {data} = await movieAPI.get<MovieDBMoviesResponse>('/now_playing');
// Primera y segunda forma son equivalentes
const movies = data.results.map((movie) => MovieMapper.fromTheMovieDBTo
Movie(movie));
const movies = data.results.map(MovieMapper.fromTheMovieDBToMovie);

console.log(JSON.stringify(movies, null, 2));
return movies;
```

El resultado sería algo así:

```
{
  "id": 1116465,
  "title": "El descubridor de leyendas",
  "description": "Un arqueólogo nota que la textura de las reliquias descubiertas durante la excavación de un glaciar se parece mucho a un colgante de jade visto en uno de sus sueños. Él y su equipo se embarcan en una expedición a las profundidades del glaciar.",
  "releaseDate": "2024-07-05T00:00:00.000Z",
  "poster": "https://image.tmdb.org/t/p/w500/jvrSYRtgHTu3oqJN1X7LWZoekB6.jpg",
  "backdrop": "https://image.tmdb.org/t/p/w500/7FDVhmCur4LM0fnXMteiSctsd0b.jpg",
  "rating": 6.8
},
{
  "id": 425274,
  "title": "Ahora me ves 3",
  "description": "Los Cuatro Jinetes regresan junto a una nueva generación de ilusionistas en una extraordinaria aventura que contiene giros asombrosos, sorpresas alucinantes y trucos de magia nunca vistos en la gran pantalla.",
  "releaseDate": "2025-11-12T00:00:00.000Z",
  "poster": "https://image.tmdb.org/t/p/w500/Aspr0HcItNVbgLC8cV0ZA00EBcc.jpg",
  "backdrop": "https://image.tmdb.org/t/p/w500/ufqytAlziHq5pljKByGJ8IKhtEZ.jpg",
  "rating": 6.301
},
{
  "id": 701387,
  "title": "Bugonia",
  "description": "Un joven captura e interroga a un hombre de negocios que cree que es un invasor alienígena. El captor, su novia, el empresario y un detective privado se enzarzan en una tensa batalla psicológica. Remake del film coreano \"Save the Green Planet\".",
  "releaseDate": "2025-10-23T00:00:00.000Z",
  "poster": "https://image.tmdb.org/t/p/w500/2C5Vf9VcoXB7SFFcG8vppW5bmtX.jpg",
  "backdrop": "https://image.tmdb.org/t/p/w500/tN3pTxkQoP96wtaEahYuRVdUWb2.jpg",
  "rating": 7.5
}
```

6. Ya podemos dejar preparado el código para los próximos pasos eliminando la salida por consola que tenemos y la llamada `nowPlayingAction()`; dado que no debe ir ahí.

2.5. Gestor de estado asíncrono

En estos momentos nuestro código puede parecer funcional pero tiene bastantes carencias:

- a. No hay gestión de carga durante la transacción de la llamada HTTP que mantenga al usuario informado.
- b. No hacemos ninguna gestión adecuada de los errores para identificar fallos.
- c. Estamos retornando una lista de `Movies` pero no sabemos si será así en todo momento.
- d. La llamada se hace una vez cada bastante tiempo y no estamos haciendo uso de recursos como la caché de memoria o la propia memoria del dispositivo para evitar llamar cuando ya tenemos los datos.

Todas estas gestiones requieren de `hooks` basados en estados para gestionar adecuadamente los recursos del dispositivo y la calidad de usanza del usuario.

Para resolver estas cuestiones vamos a utilizar una librería llamada `TanStack Query` :

TanStack | High Quality Open-Source Software for Web Developers
Headless, type-safe, powerful utilities for complex workflows like Data Management, Data Visualization, Charts, Tables, and UI Components.

 **TANSTACK**
High-quality open-source software for web developers.
Headless, type-safe & powerful utilities for Web Application, Routing, State Management, Data Visualization, DataGrids/Tables, and more.

 <https://tanstack.com/>

Nos registramos en la página a través de GitHub o Google y accedemos al siguiente enlace:

Installation | TanStack Query React Docs
You can install React Query via , or a good ol' <script> via . NPM bash
npm i @tanstack/react-query or bash pnpm add @tanstack/react-query or bash yarn add @tanstack/react-query or bash bun add
 <https://tanstack.com/query/latest/docs/framework/react/installation>

 **TanStack Query** v5
Powerful asynchronous state management for TS/JSS, React, Solid, Vue and Svelte
Toss out that granular state management, manual refetching and endless bowls of async-spaghetti code. TanStack Query gives you declarative, always-up-to-date auto-managed queries and mutations that directly improve both your developer and user experiences.

1. Instalamos la librería:

```
npm i @tanstack/react-query
```

2. Vamos al apartado de `Quick Start` y copiamos el componente que debe envolver el resultado de nuestro `RootLayout` para que este disponible en todo el sistema.

```

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
...
//Fuera del RootLayout
const queryClient = new QueryClient()

...
//Dentro del RootLayout
<QueryClientProvider client={queryClient}>
  <SafeAreaView>
    <Text className="text-4xl" >RootLayout</Text>
  </SafeAreaView>
</QueryClientProvider>

```

- Vamos a preparar la estructura para las pantallas que verá el usuario: Creamos una carpeta en raíz llamada `presentation` que contenga una carpeta `hooks` que contenga un fichero llamado `useMovies.tsx`.

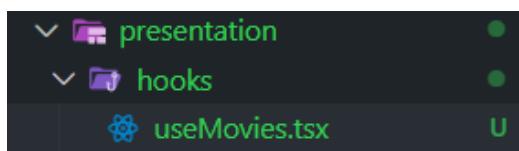
```

import { nowPlayingAction } from '@/core/actions/movies/now-playing.action';
import { useQuery } from '@tanstack/react-query';

export const useMovies = () => {
  // Queries
  const nowPlayingQuery = useQuery({
    queryKey: ['movies','nowPlaying'],
    queryFn: nowPlayingAction,
    // 1000ms * 60s * 60m * 24h → Mantiene los datos durante 24h
    staleTime: 1000 * 60 * 60 * 24
  })

  return {
    nowPlayingQuery,
  }
}

```



- Vamos a crear lo que serán nuestras pantallas de visualización `home/index.tsx` y un `index.tsx` en `app`.

```

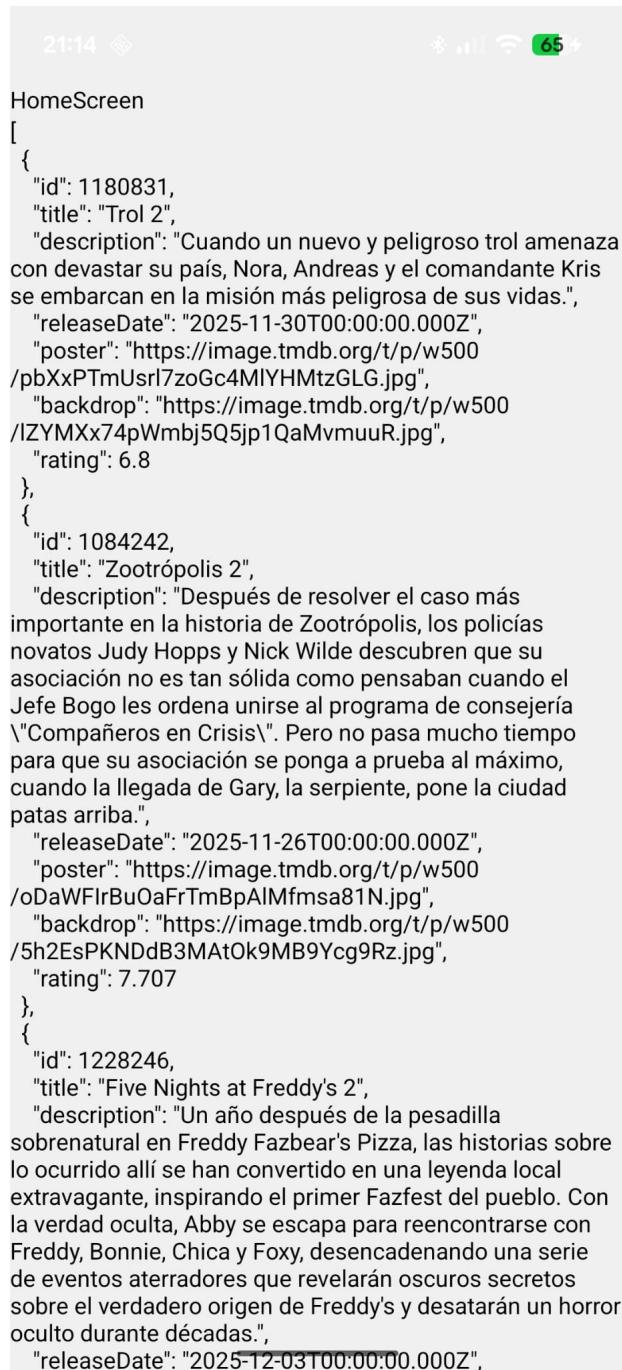
// app/index.tsx
import { Redirect } from 'expo-router';
const MoviesApp = () => {
  return (
    <Redirect href='/home' />
  )
}
export default MoviesApp

// home/index.tsx
import { useMovies } from '@/presentation/hooks/useMovies';
import { Text } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
const HomeScreen = () => {
  const {nowPlayingQuery} = useMovies();
  return (
    <SafeAreaView>
      <Text>HomeScreen</Text>
      <Text>{
        JSON.stringify(nowPlayingQuery.data, null, 2)
      }</Text>
    </SafeAreaView>
  )
}
export default HomeScreen

// app/_layout.tsx
return (
  <QueryClientProvider client={queryClient}>
    <Stack screenOptions={{headerShown: false}}/>
  </QueryClientProvider>
)

```

5. Con estos pasos nuestra aplicación se verá similar a:



21:14 65+

```
HomeScreen
[
  {
    "id": 1180831,
    "title": "Trol 2",
    "description": "Cuando un nuevo y peligroso trol amenaza con devastar su país, Nora, Andreas y el comandante Kris se embarcan en la misión más peligrosa de sus vidas.",
    "releaseDate": "2025-11-30T00:00:00.000Z",
    "poster": "https://image.tmdb.org/t/p/w500/pbXxPTmUsrl7zoGc4MlYHMtzGLG.jpg",
    "backdrop": "https://image.tmdb.org/t/p/w500/IZYMXx74pWmbj5Q5jp1QaMvmuuR.jpg",
    "rating": 6.8
  },
  {
    "id": 1084242,
    "title": "Zootrópolis 2",
    "description": "Después de resolver el caso más importante en la historia de Zootrópolis, los policías novatos Judy Hopps y Nick Wilde descubren que su asociación no es tan sólida como pensaban cuando el Jefe Bogo les ordena unirse al programa de consejería \"Compañeros en Crisis\". Pero no pasa mucho tiempo para que su asociación se ponga a prueba al máximo, cuando la llegada de Gary, la serpiente, pone la ciudad patas arriba.",
    "releaseDate": "2025-11-26T00:00:00.000Z",
    "poster": "https://image.tmdb.org/t/p/w500/oDaWF1rBuOaFrTmBpAIMfmsa81N.jpg",
    "backdrop": "https://image.tmdb.org/t/p/w500/5h2EsPKNDdB3MAtOk9MB9Ycg9Rz.jpg",
    "rating": 7.707
  },
  {
    "id": 1228246,
    "title": "Five Nights at Freddy's 2",
    "description": "Un año después de la pesadilla sobrenatural en Freddy Fazbear's Pizza, las historias sobre lo ocurrido allí se han convertido en una leyenda local extravagante, inspirando el primer Fazfest del pueblo. Con la verdad oculta, Abby se escapa para reencontrarse con Freddy, Bonnie, Chica y Foxy, desencadenando una serie de eventos aterradores que revelarán oscuros secretos sobre el verdadero origen de Freddy's y desatarán un horror oculto durante décadas.",
    "releaseDate": "2025-12-03T00:00:00.000Z"
  }
]
```

2.6. Pantalla de carga

1. Cerramos todo y vamos a nuestra pantalla `Home` .
2. Vamos a crear el contenido de la pantalla de carga con el siguiente código:

```
import { SafeAreaView, useSafeAreaInsets } from 'react-native-safe-area-context';
const HomeScreen = () => {
```

```

const safeArea = useSafeAreaInsets();
const {nowPlayingQuery} = useMovies();

if(nowPlayingQuery.isLoading){
  return(
    <SafeAreaView className="justify-center items-center flex-1">
      <ActivityIndicator color="purple" size={40}/>
    </SafeAreaView>
  )
}

return(
  <View className="" style={{paddingTop: safeArea.top}}>
    <Text className="text-3xl font-bold px-4 mb-2">HomeScreen</Text>
  </View>

/* <SafeAreaView className="">
  <Text>HomeScreen</Text>
</SafeAreaView> */
)

```

3. En el caso de hacer una primera carga, se verá una flecha indicando que hay que refrescar la aplicación.

2.7. Carrusel de imágenes

Las listas laterales para mostrar las películas según el título o tendencia o sección en la que estamos se puede realizar con `FlatList` pero vamos a usar una librería para crear un carrusel.

1. Incluimos en el fichero `tailwind.conf.js` la carpeta de `presentation` en caso de no haberlo hecho.
2. Accedemos al siguiente enlace y comprobamos en la web los efectos de nuestro carrusel.

<https://www.npmjs.com/package/react-native-reanimated-carousel>

3. Ejecutamos los comandos de instalación:

```
npm i react-native-reanimated-carousel  
npx expo install react-native-reanimated react-native-gesture-handler
```

4. Dentro de la carpeta presentation creamos fichero llamado `components/MainSlideshow.tsx` con el siguiente código:

```
import { Movie } from '@/infrastructure/interfaces/movie.interface';
import { useRef } from 'react';
import { Text, useWindowDimensions } from 'react-native';
import Carousel, { ICarousellInstance } from "react-native-reanimated-carousel";
import { SafeAreaView } from 'react-native-safe-area-context';

interface Props {
  movies: Movie[];
}

const MainSlideshow = ({movies}:Props) => {

  const ref = useRef<ICarousellInstance>(null);
  const width = useWindowDimensions().width;

  return (
    <SafeAreaView className="h-[250px] w-full">
      <Carousel
        ref={ref}
        data={movies}
        renderItem={({item}) => <Text>{item.title}</Text>}
        width={200}
        height={350}
        style={{
          width: width,
          height: 350,
          justifyContent: 'center',
          alignItems: 'center',
        }}
      mode="parallax"
    
```

```

        modeConfig={{
          parallaxScrollingScale: 0.9,
          parallaxScrollingOffset: 50,
        }}
        defaultIndex={1}
      />
    </SafeAreaView>
  )
}

export default MainSlideshow;

```

5. Ahora vamos a crear la visualización del póster con el siguiente código y la modificación del renderizador:

```

import "@/global.css";
import { Image, Pressable } from 'react-native';

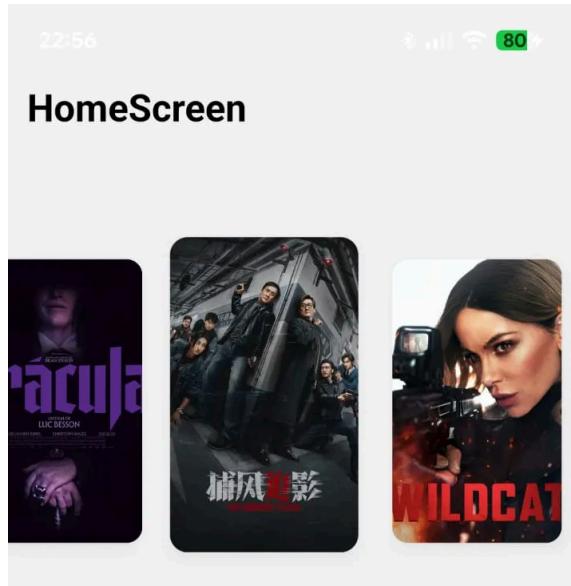
interface Props{
  id: number;
  poster: string;
  smallPoster?: boolean;
}

const MoviePoster = ({id, poster, smallPoster=false}:Props) => {
  return (
    <Pressable>
      <Image source={{uri: poster}}
        className="shadow-lg rounded-2xl w-full h-full"
        style={{
          width: smallPoster? 85 : 150,
          height: smallPoster? 130 : 250
        }}
        resizeMode="cover"
      />
    </Pressable>
  )
}

export default MoviePoster

```

```
// Modificamos el renderItem
renderItem={({item}) => <MoviePoster id={item.id} poster={item.poster}/>}
```



6. Volvemos al póster y ahora vamos a darle algún tipo de interacción a las imágenes:

```
import "@/global.css";
import { Image, Pressable } from 'react-native';

interface Props{
  id: number;
  poster: string;
  smallPoster?: boolean;
  className?: string;
}

const MoviePoster = ({id, poster, smallPoster=false, className}:Props) => {
  return (
    <Pressable className={`active:opacity-80 px-2 ${className}`}>
      <Image source={{uri: poster}}
        className="shadow-lg rounded-2xl w-full h-full"
        style={{
          width: smallPoster? 85 : 150,
          height: smallPoster? 130 : 250
        }}
        resizeMode="cover"
      />
    </Pressable>
  )
}
```

```

    )
}

export default MoviePoster

```

De esta forma ahora cuando mantenemos pulsada una película pierde algo de opacidad y cuando queramos entregar algún estilo personalizado a nuestra imagen presionable, lo aceptará y aplicará.

2.8. FlatList horizontal

Las diferentes secciones que tendrá la aplicación serán todas muy homogéneas por lo que deberíamos crear un componente reutilizable.

1. Copiamos nuestra acción de vistas actualmente y le cambiamos el nombre a `popular.action.ts`.
2. Quedará muy similar al anterior:

```

import { movieAPI } from '@/core/api/movie-api';
import { MovieDBMoviesResponse } from '@/infrastructure/interfaces/moviedb-response';
import { MovieMapper } from '@/infrastructure/mappers/movie.mapper';

export const popularMoviesAction = async() => {
  try{
    const {data} = await movieAPI.get<MovieDBMoviesResponse>('/popular');

    const movies = data.results.map(MovieMapper.fromTheMovieDBToMovie);

    return movies;
  }catch(error){
    console.log(error);
    throw 'Cannot load now playing movies'
  }
}

```

3. Vamos a nuestro `hook` `useMovies` y lo actualizamos con el nuevo servicio:

```

import { nowPlayingAction } from '@/core/actions/movies/now-playing.action';
import { popularMoviesAction } from '@/core/actions/movies/popular.action';
import { useQuery } from '@tanstack/react-query';

export const useMovies = () => {

```

```

// Queries
const nowPlayingQuery = useQuery({
  queryKey: ['movies','nowPlaying'],
  queryFn: nowPlayingAction,
  // 1000ms * 60s * 60m * 24h → Mantiene los datos durante 24h
  staleTime: 1000 * 60 * 60 * 24
})

const popularQuery = useQuery({
  queryKey: ['movies','popular'],
  queryFn: popularMoviesAction,
  // 1000ms * 60s * 60m * 24h → Mantiene los datos durante 24h
  staleTime: 1000 * 60 * 60 * 24
})

return {
  nowPlayingQuery,
  popularQuery
}
}

```

4. Creamos nuestro componente `MovieHorizontalList` en `movies` y añadimos el siguiente código:

```

import { Movie } from '@/infrastructure/interfaces/movie.interface';
import { FlatList, Text } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import MoviePoster from './MoviePoster';

interface Props{
  title?: string;
  movies: Movie[];
}

const MovieHorizontalList = ({title, movies}:Props) => {
  return (
    <SafeAreaView>
      {title && <Text className="text-3xl font-bold px-4 mb-2">{title}</Text>}

```

```

<FlatList
  horizontal
  data={movies}
  showsHorizontalScrollIndicator={false}
  keyExtractor={(item) => `${item.id}`}
  renderItem={({item}) => (
    <MoviePoster id={item.id} poster={item.poster} smallPoster/>
  )}
/>
</SafeAreaView>
)
}

export default MovieHorizontalList

```

5. Actualizamos la `Home` con la creación del siguiente componente:

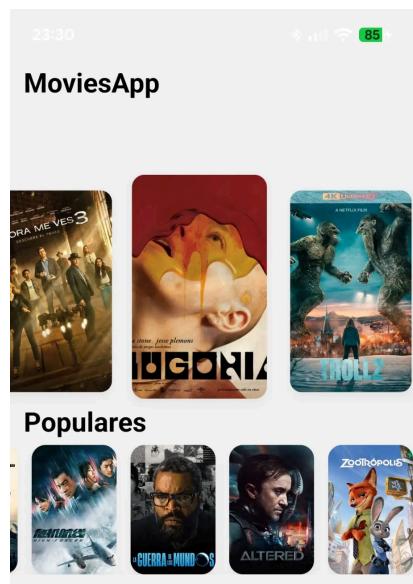
```

return(
  <View className="mt-2" style={{paddingTop: safeArea.top}}>
    <Text className="text-3xl font-bold px-4 mb-2">MoviesApp</Text>
    {/**Carrusel de imagenes */}
    <MainSlideshow movies={nowPlayingQuery.data ?? []}/>

    {/**POPULAR MoviesHorizontalList */}
    <MovieHorizontalList title="Populares" movies={popularQuery.data?? []}/>
  </View>
);

```

6. El resultado debería ser algo similar:



3. Actividad 4

Ya hemos realizado 2 ejemplos de como crear las estructuras para añadir servicios de llamada a la API y como modificar la estructura definida para añadir las nuevas llamadas.

Ahora toca realizar de forma autonoma los otros dos apartados esperados de la aplicación: **Mejor valoradas y Próximamente**. Para ello lo único necesario es conocer los puntos de entrada a la API:

- `/top_rated`
- `/upcoming`

El estilo debe ser coherente con las otras peticiones hechas.