Desafío técnico Frontend: **Gabriel Palma**
Fecha de entrega: 26/04/2023

Project name: **Veri-Movies**
Url: https://veri-movies.web.app
GitHub: https://github.com/GabrielFedericoPalma/veri-movies

1- Requerimiento:
Como usuario de un video club, accedo mediante login a la lista de películas del video club.

1- Solución:
Login implementado con Firebase.

1- Nota
Para acceder a la app crear un usuario desde la vista login.

1- Detalle:
Se implementó un middleware para proteger las rutas de accesos sin login.
Este middleware hace un await de una promise del objeto auth de Firebase para continuar.

La lógica de la conexión con Firebase está en un composable para centralizar las llamadas.

2- Requerimiento:
Las pantallas deben estar inspiradas en las plataformas de streaming de películas como Netflix, Prime Video, Paramount+, etc. El diseño debe ser responsive y se debe tener en cuenta la accesibilidad para personas con discapacidades visuales. Cualquier otra idea innovadora será bienvenida.

2- Solución:
Diseño inspirado en plataforma Netflix y maquetación responsive con foco en la accesibilidad.

2- Detalle:
La maquetación está realizada en unidad rem para poder centralizar el tamaño del layout.

Para la accesibilidad, se trabajó en: HTML semántico, UI con contraste y textos alternativos.

2- Idea innovadora: Se desarrolló un banner tipo carrusel con el objetivo de aplicar una lógica de optimización a la prueba técnica.
Esta optimización consiste en que, los items del banner, se obtienen del estado local de la aplicación. De todas maneras, las imágenes del banner no son locales (se traen por http) por este motivo la funcionalidad del banner es sólo para desktop.

3- Requerimiento:

Se solicita interactuar con la API (https://www.omdbapi.com) para:

1. Obtener el listado de películas.
2. Obtener el detalle de una película.

3- Nota
Tomé la iniciativa de utilizar otra API ya que, con la proporcionada no pude obtener más de una película por página.

La API que utilicé es:
https://developers.themoviedb.org/3/getting-started/introduction

3- Solución:
Mediante un objeto de configuración se define el tipo de llamadas que se va a hacer a la API para obtener las listas de objetos para cada escenario en particular.

3- Detalle:
El componente Items, realiza una llamada a la api de acuerdo al parámetro props type definido en el objeto de configuración.

Este parámetro sirve para obtener el tipo de llamada. (Ej, *Top related, Popular, Upcoming*)

El componente items se renderiza en la vista, de acuerdo a la cantidad de items del objeto inicial de configuración.

Para optimizar llamadas, el detalle de cada película lo trae desde el estado local de la aplicación.

Traté de utilizar nombres genéricos (ej, item, items) para desacoplar la app del tipo de API.

Para el loading de los items, se implementó un Teleport a fin de poder individualizar el spinner por cada componente Items. (enviando por props el nombre de la clase del componente)

Nota final:

Como objetivo del proyecto, traté de modularizar lo más posible los componentes y las llamadas a las APIs, para lograr reusabilidad y escalabilidad.

Antes de la entrega, noté que el middleware no está funcionando del todo bien. Hay un error que indica: no se encuentra la instancia de Nuxt.
Por lo que investigué, puede ser porque el middleware espera una llamada a Firebase que se hace desde el cliente y la app es SSR.
Teniendo en cuenta que no es requisito de la prueba técnica implementar un middleware, y, que ya estoy en fecha de entrega, lo dejo como un tema a resolver.
De todas maneras se implementó la lógica del middleware, la lógica de llamadas a Firebase, y un plugin con la configuración inicial de Firebase.