PÁGINA PELÍCULAS

LAURA GREGORIO ROSAS

ÍNDICE

ESTRUCTURA DE CÓDIGO E INVOCACIÓN DE FUNCIONES

-Html: el html en un principio es muy simple, con un para el header con el nombre de la web y un botón para descargar este pdf, un para el login vacío y otro que contiene a su vez dos , uno para la parte de buscar películas y otra para la lista de favoritos, pero de primeras también están vacíos. El html de mi web se ha ido creando en el código de javascript así que lo explicaré en ese apartado.

-CSS: He utilizado para el header un background color rosa que he seguido utilizando en varios elementos de la web. Tanto en el título como en el botón de descargar he usado iconos de fontawesome. Este último botón al hacerle hover hace display: none del primer icono para hacer display: flex de otro, además tiene position: absolute para poderlo fijar a la derecha. El login tiene el título centrado con padding a ambos lados. El botón tiene márgenes, no tiene color de borde y tiene el borde redondeado con border-radius. Además al hacerle hover cambia su background color. El botón de cambiar usuario es exactamente igual. El 'contenido', que es el padre de los 2 principales, tiene un display: grid con 2 columnas y 1 fila, además de 15px de gap entre columnas. El de la izquierda, 'buscarPeliculas', centra su contenido con un display: flex y align-items: center. Además he puesto flex-direction: column para que su contenido se vea en columna. Los botones de las páginas también tienen display: flex y align-items: center para centrar los iconos de las flechas. El buscador tiene display: flex, align-items: center y flex-direction: row para que el input y el botón de buscar (sus hijos) estén alineados. Este botón también tiene márgenes, border: none y border: radius y cambia de color al hacer hover. El de la derecha, 'miSeleccion',

tiene display: flex y flex-drection: column. Los elementos de lista (películas favoritas) tiene márgenes y paddings por todos los lados para que estén bien centrados. Los botones de borrar e info son iguales, mismo background color y hover, border: none, border: radius y cursor: pointer entre otras cosas. El título de la película cuando se muestra la información está centrado con text align: center. El botón de cerrar 'X' la información de la película tiene position: absolute y left: 0 para pegarlo a la izquierda. Las propiedades de la información están en un con display: flex y flex-direction: columna para que se vean en columna y text-align: center para que estén centrados.

Finalmente para que sea responsive he creado dos breakpoints, uno en max-width 768 px que principalmente cambia el grid del contenido a uno de dos filas y una columna. También a los 'buscarPeliculas' y 'miSeleccion le da márgenes por la izquierda y por la derecha y reduzco el tamaño del buscador. El segundo breakpoint está en max-width 390 px y añade más márgenes a los principales y reduce también el tamaño del buscador.

-Javascript: primero hay un script de type=module en el que se encuentra toda la configuración de firebase y mi proyecto. Hay dos funciones (ponValor y actualizarValor) que actualizan el contenido de un documento de una colcción que le pasemos como referencia. Esas funciones serán llamadas desde el siguiente script, que es el que contiene todo el código.

En este script lo primero que tengo es declaradas todas las variables que quiero tener a nivel global porque van a ser usadas por distintas funciones.

Le he añadido a la ventana un evento para que cada vez que se refresque llame a otra función para recoger de FireStore la lista de favoritos del último usuario. Después he utilizado localStorage para tener localizado al último usuario de forma más fácil, las películas vistas también las he almacenado en un array en localStorage.

Se comprueba si hay algún usuario en el array 'usuarios', y en tal caso el html cambia y no es el del login, sino el del buscador de películas y lista de favoritos y además se saluda a ese usuario (el último que hizo login). Si no hay ninguno en el array es que no hay nadie registrado y el html incluye el input y botón del login para registrarse.

La función recogerUsuariosFb() es invocada al refrescar la página y recibe como parámetro el nombre del último usuario que inició sesión, hacemos una referencia a la colección 'Usuarios' de firestore y la recogemos en docSnap. Creamos un objeto javascript de la data de docSnap y lo recorremos añadiendo al html la lista de películas favoritas. Cuando es la primera vez que ese usuario hace login, aparece este mensaje en consola: "No se encontró ningún documento con ese ID."

La función guardarUsuario() se invoca al clickar en el button Login, guardo en una variable el nombre de usuario del input donde se escribe el nombre de usuario. Esta variable la añado al array de usuarios() y también a la colección de firestore de 'Usuarios' con la función ponValor(). Yo he decidido que cada usuario que se guarde sea un documento de la colección con un campo que tiene un array de Favoritos. El array de 'usuarios' también lo he almacenado en localStorage para tener guardado el último que inició sesión (la última posición del array). Compruebo si el usuario que se ha guardado ya existía en mi colección usando la función getDoc, en caso de que exista, se muestra sus películas favoritas almacenadas en firestore. En caso de que no, se avisa por consola. También se comprueba si ese usuario tiene almacenadas películas vistas en el localStorage para en caso afirmativo recuperarlas.

La función cambiarUsuario() se invoca al clickar en el botón Cambiar usuario. El html cambia, desaparece el buscador de pelis y el apartado de favoritos y aparece el login

La función info() es invocada cuando introduces el nombre de una película y pulsas el botón Buscar y cuando pulsas alguna de las flechas para cambiar de página, recibe como parámetro la página actual. Utilizo la url de la API con el link de tmdb, la key, la query (el título introducido en el buscador) y la página actual. A esta url le hago una llamada utilizando el fetch, en caso de que la llamada sea correcta creamos un objeto json de la llamada a la api e itero sobre las propiedades de dicho objeto, de las cuales yo he decidido mostrar en un <details> una lista con el título, fecha de lanzamiento e idioma original.

La función mostrarBotonesPaginacion() es invocada al final de la función anterior. Se puede saber cuántas páginas hay en una búsqueda concreta ya que el objeto json que se creó en la función info() tiene una propiedad que es total_pages. Si hay más de 1 página, se crean dos botones (uno para volver y otro para avanzar) y un párrafo con el número de página y se añaden al html. He creado un evento al hacer click en el botón de volver en el que se resta 1 a la variable páginaActual y llamamos a info() pasándole este nuevo parámetro. Para el botón de avanzar hay otro evento exactamente igual pero que le suma 1 a la páginaActual. Como quiero que cuando esté en la 1 página no aparezca el botón de volver y cuando esté en la última no aparezca el de siguiente, compruebo cuál es la página actual para añadir al contendedor de los botones el botón correspondiente. En las páginas intermedias aparecen ambos.

La función guardarFavorito() es invocada al pulsar en el botón like (corazón) y recibe como parámetro el título de la película. Añado dicha película a mi array de favoritos y actualizo el campo de Favoritos del firestore pasándole en la función actuzalizarValor() el array. En el html, la parte de la derecha con las películas favoritas se actualiza y se

crea un elemento que tiene el título de la peli (del parámetro) y 3 botones, uno para borrar, otro para información y un último para marcar como visto.

La función borrarFavorito() es invocada en el botón Borrar y recibe como parámetro el título de la película. Con el parámetro creo una constante que haga referencia a él, después usando parentNode.removeChild elimino del padre esta constante. Así el de la lista de favoritos del html se cambia cada vez que se borra una. También elimino la película del array de favoritos y actualizo el firestore pasándole en la función actualizarValor() este array y sin la película.

La función marcarVisto() es invocada en el botón visto(el ojo) y recibe como parámetro el título de la película. Si el usuario ya tenía películas guardadas en en array en localStorage, se recogen para no perderlas. Se añade al array 'vistos' la película del parámetro y se guarda de nuevo en localStorage. También salta una SweetAlert que aparte de informar sobre qué película has marcado, te permite ver una lista con tus películas vistas.

La función detallesPeliculas() es invocada en el botón Info, recibe como parámetro el título de la película. Primero se comprueba si en el ya hay información de otra película, para en tal caso borrarla y que aparezca la de la nueva. Con la url de la api hago una llamada con fetch y después la convierto en un objeto, de ese objeto he añadido la información que me ha parecido relevante al html: título original, imagen cartel, fecha de lanzamiento, sinopsis, idioma original, popularidad y puntuación. simplemente vacía el que tiene la información utilizando parentNode.removeChild. El último <script> sirve para poder descargar este documento pdf al pulsar un botón.

PROBLEMAS Y SOLUCIONES

Aunque en general me ha parecido un trabajo un poco complicado en general, las partes que más me han costado han sido la paginación y Firebase, sobre todo esta última. La paginación no sabía como enfocarla hasta que descubrí que el propio objeto json de cada búsqueda tenía propiedades como 'total_pages' y total_results'. Con eso y descubriendo la función createElement, pude añadir una función que comprueba si hay más de una página con la propiedad antes mencionada y en tal caso crear los botones. A partir de ahí ya solo tuve que añadirles eventos que restasen o sumasen el número de página y llamasen a otra función.

Para Firebase en un principio solo sabía como crear un proyecto y conectarlo con mi programa, así que tuve que ver algún tutorial de como crear documentos y añadir o editar campos de estos desde el código. Crear documentos, que eran los nombres de

mis usuarios, fue fácil. Lo complicado fue que cada uno tuviera un campo con un array de sus películas favoritas, al principio porque guardaba todas en el mismo array y así cada usuario tenía las suyas más las de los anteriores. Me di cuenta de que podía crear un array nuevo para cada usurario que se registre y mandar ese array al firebase con setDoc y así funcionó. Después tuve el problema de que si un usuario ya se había registrado anteriormente apareciesen sus películas ya guardadas, asique investigué como recoger datos con getDoc a una referencia hacia ese usuario y que lo que devuelve es un objeto javascript, no json. Así pude hacer una función que se llama cada vez que se hace login y comprueba que el usuario esta en firebase y coge sus películas guardadas. Después de saber todo esto, borrarlas del firebase fue fácil ya que solo era actualizar el array y mandarlo con setDoc.

POSIBLES MEJORAS

En este trabajo creo que he abusado bastante de cambiar el html con innerHtml y en un futuro trabajo me gustaría que el html ya estuviese escrito y que con displays cambiase su visibilidad. El array de usuarios no lo utilizaría ya que solo necesito el último usuario. Por tiempo no añadí las películas vistas es Firebase, solo en localStorage. Y por último hubiese estado mucho más completa si tuviese la opción de buscar por series o actores.