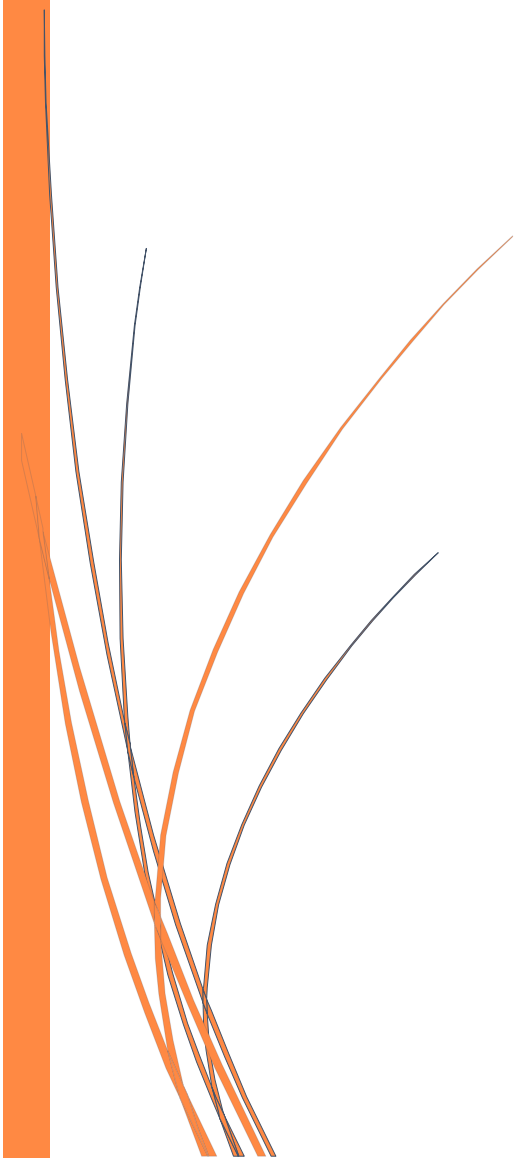
2023

Biblioteca

Documentación



Carlos Castaño && Angie Garcia

Tabla de contenido

Introducción

Bienvenidos a la documentación del proyecto de la aplicación web desarrollada por nuestro sello editorial de libros multinacional. Nuestro objetivo es ofrecer a nuestro público una forma interactiva de explorar nuestro catálogo de libros y, además, permitirles crear y gestionar su propia lista de lectura personalizada. Para ello hemos usado React y Redux como tecnologías principales.

A continuación, detallamos las consideraciones técnicas relacionadas con estas tecnologías:

- **React como Framework Frontend:** Hemos adoptado React como el framework principal para el desarrollo de la interfaz de usuario (UI). React es una biblioteca de JavaScript que nos permite construir componentes reutilizables y mantener una UI dinámica y eficiente. Aprovechamos las características clave de React, como el uso de componentes, el enlace de datos unidireccional y el enfoque basado en el árbol de componentes para garantizar un desarrollo organizado y escalable.
- **Redux para la Gestión del Estado Global:** Implementamos Redux para gestionar el estado global de la aplicación. Redux es una biblioteca de gestión de estado que nos proporciona un flujo de datos predecible y centralizado en nuestra aplicación.Utilizamos acciones y reductores para manejar los cambios en el estado global, permitiendo así una gestión clara y controlada de los datos en toda la aplicación.
- **Reutilización de Componentes:** Siguiendo las prácticas recomendadas de React, hemos creado componentes reutilizables para diversas partes de la interfaz. Esto nos ha permitido ahorrar tiempo y esfuerzo al desarrollar nuevas funcionalidades.Los componentes reutilizables también mejoran la legibilidad del código y facilitan el mantenimiento a medida que la aplicación crece y se expande.
- **Persistencia de Datos con Redux-Persist:** Para lograr la persistencia de la lista de lectura en el almacenamiento local del navegador, hemos utilizado redux-persist. Esta librería nos permite almacenar de manera segura y automática el estado de Redux en el almacenamiento del navegador, evitando que los datos se pierdan al recargar la página.

Estamos entusiasmados de presentar esta aplicación web, que brindará a nuestros usuarios una experiencia única e intuitiva para explorar y gestionar sus lecturas favoritas. Esperamos que esta documentación sea una guía completa para entender el desarrollo y funcionamiento de nuestra aplicación.

¡Disfruten explorando nuestro catálogo y creando su lista de lectura personalizada!

Carlos Castaño && Angie Garcia

Ficheros

Redux

Fichero store.js

Es el responsable de configurar el almacenamiento de Redux en la aplicación, incluyendo la persistencia de datos en el almacenamiento local para mantener el estado entre sesiones del navegador.

- **Configuración de Persistencia:** Creamos una configuración (persistConfig) para la persistencia del estado. Le asignamos una clave (key: 'root') para identificar el estado persistente en el almacenamiento local.
- **Reducer Persistente**: Utilizamos persistReducer para envolver nuestro reducer (reducer) con la configuración de persistencia (persistConfig). Esto asegura que el estado sea almacenado y recuperado del almacenamiento local.
- **Creación del Store:** Creamos el store de Redux utilizando el reducer persistente (persistedReducer) y aplicando el middleware thunk con applyMiddleware.
- **Persistor:** Creamos un persistor utilizando persistStore para mantener el estado persistente a través de las sesiones del navegador.

Fichero actions.js

Contiene las acciones que pueden ser disparadas por la interfaz de usuario o por otras partes de la aplicación, y son utilizadas para modificar el estado de la aplicación a través de los reducers. Cada acción tiene un tipo definido y opcionalmente un dato asociado (payload) que será utilizado para realizar los cambios en el estado.

- **getBooks():** Esta acción se encarga de obtener la lista de libros de la biblioteca y devuelve un objeto con el tipo de acción GET_BOOKS y la lista de libros (payload) obtenida del archivo JSON.
- **addToRead(book):** Esta acción se dispara cuando un libro es añadido a la lista de "Para leer". Recibe el objeto del libro (book) como parámetro y devuelve un objeto con el tipo de acción ADD_TOREAD y el libro (payload) que se agregará a la lista.
- **removeToRead(book):** Esta acción se dispara cuando un libro es eliminado de la lista de "Para leer". Recibe el objeto del libro (book) como parámetro y devuelve un objeto con el tipo de acción REMOVE_TOREAD y el libro (payload) que se eliminará de la lista.
- **filterBooks(genre):** Esta acción se utiliza para filtrar los libros por género. Recibe el género del libro (genre) como parámetro y devuelve un objeto con el tipo de acción FILTER y el género (payload) que se utilizará para filtrar los libros.
- **orderToRead(order):** Esta acción se dispara para ordenar la lista de libros "Para leer". Recibe el tipo de orden (order) como parámetro y devuelve un objeto con el tipo de acción ORDER y el tipo de ordenamiento (payload) que se aplicará a la lista.
- **filterPages(pages):** Esta acción se utiliza para filtrar los libros por cantidad de páginas. Recibe el número de páginas (pages) como parámetro y devuelve un objeto con el tipo de acción FILTER_PAGES y el número de páginas (payload) que se utilizará para filtrar los libros.

Fichero reducer.js

Contiene el reducer que se encarga de gestionar el estado de la aplicación y responder a las acciones disparadas en la interfaz. Cada caso en el switch maneja una acción específica y actualiza el estado en consecuencia para reflejar los cambios en la aplicación.

- **"GET_BOOKS":** Modifica el estado para obtener la lista de libros (allBooks y disponibilityBooks) y actualiza los contadores.
- **"ADD_TOREAD":** Modifica el estado para añadir un libro a la lista de "Para leer" (toRead) y ajusta los contadores.
- **"REMOVE_TOREAD":** Modifica el estado para eliminar un libro de la lista de "Para leer" (toRead) y ajusta los contadores.
- **"FILTER":** Modifica el estado para filtrar libros por género, actualizando la lista de libros disponibles (disponibilityBooks).
- **"FILTER_PAGES":** Modifica el estado para filtrar libros por cantidad de páginas (pages).

App.js

El componente raíz de la aplicación que utiliza React-Redux y Redux-Persist para conectar la aplicación al store de Redux y garantizar la persistencia del estado. El componente Home es renderizado dentro de la estructura proporcionada por los componentes Provider y PersistGate, permitiendo que la aplicación acceda y utilice el estado global almacenado en el store de Redux.

Pages

El componente representa la página de inicio de la aplicación. Utiliza React-Redux para obtener datos del estado global de Redux y carga la lista de libros disponibles al cargar el componente. El componente BooksList es renderizado y le pasamos los libros disponibles y los libros para leer como propiedades para que pueda mostrarlos en la interfaz de usuario.

- **Obtención de Datos del Estado:** Utilizamos useSelector para obtener datos del estado global de Redux. En este caso, obtenemos los libros disponibles (disponibilityBooks) y los libros para leer (toRead) desde el estado.
- **Dispatch de la Acción:** Utilizamos useDispatch para acceder a la función dispatch de Redux. Luego, dentro de useEffect, llamamos a la acción getBooks() para obtener la lista de libros y cargarla en el estado al cargar el componente. Dado que useEffect se ejecuta solo una vez (gracias al array vacío de dependencias), esta acción se dispara al inicio.
- **Renderizado del Componente BooksList:** Renderizamos el componente BooksList y le pasamos los libros disponibles (books) y los libros para leer (toRead) como propiedades.

Home

Representa la página de inicio de la aplicación. Utiliza React-Redux para obtener datos del estado global de Redux y carga la lista de libros disponibles al cargar el componente. El componente BooksList es renderizado

- **Obtención de Datos del Estado:** Utilizamos useSelector para obtener datos del estado global de Redux. En este caso, obtenemos los libros disponibles (disponibilityBooks) y los libros para leer (toRead) desde el estado.
- **Dispatch de la Acción:** Utilizamos useDispatch para acceder a la función dispatch de Redux. Luego, dentro de useEffect, llamamos a la acción getBooks() para obtener la lista de libros y cargarla en el estado al cargar el componente. Dado que useEffect se ejecuta solo una vez (gracias al array vacío de dependencias), esta acción se dispara al inicio.
- **Renderizado del Componente BooksList:** Renderizamos el componente BooksList y le pasamos los libros disponibles (books) y los libros para leer (toRead) como propiedades.

Componentes

BookDisponibility.js:

Muestra la lista de libros disponibles y permite filtrarlos por género y cantidad de páginas. Los libros disponibles se obtienen del estado global de Redux utilizando useSelector, y se utilizan en el componente para mostrarlos utilizando el componente Book. También se incluyen los componentes FilterGenre y FilterPages para permitir al usuario filtrar los libros según sus preferencias.

- **Función BooksDisponibility:** Definimos el componente BooksDisponibility que muestra los libros disponibles en la biblioteca.
- **Obtención de Datos del Estado:** Utilizamos useSelector para obtener datos del estado global de Redux. En este caso, obtenemos el contador de libros disponibles (counterDisp), la lista de libros para leer (toRead), y la cantidad de páginas utilizada para filtrar libros (pages).
- **Filtrado de Libros Disponibles:** Utilizamos el método filter() para obtener la lista de libros que están disponibles para ser leídos. Filtramos aquellos libros que no están en la lista de "Para leer" (toRead) y que cumplen con el criterio de cantidad de páginas (entre pages y 1500 páginas).
- **Renderizado de Libros Disponibles:** Utilizamos el método map() para recorrer la lista de libros disponibles (availableBooks) y renderizar cada libro utilizando el componente Book.

ToRead.js

muestra la lista de libros que están en la lista de "Para leer". Utiliza el estado global de Redux para obtener el contador de libros en lectura (counterToRead) y la lista de libros que están en la lista de "Para leer" (toRead). Los libros para leer se muestran utilizando el componente Book.

- **Obtención de Datos del Estado:** Utilizamos useSelector para obtener el contador de libros que están en la lista de "Para leer" (counterToRead) desde el estado global de Redux.

- Renderizado de Libros en "Para Leer": Utilizamos el método map() para recorrer la lista de libros que están en la lista de "Para leer" (toRead) y renderizar cada libro utilizando el componente Book.

Book.js

representa un libro individual en la lista de libros disponibles. Utiliza el estado global de Redux para gestionar la lista de libros que están en la lista de "Para leer" y “Disponibles” (toRead). El usuario puede añadir o eliminar u n libro de la lista de "Para leer" haciendo clic en el botón correspondiente.

- **Obtención de Datos del Estado:** Utilizamos useSelector para obtener la lista de libros que están en la lista de "Para leer" (toRead) desde el estado global de Redux.

- **Manejo de la Lista de "Para Leer":** La función handlerRead(book) se encarga de manejar la acción de añadir o eliminar un libro de la lista de "Para leer". Si el libro ya está en la lista, se dispara la acción removeToRead(book) para eliminarlo. Si no está en la lista, se dispara la acción addToRead(book) para añadirlo. n libro de la lista de "Para leer" haciendo clic en el botón correspondiente.

FilterGender.js

Permite filtrar los libros por género. El usuario puede seleccionar un género en el selector y se dispara la acción filterBooks para filtrar los libros por ese género específico. También se incluye una opción para mostrar todos los libros sin filtrar por género.

- **Lista de Géneros:** Definimos una lista de géneros disponibles (genres) que serán mostrados como opciones en el selector.
- **Utilización de useDispatch:** Utilizamos useDispatch para acceder a la función dispatch de Redux, que nos permite despachar acciones al store.
- **Función handlerFilter:** La función handlerFilter(e) se ejecuta cuando se selecciona una opción del selector. En este caso, se dispara la acción filterBooks(e.target.value), donde e.target.value representa el valor seleccionado (el género) que será utilizado para filtrar los libros.

FilterPages.js

Permite filtrar los libros por el número máximo de páginas. El usuario puede seleccionar el rango de páginas utilizando el slider, y se dispara la acción filterPages para filtrar los libros según el número máximo de páginas seleccionado. Además, se muestra el rango de páginas seleccionado al usuario mediante un texto.

- **Estado Local para el Rango de Páginas:** Utilizamos el estado local maxPages y el método setMaxPages del hook useState para almacenar el valor del rango de páginas seleccionado por el usuario.
- **Utilización de useDispatch:** Utilizamos useDispatch para acceder a la función dispatch de Redux, que nos permite despachar acciones al store.
- **Función handleMaxPagesChange:** La función handleMaxPagesChange(e) se ejecuta cuando el usuario cambia el valor del rango de páginas. Actualiza el estado local maxPages con el valor

seleccionado y dispara la acción filterPages(e.target.value), donde e.target.value representa el número máximo de páginas seleccionado para filtrar los libros.

Flujo de la Aplicación

El flujo de la aplicación web se describe a continuación, desde el punto de entrada hasta cómo los componentes interactúan y se comunican entre sí:

La aplicación web comienza en el archivo index.js, que actúa como el punto de entrada de la aplicación.

Configuración del Store de Redux

En index.js, se configura el store de Redux utilizando la función createStore() de la librería redux.

El store es creado a partir de los reductores combinados y se aplica el middleware thunk para admitir acciones asíncronas.

Persistencia del Store

El store configurado se pasa a la función persistStore() de redux-persist, lo que permite la persistencia de datos en el almacenamiento local del navegador.

Enrutamiento

La aplicación utiliza un sistema de enrutamiento para gestionar la navegación entre diferentes páginas y vistas.

El componente App.js se encarga de manejar el enrutamiento utilizando la librería react-router.

Home - Visualización de Libros Disponibles:

Al cargar la página de inicio (Home), se dispara una acción para obtener los libros disponibles desde la API o el almacenamiento local.

Los libros disponibles se almacenan en el estado global y se muestran en el componente BooksDisponibility.js.

Los usuarios pueden ver la lista de libros disponibles y tienen la opción de agregar un libro a su lista de lectura o quitarlo de ella.

Creación de Lista de Lectura

Cuando un usuario selecciona un libro en la lista de disponibles y hace clic en el botón "Agregar" (o "x" si el libro ya está en la lista de lectura), se dispara una acción para agregar o quitar el libro de la lista de lectura.

La actualización del estado global refleja la adición o eliminación del libro en la lista de lectura y en los libros disponibles.

Filtrado de Libros por Género

Los usuarios tienen la opción de filtrar la lista de libros disponibles por género utilizando el componente FilterGenre.js.

Al seleccionar un género, se dispara una acción para filtrar los libros disponibles según el género elegido.

Se muestra un contador actualizado con el número de libros disponibles, el número de libros en la lista de lectura y el número de libros disponibles en el género seleccionado.

Persistencia y Sincronización entre Pestañas

La persistencia del store a través de redux-persist asegura que la lista de lectura se mantenga incluso si el usuario recarga la página o cierra el navegador.

La sincronización del estado global entre pestañas se logra mediante la gestión centralizada de Redux. Cuando un usuario realiza cambios en una pestaña, los cambios se reflejan automáticamente en otras pestañas abiertas.

Visualización de Lista de Lectura

Los usuarios pueden ver su lista de lectura personalizada en la página BooksToRead.js.

Los libros en la lista de lectura se muestran en el componente BooksToRead.js, que también muestra el contador actualizado del número de libros en la lista de lectura.