Ejercicio 2 “To do List”, con React.

En cuanto al segundo ejercicio, presente mas dificultades. Logre entender varias cosas, pero hay algunas que todavía no comprendo. Sentí que React fue mucho más complejo. En comparación del anterior, en el cual amolde la información y documentación que encontraba a lo que yo necesitaba, en este ejercicio copie más código tal como estaba. Al igual que el ejercicio anterior, me pareció bastante interesante y por las tardes cuando tengo tiempo libre estoy siguiendo con el curso de React por internet, el cual utilice para armar la To do List.  
  
  
  
En cuanto al código. Comencé por un simple “html”, armando un “div” con un “h1” siendo el titulo de la lista de tareas, y luego con otro “div” 2 “h3” con el numero de tareas y las pendientes. Y continuamos con otro “div” con su respectivo “h3”, que tiene la función de botón, el cual agrega una tarea nueva. Y al final tenemos un “TodoList “ en la cual estarán todas las tareas.   
  
Pasamos a los componentes. El primero creado fue el “TodoAdd”, en el cual arme un formulario, con un input, aquí pondremos la tarea que queremos agregar. Y luegro agregamos un botón de tipo enviar que será lo que nos permitirá darle la acción.   
  
Segundo y tercer componente, creamos “TodoList”. En la cual tendrá la lista dinámica, en la cual utilice un “.map”, para recorrer la lista de tareas. Por lo tanto, creamos otro componente “TodoItem”, para desarrollar cada componente que estará en la barra de tareas. En este caso, viendo ejemplos vi uno que tenia las opciones de editar y eliminar una tarea, las cuales me parecían interesantes así que las agregué.

En el “TodoItem”, tendremos un “span” el cual tendrá la tarea y la opción de editarla, para ello crearemos otro componente llamado “TodoUpdate”.   
También tendrá los botones de editar y borrar, cada uno con su respectivo icono.  
  
Componente “TodoUpdate”. El cual es lo que nos permitirá editar la tarea.   
  
Pasamos a darle funciones a nuestra To do List.   
 Creamos el “todoreducer.js” Aquí con un switch, creamos los 4 casos de acciones que presentara nuestra To do List. Agregar, editar, si está completa, y si elimina la tarea.   
También agregamos al final un “return initialstate” si es que el “reducer” no encuentra alguna acción.

Para los casos creados, la explicación que encontré fue la siguiente.

“Add Todo”: Agrega una nueva tarea al estado actual. Utiliza el operador de propagación (...) para crear una nueva matriz que contiene todos los elementos del estado actual más el nuevo elemento que se encuentra en action.payload.

“Delete Todo”: Elimina una tarea del estado actual. Utiliza el método filter para crear una nueva matriz que excluye la tarea con el ID proporcionado en action.payload.

“Complete Todo”: Marca una tarea como completa o incompleta. Utiliza el método map para crear una nueva matriz, actualizando la propiedad done de la tarea con el ID proporcionado en action.payload.

“Update Todo”: Actualiza la descripción de una tarea. Utiliza el método map para crear una nueva matriz, actualizando la descripción de la tarea con el ID proporcionado en action.payload.  
  
Luego de esto crearemos nuestros customshooks.  
  
El primero será UseTodo.   
El cual importa funciones de “UseEffect” y “UseReducer” de react. Y también nuestro “todoreducer”.   
Definimos el Hook que se llamara “UseTodo”  
Utilizamos el hook useReducer para manejar el estado de “todos” utilizando el reductor “todoReducer”, el estado inicial “initialState” y la función de inicialización “init”.

Luego tenemos que “todos” es el estado actual y “dispatch” es la función que se utiliza para enviar acciones al reductor.  
  
Debajo de esto, tenemos los contadores que utilizaremos para el numero de tareas y las pendientes.   
  
Luego utilizamos el hook “useEffect” para realizar un efecto secundario. En este caso, actualiza “localStorage” cada vez que el estado todos cambia.  
  
Debajo tenemos las funciones que manejan nuestras acciones, para asi al final el retorno de las mismas. Las cuales estas son las que importaremos en nuestro “App.jsx” para aplicarlas en nuestra “Card-to-do“. Y en nuestra lista dinámica.  
  
Creacion de hook useForm  
Definimos al hook personalizado llamado “useForm”. Este toma un parámetro opcional “initialForm,” que es el estado inicial del formulario y se establece por defecto como un objeto vacío.   
Utilizamos el hook “useState” para crear el estado del formulario “formState” y la función “setFormState” que se utilizará para actualizar ese estado.  
  
Luego definimos una función llamada “onInputChange” que se utiliza para manejar cambios en los campos del formulario. Y también definimos “onResetForm” que restablece el formulario al estado inicial proporcionado como “initialForm” utilizando “setFormState”.

Y al final retornamos las funciones y estados para importarlas en los componentes “TodoAdd” y “TodoUpdate”.  
  
En todo add aplicamos lo siguiente.

Utilizamos el hook personalizado “useForm” para inicializar el estado del formulario. El estado “description” representa la descripción de la nueva tarea, y se proporcionan funciones "onInputChange” y “onResetForm” para manejar cambios y restablecer el formulario respectivamente.

Para manejar el formulario:  
Definimos una función llamada “onFormSubmit” que se llama cuando se envía el formulario de agregar.

Evitamos el comportamiento por defecto del formulario usando e.preventDefault().

Verificamos si la longitud de la descripción es mayor que 1 para evitar agregar tareas vacías o con descripciones muy cortas.

Creamos un nuevo objeto “newTodo” con un ID basado en la fecha actual, la descripción del formulario y marca la tarea como no completada.

Utilizamos a la función “handleNewTodo”

Utilizamos a la función “onResetForm” para restablecer el formulario después de agregar la tarea.  
  
Y por último renderizamos el formulario de adición.

Renderizamos un formulario que utiliza la función “onFormSubmit” que maneja el envio

Renderizamos un campo de entrada “<input>” que refleja el estado de “description” y permite al usuario ingresar la descripción de la nueva tarea.

Renderizamos un botón “<button>” que se utiliza para enviar el formulario y agregar la tarea.

En el todo update aplicamos lo siguiente:   
Utilizamos el hook personalizado “useForm” para inicializar el estado del formulario “updateDescription” con la descripción actual de la tarea.

Utilizamos el hook “useState” para inicializar el estado de “disabled” que determina si el campo de entrada está deshabilitado o no. Por defecto, se establece en true.

Utilizamos “useRef” para crear una referencia “focusInputRef” que se utilizará para enfocar el campo de entrada después de la actualización.  
  
Agregamos la función “On SubmitUpdate” la cual actualiza la tarea. Y hace lo siguiente.  
Define una función llamada “onSubmitUpdate“ que se llama cuando se envía el formulario de actualización.

Evitamos el comportamiento por defecto del formulario usando “e.preventDefault”.

Extraemos el ID de la tarea “todo.id” y la nueva descripción del formulario.

Y tomamos a la función “handleUpdateTodo” para actualizar la tarea.   
  
Y por último renderizamos el formulario de actualización.

Renderizamos un campo de entrada (<input>) que refleja el estado de “updateDescription” y permite al usuario actualizar la descripción de la tarea.

Aplicamos una clase condicional para aplicar un estilo de línea punteada si la tarea está marcada como completada.

Utilizamos la referencia “focusInputRef” para enfocar el campo de entrada después de la actualización.

Renderizamos un botón que utiliza el icono de edición “<FaEdit />” como su contenido.   
  
  
  
Y con esto finalizaría la “To do List”. Hay varios elementos y funciones que todavía no logro entender del todo bien. O mejor dicho, las entiendo, pero no logro aplicarlas bien. Seguramente con practica y seguir viendo tutoriales termino de entender. También, presente dificultades a la hora de importar y muchas nuevas funciones que no había visto nunca. Pero todo es parte del proceso.  
  
Les dejo la documentación, el curso que estoy viendo y los tutoriales que utilice.   
  
  
<https://devdocs.io/>

<https://www.youtube.com/watch?v=6Jfk8ic3KVk&t=25s>

<https://www.youtube.com/watch?v=7iobxzd_2wY&t=2715s>

<https://www.youtube.com/watch?v=9N7iuyYnqpg>

<https://www.youtube.com/watch?v=x-LcbVw99o8&list=PLUofhDIg_38q4D0xNWp7FEHOTcZhjWJ29&index=4>

<https://www.youtube.com/watch?v=XYpadB4VadY&list=PLUofhDIg_38q4D0xNWp7FEHOTcZhjWJ29&index=3>

<https://www.youtube.com/watch?v=8GFmoynezyA&t=96s>

<https://www.youtube.com/watch?v=LoYbN6qoQHA>