Sesión # 7 Componente Práctico

Desarrollo de Front-End web con React - Parte 2

Continuando con el desarrollo de nuestra app para crear listas de pendientes, sigue los siguientes pasos en el proyecto trabajado en la sesión 6.

1. Siguiendo con nuestro proyecto, ahora vamos a añadir un input de tipo checkbox en cada tarea de nuestra lista (TodoItem.jsx) para marcar su estado es decir, si ya fue completada o no.

```jsx
import React from "react";

export function TodoItem({ lista }) {
  const { id, task, completed } = lista;
  return (
    <li>
      <input type="checkbox">{task}</input>
    </li>
  );
}
```

2. Añade la propiedad toggleTask en el componente listado de tareas (TodList) que se llama desde App.jsx y crea una nueva función toggleTask que maneje este elemento (toggle/checkbox). Tu archivo App.jsx quedará así hasta el momento:

```jsx
import React, { Fragment, useState, useRef } from "react";

import { v4 as uuidv4 } from "uuid";

import { TodoList } from "./components/TodoList";



export function App() {

 const [listas, setListas] = useState([

   { id: 1, task: "Tarea 1", completed: false },

 ]);
```

```
 const taskRef = useRef();


 const toggleTask = (id) => {
   //copia de las tareas
   const newTasks = [...listas];
   //encontrar tarea seleccionada, según su id
   const task = newTasks.find((task) => task.id === id);
   task.completed = !task.completed; // si es true se convierte en
false, si es false se convierte en true
   setListas(newTasks); //actualizamos el listado de tareas
 };


 //Método para añadir tareas
 const handleTaskAdd = () => {
   const task = taskRef.current.value;
   if (task === "") return;
   setListas((prevTasks) => {
     return [...prevTasks, { id: uuidv4(), task, completed: false
}];
   });
   taskRef.current.value = null; //Limpia el input cuando se añade
 };


 return (
   <Fragment>
```

```
      <TodoList listas={listas} toggleTask={toggleTask} />

      <input ref={taskRef} type="text" placeholder="Nueva Tarea" />

      <button onClick={handleTaskAdd}>+</button>

      <button>-</button>

    </Fragment>

  );

}
```

3. ¿Recuerdas la propiedad toggleTask creada en el archivo App.jsx? Modifica ahora los archivos TodoList y TodoItem para añadir este nuevo parámetro

```
import React from "react";
import { TodoItem } from "./TodoItem";

export function TodoList({ listas, toggleTask }) {
  return (
    <ul>
      {listas.map((lista) => (
        <TodoItem key={lista.id} lista={lista} toggleTask={toggleTask} />
      ))}
    </ul>
  );
}
```

```
import React from "react";

export function TodoItem({ lista, toggleTask }) {
  const { id, task, completed } = lista;

  const handleTaskClick = () => {
    toggleTask(id);
  };

  return (
    <li>
      <input type="checkbox" checked={completed} onChange={handleTaskClick} />
      {task}
    </li>
  );
}
```

Como puedes notar se pasan las propiedades de padre a hijos (del componente de más arriba al más de abajo), además se evidencia la creación de una nueva función handleTaskClick en el archivo TodoItem.jsx la cual nos permitirá llamar la función toggleTask creada en App.jsx y hacer su llamado posteriormente en el método

onChange.

4. Ahora, nos hace falta crear una función para eliminar tareas de nuestro listado. Para ello en el archivo App.jsx crea una llamada handleClearAll y crea un evento en el botón de eliminar para hacer su llamado.

```
//Método para eliminar tareas
const handleClearAll = () => {
  //Hacemos una copia de las tareas creadas y
  // filtramos por aquellas que han sido seleccionadas
  const newTasks = listas.filter((task) => !task.completed);
  // Utilizamos setListas para setear los elementos,
  setListas(newTasks);
};
```

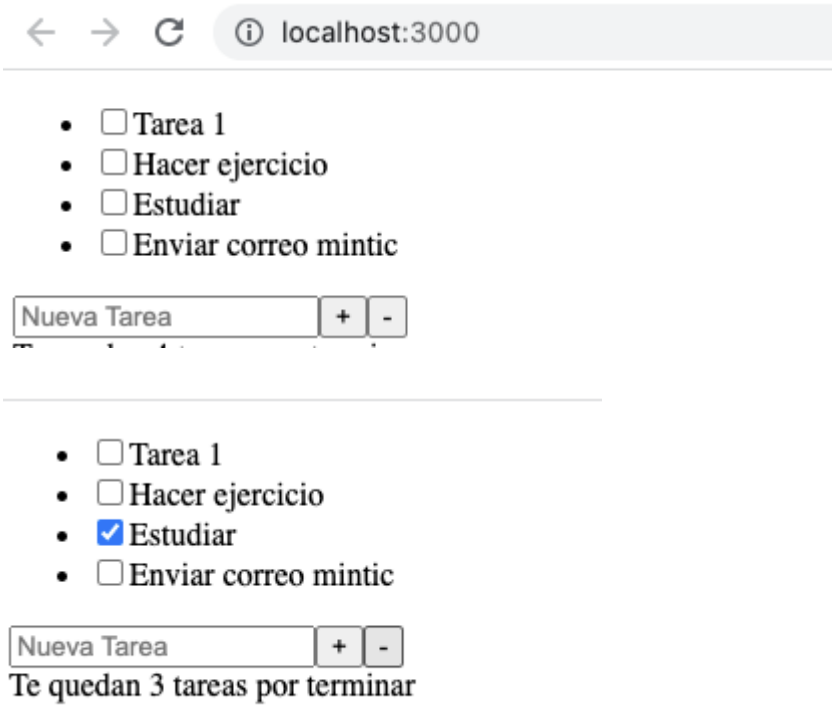
```
return (
  // Fragment se utiliza como padre para englobar
  <Fragment>
    <TodoList listas={listas} toggleTask={toggleT
    <input ref={taskRef} type="text" placeholder='
    <button onClick={handleTaskAdd}>+</button>
    <button onClick={handleClearAll}>-</button>
```

5. Como puedes ver, ya es posible remover tareas de nuestra lista. Pero podemos notar que al refrescar el sitio se borran las tareas existentes, para ello usemos un hook muy útil que nos permitirá almacenar los datos en nuestro local storage. Importa from react el hook `useEffect` y haz uso de él para almacenar en local las tareas creadas en la lista y para visualizar aquellas que ya se encuentran creadas al recargar el sitio.

```
//Para escuchar y guardar las nuevas tareas creadas
useEffect(() => {
  localStorage.setItem("listApp.lists", JSON.stringify(listas));
}, [listas]);

//Para visualizar aquellas tareas que ya se encuentren creadas
useEffect(() => {
  //Obtener tareas guardadas
  const storedTasks = JSON.parse(localStorage.getItem("listApp.lists"));
  //Validar que existan,
  if (storedTasks) {
    setListas(storedTasks);
  }
}, []);
```

6. Valida el resultado final



¡Listo, acabas de crear tu primera aplicación con react! Para ver la solución completa, revisa el siguiente repositorio: https://github.com/Misiontic-Ciclo-4A/sesion6-solucion

Ejercicio para practicar: tic-tac-toe con react