



TODOS

clase n° 17

*REC



recordá
poner a grabar la clase

creando una aplicación de “TODOs”


Introducción (5 minutos)

- Repasar conceptos clave de la clase anterior:
 - Qué son los estados.
 - Cómo se comparten estados entre componentes (props y lifting state up).
- Explicar el propósito del proyecto: "Vamos a aplicar lo aprendido para crear una aplicación sencilla de lista de tareas."

desarrollo paso a paso

Paso 1: Crear la estructura básica

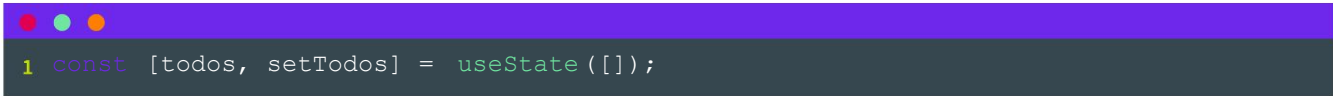
Estructura inicial de archivos:

A screenshot of a code editor window with a dark background and a purple title bar. The editor shows a file tree structure for a project. The root directory is 'src', which contains 'App.jsx' and a subdirectory 'Components'. The 'Components' directory contains three files: 'TodoForm.jsx', 'TodoList.jsx', and 'TodoItem.jsx'. The lines are numbered 1 through 6.

```
1 src/  
2   App.jsx  
3   Components/  
4     TodoForm.jsx  
5     TodoList.jsx  
6     TodoItem.jsx
```

Paso 2: Crear el estado y mostrar tareas

En App.jsx, añadir el estado inicial:

A screenshot of a code editor window with a dark background and a purple title bar. The editor shows a single line of code in App.jsx, which uses the useState hook from React to initialize a state variable named 'todos' with an empty array. The line is numbered 1.

```
1 const [todos, setTodos] = useState([]);
```

Crear el componente TodoList para mostrar tareas:

```
1 const TodoList = ({ todos }) => {  
2   return (  
3     <ul>  
4       {todos.map((todo, index) => (  
5         <TodoItem key={index} todo={todo} />  
6       ))}  
7     </ul>  
8   );  
9 };
```

Crear el componente TodoItem:

```
1 const TodoItem = ({ todo }) => {  
2   return <li>{todo.text}</li>;  
3 };
```

Integrar TodoList en App.jsx:

```
1 <TodoList todos={todos} />
```

Paso 3: Agregar nuevas tareas

Crear TodoForm con un input controlado:

```
1 const TodoForm = ({ onAddTodo }) => {
2   const [inputValue, setInputValue] = useState("");
3   const handleSubmit = (e) => {
4     e.preventDefault();
5     if (inputValue.trim()) {
6       onAddTodo(inputValue.trim());
7       setInputValue("");
8     }
9   };
10  return (
11    <form onSubmit={handleSubmit}>
12      <input
13        type="text"
14        value={inputValue}
15        onChange={(e) => setInputValue(e.target.value)}
16      />
17      <button type="submit">Agregar Tarea</button>
18    </form>
19  );
20};
```

En App.jsx, pasar la función para agregar tareas:

```
1 const addTodo = (text) => {  
2   setTodos([...todos, { text, completed: false }]);  
3 };  
4 <TodoForm onAddTodo={ addTodo } />
```

Paso 4: Marcar como completada

Modificar el estado en App.jsx para marcar tareas:

```
1 const toggleComplete = (index) => {  
2   setTodos(  
3     todos.map((todo, i) =>  
4       i === index ? { ...todo, completed: !todo.completed } : todo  
5     )  
6   );  
7 };
```

Añadir un botón en TodoItem:

```
1 const TodoItem = ({ todo, onToggle }) => {  
2   return (  
3     <li>  
4       <span  
5         style={{ textDecoration: todo.completed ? "line-through" : "none" }}  
6       >  
7         {todo.text}  
8       </span>  
9       <button onClick={onToggle}>  
10        {todo.completed ? "Desmarcar" : "Completar"}  
11      </button>  
12    </li>  
13  );  
14 };
```

Pasar la función toggleComplete desde App.jsx:

```
1 <TodoList  
2   todos={todos}  
3   onToggle={(index) => toggleComplete(index)}  
4 />
```


Paso 5: Eliminar tareas

Añadir una función para eliminar en App.jsx:

```
1 const deleteTodo = (index) => {  
2   setTodos(todos.filter((_, i) => i !== index));  
3 };
```

Añadir un botón de eliminar en TodoItem:

```
1 <button onClick={onDelete}>Eliminar</button>
```



¿tienen dudas?

*Es momento de preguntar
¡no tengan vergüenza!*



revolución*
digital_