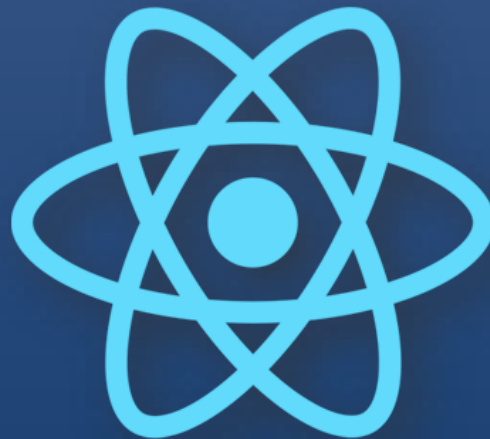


# **PROYECTO: AGENDA DE TAREAS**



## **CON NODEJS Y REACT**

**POR: NIKI VALENTINOV VELICHKOV**

I.E.S RÍO ARBA 2023-2024

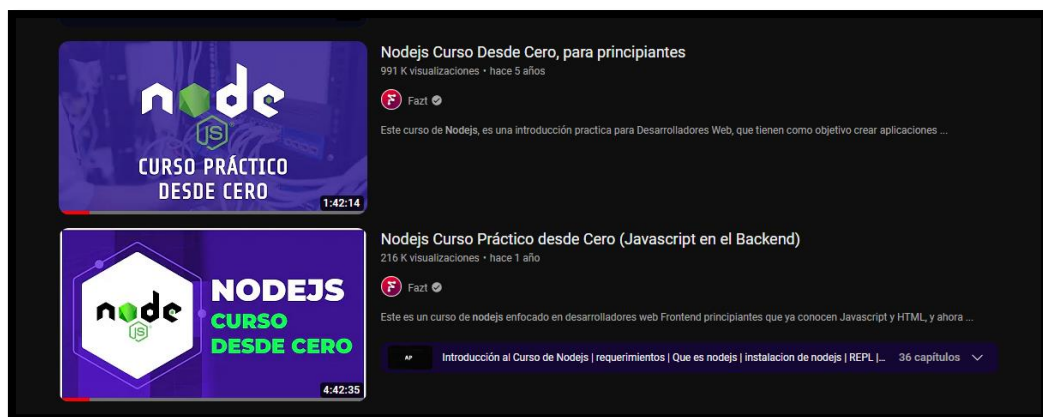
# ÍNDICE

<b>PRECEDENTES</b> .....	2
<b>CREACIÓN MY-APP</b> .....	3
<b>APP.CSS y CHECKBOX</b> .....	5
<b>TASK.JS</b> .....	7
<b>APP.JS</b> .....	9
<b>RESULTADO FINAL</b> .....	15

## PRECEDENTES

- Al inicio del proyecto de crear una aplicación de una agenda de tareas con React y Node.js, me propuse adquirir conocimientos para establecer una sólida base.

Comencé por explorar tutoriales en YouTube para familiarizarme con los conceptos básicos. Luego, decidí realizar un curso de JavaScript en Openwebinars Academy para profundizar en los fundamentos de la programación. Durante el curso, desarrollé varios códigos y completé un proyecto en JavaScript puro, con la intención de migrarlo posteriormente a React.



Una vez que adquirí una comprensión básica pero sólida de JavaScript y me sentí cómodo con los conceptos clave, me dispuse a comenzar con la aplicación.

Este enfoque paso a paso me permitió construir una base sólida de conocimientos antes de sumergirme en el desarrollo de la aplicación con React y Node.js que a continuación expondré.

# CREACIÓN MY-APP

- Tras iniciar mi proyecto desde comandos y estructurar un poco el html, me dispongo a crear el primer componente llamado **TaskForm**.

```
JS TaskForm.js X
src > JS TaskForm.js > TaskForm
1  import {useState} from "react";
2
3  export default function TaskForm({onAdd}) {
4    const [taskName, setTaskName] = useState('');
5    function handleSubmit(ev) {
6      ev.preventDefault();
7      onAdd(taskName);
8      setTaskName('');
9    }
10   return (
11     <form onSubmit={handleSubmit}>
12       <button>+</button>
13       <input type="text"
14         value={taskName}
15         onChange={ev => setTaskName(ev.target.value)}
16         placeholder="Tu nueva tarea..." />
17     </form>
18   );
19 }
```

El cual es responsable de manejar la entrada de nuevas tareas en la aplicación de lista de tareas:

- Utiliza el "hook" **useState** de las librerías de react para mantener el estado del nombre de la tarea.
- El componente ejecuta un formulario con un campo de entrada de texto y un botón.

### Manejo del formulario:

- Cuando se envía el formulario (al hacer clic en el botón o presionar “Enter”) se llama a la función **handleSubmit**.
- La función **onAdd** (creada como “prop” desde el componente padre) aparece con el nombre de la tarea.
- Finalmente, el campo de entrada se restablece a un valor vacío después de agregar la tarea.

# APP.CSS y CHECKBOX

- Posteriormente me decido por crear y personalizar el apartado de CSS.

En donde añado:

- Colores y fuentes.
- Estilos para el formulario.
- Estilos para las tareas individuales.
- Estilos para los encabezados y otros elementos como el **Checkbox** que a continuación me dispongo a crear.

```
JS Checkbox.js X
src > JS Checkbox.js > ...
1
2 export default function Checkbox({checked = false, onClick}) {
3   return (
4     <div onClick={onClick}>
5       {!checked && (
6         <div className="checkbox unchecked">
7           <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 448 512"><path d="M384 352
8             </div>
9         )}
10      {checked && (
11        <div className="checkbox checked">
12          <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 448 512"><path d="M64 352
13            </div>
14        )}
15      </div>
16    );
17  }
18
```

## Función de Checkbox:

- El componente Checkbox es una función de React que representa un checkbox interactivo.
- Se exporta como el componente predeterminado (export default).

## El componente acepta dos props:

- **checked:** Un valor booleano que indica si el checkbox está marcado o no (por defecto es false).

- **onClick:** Una función que se ejecutará cuando se haga clic en el checkbox.

#### **Renderizado del componente:**

- El componente renderiza un contenedor **<div>** que envuelve todo el contenido. Cuando se hace clic en este contenedor, se ejecuta la función **onClick**.

#### **Renderizado condicional:**

- El componente utiliza operadores ternarios para renderizar diferentes elementos según el valor de **checked**:
  - Si **checked** es **false**, se muestra un checkbox no marcado con un ícono de “visto” (checkmark).
  - Si **checked** es **true**, se muestra un checkbox marcado con un ícono de “visto” (checkmark) en un fondo diferente.

#### **Iconos SVG:**

- Los íconos de “visto” se representan utilizando elementos **<svg>** con atributos **xmlns** y **viewBox**.
- El primer ícono representa un checkbox no marcado.
- El segundo ícono representa un checkbox marcado.

# TASK.JS

- El segundo componente que creo es **Task.js**:

```
JS Task.js x
src > JS Task.js > Task
1 import Checkbox from "../Checkbox";
2 import {useState} from "react";
3
4 export default function Task({name,done,onToggle,onTrash,onRename}) {
5   const [editMode,setEditMode] = useState(false);
6   return (
7     <div className={'task ' + (done?'done':'')}>
8       <Checkbox checked={done} onClick={() => onToggle(!done)} />
9       <div className="task-name" onClick={() => setEditMode(prev => !prev)}>
10         <span>{name}</span>
11       </div>
12     )
13   )
14   {editMode && (
15     <form onSubmit={ev => {ev.preventDefault();setEditMode(false);}}>
16       <input type="text" value={name}
17         | | | | onChange={ev => onRename(ev.target.value)} />
18     </form>
19   )}
20   <button className="trash" onClick={onTrash}>
21     <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 448 512"><path d="M135.2
22   </button>
23 </div>
24 );
25 }
```

Este componente representa una tarea individual en la aplicación de la lista de tareas:

## El componente acepta varios props:

- **name:** El nombre de la tarea.
- **done:** Un valor booleano que indica si la tarea está marcada como completada.
- **onToggle:** Función que se ejecuta cuando se hace clic en el **checkbox** para cambiar el estado de la tarea.
- **onTrash:** Función que se ejecuta cuando se hace clic en el botón de eliminación.



- **onRename**: Una función que se ejecuta cuando se cambia el nombre de la tarea en el modo de edición.

#### **Renderizado condicional:**

- El componente utiliza operadores ternarios para alternar entre el modo de visualización normal y el modo de edición:
  - Si editMode es false, muestra el nombre de la tarea y permite cambiar al modo de edición al hacer clic.
  - Si editMode es true, muestra un campo de entrada de texto para editar el nombre de la tarea.

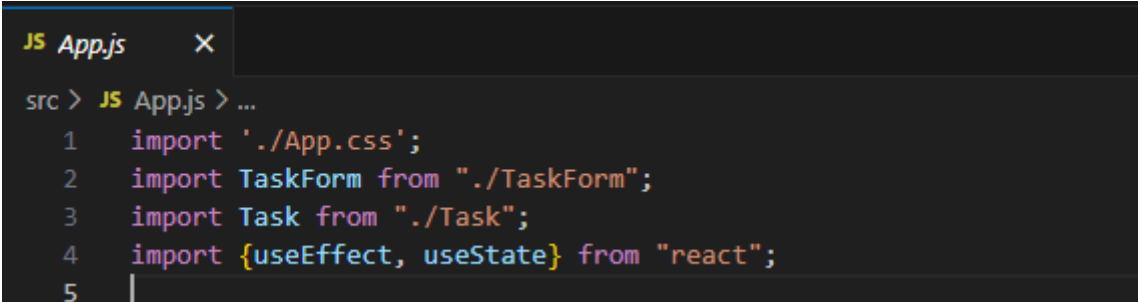
#### **Interacción con el usuario:**

- Task.js utiliza el componente **Checkbox** para representar el estado de tarea realizada.
- Al hacer clic en el nombre de la tarea, se activa el modo de edición.
- Al hacer clic en el botón de eliminación, se ejecuta la función **onTrash**.

# APP.JS

- Finalmente terminamos con el apartado de APP.JS:

## Importamos los módulos:

A screenshot of a code editor window titled 'JS App.js'. The editor shows the following code:

```
src > JS App.js > ...
1  import './App.css';
2  import TaskForm from './TaskForm';
3  import Task from './Task';
4  import {useEffect, useState} from "react";
5
```

- **import './App.css';** Importa un archivo CSS llamado App.css. Este archivo contiene estilos que se aplicarán a los componentes en la aplicación.
- **import TaskForm from './TaskForm';** Importa el componente TaskForm desde el archivo **TaskForm.js**.
- **import Task from './Task';** Importa el componente Task desde el archivo **Task.js**.
- **import {useEffect, useState} from "react";** Importa los hooks useEffect y useState desde la biblioteca de react.

### Creamos las funciones:

```
6  function App() {
7    const [tasks,setTasks] = useState([]);
8
9    useEffect(() => {
10     if (tasks.length === 0) return;
11     localStorage.setItem('tasks', JSON.stringify(tasks));
12   }, [tasks]);
13
14   useEffect(() => {
15     const tasks = JSON.parse(localStorage.getItem('tasks'));
16     setTasks(tasks || []);
17   }, []);
18
19   function addTask(name) {
20     setTasks(prev => {
21       return [...prev, {name:name,done:false}];
22     });
23   }
24
25   function removeTask(indexToRemove) {
26     setTasks(prev => {
27       return prev.filter((taskObject,index) => index !== indexToRemove);
28     });
29   }
30
31   function updateTaskDone(taskIndex, newDone) {
32     setTasks(prev => {
33       const newTasks = [...prev];
34       newTasks[taskIndex].done = newDone;
35       return newTasks;
36     });
37   }
```

### Función addTask:

- Agrega una nueva tarea al **array tasks**.
- Acepta un parámetro **name** que representa el nombre de la nueva tarea.
- Utiliza el **hook setTasks** para actualizar el estado de **tasks**.
- Crea un nuevo array copiando las tareas existentes y agregando una nueva tarea con el nombre proporcionado y el estado **done** establecido en **false**.

### Función removeTask:

- Elimina una tarea del **array tasks**.
- Acepta el parámetro **indexToRemove** que indica el índice de la tarea que se debe eliminar.
- Utiliza el método **filter** para crear un nuevo array que excluye la tarea en el índice especificado.

### Función updateTaskDone:

- Actualiza el estado de tarea completada.
- Acepta dos parámetros: **taskIndex** (el índice de la tarea) y **newDone** (el nuevo estado de tarea completada).
- Crea un nuevo **array** copiando las tareas existentes y actualizando el estado de la tarea en el índice especificado.

### Hooks useEffect:

- El primer **useEffect** guarda las tareas en el almacenamiento local.
- El segundo **useEffect** carga las tareas desde el almacenamiento local al estado de **tasks**.

### Declaramos las constantes y añadimos otras funciones:

```
39  const numberComplete = tasks.filter(t => t.done).length;
40  const numberTotal = tasks.length;
41
42  function getMessage() {
43    const percentage = numberComplete/numberTotal * 100;
44    if (percentage === 0) {
45      return '¡Alcanza tus objetivos!';
46    }
47    if (percentage === 100) {
48      return '¡Buen Trabajo!';
49    }
50    return '¡Añade tus tareas pendientes!';
51  }
52
53  function renameTask(index,newName) {
54    setTasks(prev => {
55      const newTasks = [...prev];
56      newTasks[index].name = newName;
57      return newTasks;
58    })
59  }
```

**const numberComplete = tasks.filter(t => t.done).length;**

- Filtra las tareas para obtener solo las que están marcadas como completadas.
- Luego, calcula la longitud (cantidad) de las tareas completadas.

**const numberTotal = tasks.length;**

- Obtiene la cantidad total de tareas en el **array tasks**.

**Función getMessage:**

- Calcula un mensaje basado en el porcentaje de tareas completadas:
  - Si el porcentaje es 0, devuelve "¡Alcanza tus objetivos!".
  - Si el porcentaje es 100, devuelve "¡Buen Trabajo!".
  - De lo contrario, devuelve "¡Añade tus tareas pendientes!".

### **Función renameTask:**

- Actualiza el nombre de una tarea en el **array tasks**.
- Acepta dos parámetros: **index** (el índice de la tarea) y **newName** (el nuevo nombre).
- Crea una copia del array tasks, actualiza el nombre de la tarea en el índice especificado y devuelve el nuevo array.

### **Renderizacion:**

```
return (  
  <main>  
    <h1>{numberComplete}/{numberTotal} Completadas</h1>  
    <h2>{getMessage()}</h2>  
    <TaskForm onAdd={addTask} />  
    {tasks.map((task,index) => (  
      <Task {...task}  
        onRename={newName => renameTask(index,newName)}  
        onTrash={() => removeTask(index)}  
        onToggle={done => updateTaskDone(index, done)} />  
    ))}  
  </main>  
}  
  
export default App;
```

### **Muestra las estadísticas de las tareas:**

**<h1>{numberComplete}/{numberTotal} Completadas</h1>:**

- Muestra el número de tareas completadas (**numberComplete**) y el total de tareas (**numberTotal**).

**<h2>{getMessage()}</h2>:**

- Muestra un mensaje basado en el porcentaje de tareas completadas utilizando la función **getMessage()**.

**Renderizado de componentes hijos:**

**<TaskForm onAdd={addTask} />:**

- Renderiza el componente **TaskForm** para agregar nuevas tareas.
- Proporciona la función **addTask** como prop llamada a **onAdd**.

**{tasks.map((task,index) => (<Task {...task} ... />))}:**

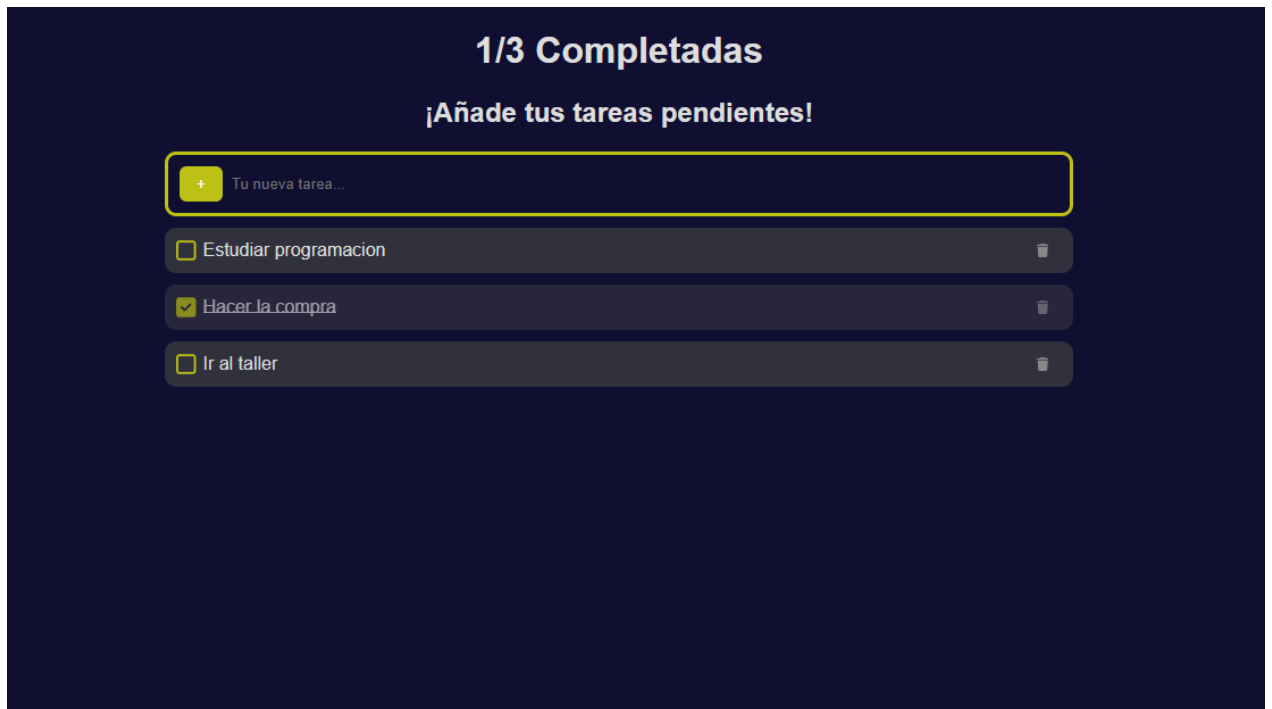
- Itera sobre el array de tareas (tasks) y renderiza un componente Task para cada tarea.
- Proporciona varias funciones (**onRename**, **onTrash**, **onToggle**) y los datos de la tarea como props.

**Componente Task:**

- El componente Task se renderiza para cada tarea en el array tasks.
- Contiene la lógica para mostrar y gestionar tareas individuales.

# RESULTADO FINAL

- Tras iniciar la aplicación este es el resultado:



Una aplicación web desarrollada con la finalidad de permitir la creación y eliminación de tareas pendientes y completadas. Estas tareas se almacenan localmente en el navegador, brindando una solución práctica para la gestión de una agenda de tareas de forma sencilla. Aunque su enfoque pueda parecer simple o básico, su funcionalidad sólida ha sido fundamental para mi aprendizaje en JavaScript y la exploración de frameworks como React.