Explicación del Cliente React (App.jsx) y su Interacción con el Servidor

🎯 Estructura del Cliente (Frontend)

1. **Importaciones y Estado Inicial**

```
import { useEffect, useState } from 'react'

function App() {
  const [tasks, setTasks] = useState([])
  const [newTask, setNewTask] = useState("")
  const [isLoading, setIsLoading] = useState(true)

  const API = "https://todo-app-full-stack-hd7e.onrender.com"
```

¿Qué hace?

- `useEffect`: Hook para ejecutar código cuando el componente se monta
- `useState`: Hook para manejar el estado de la aplicación
- `tasks`: Array que guarda todas las tareas obtenidas del servidor
- `newTask`: String que guarda el texto de la nueva tarea mientras se escribe
- `isLoading`: Boolean para mostrar indicador de carga
- `API`: URL del servidor donde están deployadas las rutas

2. **Carga Inicial de Tareas (useEffect)**

```
useEffect(() => {
  fetch(`${API}/tasks`)
    .then(res => res.json())
    .then(data => {
      setTasks(data)
      setIsLoading(false)
    })
    .catch(err => {
      console.error("Error cargando tareas:", err)
      setIsLoading(false)
    })
}, [])
```

¿Qué hace?

- Se ejecuta una sola vez cuando el componente se monta (array vacío `[]`)
- Hace una petición `GET` a `/tasks` del servidor
- Convierte la respuesta a JSON

- Actualiza el estado `tasks` con los datos recibidos
- Cambia `isLoading` a `false` para ocultar el indicador de carga

🔄 Interacción con el servidor:

```
React useEffect → GET /tasks → Servidor busca en MongoDB → Devuelve JSON → React actualiza estado
```

3. **Función para Agregar Tareas**

```
const handleAdd = async () => {
  if (newTask.trim() === "") return

  const res = await fetch(`${API}/tasks`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ text: newTask })
  })

  if (res.ok) {
    const data = await res.json()
    setTasks([...tasks, data])
    setNewTask("")
  }
}
```

¿Qué hace?

- Valida que el texto no esté vacío
- Hace una petición `POST` a `/tasks` con el texto de la nueva tarea
- Usa `JSON.stringify()` para convertir el objeto a JSON
- Si la respuesta es exitosa (`res.ok`), actualiza el estado local
- Limpia el campo de entrada (`setNewTask("")`)

🔄 Interacción con el servidor:

```
React handleAdd → POST /tasks + {text: "nueva tarea"} → Servidor crea en MongoDB → Devuelve tarea creada → React agrega al array local
```

4. **Función para Eliminar Tareas**

```
const handleDelete = async (id) => {
  const res = await fetch(`${API}/tasks/${id}`, {
    method: "DELETE"
  })
```

```
  if (res.ok) {
    setTasks(tasks.filter(task => task._id !== id))
  }
}
```

¿Qué hace?

- Recibe el `id` de la tarea a eliminar
- Hace una petición `DELETE` a `/tasks/:id`
- Si es exitosa, filtra la tarea del array local (sin recargar desde servidor)

🔄 Interacción con el servidor:

```
React handleDelete → DELETE /tasks/123abc → Servidor elimina de MongoDB → React filtra del array local
```

5. **Función para Manejar Enter**

```
const handleKeyPress = (e) => {
  if (e.key === 'Enter') {
    handleAdd()
  }
}
```

¿Qué hace?

- Escucha cuando se presiona una tecla en el input
- Si la tecla es "Enter", ejecuta la función `handleAdd()`
- Mejora la experiencia de usuario

6. **Renderizado del UI**

```
return (
  <div className="min-h-screen bg-gradient-to-br from-indigo-50 via-white to-cyan-50">
    {/* Header */}
    <div className="text-center mb-8">
      <h1 className="text-4xl font-bold text-gray-800 mb-2">
        📝 Mi Lista de Tareas
      </h1>
    </div>

    {/* Input Section */}
    <div className="bg-white rounded-xl shadow-lg p-6 mb-8">
      <input
```

```
          type="text"
          value={newTask}
          placeholder="¿Qué necesitas hacer hoy?"
          onChange={(e) => setNewTask(e.target.value)}
          onKeyPress={handleKeyPress}
        />
        <button onClick={handleAdd} disabled={!newTask.trim()}>
          ➕ Agregar
        </button>
      </div>

      {/* Tasks List */}
      <ul className="space-y-3">
        {tasks.map((task, index) => (
          <li key={task._id}>
            <span>{task.text}</span>
            <button onClick={() => handleDelete(task._id)}>
              🗑️ Eliminar
            </button>
          </li>
        ))}
      </ul>
    </div>
  )
```

¿Qué hace?

- Renderiza la interfaz de usuario con Tailwind CSS
- Input controlado: `value={newTask}` y `onChange`
- Lista dinámica: `tasks.map()` crea un elemento por cada tarea
- Eventos: `onClick`, `onKeyPress` para interacciones
- Estados condicionales: botón deshabilitado si no hay texto

🔄 Flujo Completo de Interacciones Cliente-Servidor

1. Carga Inicial de la App

```
sequenceDiagram
    participant R as React App
    participant S as Express Server
    participant M as MongoDB

    R->>S: GET /tasks
    S->>M: Task.find()
    M->>S: Array de tareas
    S->>R: JSON con tareas
    R->>R: setTasks(data)
    R->>R: setIsLoading(false)
```

2. Agregar Nueva Tarea

```
sequenceDiagram
    participant U as Usuario
    participant R as React App
    participant S as Express Server
    participant M as MongoDB

    U->>R: Escribe texto y presiona Enter
    R->>R: handleAdd()
    R->>S: POST /tasks + {text: "nueva tarea"}
    S->>M: new Task().save()
    M->>S: Tarea guardada con _id
    S->>R: JSON de tarea creada
    R->>R: setTasks([...tasks, newTask])
    R->>R: setNewTask("")
```

3. Eliminar Tarea

```
sequenceDiagram
    participant U as Usuario
    participant R as React App
    participant S as Express Server
    participant M as MongoDB

    U->>R: Click en botón eliminar
    R->>R: handleDelete(id)
    R->>S: DELETE /tasks/123abc
    S->>M: findByIdAndDelete(id)
    M->>S: Tarea eliminada
    S->>R: Status 204 (Sin contenido)
    R->>R: setTasks(tasks.filter())
```

📊 Estados de la Aplicación

Estado	Tipo	Propósito
`tasks`	Array	Almacena todas las tareas obtenidas del servidor
`newTask`	String	Texto temporal mientras se escribe nueva tarea
`isLoading`	Boolean	Controla el indicador de carga inicial

🎯 Patrón de Actualización de Estado

Optimistic Updates vs Server Sync

Para agregar tareas:

```
// ☑ Server-first: Espera confirmación del servidor
const data = await res.json()
setTasks([...tasks, data])  // Usa la tarea devuelta por el servidor
```

Para eliminar tareas:

```
// ☑ Optimistic: Actualiza UI inmediatamente
setTasks(tasks.filter(task => task._id !== id))  // No espera respuesta del
servidor
```

🛠 Tecnologías del Frontend

Tecnología	Propósito
React 18	Framework de UI
Hooks (useState, useEffect)	Manejo de estado y efectos
Fetch API	Peticiones HTTP al servidor
Tailwind CSS	Estilos y diseño
Vite	Herramienta de desarrollo

🔧 Métodos HTTP Utilizados

Método	Ruta	Propósito	Datos Enviados
`GET`	`/tasks`	Obtener todas las tareas	Ninguno
`POST`	`/tasks`	Crear nueva tarea	`{text: "texto"}`
`DELETE`	`/tasks/:id`	Eliminar tarea específica	Ninguno

📝 Características de UX Implementadas

1. **Loading State**: Indicador de carga mientras se obtienen las tareas
2. **Empty State**: Mensaje cuando no hay tareas
3. **Keyboard Shortcuts**: Enter para agregar tareas
4. **Visual Feedback**: Botones con estados hover y disabled
5. **Responsive Design**: Adaptable a móviles y desktop
6. **Error Handling**: Básico con console.error

🚀 Mejoras Posibles

Manejo de Errores

```
// Actual: Básico
.catch(err => console.error("Error:", err))

// Mejorado: UI feedback
.catch(err => {
  setError("No se pudo cargar las tareas")
  setIsLoading(false)
})
```

Optimistic Updates para Agregar

```
// Actual: Espera servidor
const data = await res.json()
setTasks([...tasks, data])

// Optimistic: Actualiza UI inmediatamente
const tempTask = { _id: Date.now(), text: newTask }
setTasks([...tasks, tempTask])
// Luego sincroniza con servidor
```

Estado de Completado

```
// Agregar campo completed a las tareas
const toggleComplete = async (id) => {
  // PATCH /tasks/:id para marcar como completada
}
```

📊 Flujo de Datos Resumido

```
Usuario Input → React State → HTTP Request → Express Route → MongoDB → HTTP
Response → React State → UI Update
```

Esta arquitectura sigue el patrón **Client-Server** con **REST API**, donde React maneja la interfaz y Express+MongoDB manejan los datos y la lógica del servidor.