**M3S23: React: Integración con Backend (CRUD)**

**Hola, te doy la bienvenida a la semana 23 de tu Bootcamp, en esta oportunidad estarás aprendiendo sobre React, integración con Backend (CRUD) y Frontend en la parte de comunicación de datos.**

**Te recuerdo que es super importante que te hayas alimentado, hidratado y que te encuentres super bien para que puedas aprender, asi que si te falta algo de lo anterior, te invito a realizarlo antes de iniciar con la lectura del contenido de esta semana.**

**Sigamos adelante...**

**ÍNDICE**

* [Frontend <> Backend](https://github.com/U-Camp/BOOT-M3-SEM23#frontend--backend)
  + [Backend: Encender nuestro servidor y esperar llamadas del cliente](https://github.com/U-Camp/BOOT-M3-SEM23" \l "backend-encender-nuestro-servidor-y-esperar-llamadas-del-cliente)
  + [Frontend: Comunicación de datos](https://github.com/U-Camp/BOOT-M3-SEM23" \l "frontend-comunicaci%C3%B3n-de-datos)

**Frontend <> Backend**

Hemos hecho un recorrido bastante largo. Es momento de conectar ambos procesos. Realizaremos una comunicación entre ellos a través de una API.

**Backend: Encender nuestro servidor y esperar llamadas del cliente**

Antes de comenzar, abramos nuestro Backend, el que trabajamos en la semana 21.

Vamos a revisar nuestro archivo .env:

.env

PORT=3005

MONGODB\_URI=mongodb://localhost:27017/ucamp-guitarras

SECRET=UCAMP

Haremos un cambio de puerto al 3005.

Finalmente, ejecutamos en terminal:

npm run start

Listo. Ahora bien. Vamos al cliente para conectarnos al API.

**Frontend: Comunicación de datos**

El objetivo en este punto es comunicarnos con nuestro Backend. Por lo tanto, es necesario implementar llamadas API desde React. Utilizaremos axios y nuestro primer paso será configurarlo.

axios.js

import axios from 'axios'

const clienteAxios = axios.create({

baseURL: process.env.REACT\_APP\_BACKEND\_URL

})

export default clienteAxios

Contemplar con este archivo que, a través de nuestras variables de entorno, nos conectaremos a nuestro servidor.

Hacemos un ajuste en nuestro archivo .env dentro de React:

.env

REACT\_APP\_BACKEND\_URL=http://localhost:3005

Perfecto. Vamos a establecer nuestro estado.

Vamos a GuitarState.js y comenzaremos el establecimiento de nuestras funciones que permitirán, a través del dispatch, modificar el estado global.

import React, { useReducer } from 'react'

import GuitarContext from './GuitarContext'

import guitarReducer from './GuitarReducer'

import axiosClient from './../../config/axios'

const GuitarState = (props) => {

// 1. ESTADO INICIAL

const initialState = {

guitars: []

}

// 2. DISPATCHING Y REDUCERS

const [globalState, dispatch] = useReducer(guitarReducer, initialState)

// 3. FUNCIONES

// NOS VAN A AYUDAR A CAPTURAR LOS EVENTOS DE LOS COMPONENTES

// A. CREAR GUITARRA

// CAPTURAREMOS A TRAVÉS DE ESTA FUNCIÓN LOS DATOS DE UN FORMULARIO QUE NOS PERMITIRÁN CREAR UNA GUITARRA

const createGuitar = async (dataForm) => {

const form = {

nombre: dataForm.nombre,

precio: dataForm.precio

}

try {

await axiosClient.post(`/crear-guitarra`, form)

getGuitars()

} catch (error) {

console.log(error)

}

}

// B. LEER GUITARRAS

// AL EJECUTAR ESTA FUNCIÓN, NOS PERMITE ACTUALIZAR NUESTRO ESTADO GLOBAL, A PARTIR DE UNA LLAMADA ASÍNCRONA "GET" HACIA EL SERVIDOR

const getGuitars = async () => {

try {

const res = await axiosClient.get(`/obtener-guitarras`)

dispatch({

type: "OBTENER\_GUITARRAS",

payload: res.data.guitarras

})

} catch (error) {

}

}

// C. ACTUALIZAR GUITARRA

// ACTUALIZA EL DATO DE UNA GUITARRA A PARTIR DE UN FORMULARIO HACIA NUESTRA BASE DE DATOS.

const updateGuitar = async (id, dataForm) => {

const form = {

id,

nombre: dataForm.nombre,

precio: dataForm.precio

}

try {

await axiosClient.put(`/actualizar-guitarra`, form)

getGuitars()

} catch (error) {

console.log(error)

}

}

// D. BORRAR GUITARRA

// BORRA UNA GUITARRA HACIA LA BASE DE DATOS, A TRAVÉS DE UNA RUTA CON EL MÉTODO DELETE

const deleteGuitar = async (id) => {

const data = { id }

try {

await axiosClient.delete(`/borrar-guitarra`, { data })

getGuitars()

} catch (error) {

console.log(error)

}

}

// 4. RETORNO DE ESTADO GLOBAL

return (

<GuitarContext.Provider

value={{

guitars: globalState.guitars,

createGuitar,

getGuitars,

updateGuitar,

deleteGuitar

}}

>

{props.children}

</GuitarContext.Provider>

)

}

export default GuitarState

Trabajamos nuestro GuitarReducer.js, con la intención de cambiar el estado global, tal cual lo hemos aprendido anteriormente.

// REDUCER - ES UNA FUNCIÓN AUTORIZADA QUE ALTERA EL ESTADO GLOBAL

const reducer = (globalState, action) => {

switch(action.type) {

case "OBTENER\_GUITARRAS":

return {

...globalState,

proyectos: action.payload

}

default:

return globalState

}

}

export default reducer

Confirmamos que tenemos nuestro GuitarContext.js correctamente establecido:

import { createContext } from 'react'

const GuitarContext = createContext()

export default GuitarContext

Y finalmente, la parte más relevante, construir la interfaz final. Entramos a nuestro componente GuitarList.js

// IMPORTAMOS TRES HOOKS. USECONTEXT, USEEFFECT, USESTATE

import React, { useContext, useEffect, useState } from 'react'

// REALIZAMOS LA IMPORTACIÓN DEL CONTEXTO

import GuitarContext from '../../context/guitars/GuitarContext'

export default function GuitarList() {

// A. ESTADOS LOCALES

// NUESTRO ESTADO LOCAL PRINCIPAL.

const [guitar, setGuitar] = useState({

nombre: "",

precio: ""

})

// NUESTRO MODO EDICIÓN. CUANDO EL USUARIO ESTÉ EDITANDO, ESTE LO DEBEREMOS CAMBIAR A TRUE

const [editMode, setEditMode] = useState(false)

// NUESTRO ID, EL CUAL LO UTILIZAREMOS EN LA EDICIÓN PARA LA GUITARRA SELECCIONADA

const [id, setId] = useState(null)

const [error, setError] = useState(null)

// B. ESTADO GLOBAL

// CONTEXTO

const ctx = useContext(GuitarContext)

// DATOS VÍA DESESTRUCTURACIÓN DE OBJETOS, CON EL CONTEXTO

const {

guitars,

createGuitar,

getGuitars,

updateGuitar,

deleteGuitar

} = ctx

// C. MANEJO ASÍNCRONO DE OBTENCIÓN DE DATOS

useEffect(() => {

getGuitars()

}, [])

// D. MANEJO DE CAMBIOS EN CADA CAMPO DE LOS FORMULARIOS

const handleChange = (event) => {

// CADA VEZ QUE EL USUARIO CAMBIE UN DATO DEL CAMPO DE TEXTO, DEBEREMOS CAMBIAR EL VALOR DEL ESTADO LOCAL.

setGuitar({

...guitar,

[event.target.name]: event.target.value

})

}

// E. ENVÍO DE EVENTOS A NUESTRO ESTADO GLOBAL, LOS CUALES SE EJECUTAN CUANDO EL USUARIO PRESIONA UN BOTÓN ESPECÍFICO

const sendDataToCreateGuitar = (event) => {

// DETIENE LA RECARGA AUTOMÁTICA DE LA PÁGINA

event.preventDefault()

// VALIDACIÓN DE FORMULARIO. NO PUEDE QUEDAR VACÍO

if (!guitar.nombre.trim() || !guitar.precio.trim()) {

// SI HUBO ERROR, GENERAR UN MENSAJE AL USUARIO

return setError("Debes llenar todos los campos de texto")

}

// CREAMOS GUITARRA HACIA EL SERVIDOR

createGuitar(guitar)

// SI HUBO UN ERROR, BORRARLO NUEVAMENTE

setError(null)

}

const sendDataToUpdateGuitar = (event) => {

// DETIENE LA RECARGA AUTOMÁTICA DE LA PÁGINA

event.preventDefault()

// VALIDACIÓN DE FORMULARIO

if (!guitar.nombre.trim() || !guitar.precio.trim()) {

return setError("Debes llenar todos los campos de texto")

}

// ACTUALIZAMOS GUITARRA AL SERVIDOR

updateGuitar(id, guitar)

// ID A NULO

setId(null)

// MANEJO DE ESTADO LOCAL A VACÍO

setGuitar({

nombre: "",

precio: ""

})

// CAMBIAMOS EL MODO EDICIÓN A FALSO

setEditMode(false)

// SI HUBO UN ERROR, BORRAR EL MENSAJE NUEVAMENTE

setError(null)

}

const sendDataToDeleteGuitar = (element) => {

// ELIMINA GUITARRA HACIA EL SERVIDOR

deleteGuitar(element.\_id)

}

const activateEditMode = (element) => {

// ACTIVAMOS MODO EDICIÓN

setEditMode(true)

// ESTABLECEMOS EL ID DEL ELEMENTO SELECCIONADO

setId(element.\_id)

// ESTABLECEMOS EL VALOR HACIA EL ESTADO LOCAL

setGuitar({

nombre: element.nombre,

precio: element.precio

})

}

return (

<div>

{/\* TÍTULO (DEPENDIENDO DEL MODO EN EL QUE ESTEMOS) \*/}

<h1>

{

editMode ?

"Edita guitarra"

:

"Crea una guitarra"

}

</h1>

{/\* NUESTRO FORMULARIO \*/}

<form

onSubmit={editMode ?

e => { sendDataToUpdateGuitar(e) }

: (e) => { sendDataToCreateGuitar(e) }}

>

<h2>Escribe el nombre de la guitarra</h2>

<input

name="nombre"

onChange={(e) => { handleChange(e) }}

value={guitar.nombre}

/>

<h2>Escribe el precio</h2>

<input

name="precio"

onChange={(e) => { handleChange(e) }}

value={guitar.precio}

type="number"

/>

<button

type="submit"

>

{editMode ? "Editar guitarra" : "Crear guitarra"}

</button>

</form>

{error ? error : null}

<h1>Lista de Guitarras</h1>

{/\* DATOS \*/}

{

guitars.map((elem) => {

return (

<div key={elem.\_id}>

<h2>{elem.nombre}</h2>

<p>Precio: ${elem.precio}</p>

<button onClick={(evt) => activateEditMode(elem)}>

Editar

</button>

<button onClick={(evt) => sendDataToDeleteGuitar(elem)}>

Borrar

</button>

</div>

)

})

}

</div>

)

}

Este será tu resultado:

Cada cambio se conecta al Backend y realiza las modificaciones en la base de datos.

Perfecto. Ve leyendo cada comentario en las líenas de código y ve ejecutando el algoritmo correspondiente.

En este punto, hemos realizado un **Fullstack React**. Todos los conceptos han sido aprendidos en estas últimas semanas.

Toma tu tiempo para revisar y anclar cada concepto.

**¿Cómo te sientes?, espero que estés aprendiendo muchísimo, recuerda que estamos siempre allí para ayudarte cuando lo necesites.**

**¡Nos vemos pronto!**