ACADEMY by NUMEN

FULL STACK

Comenzamos en unos minutos


ACADEMY
by NUMEN

Peticiones en React

ACADEMY by NUMEN

Peticiones con Fetch()

ACADEMY by NUMEN

¿Qué es fetch?

🔥 Es una interfaz JavaScript para acceder y manipular peticiones y respuestas por medio del protocolo HTTP.. También provee un método global que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.

Este tipo de funcionalidad se conseguía previamente haciendo uso de XMLHttpRequest. Fetch proporciona una alternativa mejor que puede ser empleada fácilmente por otras tecnologías modernas.

Consumiendo datos con Fetch

🔥 Para aprender a usar fetch crearemos una app clásica muy popular entre los estudiantes iniciados en React que consiste en consumir información de la API de Breaking Bad para mostrarla en forma de citas de los personajes.

Creando el JSX

Como primer paso empezaremos creando el JSX de nuestra app que consistirá en una imagen principal y un botón para mostrar una cita aleatoria cada vez que se lo presione.

```
function App() {

  return (
    <div style={{display: "flex", flexDirection: "column", width: 300}}>
      <img
           src="https://logospng.org/download/breaking-bad/logo-breaking-bad-2048.png"
           alt="breaking-bad" width="300"
      />
      <button>Obtener otra cita</button>
    </div>
  );
}

export default App;
```

Añadiento un estado por medio de useState

El siguiente paso es crear una función useState que nos permitirá establecer una cita inicial y la posibilidad de actualizarla por otra cita nueva.

```
import { useState} from "react";

const initialQuote = {
  text: 'Cita',
  author: 'Autor'
}

function App() {
          const [quote, setQuote] = useState(initialQuote)
  return (
          // JSX
  );
}
export default App;
```

Creando el componente de Citas

Ahora crearemos el componente donde se mostrará la información una vez consumida. Este componente luego lo importaremos en nuestro componente principal para pasarle props respectivas.

```
const Quote = () => {
  return (
    <p>
        Texto de la cita <br />
        <span>- Nombre del autor</span>
    </p>
  )
}

export default Quote
```

```
const updateQuote = async () => {
  setLoading(true)
  const url = "https://www.breakingbadapi.com/api/quote/random";
  const res = await fetch(url);
  const [newQuote] = await res.json();

  const {quote: text, author} = newQuote;

  setQuote({
    text,
    author
  })

  setLoading(false)
}

useEffect(() => {
  updateQuote()
}, [])
```

Llegó la hora de realizar nuestra petición con Fetch

Pasar nuestra función al evento y pasar las props

```
import { useState, useEffect } from "react";
import Quote from "./components/Quote";

function App() {
  //useState
  // Función de petición
  // useEffect
  return (
    <div style={{display: "flex", flexDirection: "column", width: 300}}>
      <img src="https://logospng.org/download/breaking-bad/logo-breaking-bad-2048.png"
           alt="breaking-bad" width="300"/>
      <button onClick={() => updateQuote()}>Get Another</button>
      <Quote quote={quote}/>
    </div>
  );
}

export default App;
```

Extrayendo las props en el componente de citas

```
const Quote = ({quote}) => {
  return (
    <p>
        {quote.text} <br />
        <span>- {quote.author}</span>
    </p>
  )
}

export default Quote
```

Mensaje de Loading...

```
const [quote, setQuote] = useState(initialQuote)
const [loading, setLoading] = useState(false)

const updateQuote = async () => {
  setLoading(true)
  const url = "https://www.breakingbadapi.com/api/quote/random";
  const res = await fetch(url);
  const [newQuote] = await res.json();

  const {quote: text, author} = newQuote;

  setQuote({
    text,
    author
  })

  setLoading(false)
}
```

Nuestro último paso es agregar una funcionalidad que nos permita mostrar un mensaje de carga (loading) a modo decorativo que indique al usuario que el servidor está esperando respuesta.

Para ello crearemos un nuevo estado y colocaremos su función actualizadora dentro de nuestra función de petición.

Programar la visibilidad

Finalmente solo tenemos que programar para que se alterne la visibilidad de nuestro mensaje de loading con nuestro componente de citas.

```
function App() {

  //useState
  // Función de petición
  // useEffect
   return (
    <div style={{display: "flex", flexDirection: "column", width: 300}}>
      <img src="https://logospng.org/download/breaking-bad/logo-breaking-bad-2048.png"
           alt="breaking-bad" width="300"/>
      <button onClick={() => updateQuote()}>Get Another</button>
      {loading ? <h2>Loading...</h2> : <Quote quote={quote}/>}
    </div>
  );
}
export default App;
```

Peticiones con Axios

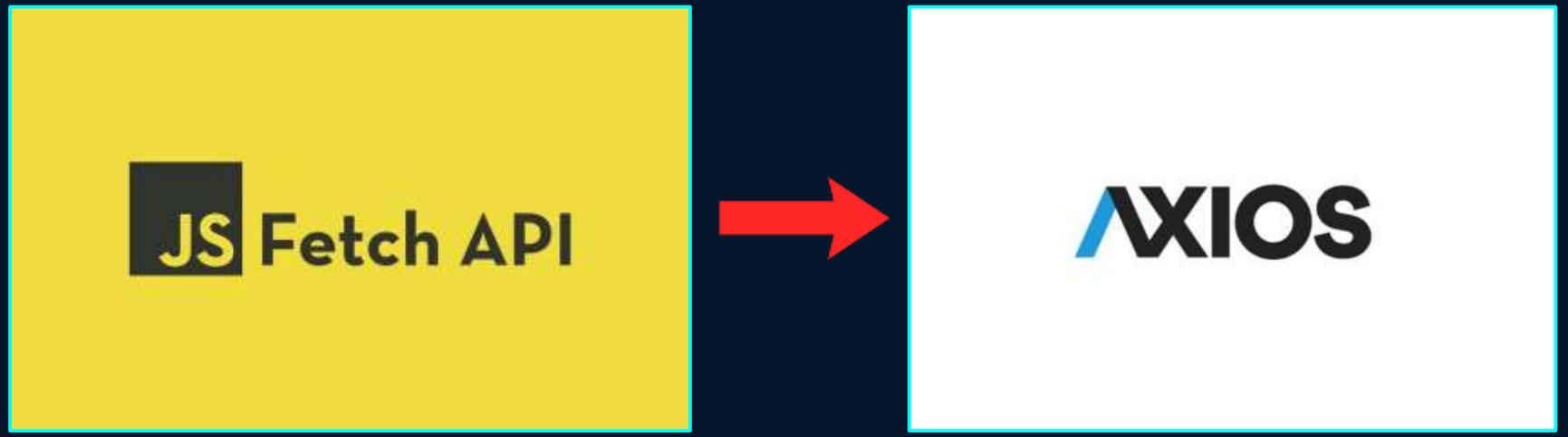
ACADEMY by NUMEN

¿Qué es Axios?

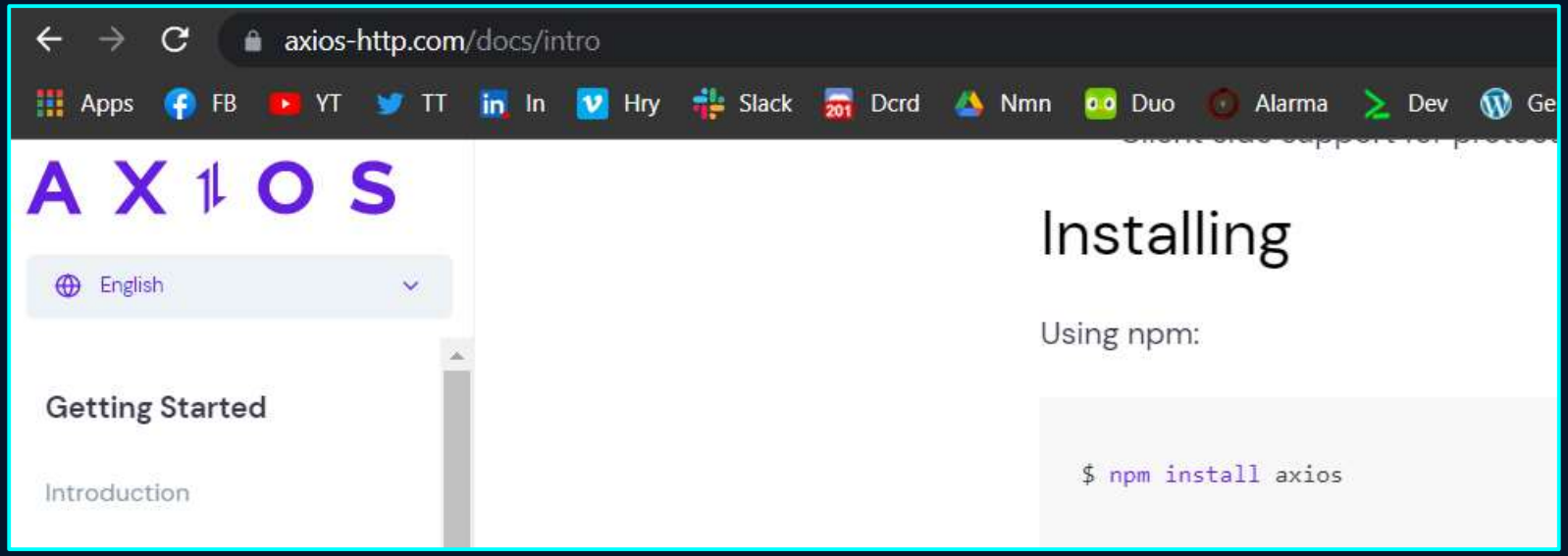
🔥 Axios es un cliente HTTP simple basado en promesas para el navegador y node.js. Axios proporciona una biblioteca fácil de usar en un paquete pequeño con una interfaz muy extensible.

Pasando nuestra APP a Axios

Pasar una aplicación desde **fetch** a **axios** es una tarea bastante sencilla si se conoce el funcionamiento de esta librería.
Lo que haremos será reemplazar las líneas de código donde involucramos fetch y hacer los cambios pertinentes para que corra con Axios.



Instalando Axios




```
PS C:\Users\NicoD\OneDrive\Escritorio\jsx> npm install axios
```

```
import axios from "axios";
```

Reemplazando fetch por axios

```
const updateQuote = async () => {
  setLoading(true)
  const url = "https://www.breakingbadapi.com/api/quote/random";
  // const res = await fetch(url);
  const res = await axios.get(url);
  // const [newQuote] = await res.json();
  const [newQuote] = await res.data;

  const {quote: text, author} = newQuote;

  setQuote({
    text,
    author
  })

  setLoading(false)
}
```

Para poder acceder al endpoint, Axios requiere de especificar el verbo a utilizar (en este caso get) y a dicho verbo se le agrega como parámetro el endpoint correspondiente.

A diferencia de **fetch**, **axios** no requiere de aplicar el método **json()** para acceder poder leer la información como objeto, sino que puede acceder a la data de manera directa.
Y eso es todo 😉

Agregando JSON-Server a nuestro carrito

ACADEMY by NUMEN

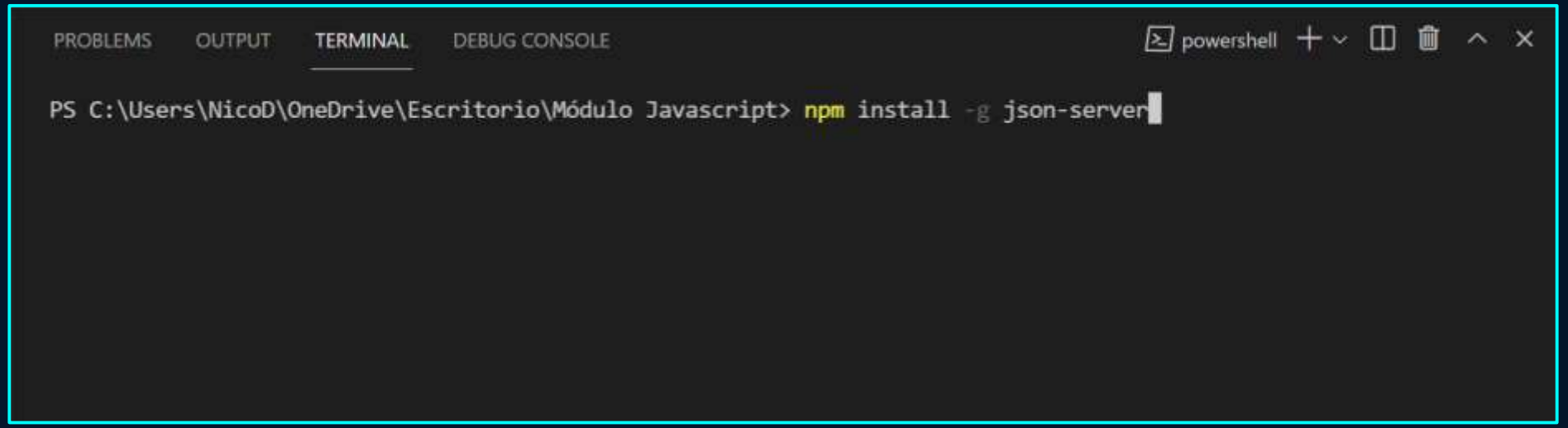
Ahora que ya aprendimos como hacer peticiones podríamos volver a nuestro carrito de compras e intentar levantar un JSON-Server y consumir desde ahí nuestros productos.

Para esto directamente usaremos axios que es el método que realmente nos interesa.

Entonces, recordemos cómo se levantaba un json-server!!

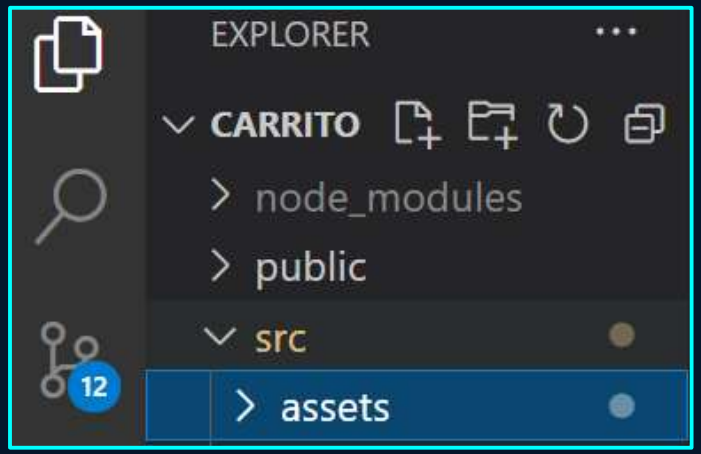
Instalando JSON Server

Teniendo instalado Node, ejecutamos el siguiente comando para instalar JSON Server en nuestro proyecto.



Conectando nuestra base de datos con JSON Server

Paso 1: Creamos una carpeta llamada assets



Paso 2: Dentro de assets creamos un archivo json llamado db.json con la información que deseamos manipular.

Paso 3: Abrimos la terminal y escribimos el siguiente comando

```
PS C:\Users\NicoD\OneDrive\Escritorio\jsx> json-server --watch src/assets/db.json
```

```json
{
  "products": [
    {
      "id": 1,
      "name": "Producto A",
      "price": 10
    },
    {
      "id": 2,
      "name": "Producto B",
      "price": 50
    },
    {
      "id": 3,
      "name": "Producto C",
      "price": 100
    },
    {
      "id": 4,
      "name": "Producto D",
      "price": 150
    },
    {
      "id": 5,
      "name": "Producto E",
      "price": 200
    }
  ],
  "cart": []
}
```

Rellenar nuestro db.json

Ahora debemos tomar todo el estado inicial de nuestro carrito y trasladarlo a nuestra hoja de **json** para que podamos levantar el servidor con dicha información.

De este modo podremos consumirla desde ahí para en un futuro retener los cambios que se realicen en el carrito de compras en lugar de que se pierdan al recargar el sitio.

Limpiando nuestro initialState

¡¡Ahora vamos al código!!

Lo primero que haremos será vaciar nuestro initialState, dado que la información ahora vendrá desde json-server.

Así es que, dejaremos los valores internos de nuestras propiedades como simples arreglos vacíos.

```
export const shoppingInitialState = {
    products: [],
    cart: []
};
```

Agregando una nueva acción

Ahora agregaremos una acción más a nuestro objeto TYPES la cual usaremos para renderizar la información consumida al momento de que se recargue la aplicación o el sitio.

```
export const TYPES = {
    READ_STATE: "READ_STATE",
    ADD_TO_CART: "ADD_TO_CART",
    REMOVE_ONE_PRODUCT: "REMOVE_ONE_PRODUCT",
    REMOVE_ALL_PRODUCT: "REMOVE_ALL_PRODUCT",
    CLEAR_CART: "CLEAR_CART"
}
```

Programando la función de petición

```
npm install axios
```

```
import axios from "axios";
```

```
const updateState = async () => {
    const productsURL = "http://localhost:3000/products";
    const cartURL = "http://localhost:3000/cart";
    const resProducts = await axios.get(productsURL);
    const resCart = await axios.get(cartURL);
    const newProduct = await resProducts.data
    const newCartItem = await resCart.data

    dispatch({type: TYPES.READ_STATE, payload: [newProduct, newCartItem]})
}

useEffect(() => {
    updateState()
}, [])
```

Una vez instalado **axios** debemos crear una petición **get** para cada endpoint.

Una vez hecho esto extraemos la información y la pasamos al objeto **action** por medio del payload para poder acceder desde la función reductora.

Programando la nueva acción en la función reductora

```
export function shoppingReducer(state, action) {
    switch (action.type) {
        case TYPES.READ_STATE: {
            return {
                ...state,
                products: action.payload[0],
                cart: action.payload[1]
            }
        }
```

Finalmente, simplemente le indicamos en el caso respectivo que deseamos que, al cargar el sitio se reemplacen los arreglos vacíos del estado inicial original por los datos extraídos desde **json-server.**

Y eso es todo. Ya tenemos el carrito corriendo desde el servidor.

ACADEMY
by NUMEN