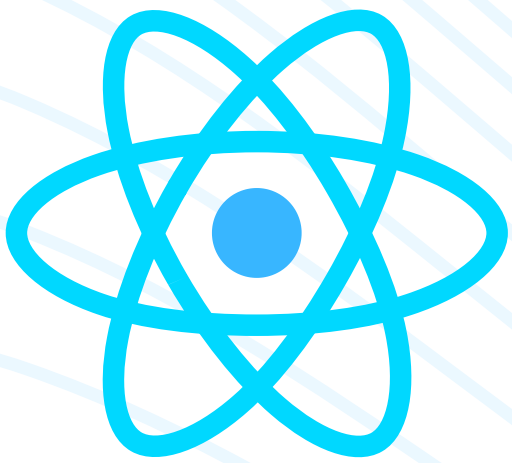
# DIFFERENT WAYS TO FETCH DATA IN



React

BY EZENAGU EMMANUEL

# THE FETCH METHOD

In React, the fetch method is a built-in JavaScript method that allows you to send HTTP requests to a server and receive responses. It is used to fetch data asynchronously from a server and update the state of a React component based on that data.

Here's an example of how to use the fetch method in React to retrieve data from an API endpoint:

# CODE SNIPPET

```
// JS TIPS BY EMMANUEL

function App() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then(response => response.json())
      .then(jsonData => setData(jsonData));
  }, []);

  return (
    <div>
      {data.map(post => (
        <div key={post.id}>
          <h2>{post.title}</h2>
          <p>{post.body}</p>
        </div>
      ))}
    </div>
  );
}
```

# AXIOS LIBRARY

Axios is a popular JavaScript library that allows developers to make HTTP requests from the browser or node.js. It provides a simple and easy-to-use API that can be used in a variety of ways, including making GET and POST requests, handling authentication, and more.

In React, Axios is commonly used to retrieve data from a server, update data on the server, or delete data from the server. It is often used in conjunction with React's useEffect hook to fetch data from a server when a component is mounted or updated.

# CODE SNIPPET

```
// JS TIPS BY EMMANUEL

function PostList() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => setPosts(response.data))
      .catch(error => console.log(error));
  }, []);

  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {posts.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

# ASYNC/AWAIT

Async/await is a syntax feature in JavaScript that simplifies asynchronous programming. It allows developers to write asynchronous code that looks and behaves like synchronous code.

In React, async/await can be used with functions that return Promises, such as the built-in fetch function used to make HTTP requests. By using async/await, developers can write code that waits for the result of an asynchronous operation to complete before proceeding with the next line of code.

Here's an example of using async/await with the fetch function in a React component:

# CODE SNIPPET

```
// JS TIPS BY EMMANUEL

function MyComponent() {
  const [data, setData] = useState([]);

  useEffect(() => {
    async function fetchData() {
      try {
        const response = await axios.get('https://jsonplaceholder.typicode.com/todos');
        setData(response.data);
      } catch (error) {
        console.log(error);
      }
    }
    fetchData();
  }, []);

  return (
    <div>
      {data.map(item => (
        <p key={item.id}>{item.title}</p>
      ))}
    </div>
  );
}
```

# CUSTOM HOOK

In React, a custom hook is a JavaScript function that uses built-in React hooks to encapsulate reusable logic. Custom hooks allow you to extract common logic from your components and reuse it across your application.

Custom hooks follow the naming convention of starting with use, which is a requirement for all hooks in React. They can use any of the existing hooks, such as useState, useEffect, or useContext, to create new hooks with a specific behavior.

Here's an example of a custom hook that uses useState to create a counter:

# CODE SNIPPET

```
// JS TIPS BY EMMANUEL

import { useState } from "react";

function useCounter(initialValue = 0, step = 1) {
  const [count, setCount] = useState(initialValue);

  function increment() {
    setCount(count + step);
  }

  function decrement() {
    setCount(count - step);
  }

  function reset() {
    setCount(initialValue);
  }

  function setTo(value) {
    setCount(value);
  }

  return { count, increment, decrement, reset, setTo };
}
```

# REACT QUERY LIBRARY

React Query is a popular library in React that provides a simple and efficient way to manage and cache remote data in a React application. It allows developers to easily fetch, cache, and update data from remote sources such as REST APIs, GraphQL endpoints, and more.

# CODE SNIPPET

```
// JS TIPS BY EMMANUEL

import React from 'react';
import { useQuery } from 'react-query';

const fetchPosts = async () => {
  const response = await fetch('https://jsonplaceholder.typicode.com/posts');
  return response.json();
}

const Posts = () => {
  const { data, isLoading, error } = useQuery('posts', fetchPosts);

  if (isLoading) {
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>Error fetching data</div>;
  }

  return (
    <ul>
      {data.map(post => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
}

export default Posts;
```

# DONT FORGET TO LIKE, SHARE AND SAVE IF YOU LIKE THIS POST

EZENAGU EMMANUEL