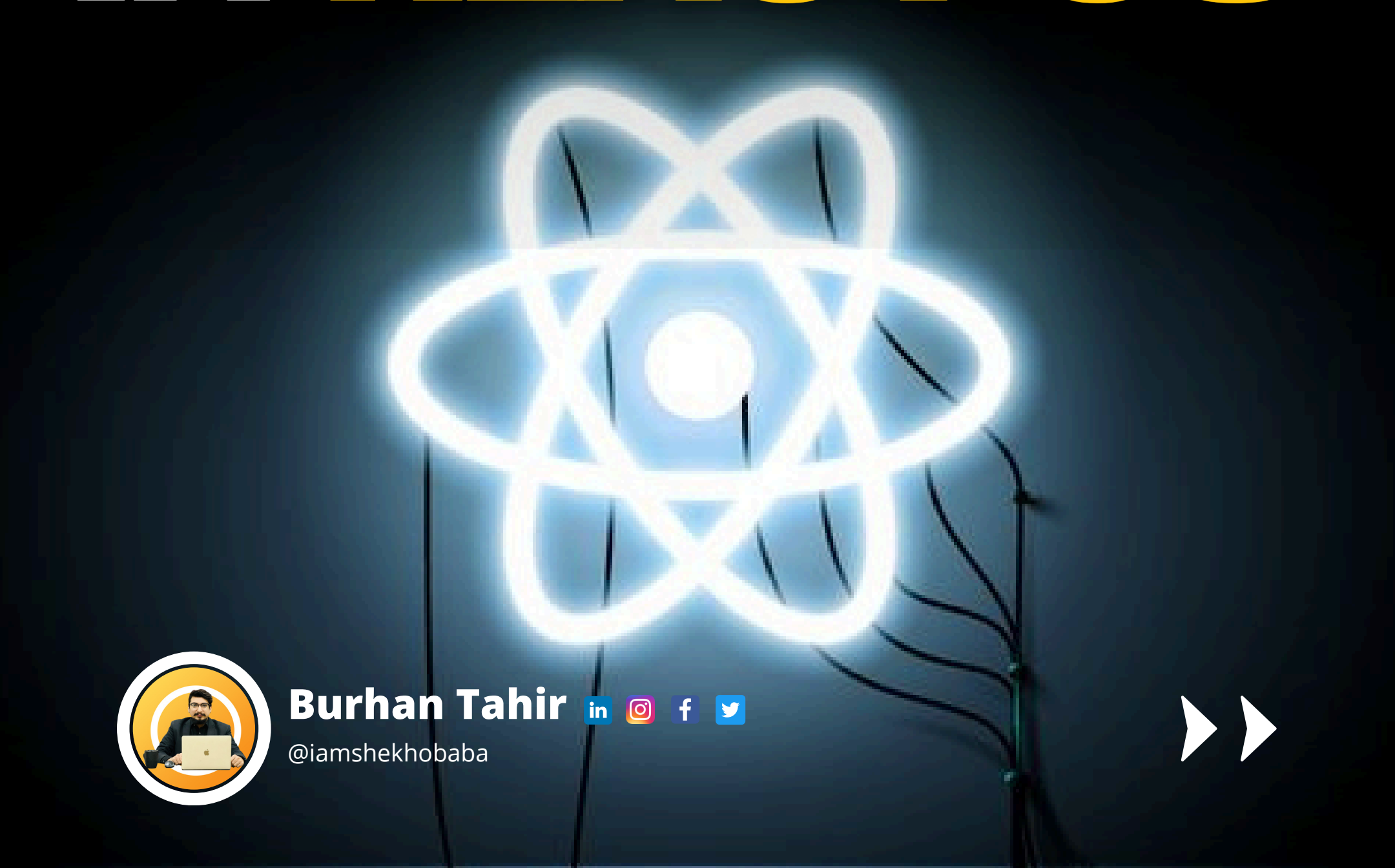
SHEKHOBABA.COM

DIFFERENT WAYS TO FETCH DATA IN REACT JS



Burhan Tahir

@iamshekhobaba

01

1. FETCH API

The Fetch API is a native JavaScript option available in modern browsers for making HTTP requests. It returns promises and is very straightforward to use.

```
useEffect(() => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));
}, []);
```

Burhan Tahir

@iamshekhobaba

02

2. AXIOS

Axios is a promise-based HTTP client for the browser and node.js. It provides more features than the Fetch API, such as intercepting requests and responses, and automatic transforms of JSON data.

```
import axios from 'axios';

useEffect(() => {
  axios.get('https://api.example.com/data')
    .then(response => console.log(response.data))
    .catch(error => console.error('Error:', error));
}, []);
```

Burhan Tahir

@iamshekhobaba

03

3. ASYNC/AWAIT SYNTAX

Using async/await syntax simplifies the handling of promises, making your asynchronous code appear more like synchronous code. It is particularly useful when combined with Fetch API or Axios.

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await fetch('https://api.example.com/data');
      const data = await response.json();
      console.log(data);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  fetchData();
}, []);
```

Burhan Tahir

@iamshekhobaba

4. REACT QUERY

React Query is a library that provides powerful tools for fetching, caching, synchronizing, and updating server state in React applications. It is highly efficient for handling server state in modern React applications.

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

const fetchProjects = async () => {
  const res = await fetch('https://api.example.com/data');
  return res.json();
};

function App() {
  const { data, error, isLoading } = useQuery('projects', fetchProjects);

  if (isLoading) return 'Loading...';
  if (error) return 'An error occurred: ' + error.message;

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

Burhan Tahir

@iamshekhobaba

5. CUSTOM HOOK

Creating a custom hook for data fetching allows you to encapsulate fetching logic into a reusable function. This hook can use any of the above methods internally and provide a clean interface for data fetching across your components.

Burhan Tahir
@iamshekhobaba

06

5. CUSTOM HOOK

```
function useCustomFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        const json = await response.json();
        setData(json);
        setLoading(false);
      } catch (error) {
        setError(error);
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
}
```

Burhan Tahir

@iamshekhobaba

SHEKHOBABA.COM

SHEKHOBABA.COM

FOLLOW ME FOR MORE INFORMATIVE CONTENT