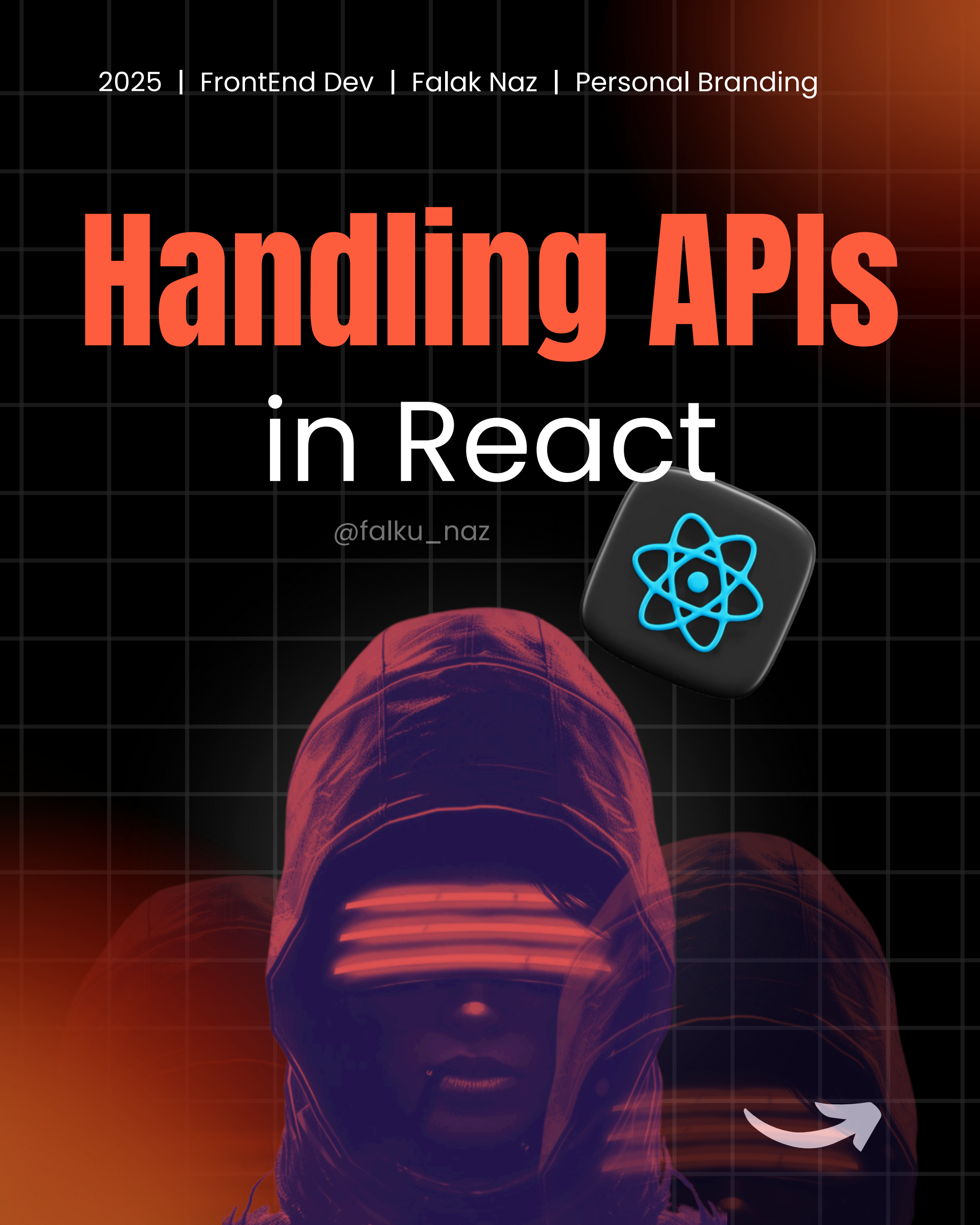
2025 | FrontEnd Dev | Falak Naz | Personal Branding

Handling APIs in React

@falku_naz



APIs are typically used to fetch data from a server and render that data dynamically within the application.

This can be done through different methods,

but the key is to use APIs to interact with backends, databases, or external services.

2025

REPOST ❤️

Main Purpose of Handling APIs in React

The **main purpose** of handling APIs in React is to fetch, send, or update data from/to a backend or external server and use that data to update the React component's state.

This helps in making the app dynamic, as data can be updated without reloading the entire page.

React allows developers to work with APIs through JavaScript and asynchronous operations, making it more efficient to build interactive web apps.

2025 REPOST ❤️

What is Fetch API?

fetch is a built-in JavaScript function used to make network requests.

It is part of the modern web API that provides a simple and clean way to make HTTP requests to a server.

It returns a Promise, so you can handle responses asynchronously.

```
fetch('https://api.example.com/data')
  .then(response ⇒ response.json())
  .then(data ⇒ console.log(data))
  .catch(error ⇒ console.error('Error:', error));
```

2025 REPOST ❤️

RESTful API

A RESTful API (Representational State Transfer) is an architectural style for designing networked applications.

It is stateless and relies on HTTP methods like

- GET
- POST
- PUT
- DELETE

to communicate between client and server.
It uses standard HTTP status codes to indicate the outcome of API requests.

2025

REPOST ❤️

Fetch using Async/Await

Using async/await allows for cleaner and more readable asynchronous code. You can handle Promises in a synchronous manner.

```
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);
  }
};
```

2025

REPOST ❤️

Fetch using Promises

With Promises, you chain .then() and .catch() methods to handle the API call's success or failure.

```
fetch('https://api.example.com/data')
  .then(response ⇒ response.json())
  .then(data ⇒ console.log(data))
  .catch(error ⇒ console.error('Error:', error));
```

2025 REPOST ❤️

Fetch using Callback

In this method, you pass a callback function to handle the response when the API call finishes. However, callbacks are less modern and can lead to "callback hell."

```
const fetchData = (callback) => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => callback(data))
    .catch(error => console.error('Error:', error));
};

fetchData(data => {
  console.log(data);
});
```

2025 REPOST ❤️

Fetch using Axios

Axios is a popular third-party library that simplifies working with APIs. It supports promises and is widely used for its ease of use, request and response interception, and more advanced features.

```
import axios from 'axios';

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

2025

REPOST ❤️

Advantages of Axios over Fetch:

- Automatic JSON parsing.
- Ability to handle requests and responses globally.
- Simplifies working with POST requests and error handling.

2025

REPOST ❤️

That's it!

Follow me!

Like this!

Share it!