API Calls with Promises in React:

Simplify Your Async Operations!

SUMANTH M

Frontend developer

Why Promises Matter in React

- Handle asynchronous operations more cleanly.
- Avoid callback hell with promise chaining.
- Simplify data fetching, sequencing, and more.
- Improve code readability and maintainability.

SUMANTH M
Frontend developer

Fetching Data from an API

- Promises streamline data retrieval.
- Avoids deeply nested callbacks.
- Ensures error handling in one place.
- Makes your code more predictable.

SUMANTH M
Frontend developer

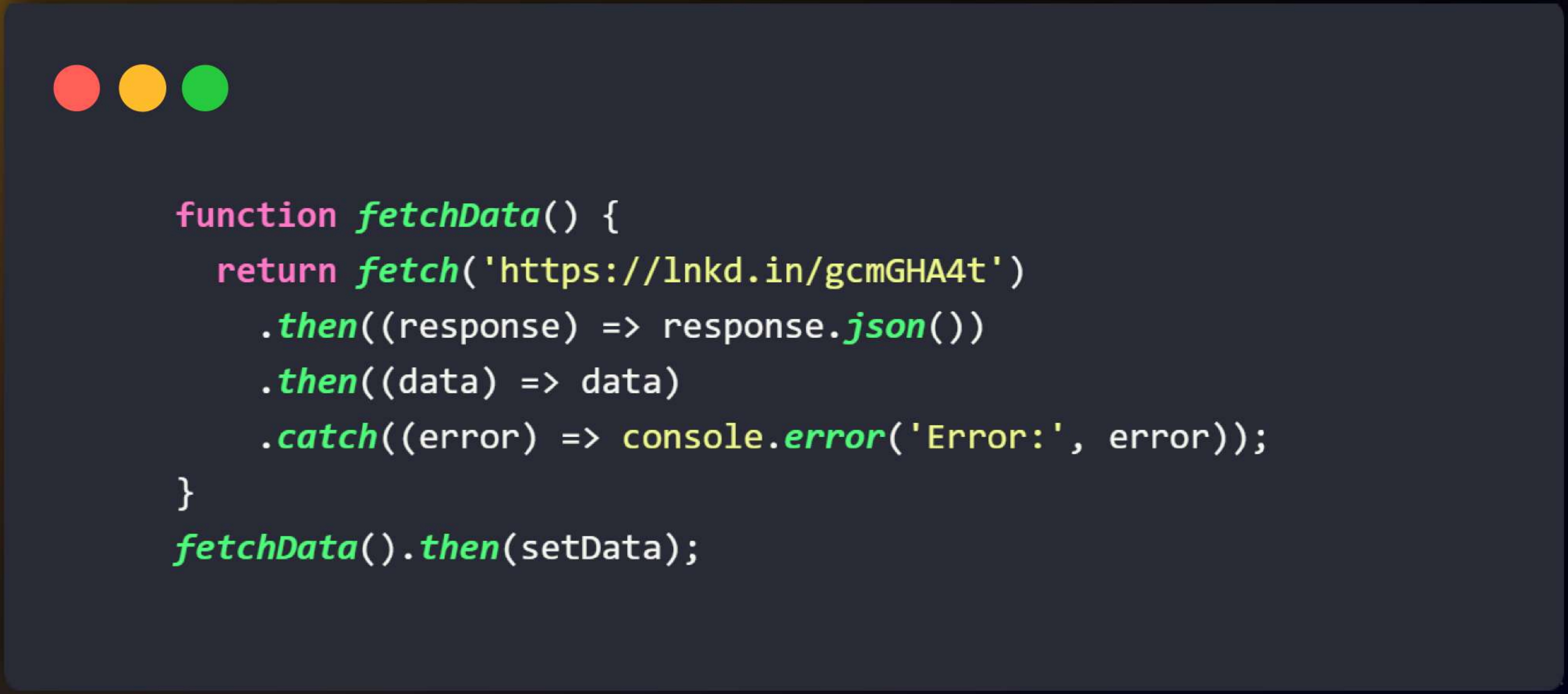
Without Promises

```
function fetchData(callback) {
  const xhr = new XMLHttpRequest();
  xhr.open('GET', 'https://lnkd.in/gvDcCTes', true);
  xhr.onload = function () {
    if (xhr.status === 200) {
      callback(null, JSON.parse(xhr.responseText));
    } else {
      callback(xhr.statusText);
    }
  };
  xhr.send();
}
```

Example: Fetching Data from an API

SUMANTH M
Frontend developer

With Promises



```
function fetchData() {
  return fetch('https://lnkd.in/gcmGHA4t')
    .then((response) => response.json())
    .then((data) => data)
    .catch((error) => console.error('Error:', error));
}
fetchData().then(setData);
```

Key Takeaway: Promises make API calls cleaner and more manageable

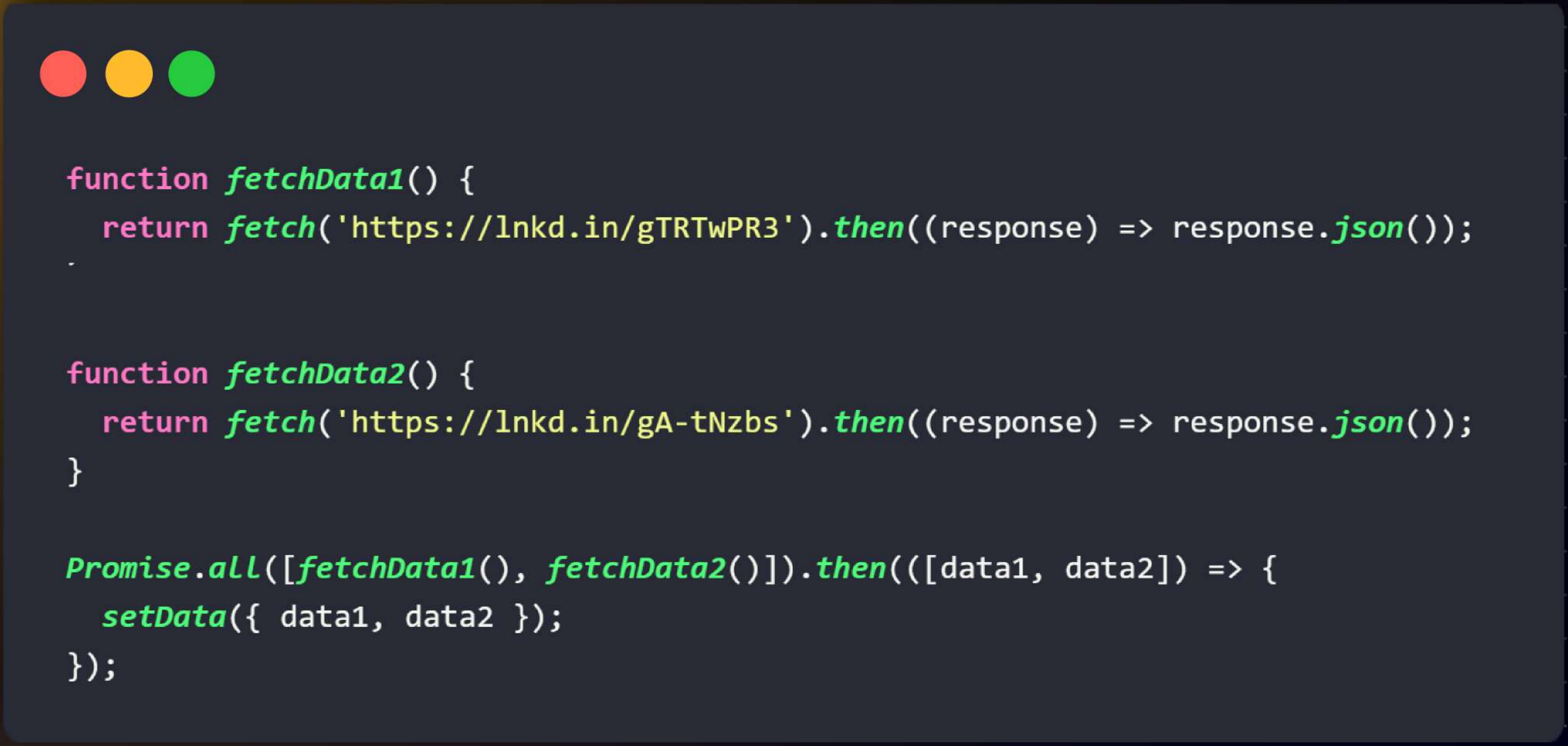
SUMANTH M
Frontend developer

Handling Multiple Asynchronous Operations

- Manage multiple async operations effortlessly.
- Wait for all operations to complete before proceeding.
- Avoid complex nested callbacks.
- Use Promise.all for cleaner code.

SUMANTH M
Frontend developer

With Promises



```
function fetchData1() {
  return fetch('https://lnkd.in/gTRTwPR3').then((response) => response.json());


function fetchData2() {
  return fetch('https://lnkd.in/gA-tNzbs').then((response) => response.json());
}

Promise.all([fetchData1(), fetchData2()]).then(([data1, data2]) => {
  setData({ data1, data2 });
});
```

Key Takeaway: Promise.all simplifies handling multiple async operations.

SUMANTH M

Frontend developer

Without Promises

```
function fetchSequentialData(callback) {
  fetchData1((err, result1) => {
    if (err) return callback(err);
    fetchData2(result1.id, (err, result2) => {
      if (err) return callback(err);
      callback(null, result2);
    });
  });
}
]
```

Example: Fetching Multiple Data

SUMANTH M
Frontend developer

Sequential Asynchronous Operations

- Handle dependent operations with promise chaining.
- Each operation waits for the previous one to complete.
- Avoid deeply nested callbacks.
- More intuitive and readable flow.

SUMANTH M
Frontend developer

Without Promises

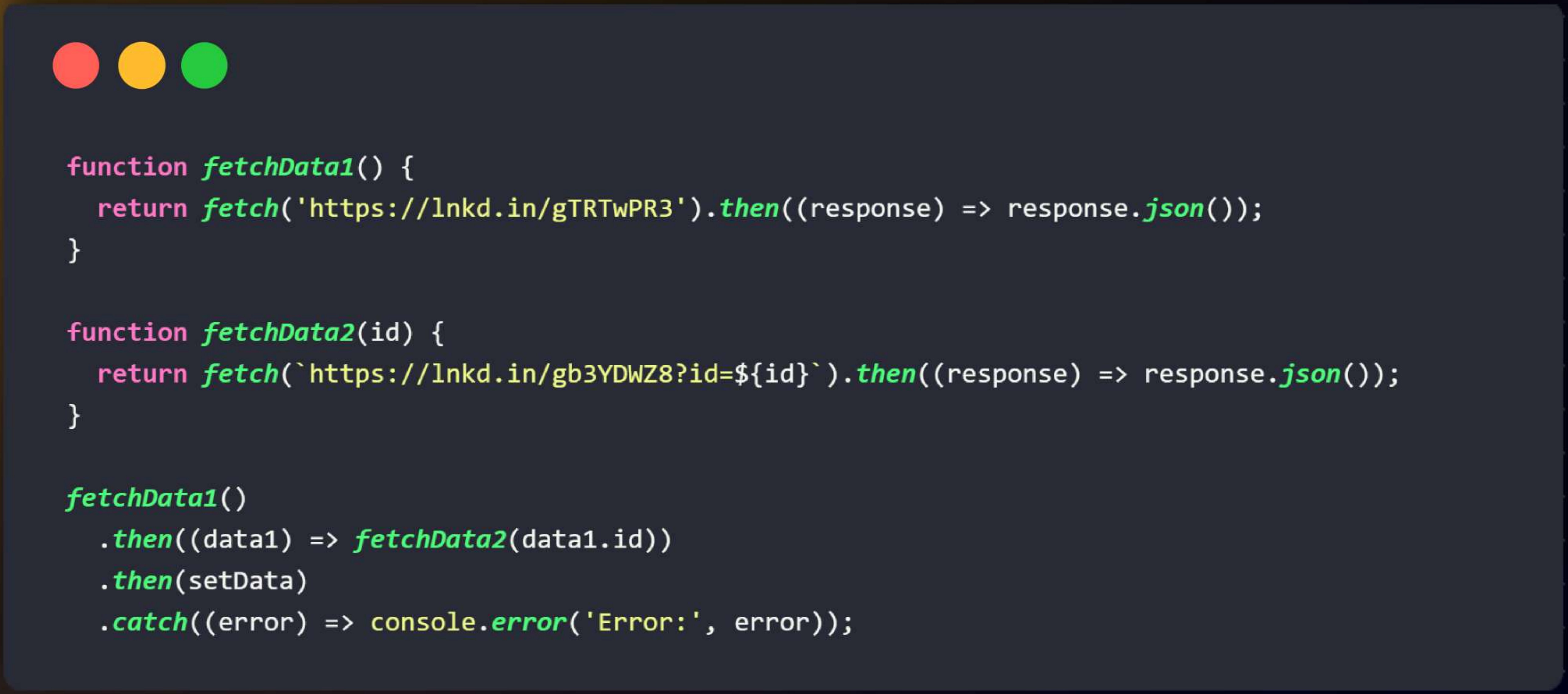
```
function fetchSequentialData(callback) {
  fetchData1((err, result1) => {
    if (err) return callback(err);
    fetchData2(result1.id, (err, result2) => {
      if (err) return callback(err);
      callback(null, result2);
    });
  });
}
```

Example: Sequential Operations

SUMANTH M

Frontend developer

With Promises



```
function fetchData1() {
  return fetch('https://lnkd.in/gTRTwPR3').then((response) => response.json());
}

function fetchData2(id) {
  return fetch(`https://lnkd.in/gb3YDWZ8?id=${id}`).then((response) => response.json());
}

fetchData1()
  .then((data1) => fetchData2(data1.id))
  .then(setData)
  .catch((error) => console.error('Error:', error));
```

Key Takeaway: Chain promises for sequential operations

SUMANTH M
Frontend developer

CONCLUSION:

Embrace Promises in React

- Use promises to simplify asynchronous code.
- Avoid callback hell with chaining and Promise.all.
- Write more maintainable and error-proof code.

Discussion

How do you use promises in your React projects?

SHARE IN THE COMMENTS!

SUMANTH M
Frontend developer