Note

Before starting the session, ensure that you have Node.js and VS Code installed on your computer. In addition, download the repository and install the relevant packages following the readme instructions.

github.com/emerge-tech-workshop/23-react

Introduction to Single Page Applications

Conduct by

Roshan Jayathunga
Software Engineer
www.linkedin.com/in/jayathunga

Hashan Dewaraja
Software Engineer
www.linkedin.com/in/dmhashan

Outline

Single Page Application

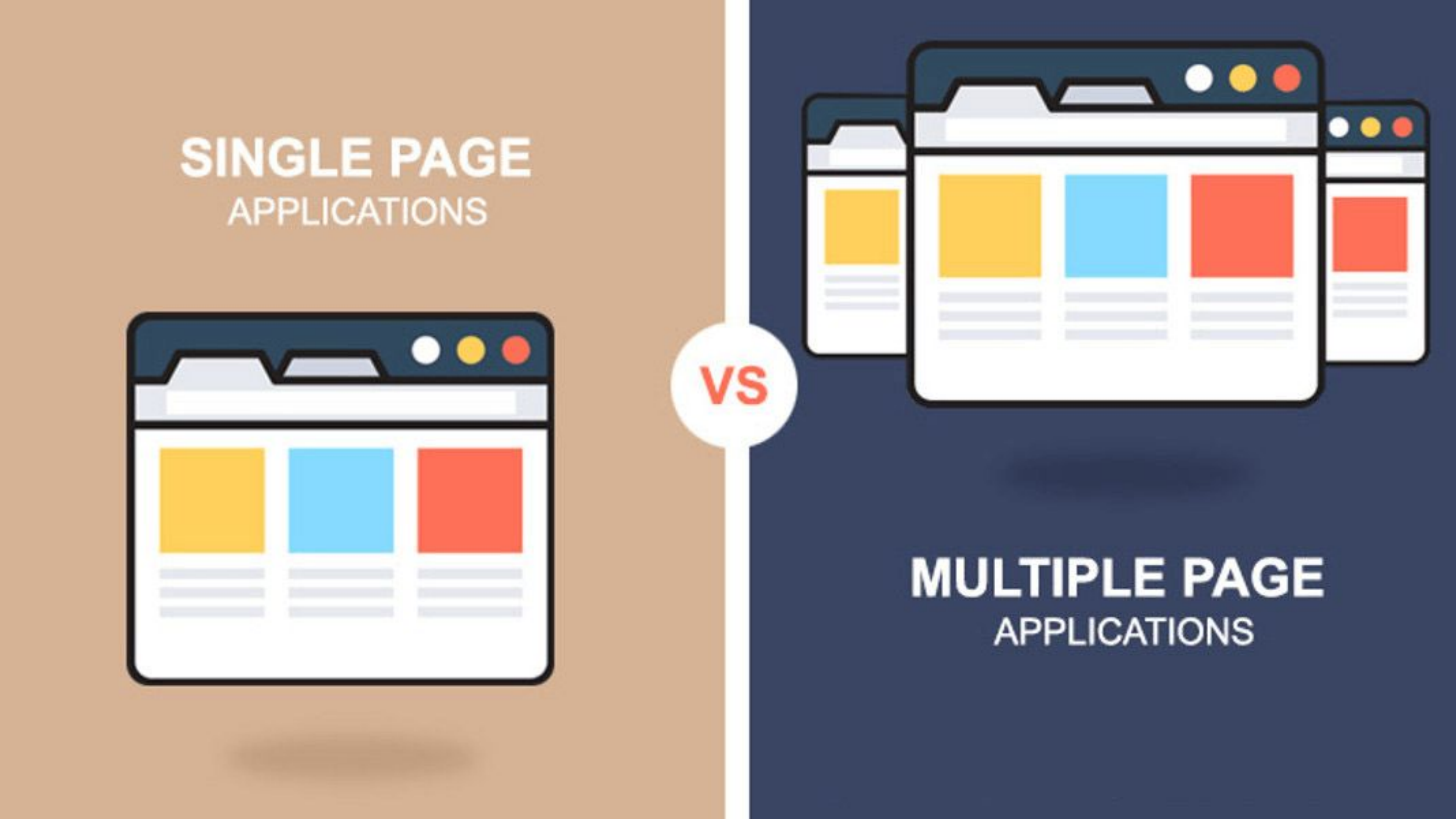
Front End Frameworks

SPA Development with React JS

Data Flow in React

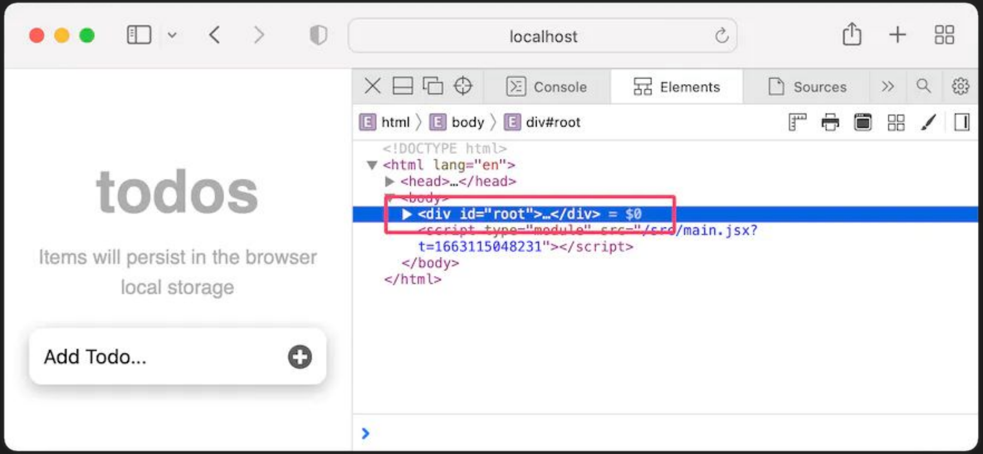
Component Lifecycle

Consume REST APIs


SINGLE PAGE
APPLICATIONS
VS
MULTIPLE PAGE
APPLICATIONS

What is a Single-Page Application?

Dynamically load content into the current page without loading an entire page from the server.



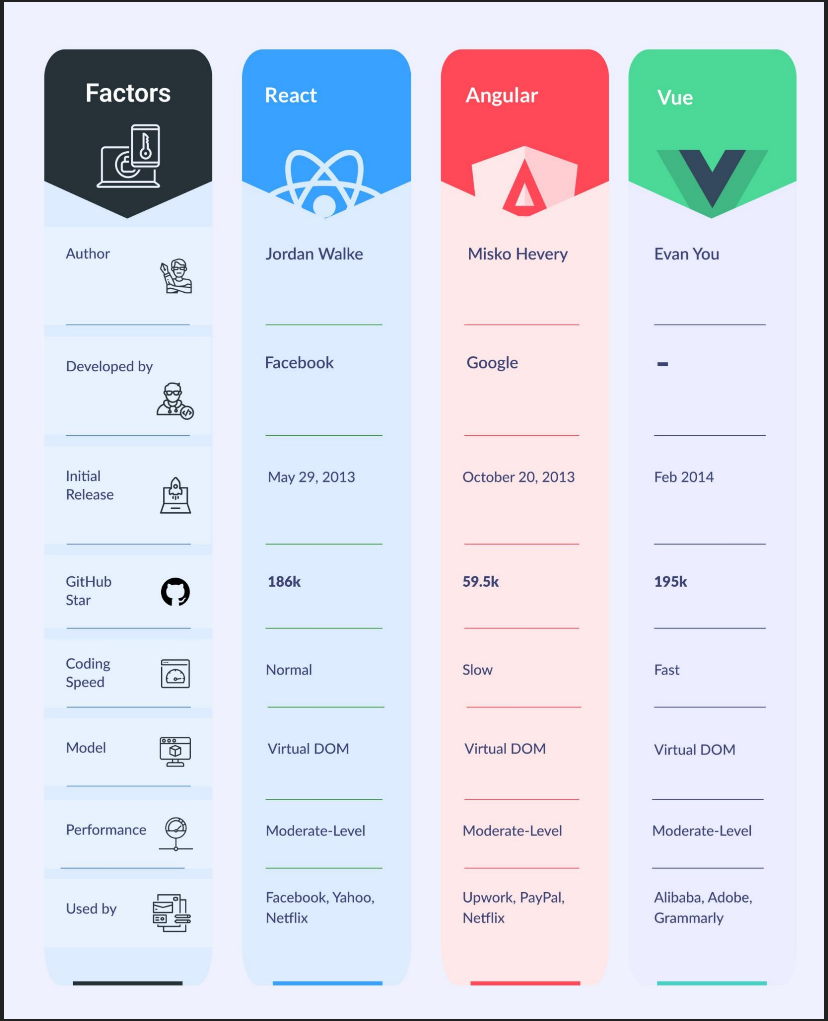
```
<!DOCTYPE html>
▼ <html lang="en">
  ▶ <head>…</head>
  ▼ <body>
    ▶ <div id="root">…</div> = $0
      <script type="module" src="/src/main.jsx?
      t=1663115048231"></script>
    </body>
  </html>
```

Exercise 01

Create a basic webpage using HTML and JavaScript that consists of a navigation menu and dynamically modifies its content.

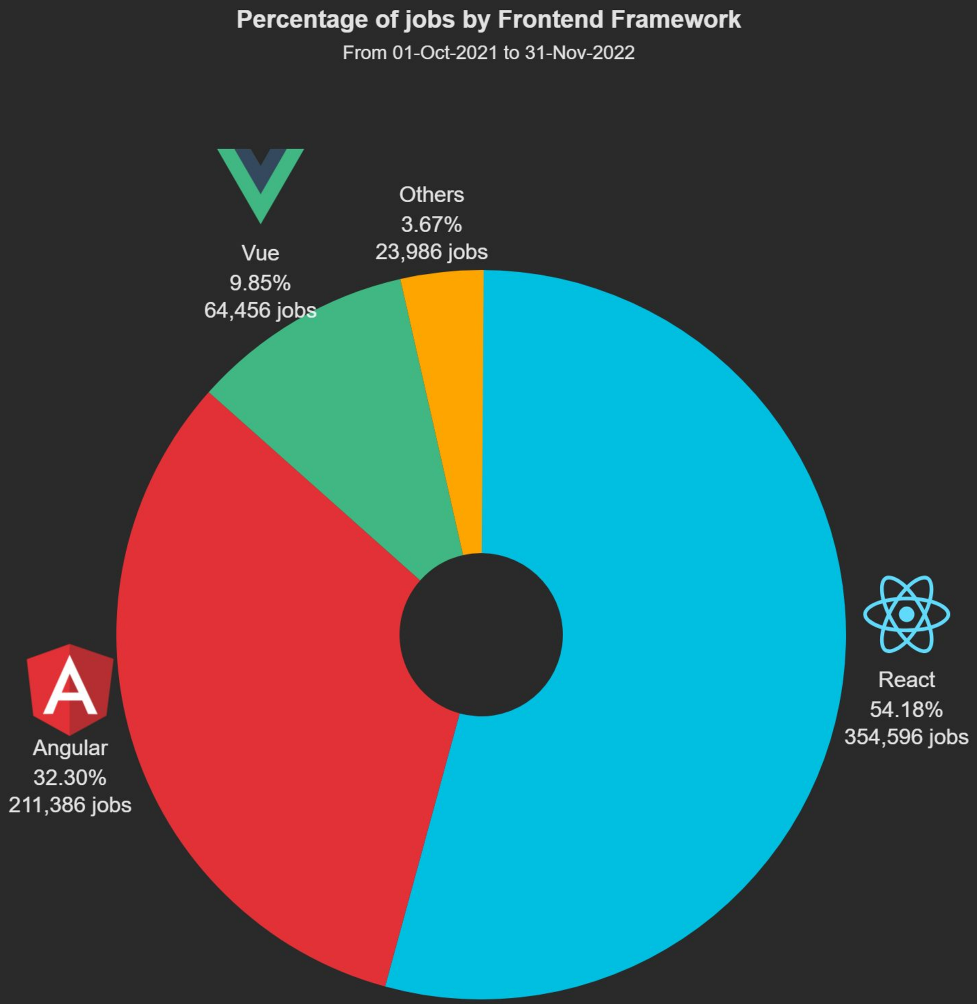
React / Angular / Vue
Vanilla JS

Front End Frameworks



Factors	React	Angular	Vue
Author	Jordan Walke	Misko Hevery	Evan You
Developed by	Facebook	Google	-
Initial Release	May 29, 2013	October 20, 2013	Feb 2014
GitHub Star	**186k**	**59.5k**	**195k**
Coding Speed	Normal	Slow	Fast
Model	Virtual DOM	Virtual DOM	Virtual DOM
Performance	Moderate-Level	Moderate-Level	Moderate-Level
Used by	Facebook, Yahoo, Netflix	Upwork, PayPal, Netflix	Alibaba, Adobe, Grammarly

Why should we choose React



Create a React JS App

1. Create a new app using create-react-app
2. Folder structure
3. Create and Reuse components
4. Setup UI Library
5. Routing

UI Libraries

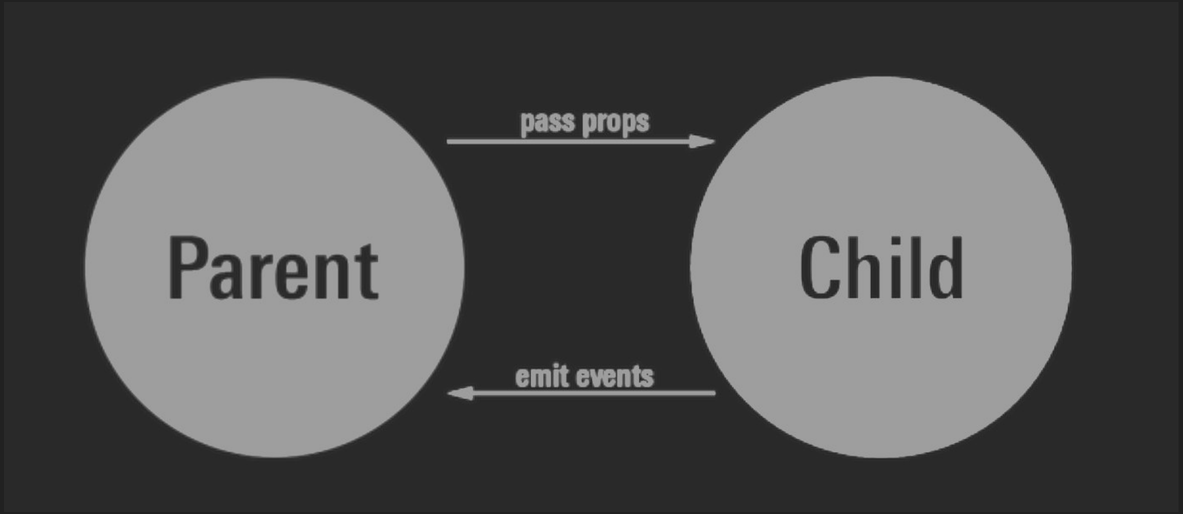
1. Bootstrap - https://getbootstrap.com
2. React Material UI - https://mui.com
3. Ant Design - https://ant.design
4. Chakra UI - https://chakra-ui.com
5. Tailwind CSS - https://tailwindcss.com

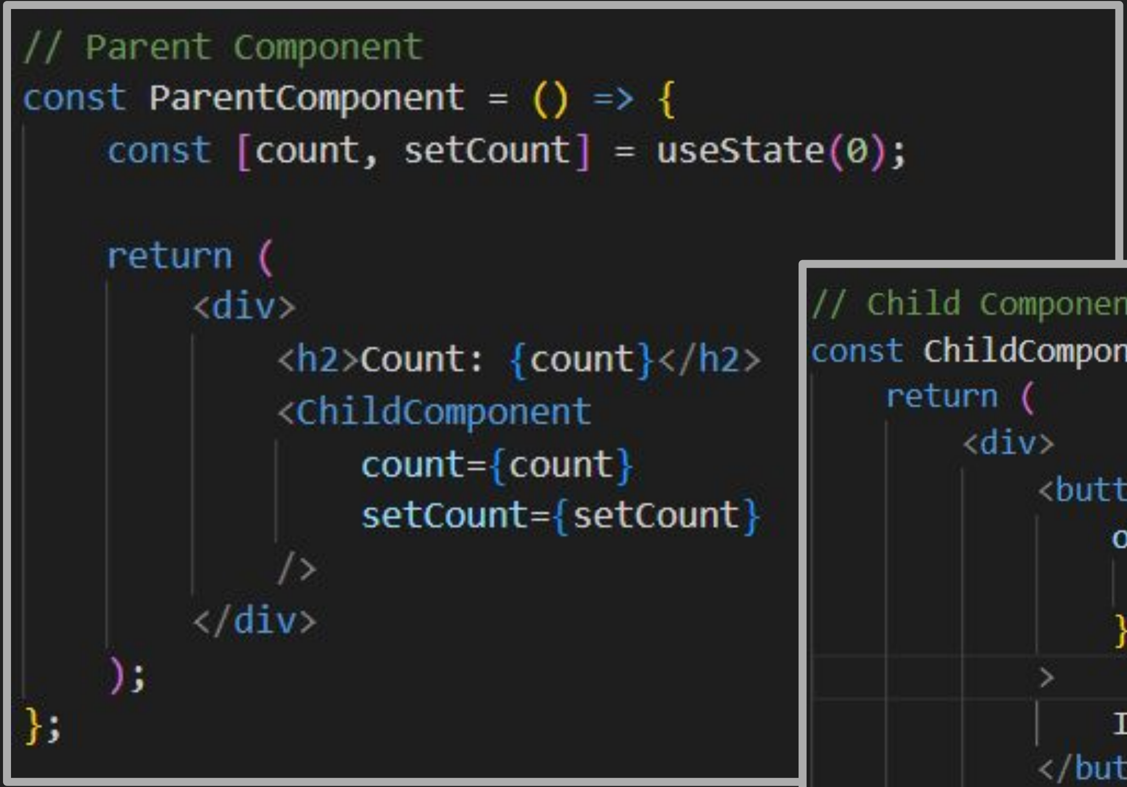
Break!

Data Flow in React

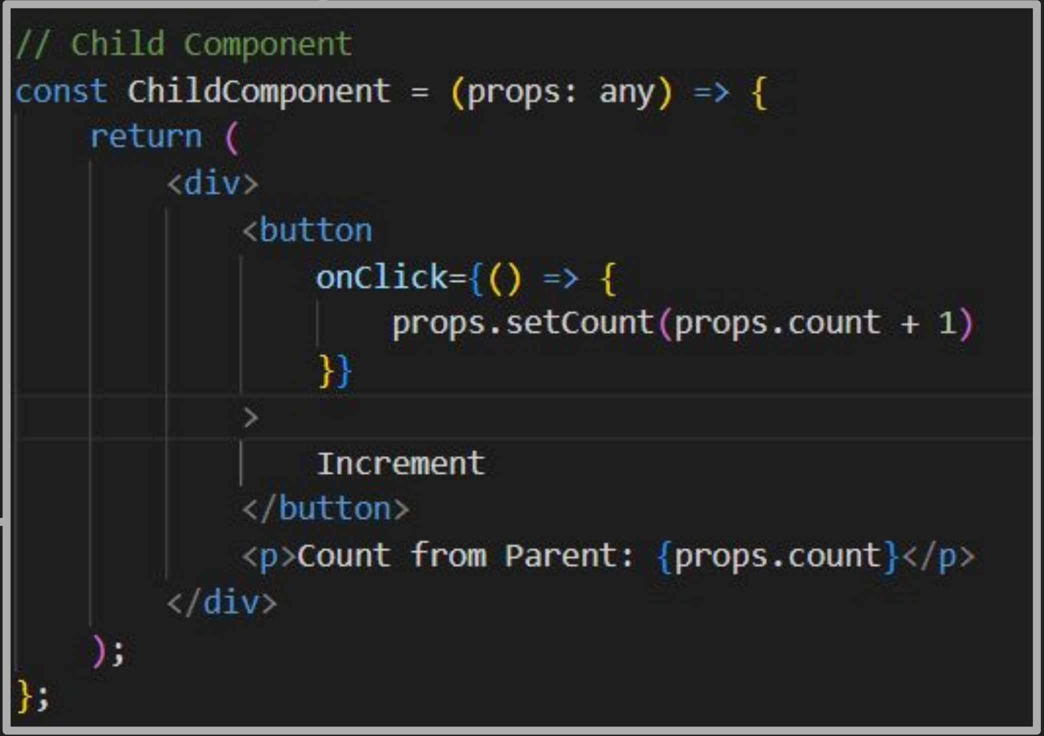
Unidirectional Data Flow

One-Way Data Binding





```
// Parent Component
const ParentComponent = () => {
    const [count, setCount] = useState(0);

    return (
        <div>
            <h2>Count: {count}</h2>
            <ChildComponent
                count={count}
                setCount={setCount}
            />
        </div>
    );
};
```

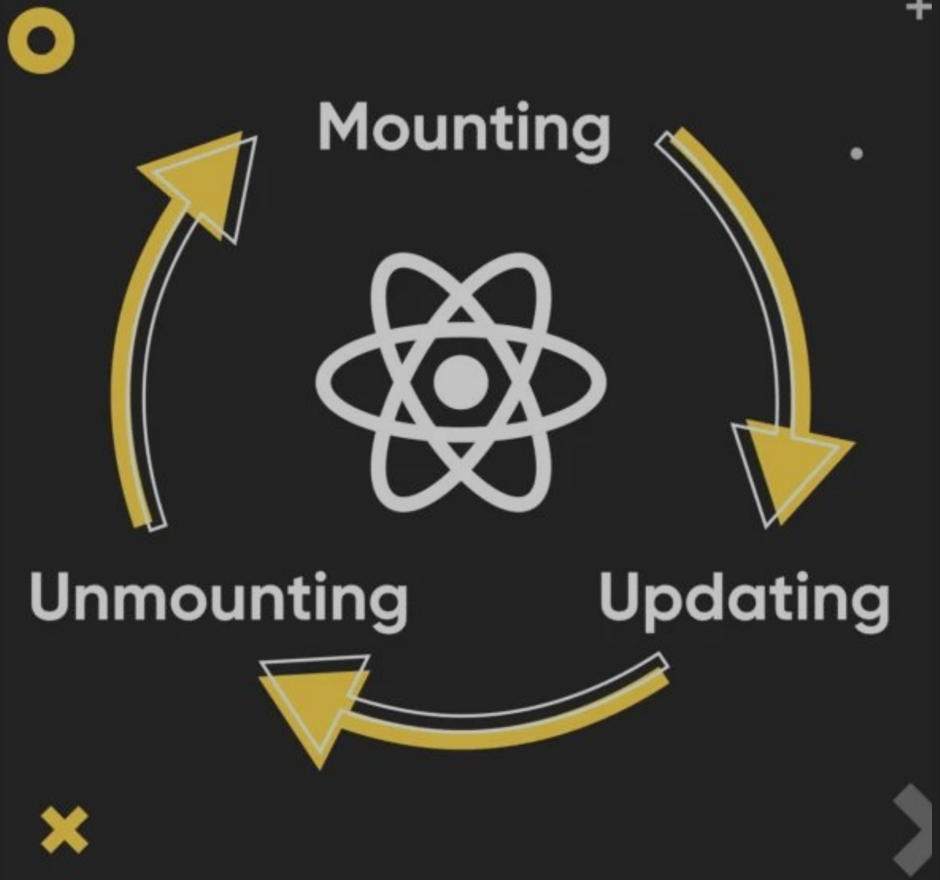


```
// Child Component
const ChildComponent = (props: any) => {
    return (
        <div>
            <button
                onClick={() => {
                    props.setCount(props.count + 1)
                }}
            >
                Increment
            </button>
            <p>Count from Parent: {props.count}</p>
        </div>
    );
};
```

Component Lifecycle

Class components have a lifecycle that is managed through a set of lifecycle methods

Function components have a simpler lifecycle based on hooks.



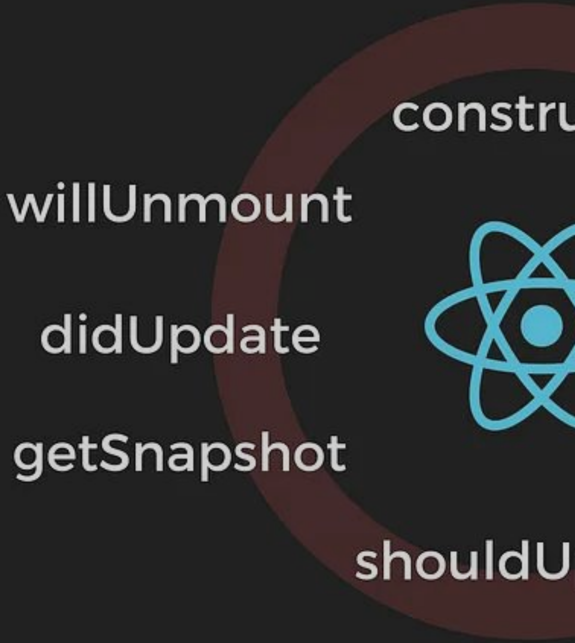
Class Component Lifecycle

Phase	Method

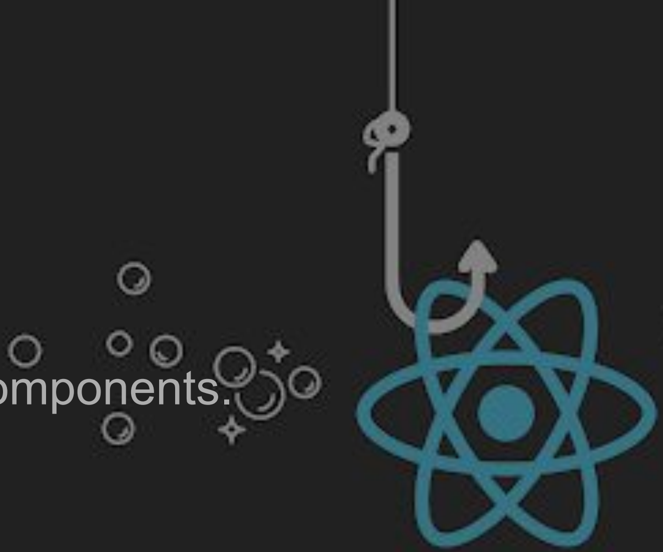
Mounting	constructor()
	componentWillMount()
	componentDidMount()

Updating	componentWillReceiveProps(nextProps)
	shouldComponentUpdate(nextProps, nextState)
	componentWillUpdate(nextProps, nextState)
	render()
	componentDidUpdate(prevProps, prevState)

Unmounting	componentWillUnmount()



Hooks



Introduced in Version 16.8

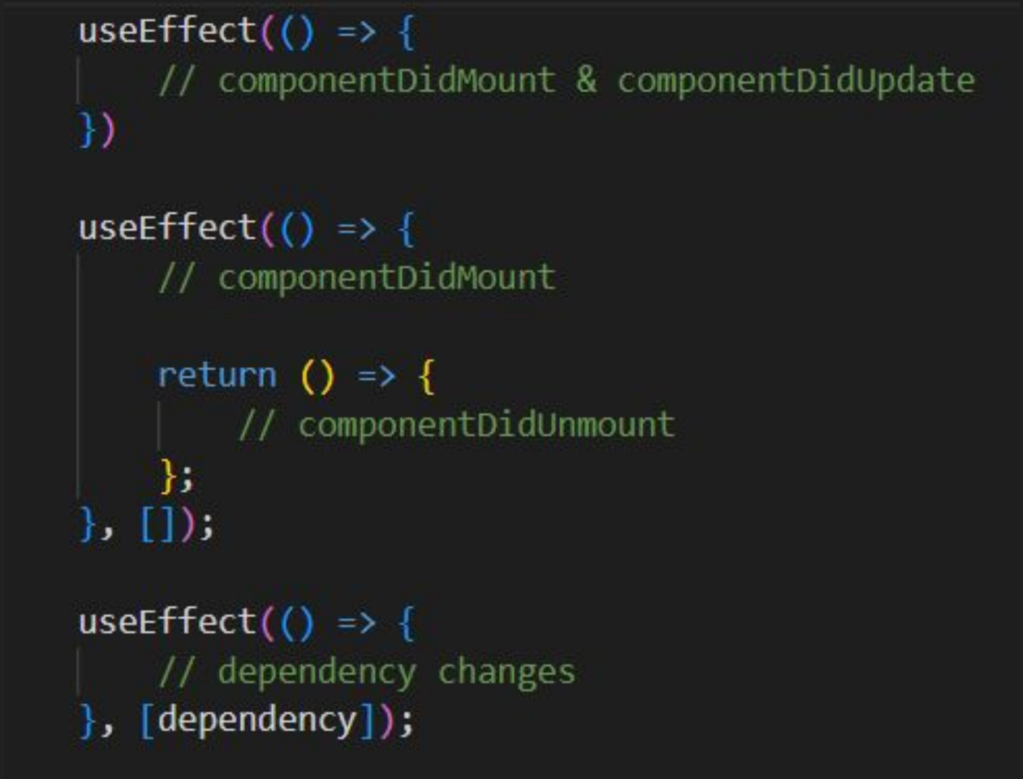
Hooks let you use different React features from your components.

State Hooks **Effect Hooks**

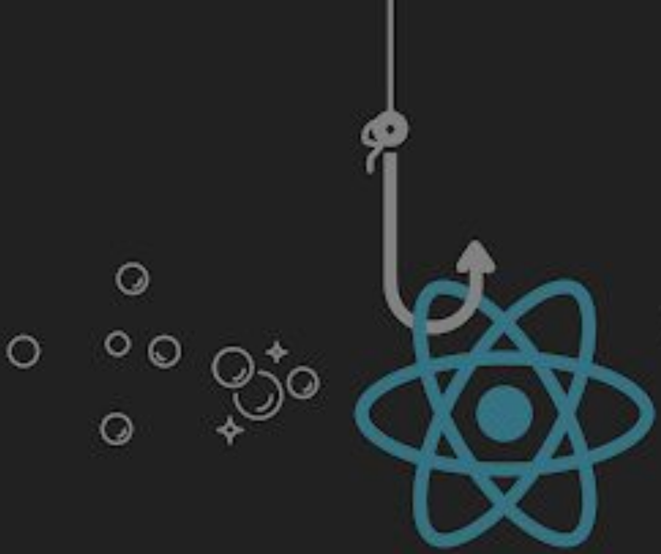
Ref Hooks Context Hooks Performance Hooks

and etc.

Function Component Lifecycle



```
useEffect(() => {
    // componentDidMount & componentDidUpdate
})

useEffect(() => {
    // componentDidMount

    return () => {
        // componentDidUnmount
    };
}, []);

useEffect(() => {
    // dependency changes
}, [dependency]);
```



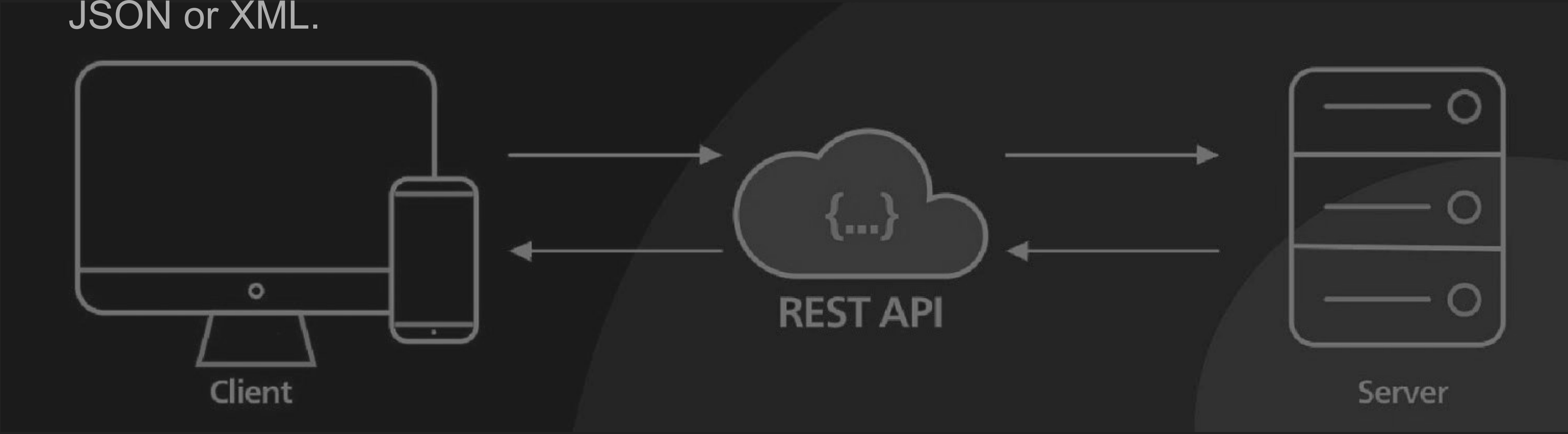
Exercise 02

Implement a Todo App using a state array.

REST APIs

A REST API allows clients to perform operations on resources (such as retrieving, creating, updating, or deleting) by sending HTTP requests to the server.

The server processes these requests, performs the necessary actions, and sends back HTTP responses to the client, typically in a standardized format such as JSON or XML.



Consume REST APIs

Fetch API (a browser in-built web API)

{ fetch }

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Axios (a promise-based HTTP client)

AXIOS

https://github.com/axios/axios

How to Install and Configure an Axios Instance

Axios, unlike the Fetch API, is not built-in

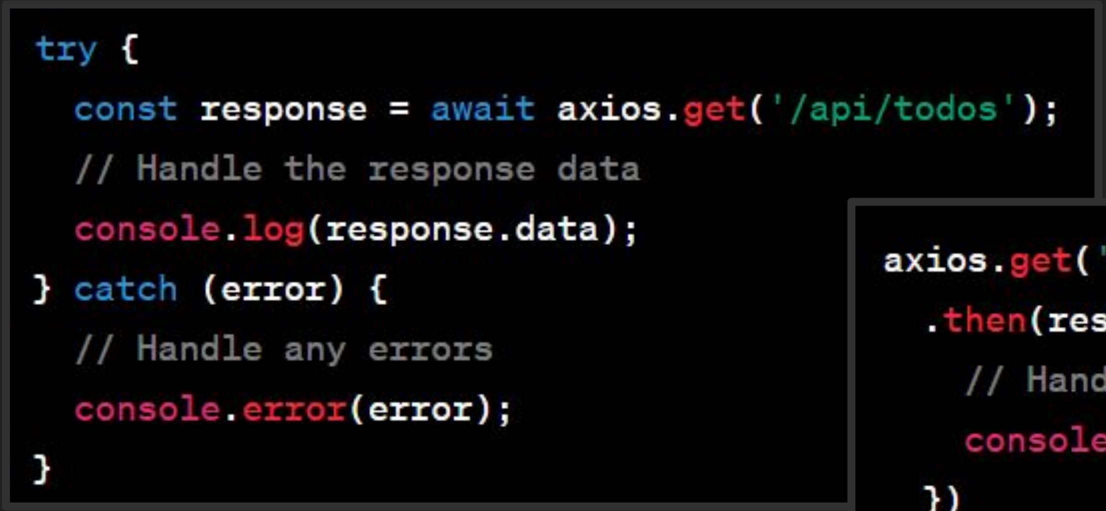
Add Axios to your project by running the following command:

```
npm install axios
```

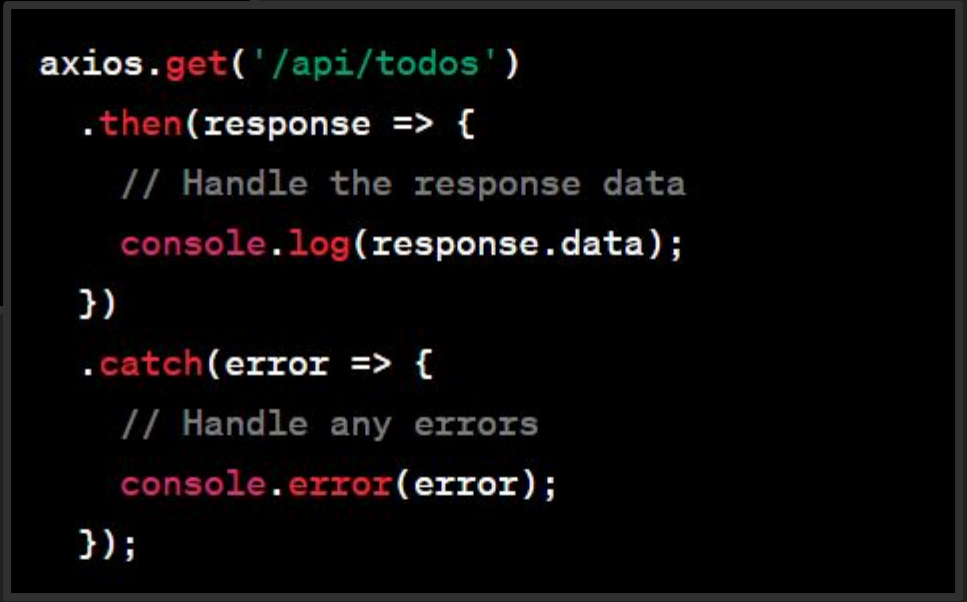
Now, we can proceed to create an instance, which is optional but recommended as it saves us from unnecessary repetition.

```
import axios from "axios";

const client = axios.create({
   baseURL: "https://jsonplaceholder.typicode.com/posts"
});
```

Perform a GET Request with Axios

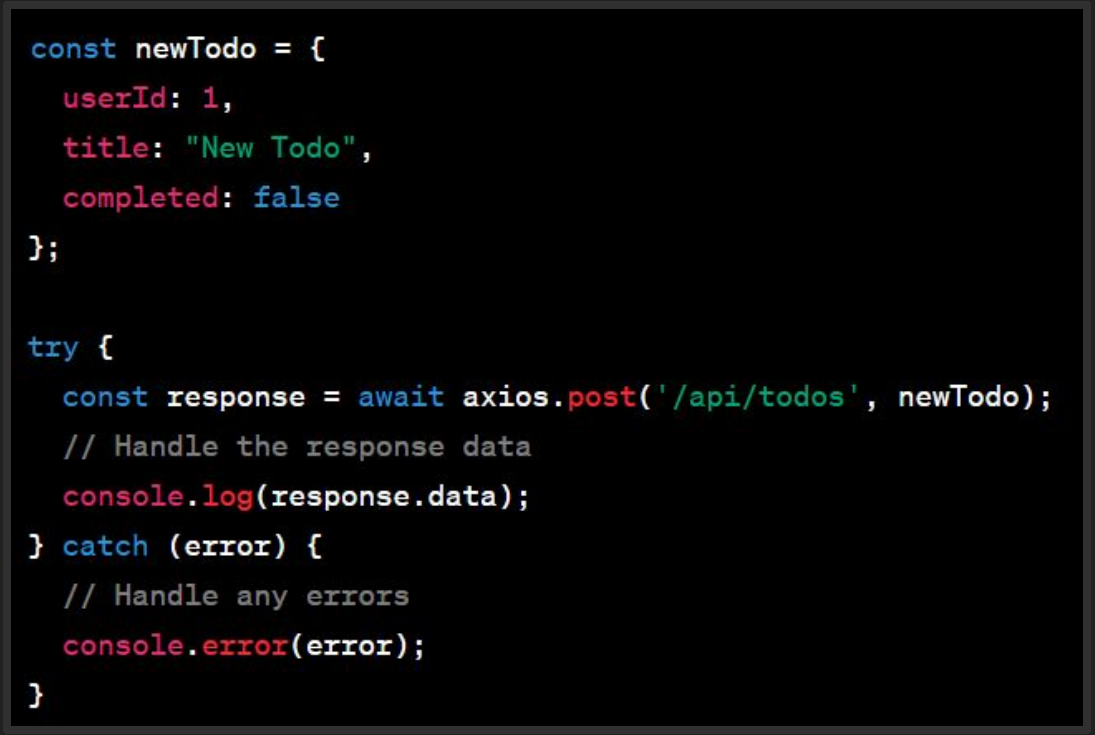


```
try {
  const response = await axios.get('/api/todos');
  // Handle the response data
  console.log(response.data);
} catch (error) {
  // Handle any errors
  console.error(error);
}
```

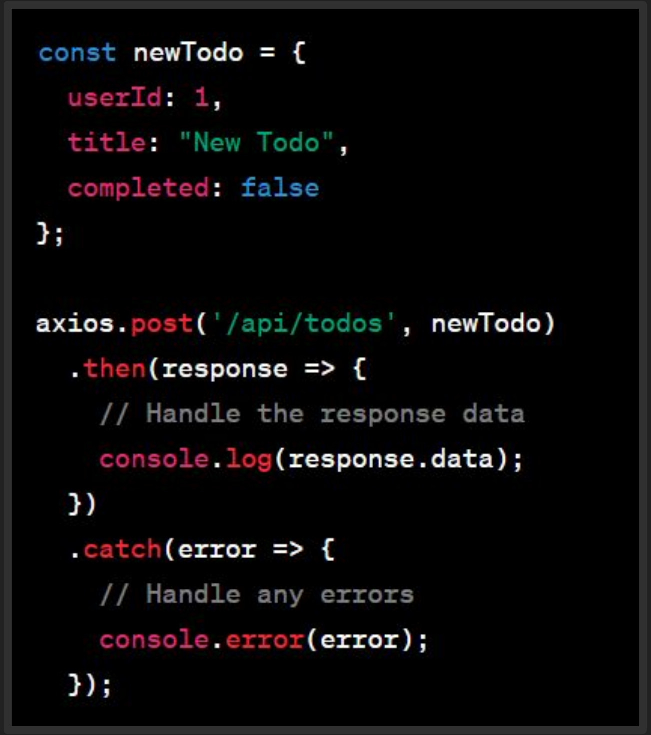


```
axios.get('/api/todos')
  .then(response => {
    // Handle the response data
    console.log(response.data);
  })
  .catch(error => {
    // Handle any errors
    console.error(error);
  });
```

Perform a POST Request with Axios

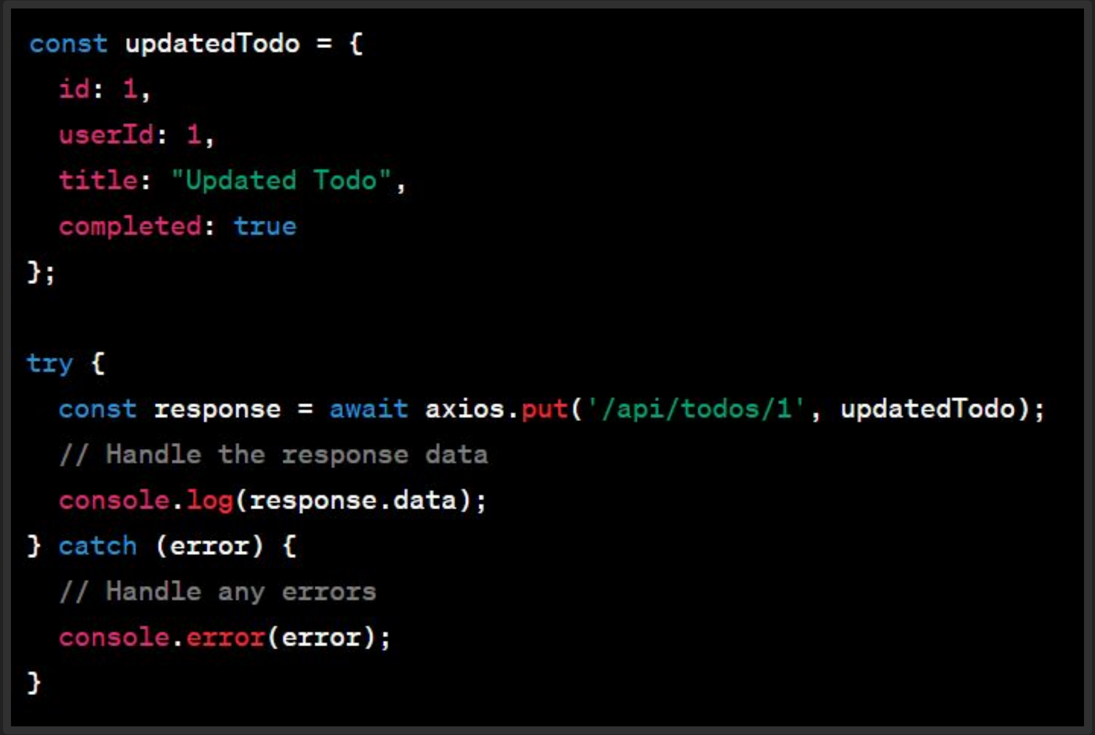


```javascript
const newTodo = {
  userId: 1,
  title: "New Todo",
  completed: false
};

try {
  const response = await axios.post('/api/todos', newTodo);
  // Handle the response data
  console.log(response.data);
} catch (error) {
  // Handle any errors
  console.error(error);
}
```

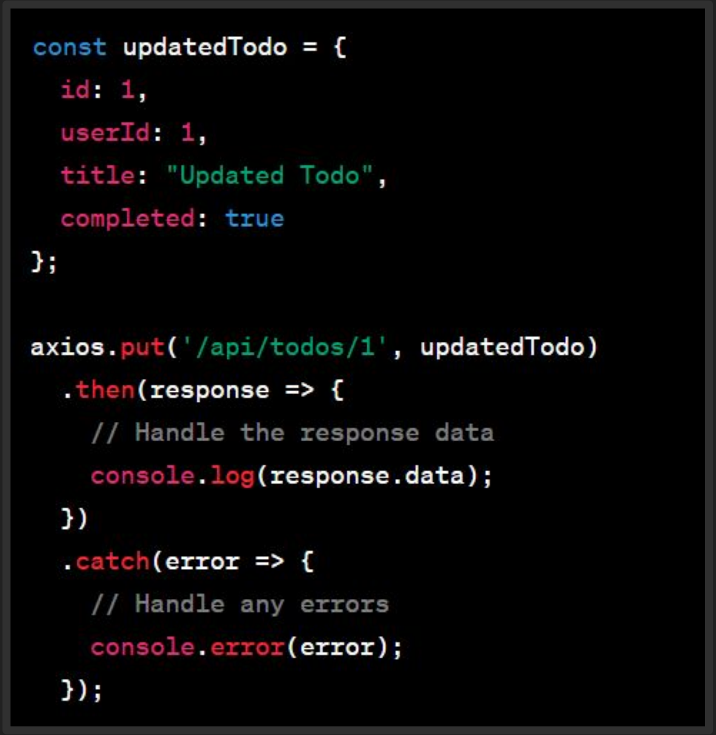


```javascript
const newTodo = {
  userId: 1,
  title: "New Todo",
  completed: false
};

axios.post('/api/todos', newTodo)
  .then(response => {
    // Handle the response data
    console.log(response.data);
  })
  .catch(error => {
    // Handle any errors
    console.error(error);
  });
```

Perform a PUT Request with Axios

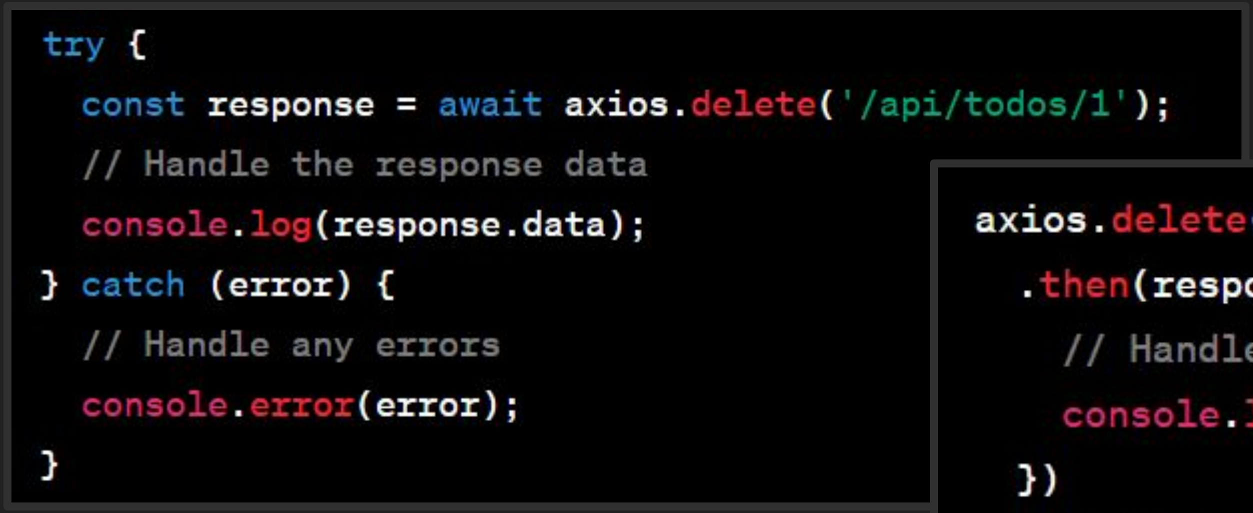


```javascript
const updatedTodo = {
  id: 1,
  userId: 1,
  title: "Updated Todo",
  completed: true
};


try {
  const response = await axios.put('/api/todos/1', updatedTodo);
  // Handle the response data
  console.log(response.data);
} catch (error) {
  // Handle any errors
  console.error(error);
}
```

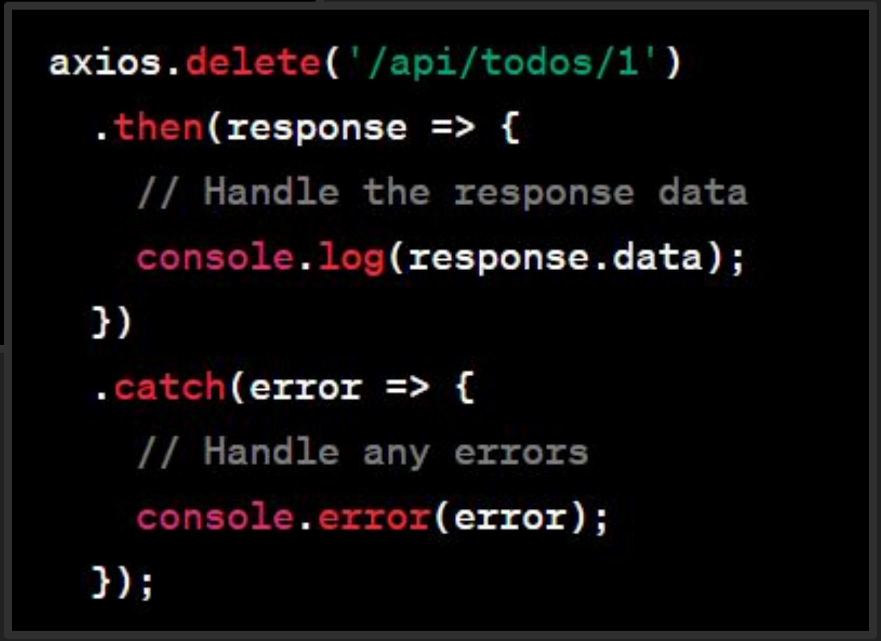


```javascript
const updatedTodo = {
  id: 1,
  userId: 1,
  title: "Updated Todo",
  completed: true
};


axios.put('/api/todos/1', updatedTodo)
  .then(response => {
    // Handle the response data
    console.log(response.data);
  })
  .catch(error => {
    // Handle any errors
    console.error(error);
  });
```

Perform a DELETE Request with Axios



```
try {
  const response = await axios.delete('/api/todos/1');
  // Handle the response data
  console.log(response.data);
} catch (error) {
  // Handle any errors
  console.error(error);
}
```



```
axios.delete('/api/todos/1')
  .then(response => {
    // Handle the response data
    console.log(response.data);
  })
  .catch(error => {
    // Handle any errors
    console.error(error);
  });
```

How to Handle Errors with Axios

Promise-based requests	async/await
.then() and.catch () methods	try...catch block

```
client.get('/todos')
  .then((response) => {
    console.log(response);
  })
  .catch((error) => {
    console.log(error);
  })
```

```
try {
  let response =
    await client.get('/todos');
  console.log(response);
} catch (error) {
  console.log(error);
}
```

Exercise 03

Modify the previously developed Todo application using this REST api.

https://jsonplaceholder.typicode.com/todos

Listing all todos	GET	https://jsonplaceholder.typicode.com/todos	{ "userId": 1, "id": 1, "title": "todo note", "completed": false }
Creating a todo	POST		
Updating a todo	PUT	https://jsonplaceholder.typicode.com/todos/{id}	
Deleting a todo	DELETE		