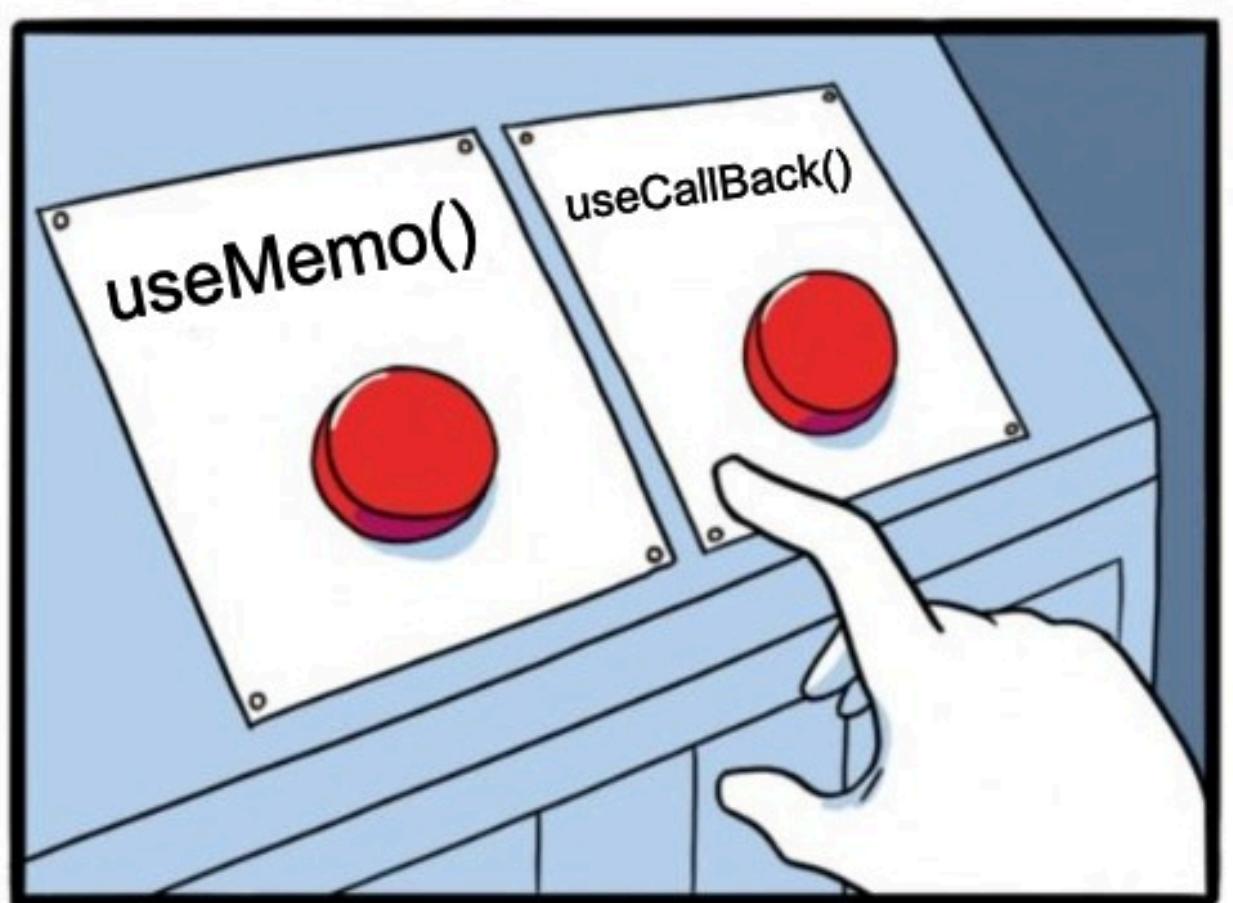


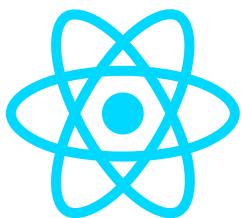
What's New in

# React v19

Let's Explore







# Compiler





PERFORMANCE  
OPTIMISATION HOOKS &

REACT  
COMPILER



## React Complier Playground



### React Compiler Playground

```
1 export default function MyApp() {  
2   return <div>Hello World</div>;  
3 }  
4
```

- JS

```
import { c as _c } from "react/compiler-runtime";  
export default function MyApp() {  
  const $ = _c(1);  
  let t0;  
  if ($[0] === Symbol.for("react.memo_cache_sentinel")) {  
    t0 = <div>Hello World</div>;  
    $[0] = t0;  
  } else {  
    t0 = $[0];  
  }  
  return t0;  
}
```

SourceMap

EnvironmentConfig

HIR

DropManualMemoization

PruneMaybeThrows

EliminateRedundantPhi

SSA

MergeConsecutiveBlocks

InlineImmediatelyInvokedFunctionExpressions

InferTypes

AnalyseFunctions

OptimizePropsMethodCalls

RewriteInstructionKindsBasedOnReassignment

InferReactivePlaces

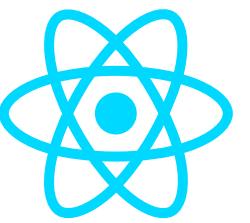
InferMutableRanges

DeadCodeElimination

Reset

Share

Copy



# Errors



# Existing Logs



The screenshot shows a browser's developer tools with the 'Console' tab selected. The log area displays several error messages:

- Warning: Text content did not match. Server: "Server" Client: "Client"  
at span  
at App
- Warning: An error occurred during hydration. The server HTML was replaced with client content in <div>.
- Warning: Text content did not match. Server: "Server" Client: "Client"  
at span  
at App
- Warning: An error occurred during hydration. The server HTML was replaced with client content in <div>.
- Uncaught Error: Text content does not match server-rendered HTML.  
at checkForUnmatchedText  
...

# Improvised Logs

The screenshot shows a browser's developer tools console tab labeled "Console". An error message is displayed:

✖ Uncaught Error: Hydration failed because the server rendered HTML didn't match the client. As a result this tree will be regenerated on the client. This can happen if an SSR-ed Client Component used:

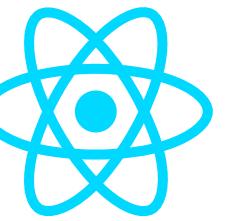
- A server/client branch `if (typeof window !== 'undefined')`.
- Variable input such as `Date.now()` or `Math.random()` which changes each time it's called.
- Date formatting in a user's locale which doesn't match the server.
- External changing data without sending a snapshot of it along with the HTML.
- Invalid HTML tag nesting.

It can also happen if the client has a browser extension installed which messes with the HTML before React loaded.

<https://react.dev/link/hydration-mismatch>

```
<App>
  <span>
+   Client
-   Server

  at throwOnHydrationMismatch
  ...
```

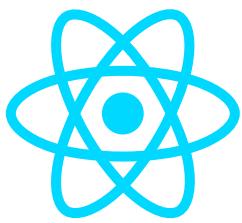


# SEO



# Support for MetaData

```
function BlogPost({post}) {
  return (
    <article>
      <h1>{post.title}</h1>
      <title>{post.title}</title>
      <meta name="author" content="Josh" />
      <link rel="author" href="https://twitter.com/joshcstory/" />
      <meta name="keywords" content={post.keywords} />
      <p>
        Eee equals em-see-squared...
      </p>
    </article>
  );
}
```



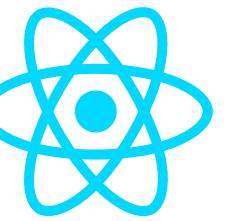
# Stylesheets



# Support for Stylesheets

```
function ComponentOne() {
  return (
    <Suspense fallback="loading...">
      <link rel="stylesheet" href="foo" precedence="default" />
      <link rel="stylesheet" href="bar" precedence="high" />
      <article class="foo-class bar-class">
        {...}
      </article>
    </Suspense>
  )
}

function ComponentTwo() {
  return (
    <div>
      <p>...</p>
      <link rel="stylesheet" href="baz" precedence="default" /> <-- will be inserted between foo & bar
    </div>
  )
}
```



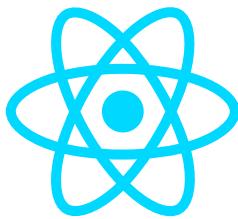
<script/>



# Support for async scripts

```
function MyComponent() {
  return (
    <div>
      <script async={true} src="..." />
      Hello World
    </div>
  )
}

function App() {
  <html>
    <body>
      <MyComponent>
      ...
      <MyComponent> // won't lead to duplicate script in the DOM
    </body>
  </html>
}
```



# Shhh!

**It's a Suspense**



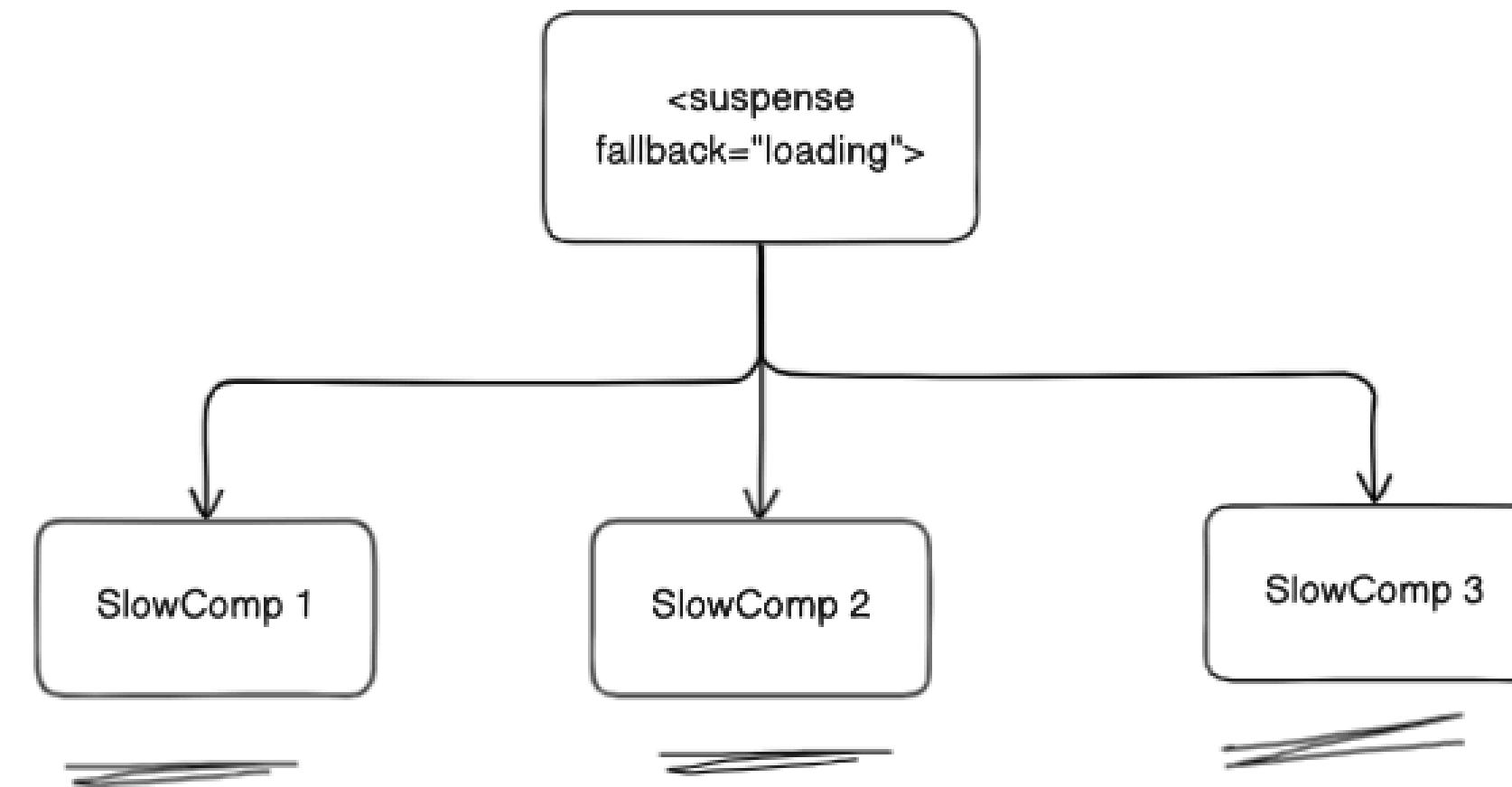
# Snippet

```
1 import React, { Suspense } from "react";
2 import { use } from "react";
3
4 const delayedPromise = (sec) => {
5   return new Promise((resolve) => setTimeout(resolve, sec));
6 };
7
8 const SlowComponent = (props) => {
9   use(delayedPromise(props.time));
10  return <div>SlowComponent {props.time}</div>;
11};
12
13 const SuspenseDemo = () => {
14  return (
15    <Suspense fallback=<div>Loading...</div>>
16      <SlowComponent time={1000} />
17      <SlowComponent time={2000} />
18      <SlowComponent time={3000} />
19    </Suspense>
20  );
21};
22
23 export default SuspenseDemo;
```



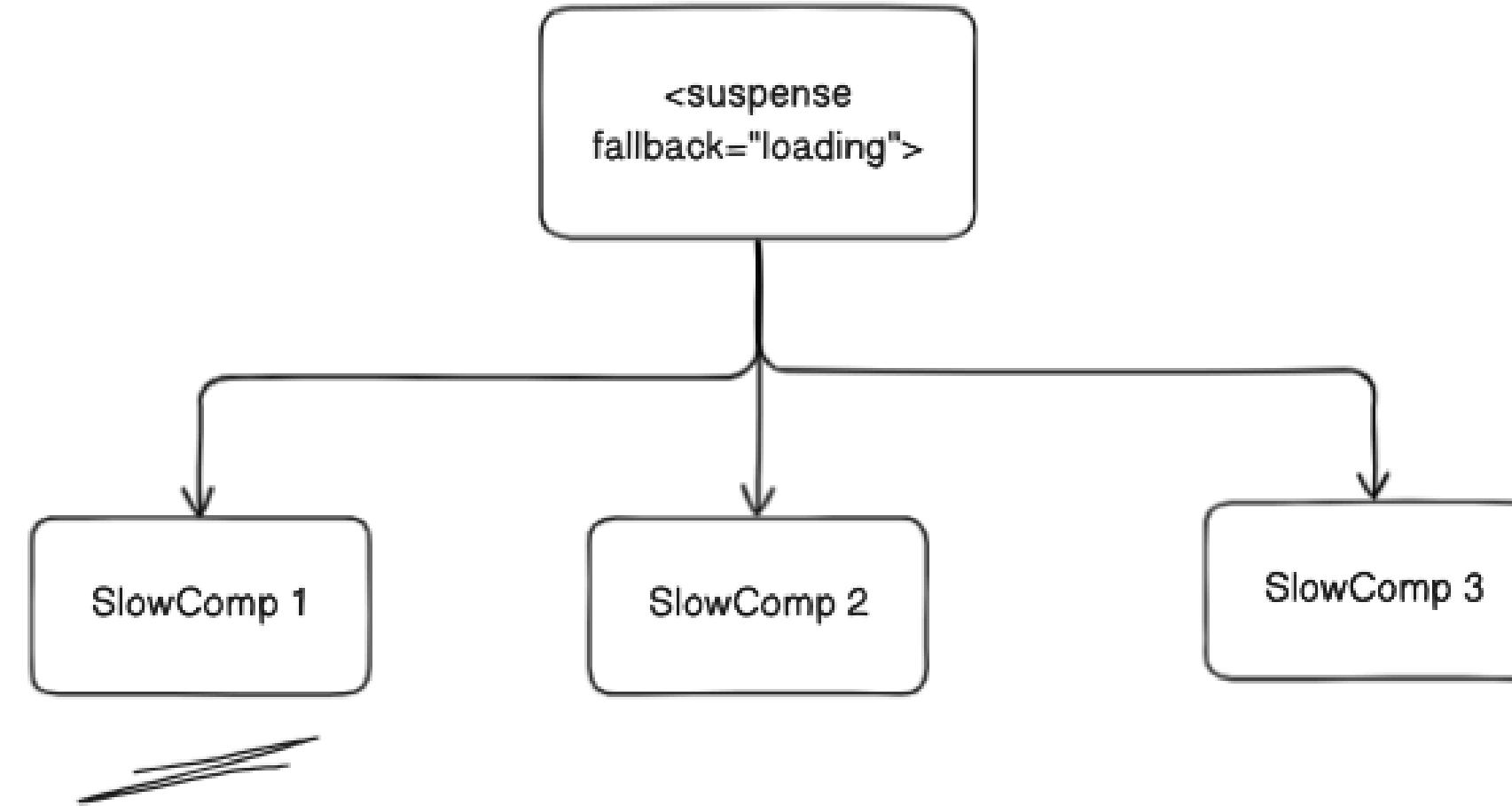
Editor

## Existing Suspense



when a component is suspended, the suspended siblings were rendered and then the fallback was committed.

# v19 Suspense



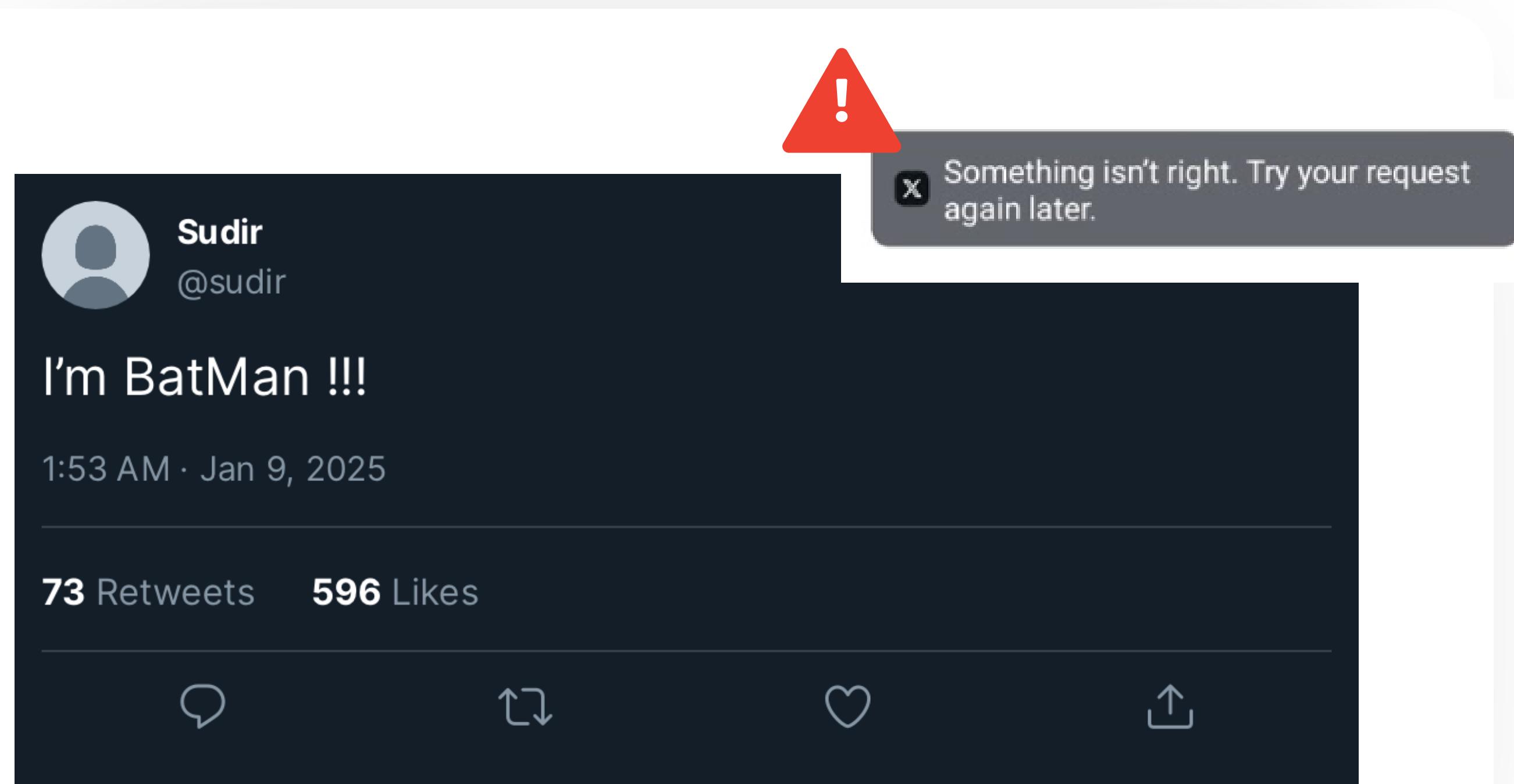
In React 19, when a component suspends,  
the fallback is committed and then the suspended siblings are rendered.





# Editing a Tweet







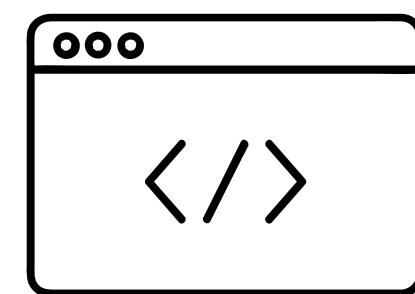
**Hooks**



# useOptmistic()



# Code



New API

use()





# Usage

**use API**

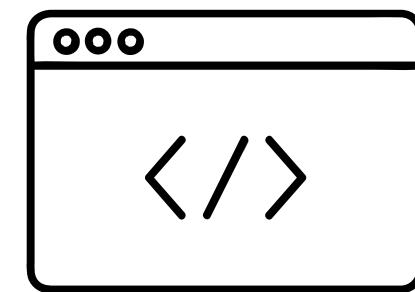
1

Promise Handling

2

Context Consumption

# Code



# Benefits use API



1

Less Declarative  
Code

2

Automatic Suspense  
Integration

3

Cleaner  
Components



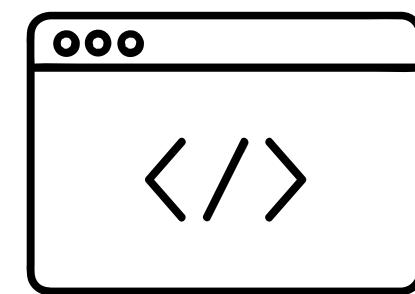
**Hooks**

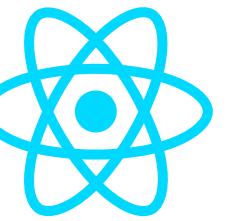


# Context API



# Code

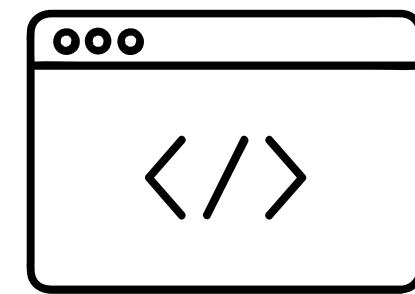




# Refs



# Code

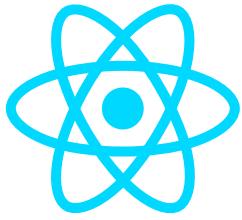


# Cleanup functions for refs

```
1 <input  
2   ref={(ref) => {  
3     // ref created  
4  
5     // cleanup function  
6     return () => {  
7       // ref cleanup  
8     };  
9   }}  
10  />
```



Well There's  
more



# Forms Actions



Form actions introduce a streamlined way to handle form submissions, making it easier to manage operations directly within the form components. This feature allows to specify functions that can be executed when a form is submitted, **significantly simplifying the process of connecting forms to server-side logic.**

### **Key Features of Form Actions:**

- **Direct Function Specification:** Developers can pass a function directly as the **action prop** of a `<form>`, `<input>`, or `<button>`. This function can handle the submission logic without needing separate API endpoints or manual state management.
- **Automatic Form Reset:** When a form action succeeds, React automatically resets the form for uncontrolled components, enhancing user experience by clearing inputs after submission.
- **Integration with New Hooks:** The `useActionState` hook can be used alongside form actions to manage form state and submission status seamlessly. This hook provides the current state, a function to trigger the action, and a boolean indicating if the submission is pending.
- **Server-Side Processing:** Actions can include server-side processing directly within the specified function using "**use server**" syntax, allowing for immediate handling of form data without additional setup.



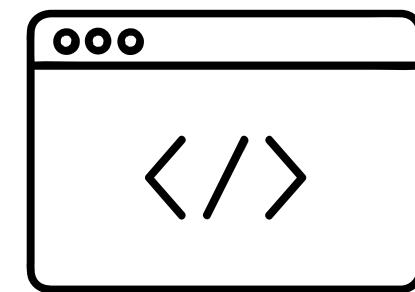
**Hooks**



# useActionState()



# Code





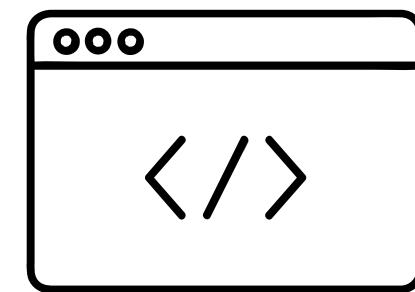
**Hooks**



# useFormStatus()



# Code





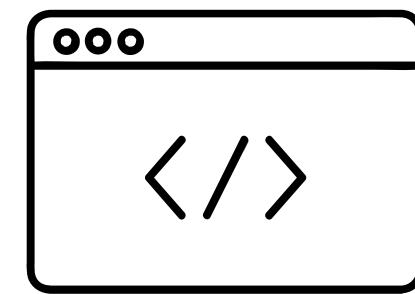
**Hooks**

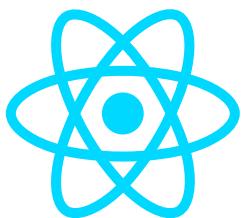


# useTransition()



# Code





# Server Actions



Server Actions in React 19 are a powerful feature that allows developers to define asynchronous functions that **run on the server side and can be invoked from client components**. This capability simplifies the interaction between client and server, **enabling direct access to server-side resources like databases without the need for separate API endpoints**.

### **Key Features of Server Actions:**

**Asynchronous Execution:** Server Actions must be defined as async functions, allowing them to perform operations such as database queries or file system access.

**Use of "use server" Directive:** To designate a function as a Server Action, you need to include the "use server" directive at the top of the function. This indicates that the function will run on the server.

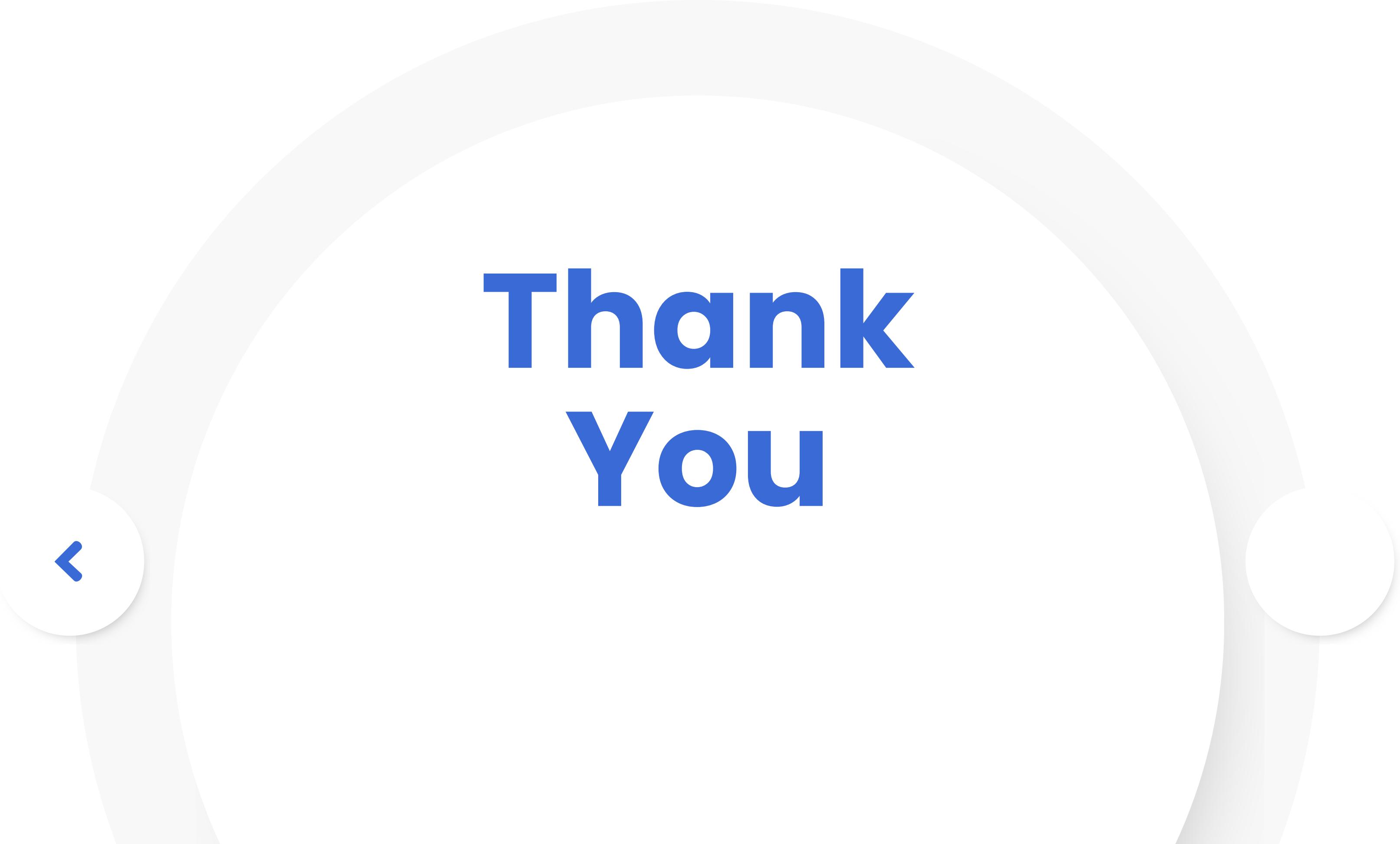
**Integration with Client Components:** Server Actions can be invoked from client components through forms or event handlers, making it easy to submit data back to the server.

**Progressive Enhancement:** Server Actions support progressive enhancement, meaning forms will still work even if JavaScript is not fully loaded or enabled.

**Automatic Handling of Form Submissions:** When a form is submitted, React can handle it without refreshing the page, providing a smoother user experience.

# Snippet

```
● ● ●  
1 // actions.js (Server Action file)  
2 export async function createUser(data) {  
3     "use server"; // Designate this function as a Server Action  
4     const { name, email } = data;  
5  
6     // Simulate database operation  
7     await db.users.create({ name, email });  
8  
9     return { success: true };  
10 }  
11  
12 // UserForm.js (Client Component)  
13 import { useState } from "react";  
14 import { createUser } from './actions';  
15  
16 export function UserForm() {  
17     const [state, submitAction, isPending] = useState(createUser, { success: false });  
18  
19     return (  
20         <form action={submitAction}>  
21             <input type="text" name="name" placeholder="Name" required />  
22             <input type="email" name="email" placeholder="Email" required />  
23             <button type="submit" disabled={isPending}>Create User</button>  
24             {state.success && <p>User created successfully!</p>}  
25         </form>  
26     );  
27 }  
28
```



**Thank  
You**



