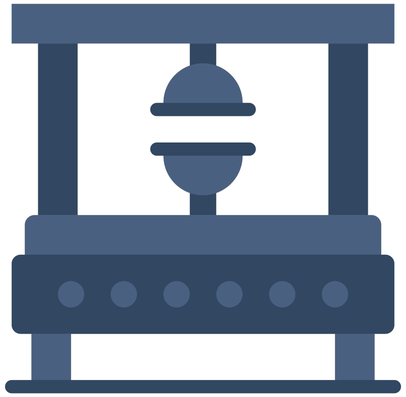
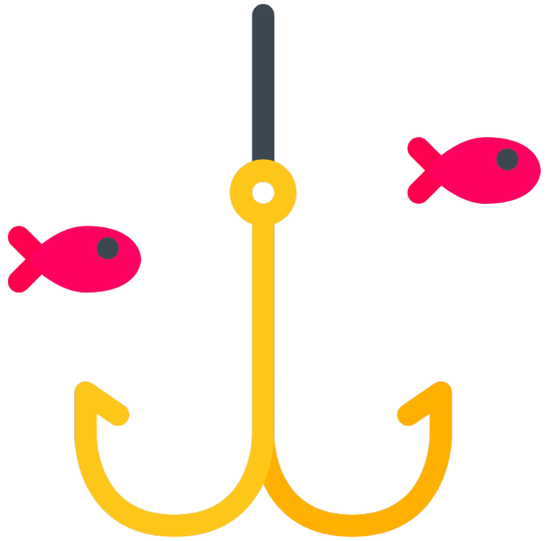
Actions

Directives

Compiler



New Hooks



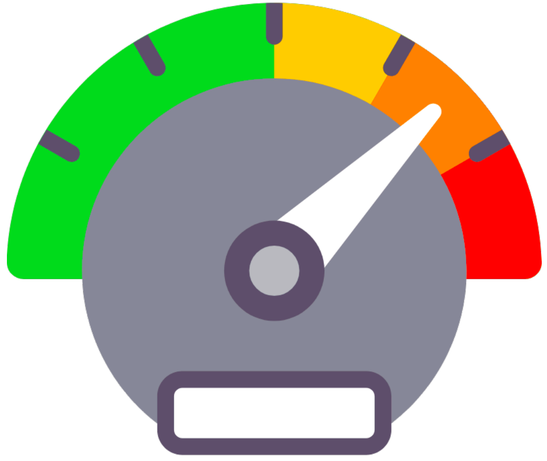
React 19

use() hook

Less Code

Optimistic

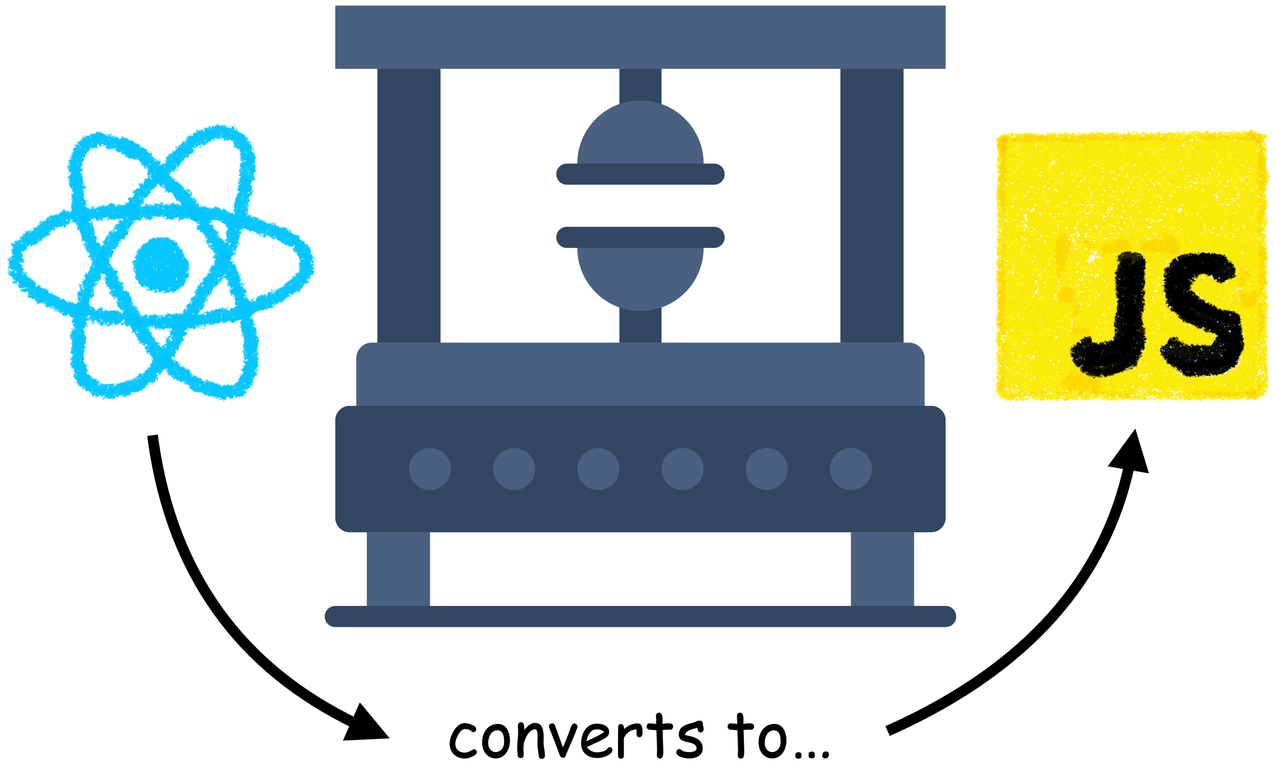
Better Perf



Install React 19

```
npm i react@canary react-dom@canary
```

React Compiler



- compiler increases performance
- removes need to write certain hooks
- reduces code you need to write

No More Memoization

new React compiler
removes the need to use:

~~useCallback()~~

~~useMemo()~~

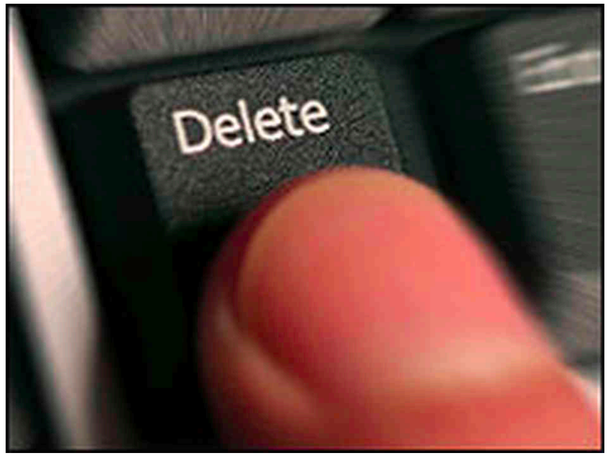
~~memo()~~

```
const [count, setCount] = useState(0)

const increment = () => setCount((c) => c + 1)

const doubleCount = count * 2

return (
  <>
    <div>Count: {count}</div>
    <div>Double Count: {doubleCount}</div>
    <Button increment={increment} />
  </>
)
```

ReactBootcamp.dev

No More forwardRef

```
import { forwardRef } from 'react'

const MyInput = forwardRef(function MyInput(props, ref) {
  // ...
});
```



Without forwardRef

```
function App() {
  const buttonRef = useRef()

  const onButtonClick = () => console.log(buttonRef.current)

  return (
    <Button ref={buttonRef} onClick={onButtonClick}>
      Click Me
    </Button>
  )
}
```

✓ access ref like any other prop

```
const Button = ({ ref, ...props }) => {
  return <button ref={ref} {...props} />
}
```

ReactBootcamp.dev

NEW!

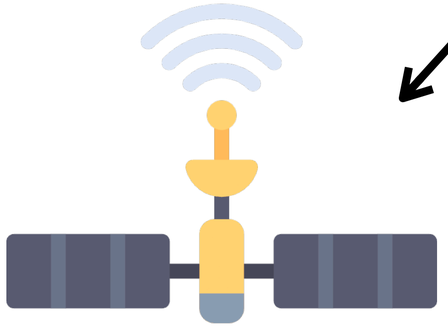
use() hook

lets you load resources asynchronously

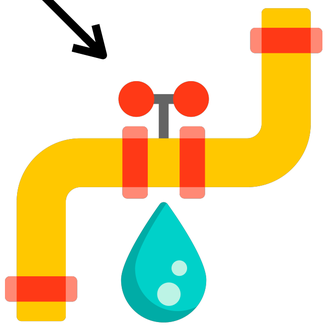
- promises
- context

multi-purpose hook

use() replaces

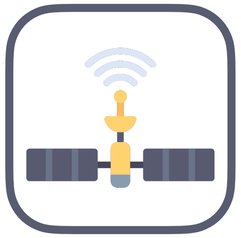


useEffect()
for data fetching



useContext()
for reading context

ReactBootcamp.dev



Fetch data - use()

```
function Person() {
  const person = use(fetchPerson())
  return <h1>{person.name}</h1>
}

function App() {
  return (
    <Suspense fallback={<h1>Loading...</h1>}>
      <Person />
    </Suspense>
  )
}

async function fetchPerson() {
  const response = await fetch("https://swapi.dev/api/people/1")
  return response.json()
}
```

1. resolve with use()

3. display data in UI

2. show fallback

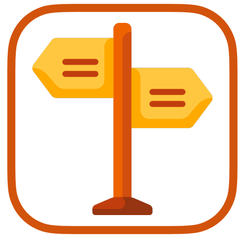
ReactBootcamp.dev

Read context - use()

```
const UserContext = createContext("")

function User() {
  const user = use(UserContext)
  return <h1>Hello, {user}!</h1>
}

function App() {
  return (
    <UserContext.Provider value="Dave">
      <User />
    </UserContext.Provider>
  )
}
```

✓ replace useContext with use()

ReactBootcamp.dev

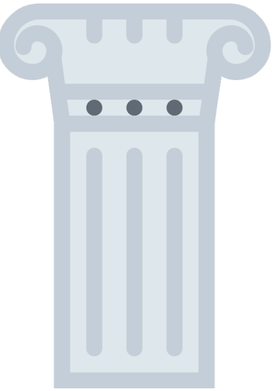


Directives

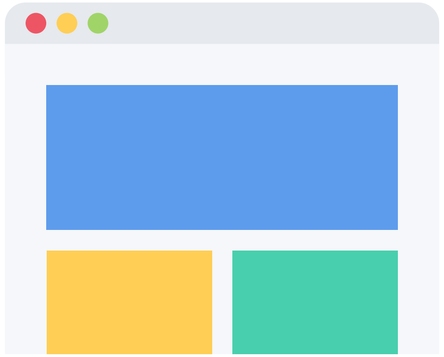
commands to run React code on server or client

add to top of component

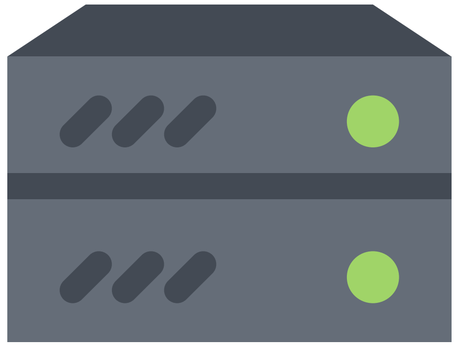
used already in Next.js



big, but simple



"use client"



"use server"

ReactBootcamp.dev

Actions

make forms much easier

function connected to form

```
function myAction(formData) {}

<form action={formAction}>
```

NEW Can be run on server **or** client

ReactBootcamp.dev

Client Actions

```
"use client"

function App() {
  function formAction(formData) {
    alert("You typed: " + formData.get("name"))
  }

  return (
    <form action={formAction}>
      <input type="text" name="name" />
      <button type="submit">Submit</button>
    </form>
  )
}
```

1. mark as client component

3. get input value from formData

2. connect function to action prop

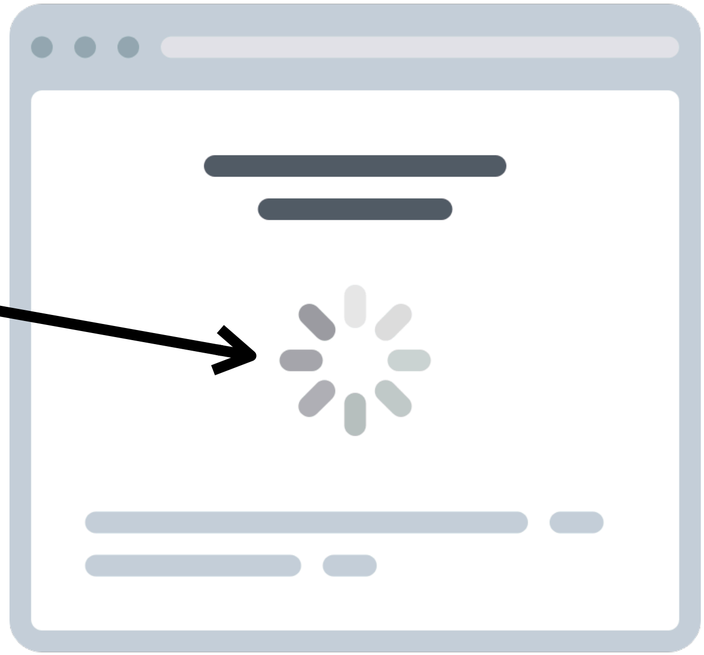
ReactBootcamp.dev

useFormStatus()

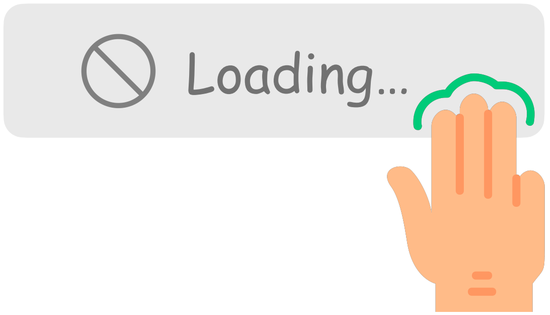
it will usually take time
for actions to complete

```
import { useFormStatus } from "react-dom"
```

submission is pending



disable submit
while pending



ReactBootcamp.dev

useFormStatus()

```
function Submit() {
  const { pending } = useFormStatus()

  return <button disabled={pending}>Submit</button>
}

function App() {
  async function formAction(formData) {/*...*/}

  return (
    <form action={formAction}>
      <input name="name" />
      <Submit />
    </form>
  )
}
```

2. get pending from hook

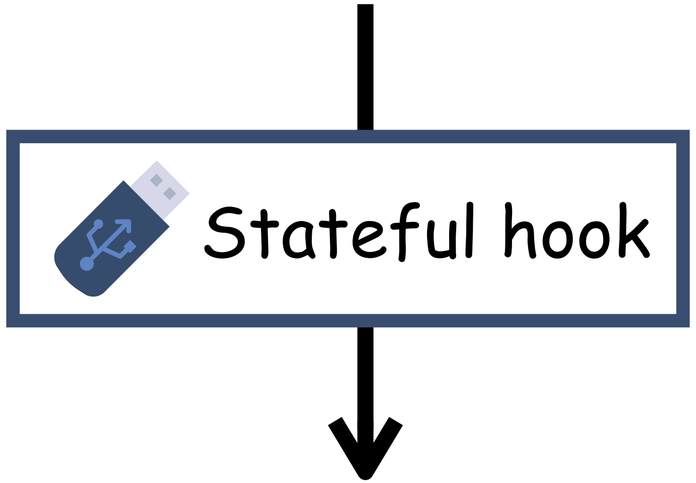
3. pass pending to disabled prop

1. nest component inside form

ReactBootcamp.dev

useFormState()

What if you want the data returned from your action?



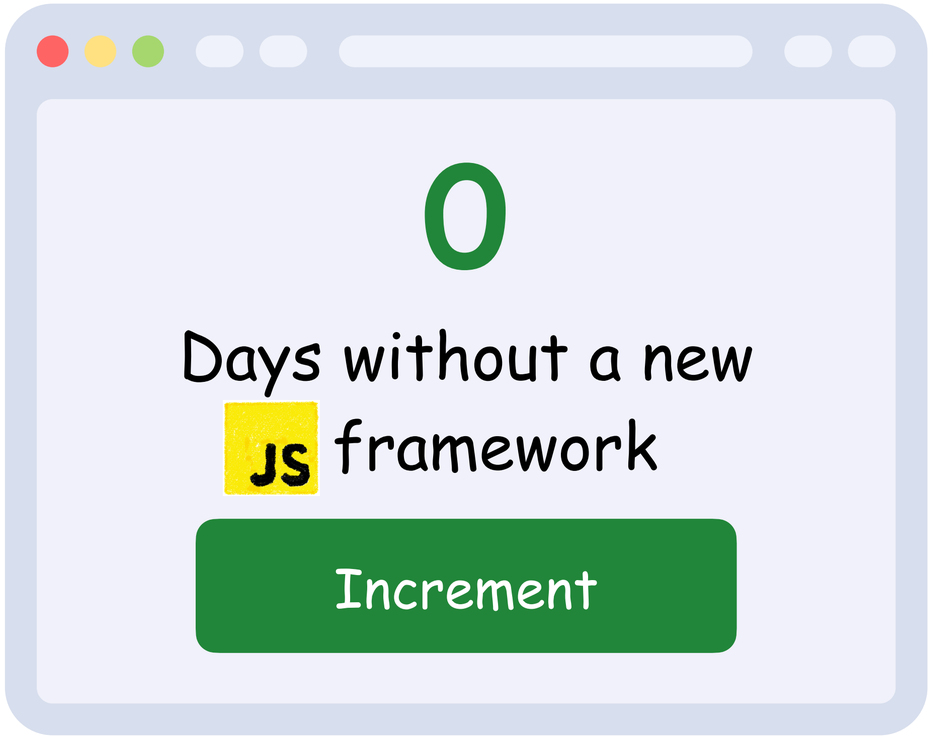

```
import { useFormState } from "react-dom"
```

ReactBootcamp.dev

Basic - useFormState()

```
async function increment(previousState, formData) {
  return previousState + 1
}

function StatefulForm() {
  const [state, formAction] = useFormState(increment, 0)

  return (
    <form>
      {state}
      <button formAction={formAction}>Increment</button>
    </form>
  )
}
```

Preview



ReactBootcamp.dev

Basic - useFormState()

```
async function increment(previousState, formData) {
  return previousState + 1
}

function StatefulForm() {
  const [state, formAction] = useFormState(increment, 0)

  return (
    <form>
      {state}
      <button formAction={formAction}>Increment</button>
    </form>
  )
}
```

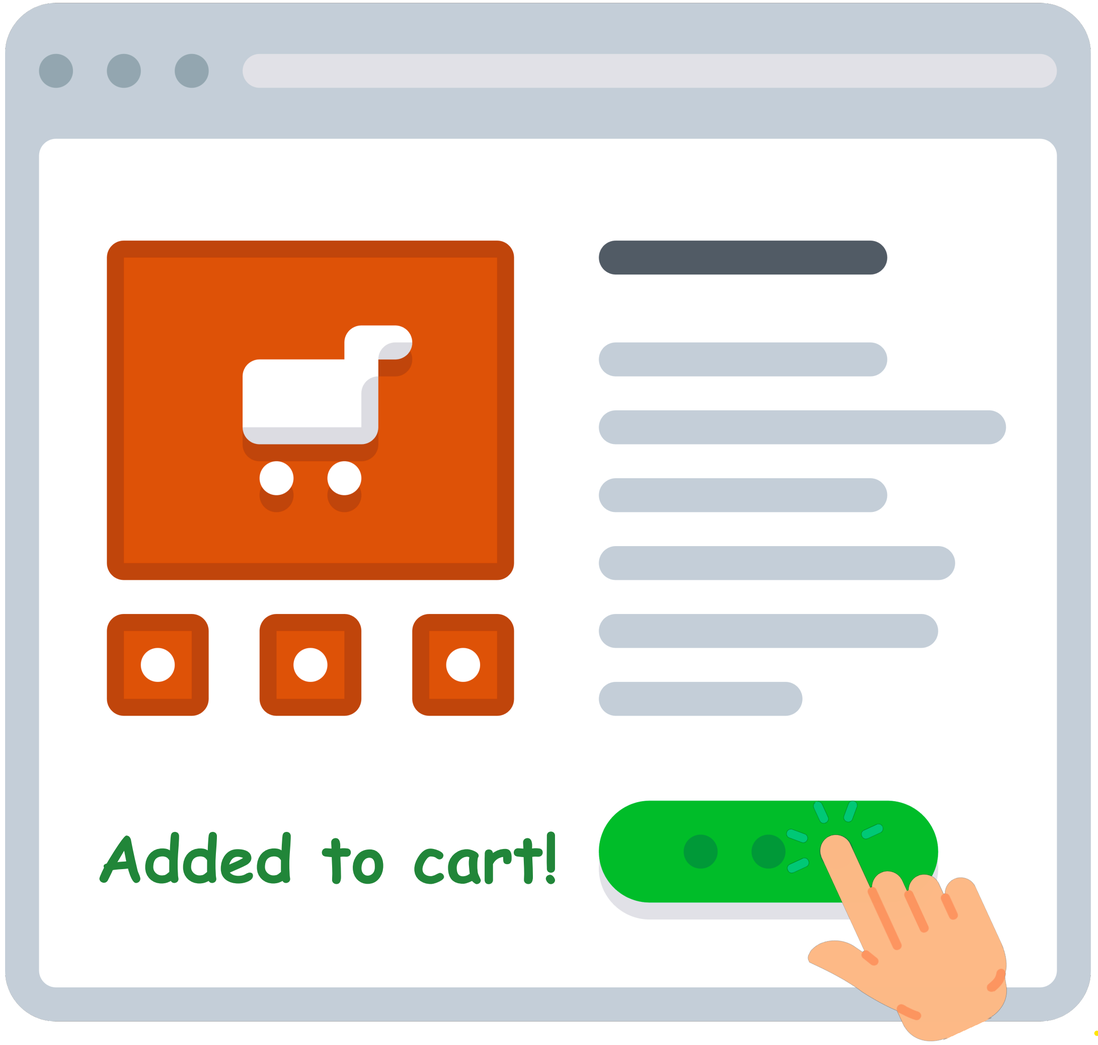
2. it gets prev state + formData

1. hook takes action + initial value

3. use returned state from action

ReactBootcamp.dev

Cart - useFormState()



ReactBootcamp.dev

Cart - useFormState()

```
function addToCart(prevState, formData) {
  const productId = formData.get("productId")
  return productId === "1"  ?  "Added to cart!" :  "Out of stock"
}

function App() {
  const [message, formAction] = useFormState(addToCart, null)

  return (
    <form action={formAction}>
      <h2>My Product</h2>
      <input type="hidden" name="productId" value="1" />
      <button>Submit</button>
      {message && <p>{message}</p>}
    </form>
  )
}
```

1. call addToCart action

ReactBootcamp.dev

Cart - useFormState()

```
function addToCart(prevState, formData) {
  const productId = formData.get("productId")
  return productId === "1"  ?  "Added to cart!" :  "Out of stock"
}

function App() {
  const [message, formAction] = useFormState(addToCart, null)

  return (
    <form action={formAction}>
      <h2>My Product</h2>
      <input type="hidden" name="productId" value="1" />
      <button>Submit</button>
      {message && <p>{message}</p>}
    </form>
  )
}
```

3. return message

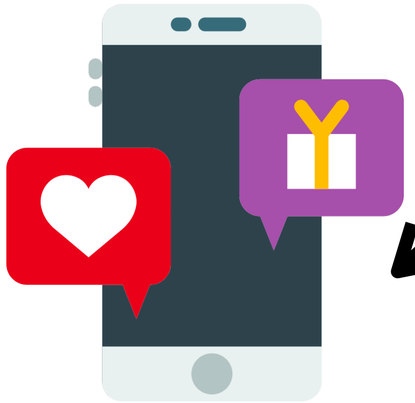
1. call addToCart action

2. pass productId ("1") via input

4. display in UI

ReactBootcamp.dev

useOptimistic()



Ideal for
real-time apps

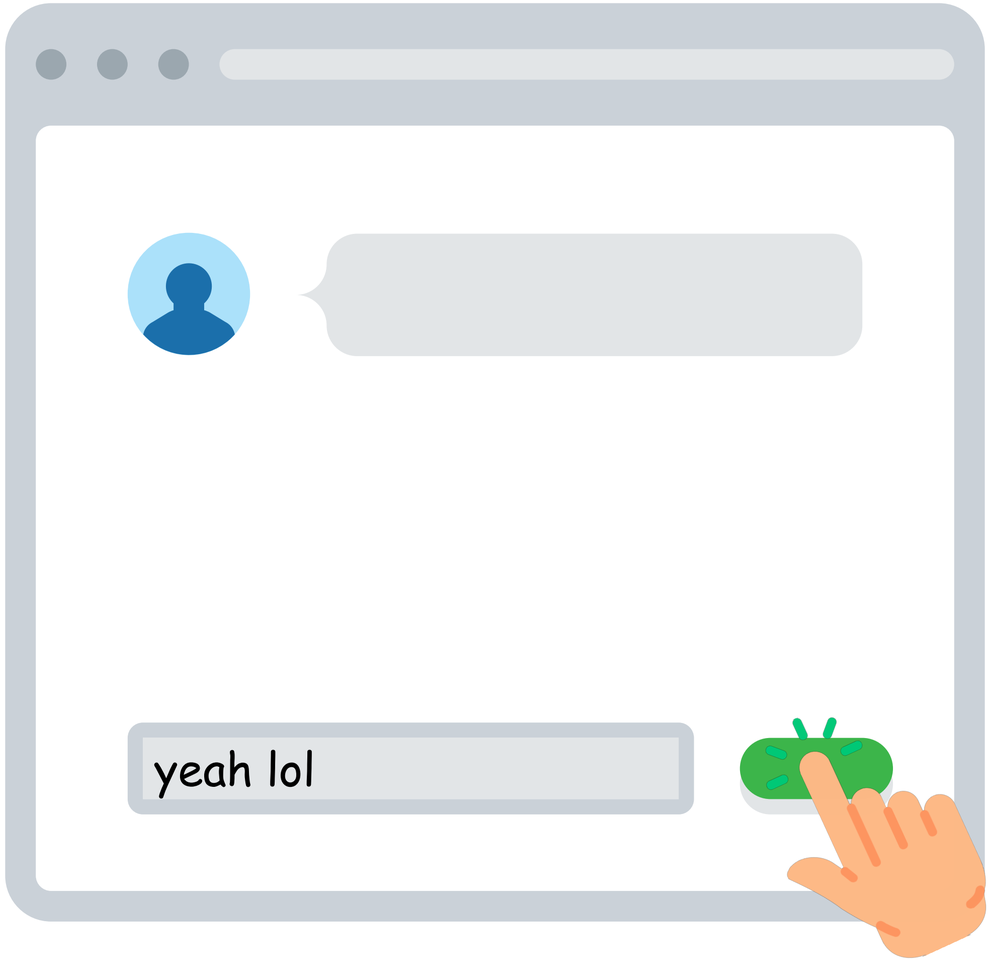
What to do while we're waiting for action to finish running?

optimistic update

```
const [optimisticState, addOptimistic] = useOptimistic(
   state,
   // updateFn
   (currentState, optimisticValue) => {
      // merge and return new state
      // with optimistic value
    }
)
```

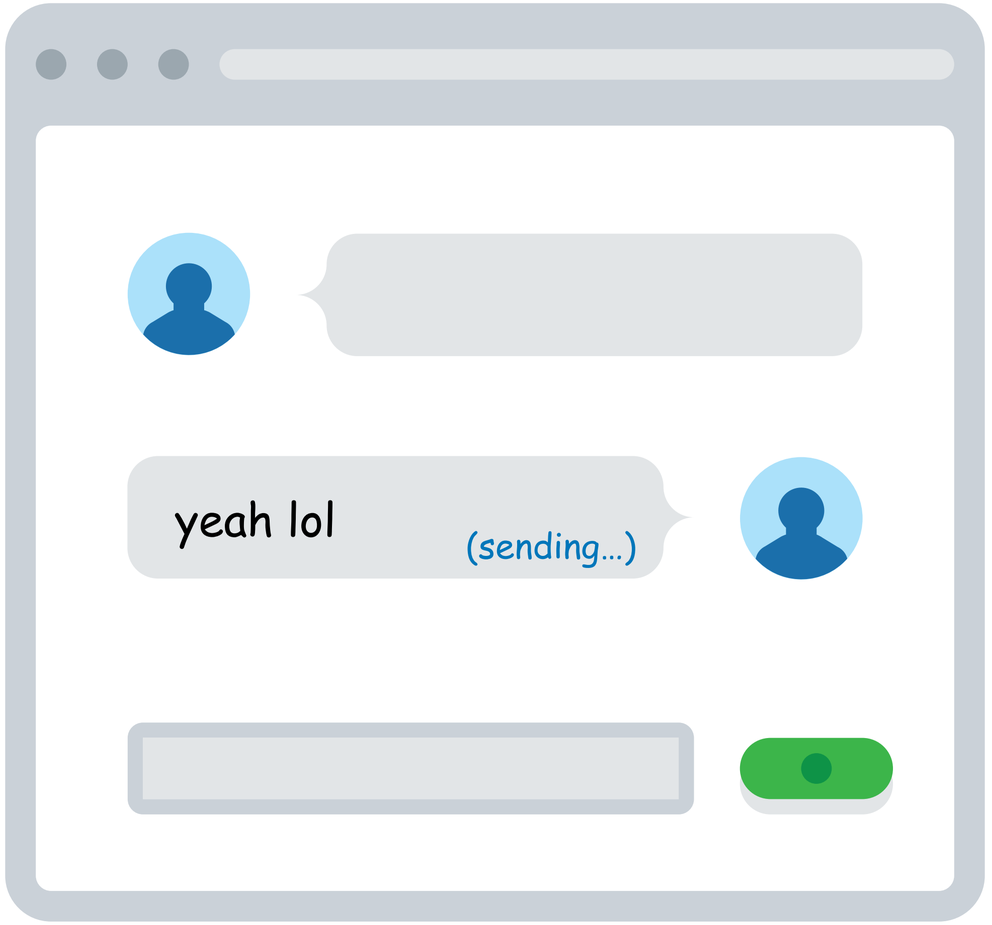
ReactBootcamp.dev

Chat - useOptimistic()



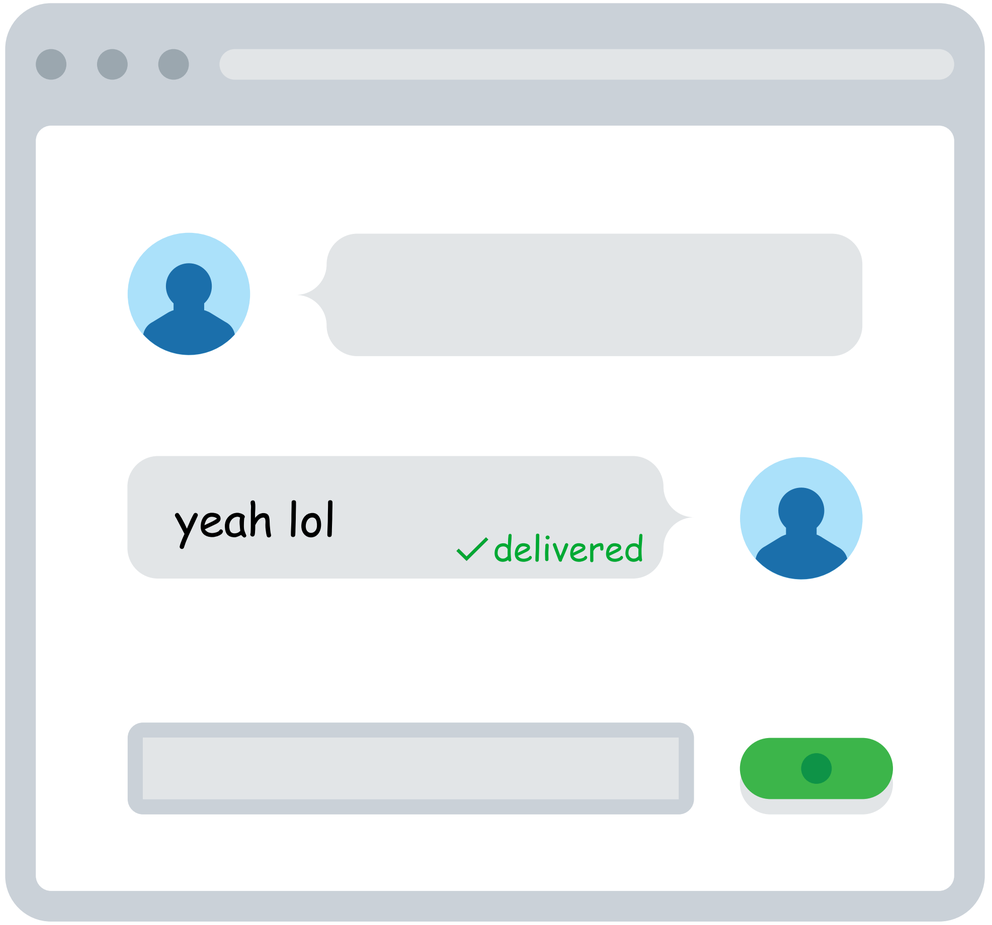
ReactBootcamp.dev

Chat - useOptimistic()



ReactBootcamp.dev

Chat - useOptimistic()



ReactBootcamp.dev

Chat - useOptimistic()

```
function ChatApp() {
  const [messages, setMessages] = useState([])

  const [optimisticMessages, addOptimisticMessage] = useOptimistic(
    messages,
    (state, newMessage) => [...state, { text: newMessage, sending: true }]
  );

  async function formAction(formData) {
    const message = formData.get("message");
    addOptimisticMessage(message);
    const createdMessage = await createMessage(message);
    setMessages((messages) => [...messages, { text: createdMessage }]);
  }
  //...
}
```

1. store messages in state
2. pass messages to hook
4. perform temp update
3. add optimistic message
5. replace w/ saved message

ReactBootcamp.dev