

PRAIRIE DEV CON

WEB | DEV | CLOUD | AI



EXCELLENCE IN RECRUITMENT



Hands-on with React 19

A roadmap to upgrading your applications

Jerecho Giba
Lead Developer
Consultant

Mechanical Keyboard Enthusiast

AGENDA

- 01 Why React?
- 02 How would it affect you?
- 03 What's new in React 19?
- 04 Upgrading to React 19
- 05 Q & A

Why React?

React Usage Statistics



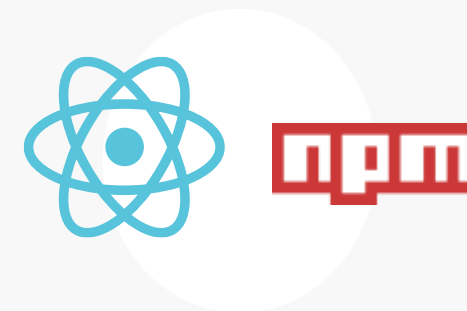
700,971

Websites in Canada



186,420 (18%)

Top 1 million sites by traffic



25,218,845

npm weekly downloads

Reference:

<https://trends.builtwith.com/javascript/React/Canada>

<https://trends.builtwith.com/websitelist/React/Top-Million-Sites-by-Traffic>

<https://www.npmjs.com/package/react>

Results. Guaranteed.

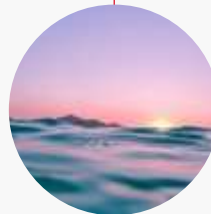
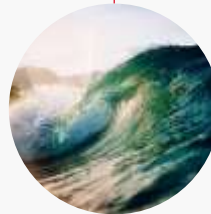


Initial Release
Date

May 29, 2013

February 2019

Hooks was introduced on
React 16.8



October 20, 2020

React 17 was released



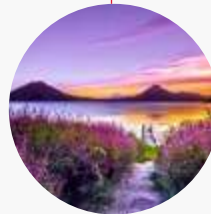
March 2022

React 18 was released



MAY 2023

Canary Release Channel rollout



April 25, 2024

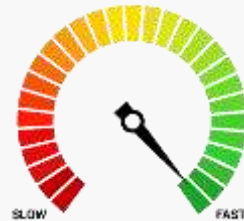
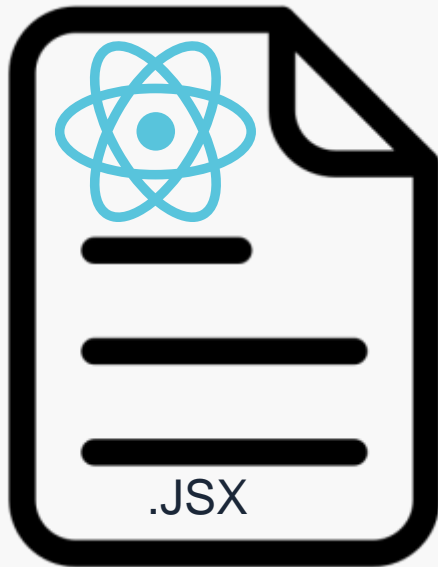
React 19 Beta! React 19 was upgraded from
Canary to RC (Release Candidate)

How would it affect me?

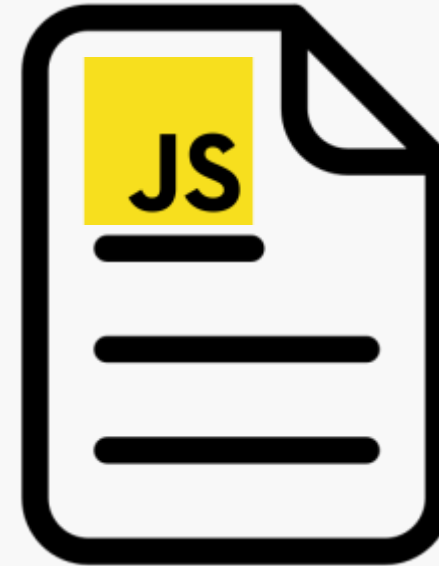
React Compiler

React Compiler

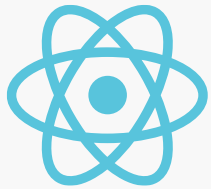
Speaker Notes: React Compiler is a build-time only tool that optimizes your React applications by automatically memoizing your code



Optimized! ✨



React Compiler



React Hooks and API for Optimization

`useCallback()`

cache a function definition

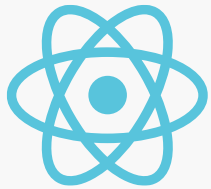
`useMemo()`

cache the result of a calculation

`memo()`

skips unnecessary rendering

React Compiler



React Hooks and API for Optimization

`useCallback()` Manual a function definition

NO MORE MEMOIZATION!

`memo()`

cache the result of a calculation

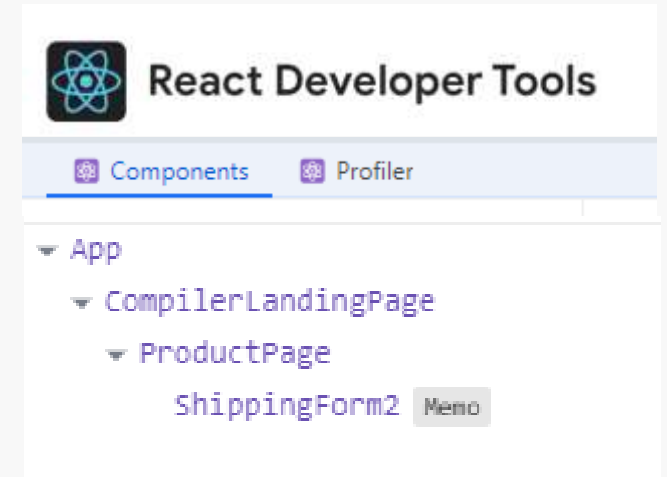
skips unnecessary rendering



With Memoization

```
function ProductPage({ productId, referrer, theme }) {  
  const handleSubmit = useCallback((orderDetails) => {  
    post('/product/' + productId + '/buy', {  
      referrer,  
      orderDetails,  
    });  
  }, [productId, referrer]);  
  
  return (  
    <div className={theme}>  
      <ShippingForm onSubmit={handleSubmit} />  
    </div>  
  );  
}
```

Reference: <https://react.dev/reference/react/useCallback>



memo component

```
const ShippingForm =  
memo(function ShippingForm({ onSubmit})
```

React 19 Compiler

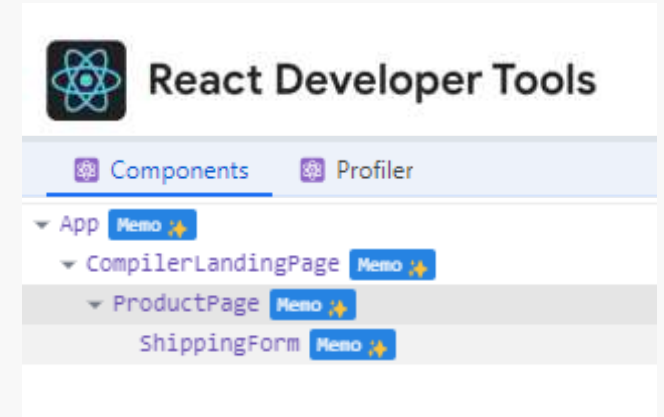
Without Memoization

```
function ProductPage({ productId, referrer, theme }) {  
  const handleSubmit = (orderDetails) => {  
    post('/product/' + productId + '/buy', {  
      referrer,  
      orderDetails,  
    });  
  };  
  
  return (  
    <div className={theme}>  
      <ShippingForm onSubmit={handleSubmit} />  
    </div>  
  );  
}
```

memoized ✨

memoized ✨

function ShippingForm({ onSubmit })



React Compiler

Playground

<https://playground.react.dev/>

Playground

<https://playground.react.dev/>

React Compiler Playground

```
1 export default function ProductPage({ productId, referrer, theme }:  
  ProductPageProps) {  
2   const handleSubmit = useCallback((orderDetails) => {  
3     post('/product/' + productId + '/buy', {  
4       referrer,  
5       orderDetails,  
6     });  
7   }, [productId, referrer]);  
8  
9   return (  
10    <div className={theme}>  
11      <ShippingForm onSubmit={handleSubmit} />  
12    </div>  
13  );  
14 }
```

with memoization

-JS

```
function ProductPage(t0) {  
  const $ = _c(8);  
  const { productId, referrer, theme } = t0;  
  let t1;  
  if ($[0] !== productId || $[1] !== referrer) {  
    t1 = (orderDetails) => {  
      post("/product/" + productId + "/buy", {  
        referrer,  
        orderDetails,  
      });  
    };  
    $[0] = productId;  
    $[1] = referrer;  
    $[2] = t1;  
  } else {  
    t1 = $[2];  
  }  
  const handleSubmit = t1;  
  let t2;  
  if ($[3] !== handleSubmit) {  
    t2 = <ShippingForm onSubmit={handleSubmit} />;  
    $[3] = handleSubmit;  
    $[4] = t2;  
  } else {  
    t2 = $[4];  
  }  
  let t3;  
  if ($[5] !== theme || $[6] !== t2) {  
    t3 = <div className={theme}>{t2}</div>;  
    $[5] = theme;  
    $[6] = t2;  
    $[7] = t3;  
  } else {  
    t3 = $[7];  
  }  
  return t3;  
}
```


Playground

<https://playground.react.dev/>

React Compiler Playground

```
1 export default function ProductPage({ productId, referrer, theme }:  
  ProductPageProps) {  
2   const handleSubmit = (orderDetails) => {  
3     post('/product/' + productId + '/buy', {  
4       referrer,  
5       orderDetails,  
6     });  
7   };  
8  
9   return (  
10    <div className={theme}>  
11      <ShippingForm onSubmit={handleSubmit} />  
12    </div>  
13  );  
14 }
```

without memoization

- JS

```
function ProductPage(t0) {  
  const $ = _c(8);  
  const { productId, referrer, theme } = t0;  
  let t1;  
  if ($[0] !== productId || $[1] !== referrer) {  
    t1 = (orderDetails) => {  
      post("/product/" + productId + "/buy", {  
        referrer,  
        orderDetails,  
      });  
    };  
    $[0] = productId;  
    $[1] = referrer;  
    $[2] = t1;  
  } else {  
    t1 = $[2];  
  }  
  const handleSubmit = t1;  
  let t2;  
  if ($[3] !== handleSubmit) {  
    t2 = <ShippingForm onSubmit={handleSubmit} />;  
    $[3] = handleSubmit;  
    $[4] = t2;  
  } else {  
    t2 = $[4];  
  }  
  let t3;  
  if ($[5] !== theme || $[6] !== t2) {  
    t3 = <div className={theme}>{t2}</div>;  
    $[5] = theme;  
    $[6] = t2;  
    $[7] = t3;  
  } else {  
    t3 = $[7];  
  }  
  return t3;  
}
```

Speaker notes 1a:
orderDetails is **not** checked here because will not cause a re-render of the Product Page

1b: but is part of the check on the handleSubmit function here since it can cause a re-render on the Shipping Form

React Compiler

How to use?



Speaker Notes: React Compiler is still experimental, rolling out the compiler to production for your app will depend on the health of your codebase and how well you've followed the [Rules of React](#).

How to use?

1 – Check for Compatibility (Rules of React)

```
npx react-compiler-healthcheck@experimental
```

```
Successfully compiled 4 out of 4 components.  
StrictMode usage found.  
Found no usage of incompatible libraries.
```

2 – Install the eslint plugin

```
npx install eslint-plugin-react-compiler@experimental
```

Speaker Notes: Use these libraries to determine if your application adheres to the Rules of React.

3 – Install the react compiler plugin

```
npx install babel-plugin-react-compiler@experimental
```

How to use?

New Project

```
const ReactCompilerConfig = { };

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react({
    babel: {
      plugins: [
        ["babel-plugin-react-compiler",
ReactCompilerConfig],
      ],
    },
  })],
})
```

Existing Project


```
const ReactCompilerConfig = {sources: (filename) =>
{
  return filename.indexOf('src/path/to/dir') !== -1;
}, };

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react({
    babel: {
      plugins: [
        ["babel-plugin-react-compiler",
ReactCompilerConfig],
      ],
    },
  })],
})
```

Ref as a prop

With forwardRef

```
const MyInput = forwardRef(function MyInput(props, ref) {  
  return <input placeholder={props.placeholder} ref={ref} />  
})  
  
function MyForm() {  
  const ref = useRef(null);  
  const handleClick = () => ref.current.focus();  
  return (  
    <form>  
      <MyInput placeholder="Enter your name" ref={ref} />  
      <button type="button" onClick={handleClick}>  
        Edit  
      </button>  
    </form>  
  );  
}
```



Use `forwardRef` to let your component receive a `ref`


Create a ref with `useRef`

Pass `ref` as a prop

With forwardRef

```
const MyInput = forwardRef(function MyInput(props, ref) {  
  return <input placeholder={props.placeholder} ref={ref} />  
})  
  
function MyForm() {  
  const ref = useRef(null);  
  const handleClick = () => ref.current.focus();  
  return (  
    <form>  
      <MyInput placeholder="Enter your name" ref={ref} />  
      <button type="button" onClick={handleClick}>  
        Edit  
      </button>  
    </form>  
  );  
}
```

React 19 ✨ Without forwardRef



```
function MyInput({ ref, ...props }) {  
  return <input placeholder={props.placeholder} ref={ref} />  
}  
  
function MyForm() {  
  const ref = useRef(null);  
  const handleClick = () => ref.current.focus();  
  return (  
    <form>  
      <MyInput placeholder="Enter your name" ref={ref} />  
      <button type="button" onClick={handleClick}>  
        Edit  
      </button>  
    </form>  
  );  
}
```

Access `ref` as a prop

React 19 ✨ Without forwardRef

```
function MyInput(props) {  
  return <input placeholder={props.placeholder} ref={props.ref} />  
}  
  
function MyForm() {  
  const ref = useRef(null);  
  const handleClick = () => ref.current.focus();  
  return (  
    <form>  
      <MyInput placeholder="Enter your name" ref={ref} />  
      <button type="button" onClick={handleClick}>  
        Edit  
      </button>  
    </form>  
  );  
}
```

Context as a provider

Context as a provider

<Context.Provider>

```
const ThemeContext = createContext('');

function App({children}) {
  return (
    <ThemeContext.Provider value="dark">
      {children}
    </ThemeContext.Provider>
  );
}
```

Context as a provider

<Context>

```
const ThemeContext = createContext('');

function App({children}) {
  return (
    <ThemeContext value="dark">
      {children}
    </ThemeContext>
  );
}
```

New API: use

New API: use

`use(promise)`

`use(context)`

New API: use

use(promise)

```
function Comments({commentsPromise}) {  
  // `use` will suspend until the promise resolves.  
  const comments = use(commentsPromise);  
  return comments.map(comment =>  
    <p key={comment.id}>{comment}</p>);  
}  
  
function Page({commentsPromise}) {  
  // When `use` suspends in Comments,  
  // this Suspense boundary will be shown.  
  return (  
    <Suspense fallback={<div>Loading...</div>}>  
      <Comments commentsPromise={commentsPromise} />  
    </Suspense>  
  )  
}
```

New API: use

use(context)

```
function Comments({commentsPromise}) {  
  // `use` will suspend until the promise resolves.  
  const comments = use(commentsPromise);  
  const theme = use(ThemeContext);  
  return comments.map(comment =>  
    <p key={comment.id} style={{color: theme.color}}>{comment}</p>);  
}  
  
function Page({commentsPromise}) {  
  // When `use` suspends in Comments,  
  // this Suspense boundary will be shown.  
  return (  
    <Suspense fallback={<div>Loading...</div>}>  
      <Comments commentsPromise={commentsPromise} />  
    </Suspense>  
  )  
}
```


What's New?

Actions

React Action is a pattern for asynchronous data updates in response to user input

Pre-React 19

```
const [name, setName] = useState("");
const [isPending, setIsPending] = useState(false);
const [error, setError] = useState("");
```

```
const handleUpdate = async () => {
  setIsPending(true);
  const response = await updateName(name);
  setIsPending(false);
  //... setError(response.error);
};
```

```
return (
  <div>
    <input onChange={(e) => setName(e.target.value)} />
    <button onClick={handleUpdate}>
      {isPending ? "UPDATING.." : "UPDATE"}
    </button>
    {error && <span> error </span>}
  </div>
)
```

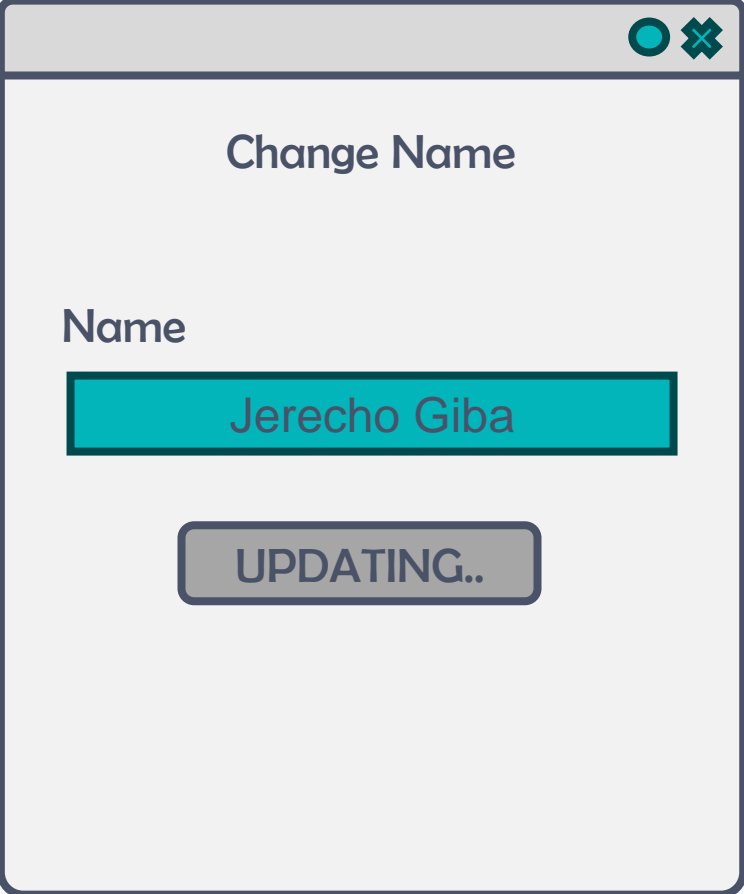


Pre-React 19

```
const [name, setName] = useState("");
const [isPending, setIsPending] = useState(false);
const [error, setError] = useState("");
```

```
const handleUpdate = async () => {
  setIsPending(true);
  const response = await updateName(name);
  setIsPending(false);
  //... setError(response.error);
};
```

```
return (
  <div>
    <input onChange={(e) => setName(e.target.value)} />
    <button onClick={handleUpdate}>
      {isPending ? "UPDATING.." : "UPDATE"}
    </button>
    {error && <span> error </span>}
  </div>
)
```



Change Name

Name

Jerecho Giba

UPDATING..



Pre-React 19

```
const [name, setName] = useState("");
const [isPending, setIsPending] = useState(false);
const [error, setError] = useState("");
```

```
const handleUpdate = async () => {
  setIsPending(true);
  const response = await updateName(name);
  setIsPending(false);
  //... setError(response.error);
};
```

```
return (
  <div>
    <input onChange={(e) => setName(e.target.value)} />
    <button onClick={handleUpdate}>
      {isPending ? "UPDATING.." : "UPDATE"}
    </button>
    {error && <span> error </span>}
  </div>
)
```



Actions

`useTransition()`

`useActionState()`

`useOptimistic()`

`<form action={...}>`

`useFormStatus()`

Speaker notes: is a Hook that lets you update the state without blocking the UI.

```
const [isPending, startTransition] = useTransition()
```

```
startTransition()
```

```
function MultiStepForm() {  
  const [currentStep, setCurrentStep] = useState('PERSONAL_INFO');  
  
  function moveNextStep(nextStep) {  
    startTransition(async () => {  
      //React 19 – Async support  
      await fn();  
      setCurrentStep(nextStep);  
    });  
  }  
  // ...  
}
```


Pre-React 19

```
const [name, setName] = useState("");  
const [isPending, setIsPending] = useState(false);  
const [error, setError] = useState("");
```

```
const handleUpdate = async () => {  
  setIsPending(true);  
  const response = await updateName(name);  
  setIsPending(false);  
  //... setError(response.error);  
};
```

```
return (  
  <div>  
    <input onChange={(e) => setName(e.target.value)} />  
    <button onClick={handleUpdate}>  
      {isPending ? "UPDATING.." : "UPDATE"}  
    </button>  
    {error && <span> error </span>}  
  </div>  
)
```

Change Name

Name

Jerecho Giba

UPDATE

useTransition()

```
const [name, setName] = useState("");
const [isPending, startTransition] = useTransition();
const [error, setError] = useState("");
```

```
const handleUpdate = async () => {
  startTransition(async () => {
    const response = await updateName(name);
    setError(response.error);
  });
};
```

```
return (
  <div>
    <input onChange={(e) => setName(e.target.value)} />
    <button onClick={handleUpdate}>
      {isPending ? "UPDATING.." : "UPDATE"}
    </button>
    {error && <span> error </span>}
  </div>
)
```



Actions

`useTransition()`

`useActionState()`

`useOptimistic()`

`<form action={...}>`

`useFormStatus()`

Actions

Pre-React 19

`<form action={...}>`

```
//HTML Action  
<form action="/success_page">
```

```
//Event Handler  
<form onSubmit={handleSubmit}>
```

```
//Event Handler  
<form>  
  <button onClick={handleCustomSubmit}>  
    Submit  
  </button>  
</form>
```

Actions

```
<form action={...}>
```

```
const [isPending, setIsPending] = useState(false);
```

```
function submitAction(formData) => {  
  setIsPending(true);  
  const name = formData.get("name");  
  await updateName(name);  
  setIsPending(false);  
};
```

```
<form action={submitAction}>  
  <input name="name" />  
  <...>  
  <button type="submit"  
    disabled={isPending}>SUBMIT</button>  
</form>
```

Form

Name

Jerecho Giba

SUBMIT

Actions

```
<form action={...}>
```

```
const [isPending, setIsPending] = useState(false);
```

```
function saveAction(formData) => {  
  setIsPending(true);  
  const name = formData.get("name");  
  await updateName(name);  
  setIsPending(false);  
};
```

```
<form action={submitAction}>  
  <input name="name" />  
  <...>  
  <button formAction={saveAction}>Save</button>  
</form>
```

Form

Name

Jerecho Giba

Save SUBMIT

Actions

`useTransition()`

`useActionState()`

`useOptimistic()`

`<form action={...}>`

`useFormStatus()`

Speaker notes: is a Hook that gives you status information of the last form submission (isPending status, form data that was submitted)

useFormStatus()

```
function SubmitButton() {  
  const { pending } = useFormStatus();  
  return (  
    <button type="submit" disabled={pending}>  
      SUBMIT</button>  
  );  
}
```

```
<form action={submitAction}>  
  <input name="name" />  
  <...>  
  <SubmitButton />  
</form>
```

Form

Name

Jerecho

SUBMIT

useFormStatus()

```
function SubmitButton() {  
  const { pending, data } = useFormStatus();  
  return (  
    <div>  
      <p>{data ? `Updating name to  
        ${data?.get("name")}...` : ""}</p>  
      <button type="submit" disabled={pending}>  
        SUBMIT  
      </button>  
    </div>  
  );  
}
```

Form

Name

Jerecho

Updating name to Jerecho...

SUBMIT

Actions

`useTransition()`

`useActionState()`

`useOptimistic()`

`<form action={...}>`

`useFormStatus()`

Speaker notes: a Hook that allows you to update state based on the result of a form action

```
const [state, form action, isPending] = useActionState(function, initialState)
```

state returned by
the action

updater function

pending status

action
implementation

initial state value

accepts two
parameters.
(currentState,
formState)

Pre-React 19

```
const [name, setName] = useState("");
const [isPending, setIsPending] = useState(false);
const [error, setError] = useState("");

const handleSubmit = async (e) => {
  e.preventDefault();
  setIsPending(true);
  const errorRes = await updateName(name);
  setIsPending(false);
  if (errorRes) setError(errorRes);
};

return (
  <form onSubmit={handleSubmit}>
    <input value={name}
      onChange={(e) => setName(e.target.value)} />
    <...>
    <button type="submit"
      disabled={isPending}>SUBMIT</button>
  </form>
)
```

Form

Name

Jerecho Giba

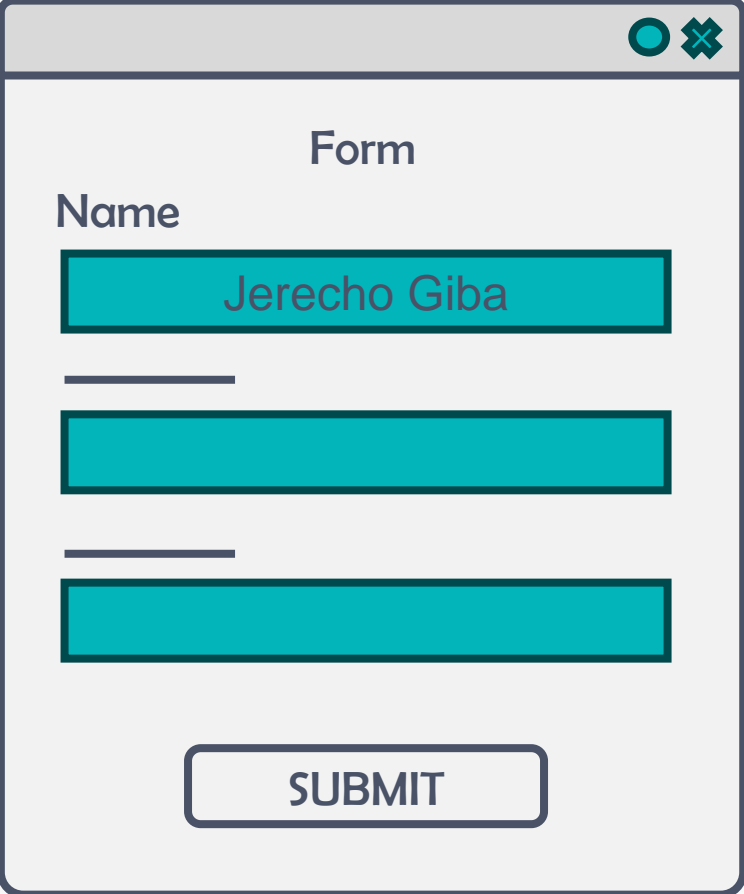
SUBMIT

Actions

useActionState()

```
const [error, submitAction, isPending] = useActionState(
  async (currentState, formData) => {
    const error = await updateName(formData.get("name"));
    if (error) return error; //return result of Action
    return null; //or handle success
  }, null);

return (
  <form action={submitAction}>
    <input name="name" />
    <...>
    <button type="submit"
      disabled={isPending}>SUBMIT</button>
  </form>
)
```



Form

Name

Jerecho Giba

SUBMIT

Actions

`useTransition()`

`useActionState()`

`useOptimistic()`

`<form action={...}>`

`useFormStatus()`

Actions

useOptimistic()

```
//Resulting optimistic state, dispatching function  
const [optimisticState, optimisticAction] = useOptimistic(  
  //Component State  
  state,  
  //Update Function  
  (currentState, optimisticValue) => {  
    //return merged current state and optimisticValue  
  }  
);
```

state returned by the
optimistic action

dispatch function

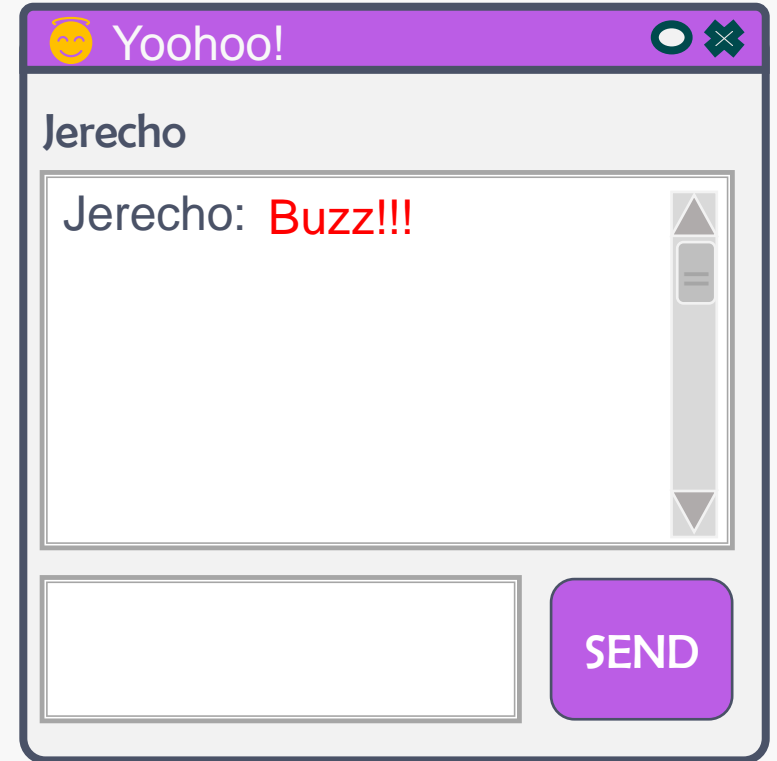
initial returned value when
no action is pending

a function that takes the
current state and the
optimistic value passed to
the **optimistic action**

useOptimistic()

```
function ChatWindow({messages, sendMessage}) {
  const [optimisticMsgs, addOptimisticMsg] = useOptimistic(
    messages,
    (currentState, newMessage) => {
      return [...currentState,
        { text: newMessage, sending: true } ]
    }
  );
  async function sendAction(formData) {
    addOptimisticMessage(formData.get("message"));
    await sendMessage(formData);
  }
  return (<>
    {optimisticMsgs.map((message, index) => (
      <div key={index}>
        {message.text}
        {!!message.sending && <small> (Sending...)</small>}
      </div>
    ))}

    <form action={sendAction}>
      <input type="text" name="message" />
      <button type="submit">Send</button>
    </form> </>)
}
```



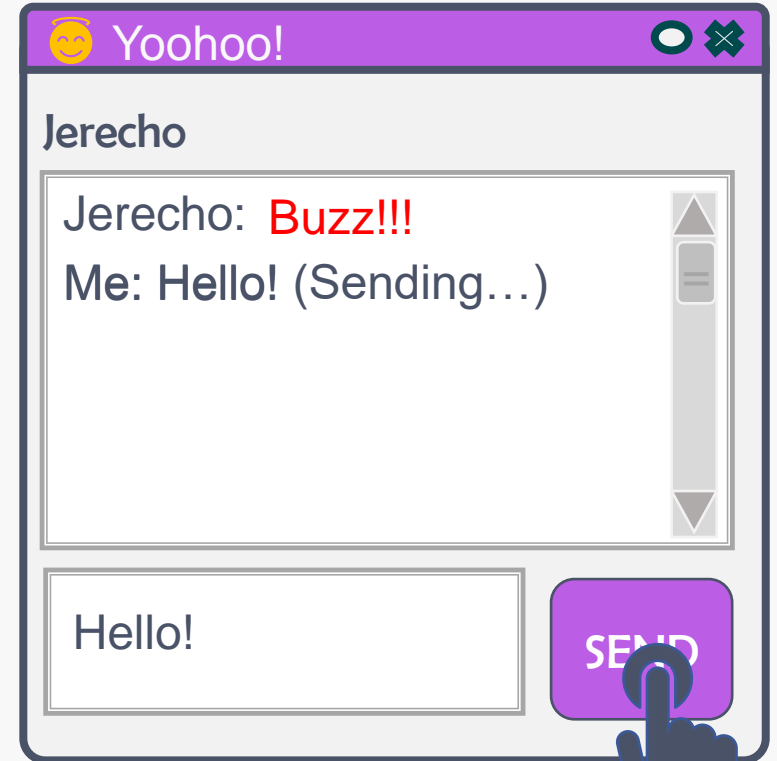
Actions

useOptimistic()

```
const [optimisticMsgs, addOptimisticMsg] = useOptimistic(
  messages,
  //update function
  (currentState, newMessage) => {
    //return merged current state and optimisticValue
    return [...currentState,
      { text: newMessage, sending: true}]
  }
);
```

```
<div>
  {optimisticMsgs.map((message, index) => (
    <div key={index}>
      {message.text}
      {!!message.sending && <small> (Sending...)</small>}
    </div>
  ))}
</div>
```

parent state
updated



Actions

useOptimistic()

```
function MessengerApp() {  
  const [messages, setMessages] = useState([]);  
  
  const [optimisticMessages, addOptimisticMessage] = useOptimistic(  
    messages,  
    (state, newMsg) => [...state, { text: newMsg, sending: true }]  
  );  
  
  async function sendAction(formData) {  
    addOptimisticMessage(formData.get("message"));  
    const sentMessage = await sendMessage(formData);  
    setMessages((messages) => [...messages, { text: sentMessage }]);  
  }  
  /* ... */  
}
```

1. store 'messages' in component state

2. pass state 'messages' into optimistic hook

4. temp optimistic update function

3. call optimistic dispatch function

5. update component state

Other new features

React Server Components

A type of Component that renders ahead of time, before bundling in an environment separate from your client application or SSR Server.

Support for Document Metadata

Render document metadata tags in components natively such as `<title>`, `<meta>`, `<link>` and move them to the `<head>` section.

Support for Stylesheets

Added 'precedence' property to allow you to manage the insertion order of the stylesheet and ensure it is loaded before revealing the content

React Server Components

Pre React 19

```
function Page({page}) {  
  const [content, setContent] = useState('');  
  // Loads *after* first page render.  
  useEffect(() => {  
    fetch(`/api/content/${page}`).then((data) => {  
      setContent(data.content);  
    });  
  }, [page]);  
  
  return <div>{sanitizeHtml(marked(content))}</div>;  
}
```

Speaker notes: Fetching static data in a CMS.

React Server Components

```
async function Page({page}) {  
  // Loads *during* render, when the app is built.  
  const content = await file.readFile(`${page}.md`);  
  
  return <div>{sanitizeHtml(marked(content))}</div>;  
}
```

Speaker notes: Server components can run at build time to read from the filesystem or fetch static content, so a web server is not required

Server Components are not sent to the browser, so they cannot use interactive APIs like `useState`

Support for Document 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>  
  );  
}
```

<title>, <meta>, <link> tags are automatically hoisted to the <head> section of document

Support for Stylesheets

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

function ComponentTwo() {
  <link rel="stylesheet" href="three" precedence="default" />
}

function App() {
  return <>
    <ComponentOne />
    ...
    <ComponentOne /> // not duplicated
  </>
}
```

precedence will manage the insertion order in DOM and ensures that the stylesheet is loaded before revealing dependent components

stylesheet **“three”** will be inserted after **“one”**

rendered component from multiple places will only include the stylesheet in the document **once**

Upgrading to React 19

Speaker notes:

recommended approach is
render-as-you-fetch
e.g. fetch early or using
loaders (remix/react-router)



Issue with Suspense on sibling
components that 'fetch-on-render'
causing waterfall requests.

<https://github.com/facebook/react/issues/29898>



Upgrading to React 19

```
npm install --save-exact react@rc react-dom@rc
```

<https://react.dev/blog/2024/04/25/react-19-upgrade-guide>

Speaker notes: There's also codemods that will automatically update your code to many of the new APIs and patterns in React 19

react.dev/blog/2024/04/25/react-19



Questions?

linkedin.com/in/jerechogiba/
jgiba@obsglobal.com





THANK YOU

Enjoy the rest of your day.