

# Managing Local State and Remote State



**Cory House**

React Consultant and Trainer

@housecor | [reactjsconsulting.com](http://reactjsconsulting.com)

# Agenda



## Build a shoe store app

### Local state

- useState

### Remote State

- Use useEffect for async calls
- Loading state
- Error handling and error boundaries
- Promises and async/await
- Custom Hooks



# Installs



**Git**  
**Source Control**  
[git-scm.com](https://git-scm.com)



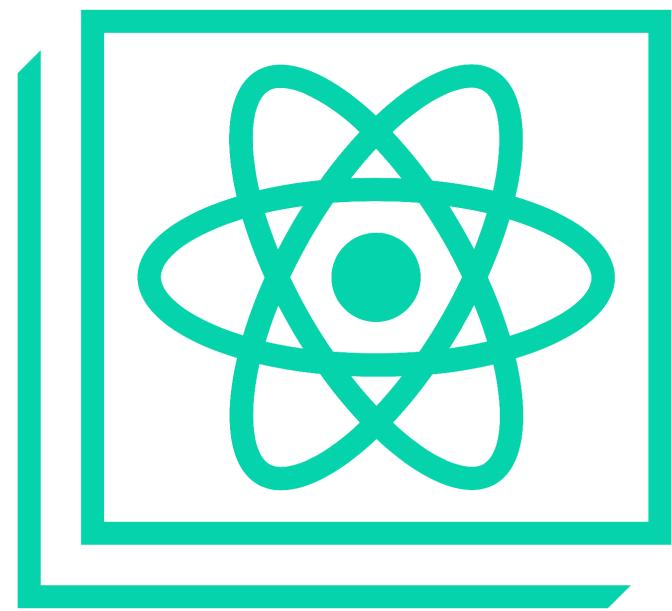
**Node**  
**JavaScript outside the browser**

[nodejs.org](https://nodejs.org)

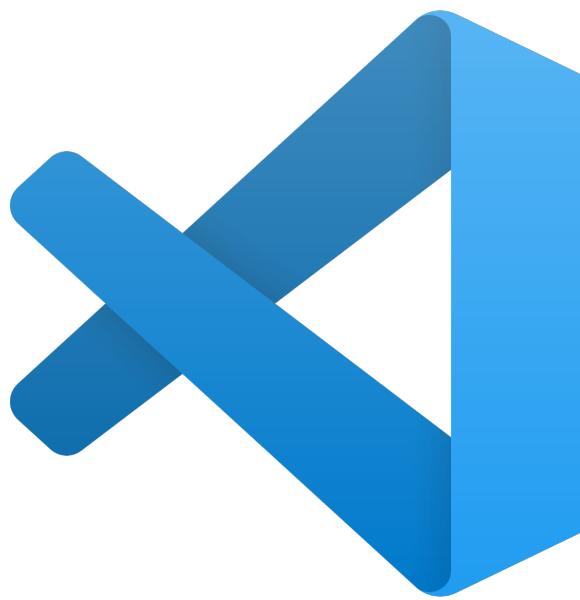
**Node 16+ required**  
**Check Your version: node -v**



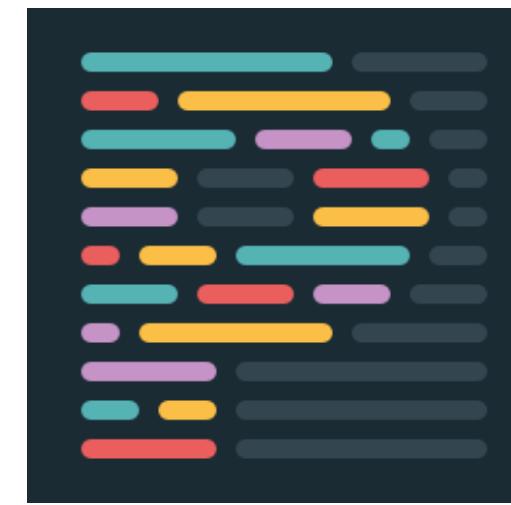
# Demo Setup



**create-react-app**  
**Generator**



**VSCode**  
**Editor**



**Prettier**  
**Code formatter**



# Demo Caveats



**Not using PropTypes or TypeScript**

**Using plain CSS**

**I'll provide some code to copy/paste**



## Demo



### Implement shoe listing page

- Why state is necessary
- Fetch and store data
- Handle immutable state
- Implement filter and update state
- Display error page
- Rules of hooks
- Implement custom hook



# Hooks API Reference

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

This page describes the APIs for the built-in Hooks in React.

If you're new to Hooks, you might want to check out the [overview](#) first. You may also find useful information in the [frequently asked questions](#) section.

- [Basic Hooks](#)
  - [useState](#)
  - [useEffect](#)
  - [useContext](#)
- [Additional Hooks](#)
  - [useReducer](#)
  - [useCallback](#)
  - [useMemo](#)
  - [useRef](#)
  - [useImperativeHandle](#)
  - [useLayoutEffect](#)
  - [useDebugValue](#)

We'll use these a lot

Dedicated modules

INSTALLATION ▾

MAIN CONCEPTS ▾

ADVANCED GUIDES ▾

API REFERENCE ▾

HOOKS ^

1. Introducing Hooks
2. Hooks at a Glance
3. Using the State Hook
4. Using the Effect Hook
5. Rules of Hooks
6. Building Your Own Hooks
- 7. Hooks API Reference**
8. Hooks FAQ

TESTING ▾

CONCURRENT MODE (EXPERIMENTAL) ▾

CONTRIBUTING ▾

FAQ ▾



## Rules of Hooks

**Only are for function components**

- Can be consumed in classes too

**Start with “use”**

**Only call them at the top level**

- Consistent order
- Can't be called inside functions, conditionals, or loops



# Breaking the Rules of Hooks

Nav.js

```
function Nav(props) {  
  // WRONG. Can't wrap in conditionals  
  if (props.isAdmin) {  
    const [role, setRole] = useState('');  
  }  
  // ...  
}
```

Nav.js

```
function Nav() {  
  let on = false;  
  let setOn = () => {};  
  
  function enableAdmin() {  
    // WRONG. Can't nest in func.  
    [on, setOn] = useState(false);  
  }  
  
  return (  
    <button onClick={enableAdmin}>  
      Enable admin  
    </button>  
  );  
}
```





## Rules of Hooks

Only work in function components

Start with “use”

Only call them at the top level

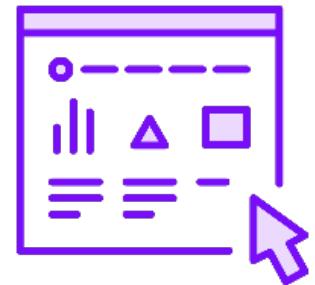
- Consistent order
- Can’t be called inside functions, conditionals, or loops

Need to run a hook conditionally?

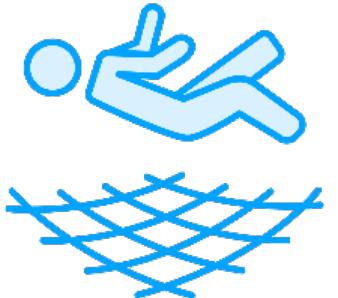
Place condition inside the hook



# Synthetic Events



**Similar to browser's native event system**



**Assures events operate consistently cross-browser**



**Improves performance**





# Event Handlers

**onClick**

**onChange**

**onHover**

**onBlur**

**onSubmit**

**onMouseover**

**Etc....**

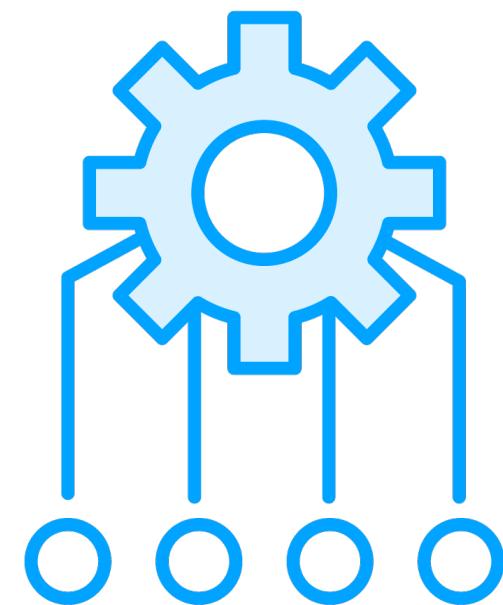


# When Does React Render?



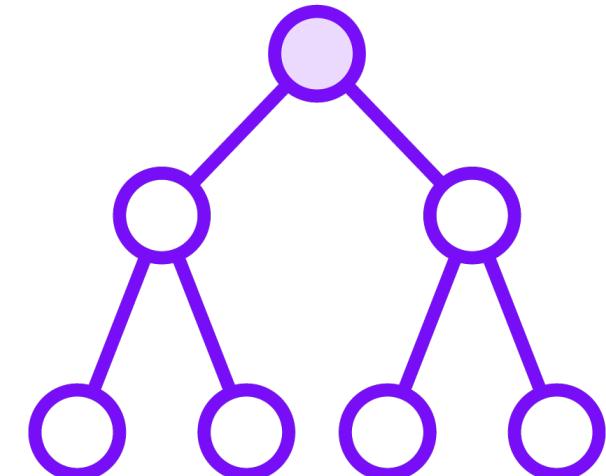
**State change**

Can skip render via:  
`shouldComponentUpdate`  
`React.memo`

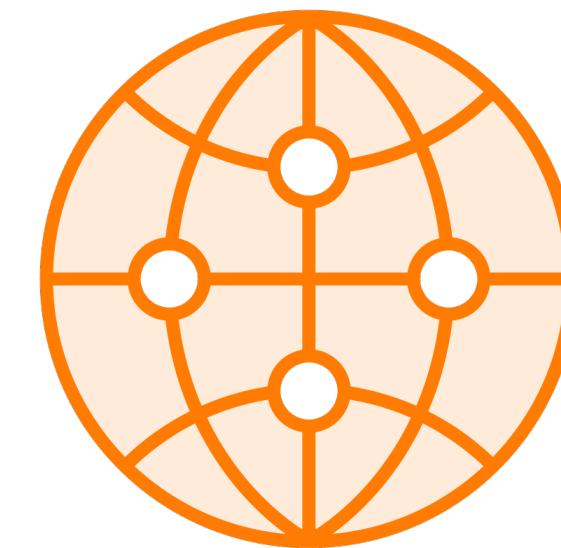


**Prop Change**

Can skip render via:  
`shouldComponentUpdate`  
`React.memo`  
`PureComponent`



**Parent Render**



**Context Changes**



# Four Ways to Handle HTTP Calls

1

Inline

Call `fetch/Axios/etc` in your component  
Not recommended



# Maturity Level 1: Inline

```
import React, { useState, useEffect } from "react";

export default function Demo() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch(`users`)
      .then(resp => resp.json())
      .then(json => setUsers(json));
  }, [ ]);

  return users[0].name;
}
```

Inline calls are hard to handle consistently across the app.



# Four Ways to Handle HTTP Calls

1

Inline

Call `fetch/Axios/etc` in your component  
**Not recommended**

2

Centralized functions

Call separate function



# Maturity Level 2: Centralized Functions

```
export async function getUsers() {  
  const response = await fetch("users");  
  if (response.ok) return response.json();  
  throw response;  
}
```



# Maturity Level 2: Centralized Functions

```
import React, { useState, useEffect } from "react";
import { getUsers } from "./services/userService";

export default function Demo() {
  const [users, setUsers] = useState([ ]);

  useEffect(() => {
    getUsers().then(json => setUsers(json))
  , [ ]);

  return users[0].username;
}
```



# Four Ways to Handle HTTP Calls

1

Inline

**Call fetch/Axios/etc in your component**  
**Not recommended**

2

Centralized functions

**Call separate function**  
**We just did this.**



# Four Ways to Handle HTTP Calls

- 1** **Inline** **Call fetch/Axios/etc in your component**  
**Not recommended**
- 2** **Centralized functions** **Call separate function**  
**We just did this.**
- 3** **Custom hook** **Create and call your own custom hook**  
**We'll do this later in this module.**
- 4** **Library** **Call library (react-query, swr, etc)**  
**We'll do this in the final module.**



# Four Ways to Handle HTTP Calls

- 1** **Inline** **Call fetch/Axios/etc in your component**  
**Not recommended**
- 2** **Centralized functions** **Call separate function**  
**We just did this.**
- 3** **Custom hook** **Create and call your own custom hook**  
**Let's do this next.**
- 4** **Library** **Call library (react-query, swr, etc)**



# Inline

```
import { useState, useEffect } from "react";
export default function App() {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    getProducts()
      .then((resp) => resp.json())
      .then((json) => setProducts(json))
      .catch((err) => {
        console.error(err);
        throw err;
      })
      .finally(() => setLoading(false));
  }, []);

  if (loading) return "Loading...";
  if (error) throw error;
  return //JSX HERE
}
```



# Custom Hook

```
import React from "react";
import useFetch from "./useFetch";

export default function App() {
  const { data: products, loading, error } = useFetch("products");
  if (loading) return "Loading...";
  if (error) throw error;
  return //JSX HERE
}
```



# Maturity Level 4: Library

```
import React from "react";
import { getProducts } from "./services/productService";
import { useQuery } from "react-query";

export default function ReactQueryDemo() {
  const { data, isLoading, error } = useQuery("products", getProducts);
  if (isLoading) return "Loading...";
  if (error) throw error;
  return data[0].name;
}
```



# Summary



## Local state

- useState

## Remote state

- useEffect
- Async calls
- Promises and async/await
- Loading state
- Declared an Error Boundary
- Error handling and Error Boundaries

Created a custom hook



**Up Next:**

# **Route State**

---

