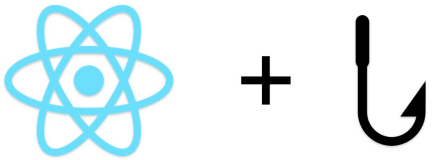
Hooks/Custom-Hooks

Hooks are optimized solutions for various problems that previously required different techniques or tools.

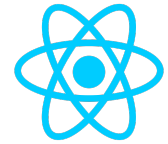


Classes confuse both people and machines

In addition to making code reuse and code organization more difficult, we've found that classes can be a large barrier to learning React. You have to understand how `this` works in JavaScript, which is very different from how it works in most languages. You have to remember to bind the event handlers. Without ES2022 public class fields, the code is very verbose. People can understand props, state, and top-down data flow perfectly well but still struggle with classes. The distinction between function and class components in React and when to use each one leads to disagreements even between experienced React developers.

Hooks let you use more of React's features without classes.

Hooks/Custom-Hooks



```
const [state, setState] = useState()
```

The **Last** State

Before any change or update, the **Last** State is equal to the **Initial** Sate

A **function** to change the Last State

Must start with **set** and finish with the **Last** State (camelCase).

The **Initial** State

Boolean (true/false)
Number (2, 4.5, -3)
String ("t", "Hello")
Array ([])
Etc.

EX:

```
const [bool, setBool] = useState(false);
```

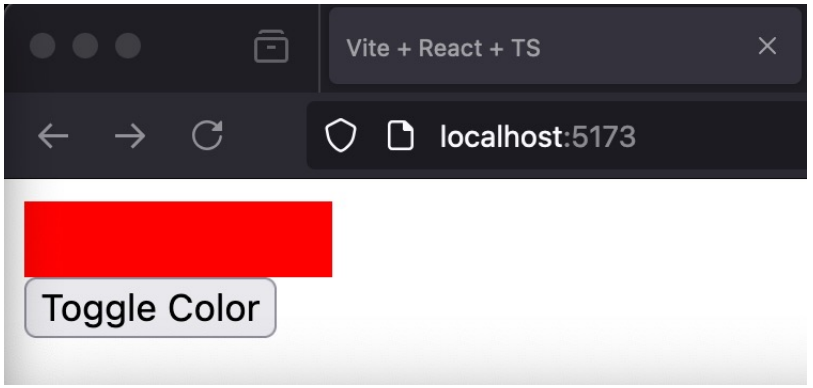
The **useState()** Hook, manages the "state" of components.

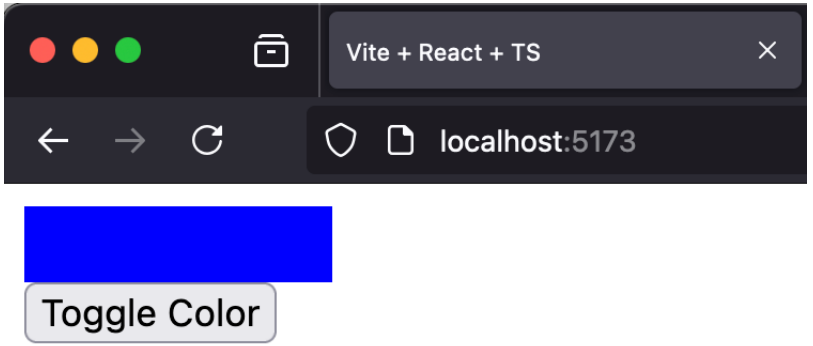
Note:
Since data-flow in React is unidirectional, the only way a child component could change the state of a parent component is through a **useState()** hook and **props** (covered next lecture).

Hooks/Custom-Hooks

EX:

```jsx
import { useState } from "react";

export default function App() {
    const [color, setColor] = useState(true);

    function changeColor() {
        setColor(!color);
    }

    return (
        <>
            <div
                style={{
                    height: "10vh",
                    width: "20vw",
                    backgroundColor: color ? "blue" : "red"
                }}
            >
            </div>
            <button onClick={changeColor}>Toggle Color</button>
        </>
    );
}
```





Hooks/Custom-Hooks

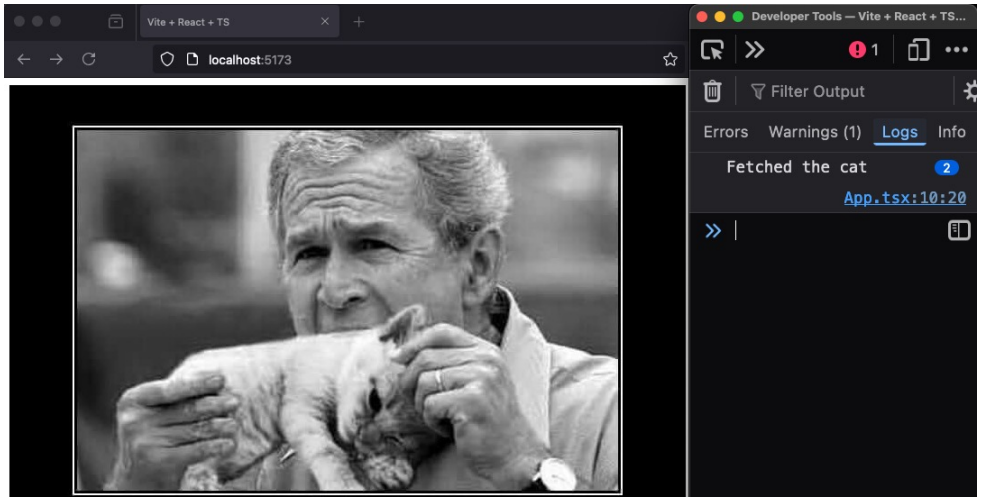
```
useEffect(() => {}, []);
```

The **useEffect()** Hook accept an anonymous function that returns an object, and re-renders the hosting component based on optional dependencies added to an array

The **useEffect()** hook handles potential side-effects when you connect your Component to an external source (DB, API, service-provider, servers, etc.)

EX:

```
const [data, setData] = useState("");

useEffect(() => {
    try {
        const cat = "https://http.cat/405";
        setData(cat);
        console.log("Fetched the cat");
    } catch (e) {
        console.log(e + " Couldn't fetch the cat");
    }
}, []);
```



EX:

Hooks/Custom-Hooks

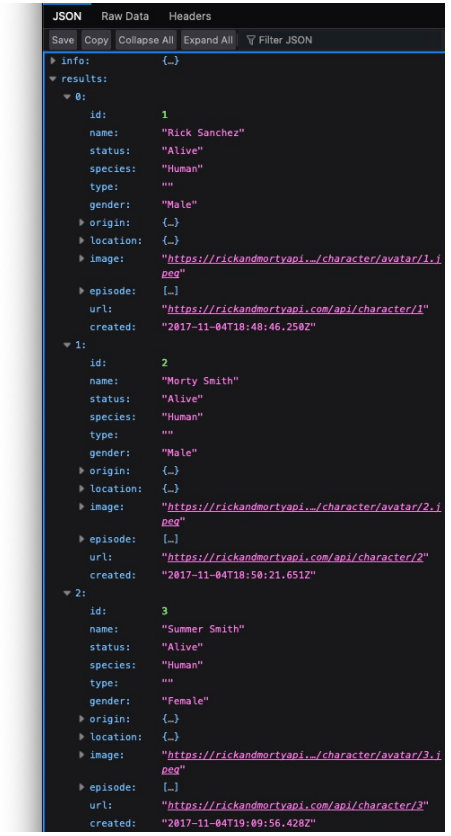
The **useState()** hook and the **useEffect()** hook are commonly used together when you want to retrieve data from on outside source.

```tsx
export default function App(){

    // useState Hook to store Data.
    const [data, setData] = useState<Character[]>([]);

    // useEffect Hook for error handling and re-rendering.
    useEffect(() => {
        async function fetchData(): Promise<void> {
            const rawData = await fetch("https://rickandmortyapi.com/api/character");
            const {results} : {results: Character[]} = await rawData.json();
            setData(results);
        }
        fetchData()
            .then(() => console.log("Data fetched successfully"))
            .catch((e: Error) => console.log("This was the error: " + e));
    }, [data.length]);

    return (
        <>
            {
                data.map((char: Character) =>
                    <div key={char.id}>
                        <h1>{char.name}</h1>
                        <img src={char.image} alt={`image of ${char.name}`} />
                    </div>
                )
            }
        </>
    );
}
```

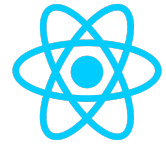
Rick Sanchez

Morty Smith

Summer Smith



Hooks/Custom-Hooks



React **Hooks**

Hooks are predefined JS. Functions in React, that we could easily import, and use for a variety of different purposes.

Hooks let you use different React features from your components. You can either use the built-in Hooks or combine them to build your own.

- **useState**: Allows functional components to manage local state.
- **useEffect**: Enables side effects in functional components, such as data fetching, DOM manipulation, and more.
- **useContext**: Provides access to the context of a parent component.
- **useReducer**: A more advanced alternative to useState for managing complex state logic.
- **useRef**: Provides access to a mutable reference object that can be used to interact with DOM elements directly.
- **useMemo**: Memoizes the result of a function, useful for optimizing performance by caching values.
- **useCallback**: Memoizes a function, similar to useMemo, but for functions.
- **useLayoutEffect**: Similar to useEffect, but it runs synchronously after all DOM mutations.
- **useDebugValue**: Used for custom hooks to display debugging information in React DevTools.

Hooks/Custom-Hooks

As mentioned in lecture, **Hooks** in **React**, embody functions and variables that have been used in the Web Industry for many years.

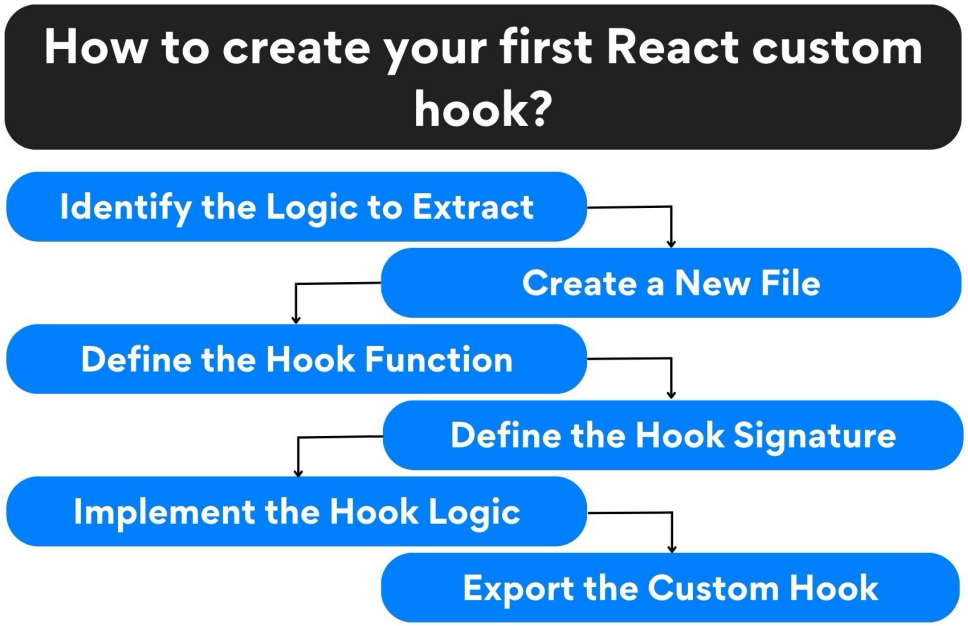
To reduce code verbosity, these frequently used functions and variables are optimized under the context of **Hooks.** So, instead of writing repetitive code from scratch, developers could just invoke these **Hooks**, and just change parts of it that pertains to their instance or application.

When you find yourself repeating a logic or algorithm, you should try to reduce code verbosity by replacing that repetitive work with a Hook, but if you couldn't find a Hook for your repetitive actions, then you could build your own Hook.

Hooks/Custom-Hooks

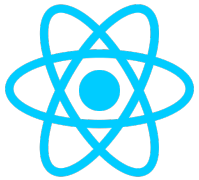
*So when do we need a **Custom React Hook**?*

- When we can identify repetitive operations (functions/Hooks).
- And there are no existing optimized alternatives.



Ex:

Hooks/Custom-Hooks



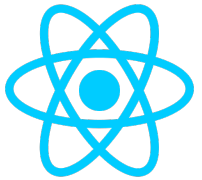
Custom Hook (useDataFetching.js).

```
  import { useState, useEffect } from 'react';

  function useDataFetching(dataSource) {
+   const [loading, setLoading] = useState(false);
+   const [data, setData] = useState([]);
+   const [error, setError] = useState('');

    return [];
  }

  export default useDataFetching;
```

Another **Custom Hook** designed by **Vercel Inc. is useSWR()**, which has optimized the **API calling** process.

```
import useSWR from 'swr'

function Profile() {
  const { data, error, isLoading } = useSWR('/api/user', fetcher)

  if (error) return <div>failed to load</div>
  if (isLoading) return <div>loading...</div>
  return <div>hello {data.name}!</div>
}
```

Hooks/Custom-Hooks



Ex:

Regardless of which option you chose, you still have to save the data from those calls in an **array []**, so that you can **map()** over its content.

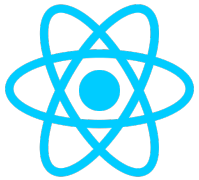
```
data.results.map((char) => (
    <div key={char.id}>
        <h1>{char.name}</h1>
        <img src={char.image} alt="Rick and Morty Character" />
    </div>
))
```

Sometime you may end up with multiple **arrays []** from multiple **API calls**, and you may need to do more with those **arrays []** than just iterations. For example, you may need to combine them, slice them, or manipulate them in certain way. So you may want to create a new **Custom Hook** to optimize these manipulations.

```
// Custom hook for managing arrays with various utility functions
1+ usages
export default function useArray(defaultValue) {...} {
    const [array, setArray] = useState(defaultValue);
```

Note:
Advance **Custom Hooks** like **useSWR()**, often have an Array manipulating schema built in. So if you decided to go with **useSWR()** for your **API calls**, read their documentations before you create a **Custom Hook**:
https://swr.vercel.app/docs/pagination

Hooks/Custom-Hooks



Rules of Hooks

Top-Level Calls:
Call hooks at the top level of a functional component or custom hook.

React Functions Only:
Use hooks only within React functional components or custom hooks, not in regular JavaScript functions.

Functional Components:
Exclusively use hooks in functional components, not in class components.

Additional Recourses:
https://legacy.reactjs.org/docs/hooks-rules.html
https://react.dev/learn/reusing-logic-with-custom-hooks

Additional Examples:
https://shorturl.at/vBOSX
https://shorturl.at/dktvZ

Rules of Custom Hooks

Start with "use":
Name with "use" prefix for convention.

Focus & Simplicity:
Keep hooks simple, focused on one concern.

Parameterize for Reusability:
Use parameters for flexibility.

Consistent Return Values:
Maintain a predictable return pattern.

Clear Documentation:
Concise documentation for usage, parameters, and effects.

Follow ESLint Rules:
Enforce best practices with ESLint.

Show & Tell

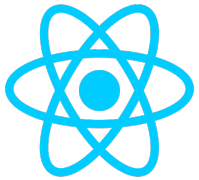
A cool feature is the **Drag&Drop** from **HTML5 Drag and Drop API**

This **API** is already built-in to your browsers, and it can be added as an **attribute** inside your **tags**.

```
<div class="container">
  <div draggable="true" class="box">A</div>
  <div draggable="true" class="box">B</div>
  <div draggable="true" class="box">C</div>
</div>
```

Then using **JS/React**, you can **handle** the the **movement/state** of objects, while being picked, dragged, and dropped.

Show & Tell



The **onDragStart()** event, starts the dragging operation.

```
function onDragStart(e, id) : void  {
    e.dataTransfer.setData( format: 'id', id);
}
```

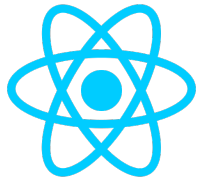
By default, it's impossible to drop elements into another element. This can be prevented by calling the **preventDefault()** method for the **onDragOver()** event

```
function onDragOver(e) : void  {
    e.preventDefault();
}
```

The **onDrop()** event handles the final state of the dropped element, so it should be called from the **component** that handles that **jsx-element**

```
function onDrop(e, laneId) : void  {
    const id : string  = e.dataTransfer.getData( format: 'id');
```

Show & Tell



There are many other cool features that you could implement in **React**.

&

If you prepare a **slide show** and a **small project** about one of these features you will get **%3 Extra-Credit**

React Utilities Provide by Third Parties

https://mui.com/material-ui/transitions/

https://madewithreactjs.com/utilities

You have to present your slide show and project in class and share your code on GitHub

Let's Look at some examples