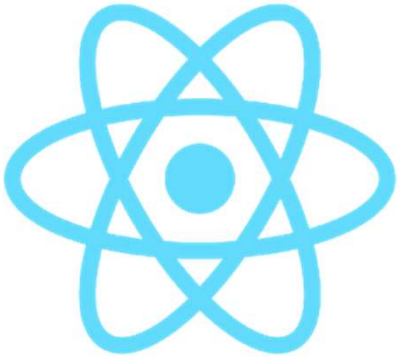


React Fundamentals Module – React Hooks

Peter Kassenaar –
info@kassenaar.com

React Hooks

Using the standard React Hooks in function components, creating your own hooks.

React Hooks introduction

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

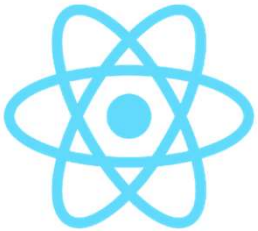
Introducing Hooks

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

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

https://reactjs.org/docs/hooks-intro.html

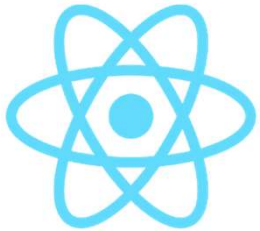
What is a Hook?



"A Hook is a special function that lets you "hook into" React features. For example, `useState` *is a Hook that lets you add React state to function components. We'll learn other Hooks later."*

https://reactjs.org/docs/hooks-state.html

React Hooks…



- Are completely opt-in. You don't *have* to use them.
- Don't replace current React knowledge.
- No plans to remove classes from React.
- Only available in `function`-based components.
- …can be a replacement for lifecycle hooks.
- More in-depth info:
  - https://reactjs.org/docs/hooks-intro.html#its-hard-to-reuse-stateful-logic-between-components

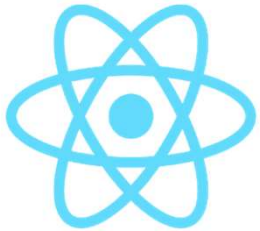
Default Hooks

- State Hook: `useState()`
- Effect Hook: `useEffect()`
- Custom Hooks: create your own. Follow the naming conventions for getting/setting data
  - `useMyHook()`

Using the state hook

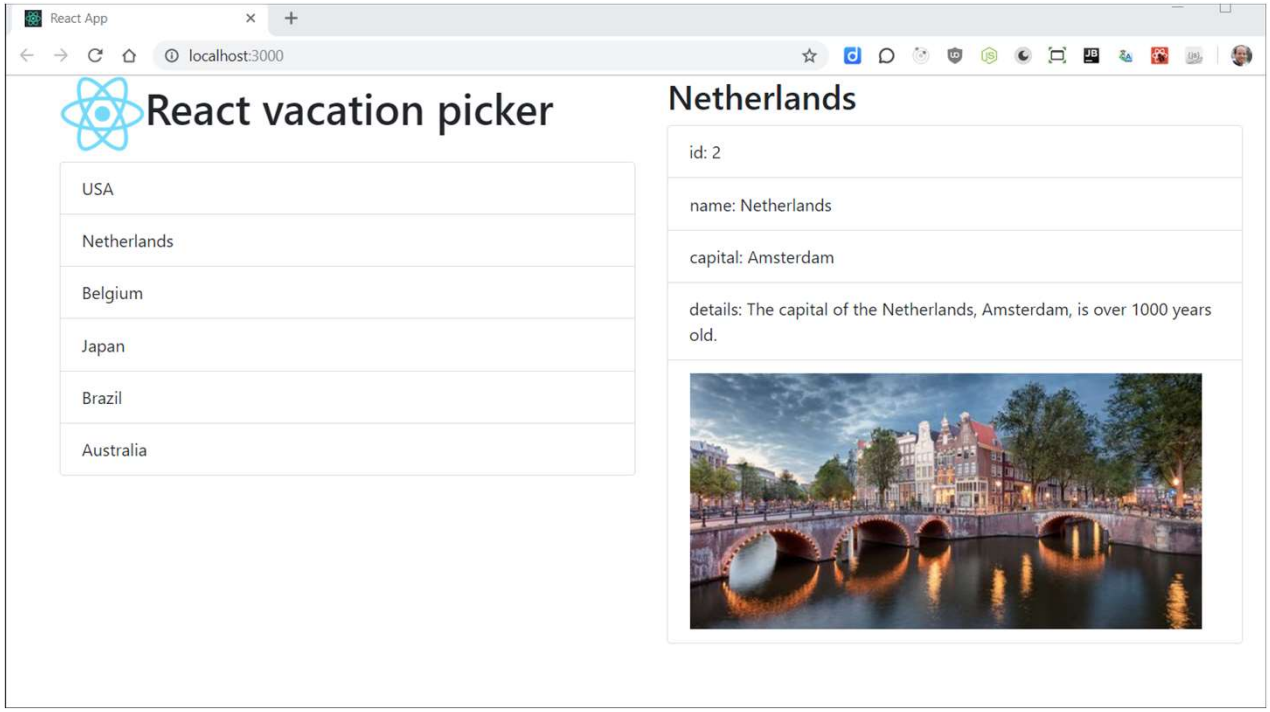
Defining state in your components, using hooks instead of a `state`-object

Let’s rewrite our `<VacationPicker />`



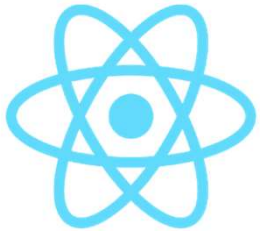
`<VacationPicker>` is going to use Hooks for its state

We removed routing, other components and so on, to avoid bloated code



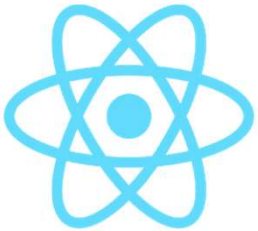
Starting point: `../examples/220-image-binding`

1. Create functions instead of classes



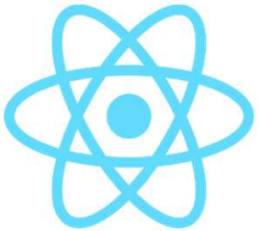
- `class App extends React.Component {…}` becomes `function App()`
- Replace the `render()` method with `return (…)` statement
- Create functions-inside-functions (instead of class methods) and remove the `this` keyword everywhere in your code

Function based components



```
// App.js - now using function components and React Hooks
function App() {

    function selectCountry(country) {
        …
    }

    // 4. Render UI, nothing special
    return (
        <div className="container">
        …
        </div>
    )
}

export default App;
```

2. Using Hooks



- Import `{useState} from 'react'`
- Replace `state`-object with the `useState()` hook
- Replace `this.setState(…)` calls with the `set`-Hook
- In Detail components, add the `props` parameter to the function
  - The state is passed down, but now as an argument to the component function

Importing Hooks

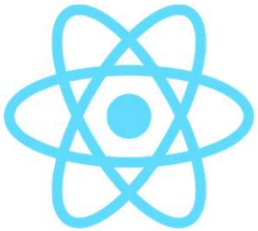
```
import React, {useState} from 'react';
```

Using ES6 destructuring for the hook function

```
// 1. Using a state hook here, only a getter. No setter.
const [countries] = useState(countryData.countries);

// 2. Directly using the variable defined in the line above
const [currentCountry, setCountry] = useState(countries[0]);

// 3. Helper function, selecting a new country and passing
// it to the Detail component
function selectCountry(country) {
    const newIndex = countries.indexOf(country);
    // 3a. Using the set Hook here to set the new state
    setCountry(countries[newIndex]);
}
```

Using the setter to define new state for `currentCountry`

3. Rewriting child components

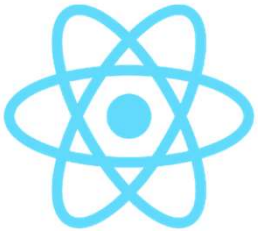


- Same procedure
  - Replace `classes` with `functions`
  - Pass in `props`
  - Replace `render()` with `return()`

Pass in, and loop over `props`

```
// VacationPicker.js
function VacationPicker(props) {
    return (
        <div>
            <ul className="list-group">
                {props.countries.map(country =>
                    …
                        {country.name}
                    )}
            </ul>
        </div>
    );
}
export default VacationPicker
```

Inside the `useState()` Hook



- `useState()` is just a function
- It returns a *pair*: the current state, and the function that lets you update it.
- The only argument to `useState()` is the initial state
- b/c a JavaScript function can only return a single value, the hook returns them in an array

```
const [countries, setCountries] = useState(countryData.countries);
```

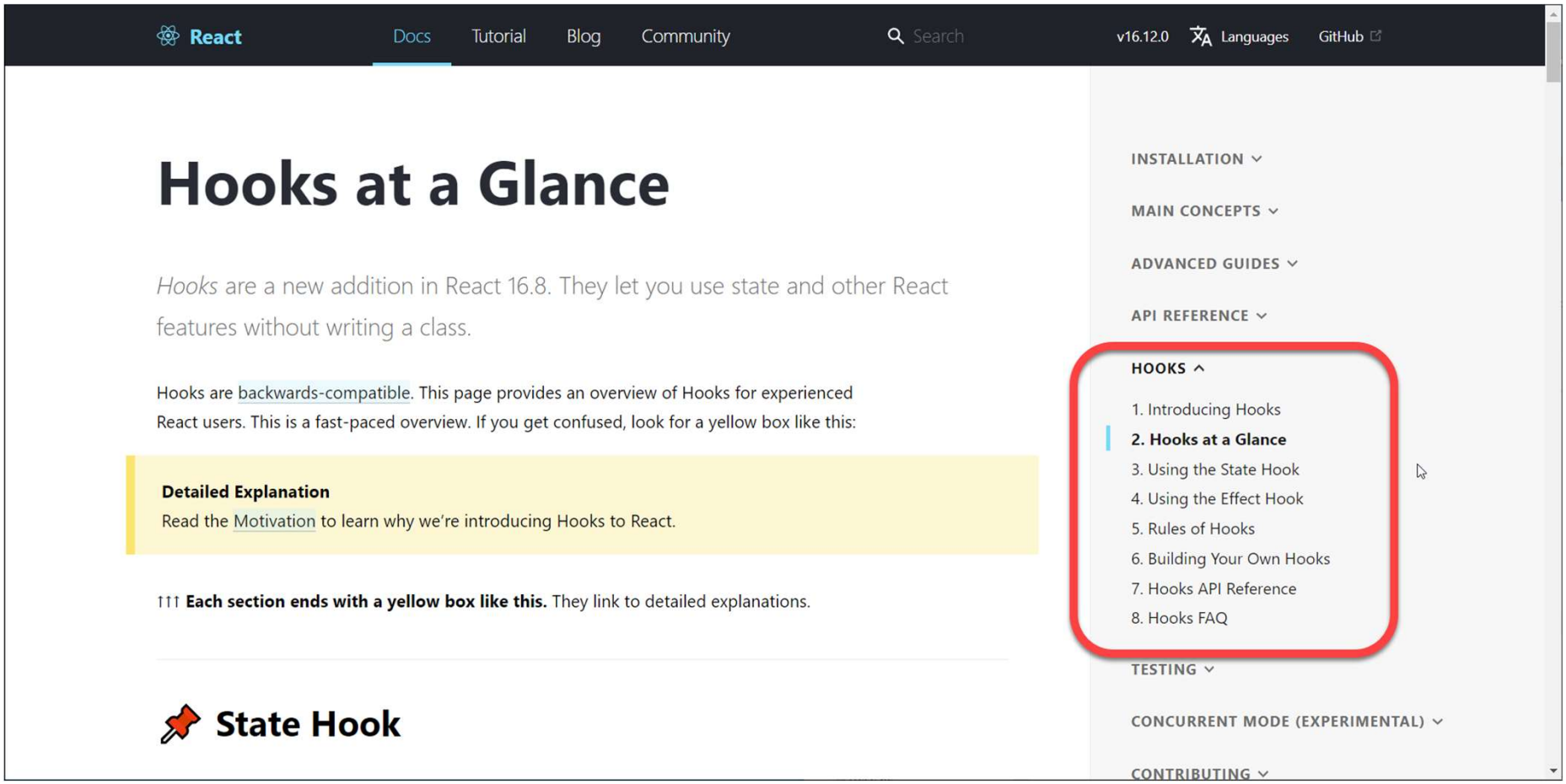
Array destructuring

- *ES6 Array destructuring syntax* gives different names to the variables we declared
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment#Array_destructuring
- You can use the state hook multiple times

```
// Declare as many state variables as needed for this component!
const [countries, setCountries] = useState(countryData.countries);
const [currentCountry, setCountry] = useState(countries[0]);
const [counter, setCounter]=useState(0);
```

https://reactjs.org/docs/hooks-overview.html

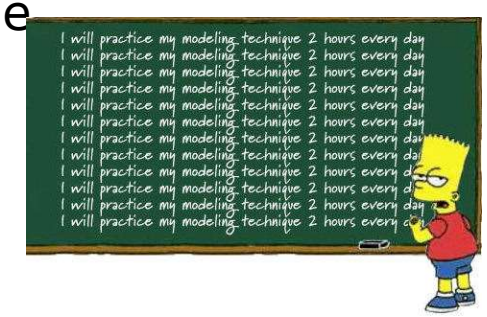
More info



reactjs.org/docs/hooks-overview.html

Workshop

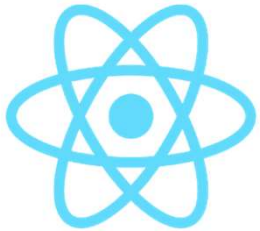
- Rewrite one of your own `class`-based apps to a `function`-based app, utilizing Hooks
- OR: start with example `../700-hooks-intro` and
  - Add state for a `name` property to `App.js`
  - Create a textfield and a button, that updates the name
  - Create a new component, pass the `name` as a prop to that component and display it.
  - Optional: make it possible to change the name in the detail component and pass it back to the parent – which in turn of course updates the state…



The Effect hook

The perfect replacement for class-based lifecycle hooks

What is the `Effect` hook?



"Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects. Whether or not you're used to calling these operations "side effects" (or just "effects"), you've likely performed them in your components before."

https://reactjs.org/docs/hooks-effect.html

Replacement for lifecycle hooks

- The `useEffect()` hook runs *after every render*
- So, no need for different `componentDidMount()` and `componentDidUpdate()` functions
- You can pass in a second parameter to `useEffect()`, as the value to watch for
- Let's say we want to show how often our detail component is updated

Counting updates

- Introducing some local state
- Introducing `useEffect()` to update the state.


```
function CountryDetail (props) {
    // introducing some local state, to keep track how often this component
    // has updated
    const [count, setCount] = useState(0);

    // updating the state in an effect hook
    useEffect(()=>{
        setCount(count + 1);
    })
…
}
```

First try: update the state for `count`

Counting updates…

(component updated 6722 times)

We created an infinite loop!

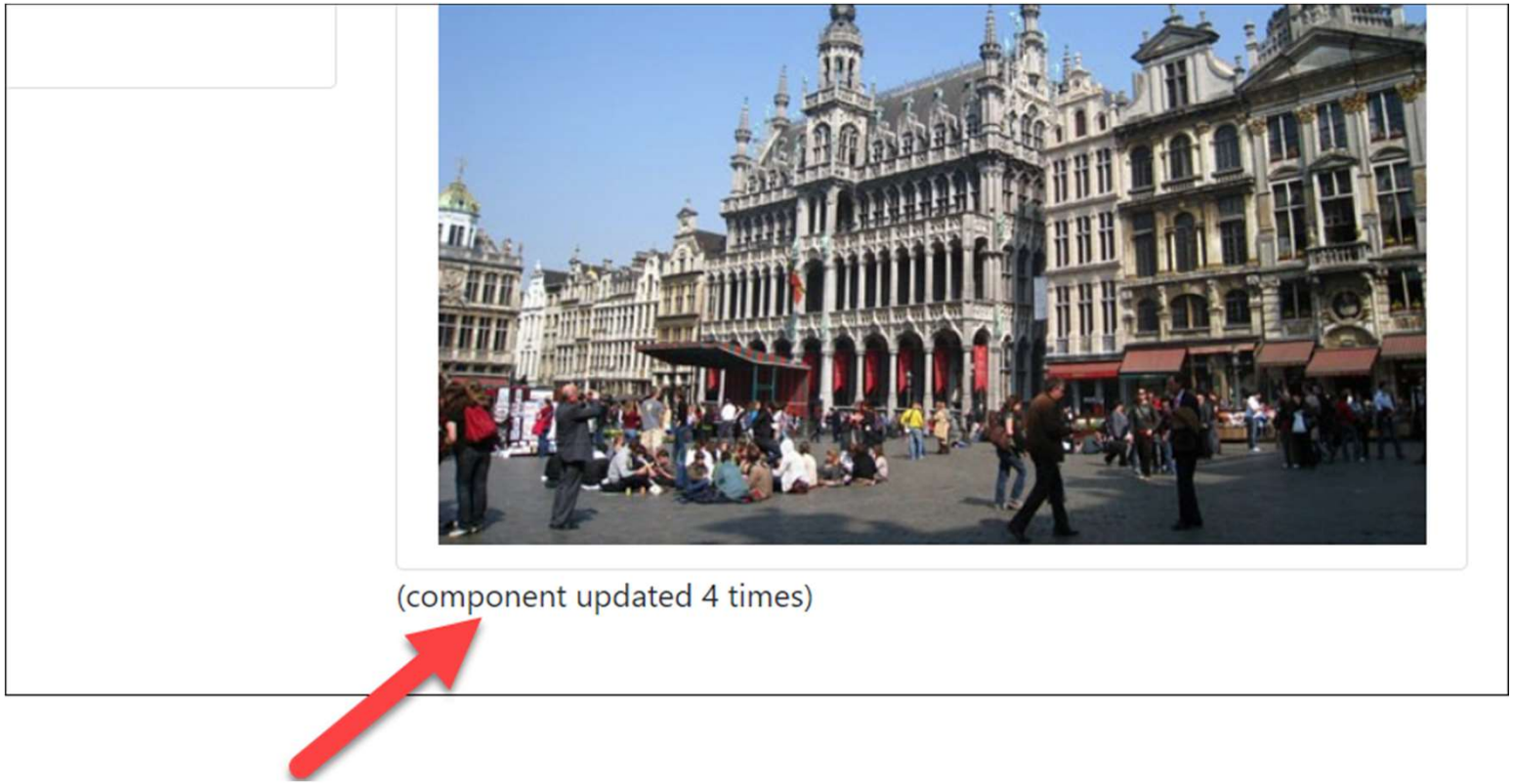
Second try

- Updating the state while watching for specific changes

```
// updating the state in an effect hook
useEffect(()=>{
    setCount(count + 1);
}, [props.country])
// pass in props.country as the second parameter.
```

Watch for `props.country` to change. Then perform the effect

Now it works…



Official documentation on skipping effects

Tip: Optimizing Performance by Skipping Effects

In some cases, cleaning up or applying the effect after every render might create a performance problem. In class components, we can solve this by writing an extra comparison with `prevProps` or `prevState` inside `componentDidUpdate`:

```
componentDidUpdate(prevProps, prevState) {
  if (prevState.count !== this.state.count) {
    document.title = `You clicked ${this.state.count} times`;
  }
}
```

This requirement is common enough that it is built into the `useEffect` Hook API. You can tell React to *skip* applying an effect if certain values haven't changed between re-renders. To do so, pass an array as an optional second argument to `useEffect`:

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]); // Only re-run the effect if count changes
```

In the example above, we pass `[count]` as the second argument. What does this mean? If the `count` is `5`, and then our component re-renders with `count` still equal to `5`, React will compare `[5]` from the previous render and `[5]` from the next render. Because all items in the

https://reactjs.org/docs/hooks-effect.html#tip-optimizing-performance-by-skipping-effects

Rule

- Only call hooks from React functions
  - Not from regular JavaScript functions
- Only call hooks at the top level.
- *Never* call hooks inside loops, conditions or nested functions!
- Pass in an empty array if you only want to perform on first render (like `componentDidMount()`)

Rules of Hooks

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

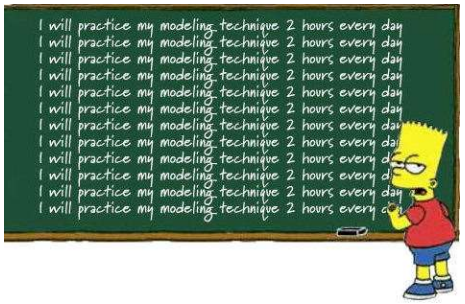
Hooks are JavaScript functions, but you need to follow two rules when using them. We provide a linter plugin to enforce these rules automatically:

Only Call Hooks at the Top Level

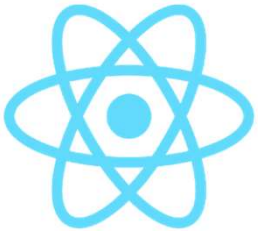
https://reactjs.org/docs/hooks-rules.html

Workshop

- Start with example `../500-api-call` and
  - Rewrite `App.js` to a function based component, using Hooks
  - Rewrite `componentDidMount()` to `useEffect()`
- Optional: rewrite the other components in the app to using `function` components and hooks

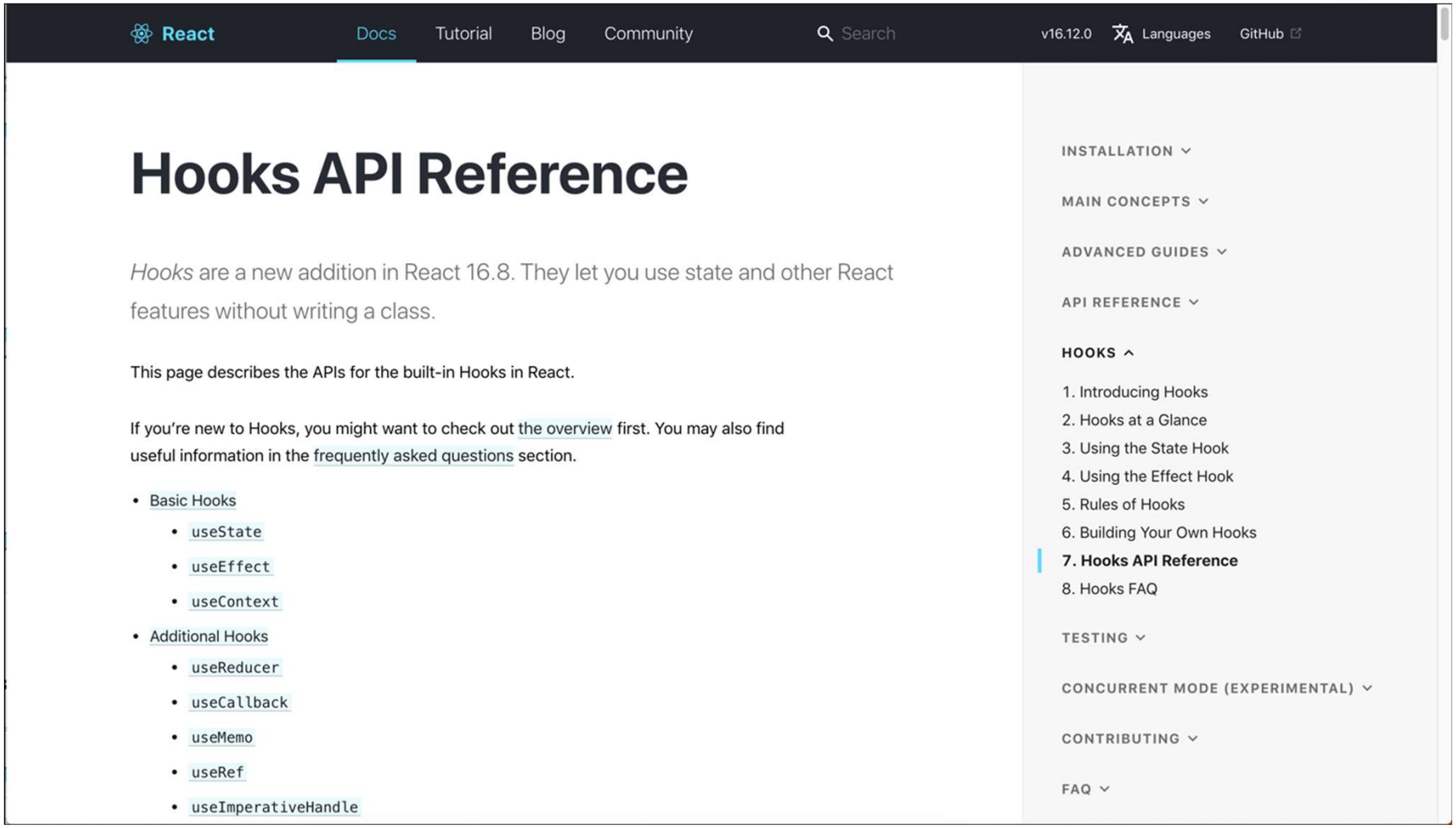


More default hooks



- We’ve looked at `useState()` and `useEffect()`.
- React also has:
  - `useContext()`
  - `useReducer`
  - `useCallback()`
  - `useMemo()`
  - `useRef`
  - `useImperativeHandle()`
  - `useLayoutEffect()`
  - `useDebugValue()`

Other React Hooks

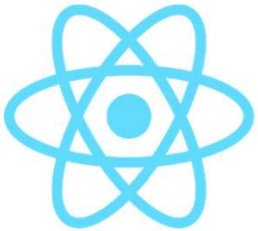


https://reactjs.org/docs/hooks-reference.html

Writing custom hooks

The perfect replacement for class-based lifecycle hooks

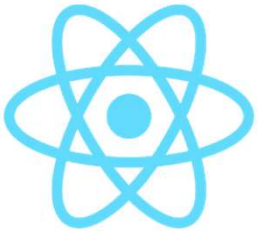
Writing custom hooks



"Building your own Hooks lets you extract component logic into reusable functions."

Custom hooks can be a replacement for `render props` and *High Order Components* (HOC)

https://reactjs.org/docs/hooks-custom.html



"Let's say you have 2 functions (components) which implement some common logic. You can create a third function with this common logic and implement it in the other two functions. After all, hooks are just functions.

Custom hooks means fewer keystrokes and less repetitive code."

https://blog.bitsrc.io/writing-your-own-custom-hooks-4fbcf77e112e

Start simple – a custom counter hook

- Traditional `<Counter />` component – nothing special:

```
// CounterStep1.js - a simple counter component, using a state hook
import React, {useState} from "react";

function CounterStep1() {
   const [count, setCount] = useState(0);

   const increment = () => setCount(count + 1);
   const decrement = () => setCount(count - 1);

   return (
      <div>
         <h2>Counter - Step 1, inline state: {count}</h2>
         <button onClick={increment}>Increment
         </button>
         <button onClick={decrement}>Decrement</button>
      </div>
   );
};

export default CounterStep1
```

Why custom hooks?

- Problem: what if we want to reuse the `count`, `increment` and `decrement` functionality in another component?
- Solution: write a custom hook
- *D.R.Y: Don't Repeat Yourself*

"A custom Hook is a JavaScript function whose name starts with '`use`' and that may call other Hooks"

1. Creating a `useCounter` hook

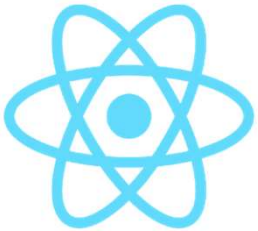
```
import {useState} from 'react';

// receive the initial state as a prop, instead of hardcoding it to 0.
function useCounter(initialState) {

   const [count, setCount] = useState(initialState);

   const increment = () => {
      setCount(count + 1);
   };
   const decrement = () =>{
      setCount(count - 1);
   };

   // ******* return items
   return [count, increment, decrement];
}

export default useCounter
```

Local state – we call another hook inside this hook

Functionality

Return items

2. Returning items from hooks



- `useCounter` returns an array
- First value is the `count` state property
- Second and third value are the helper functions to change the value of `count`

```
return [count, increment, decrement];
```

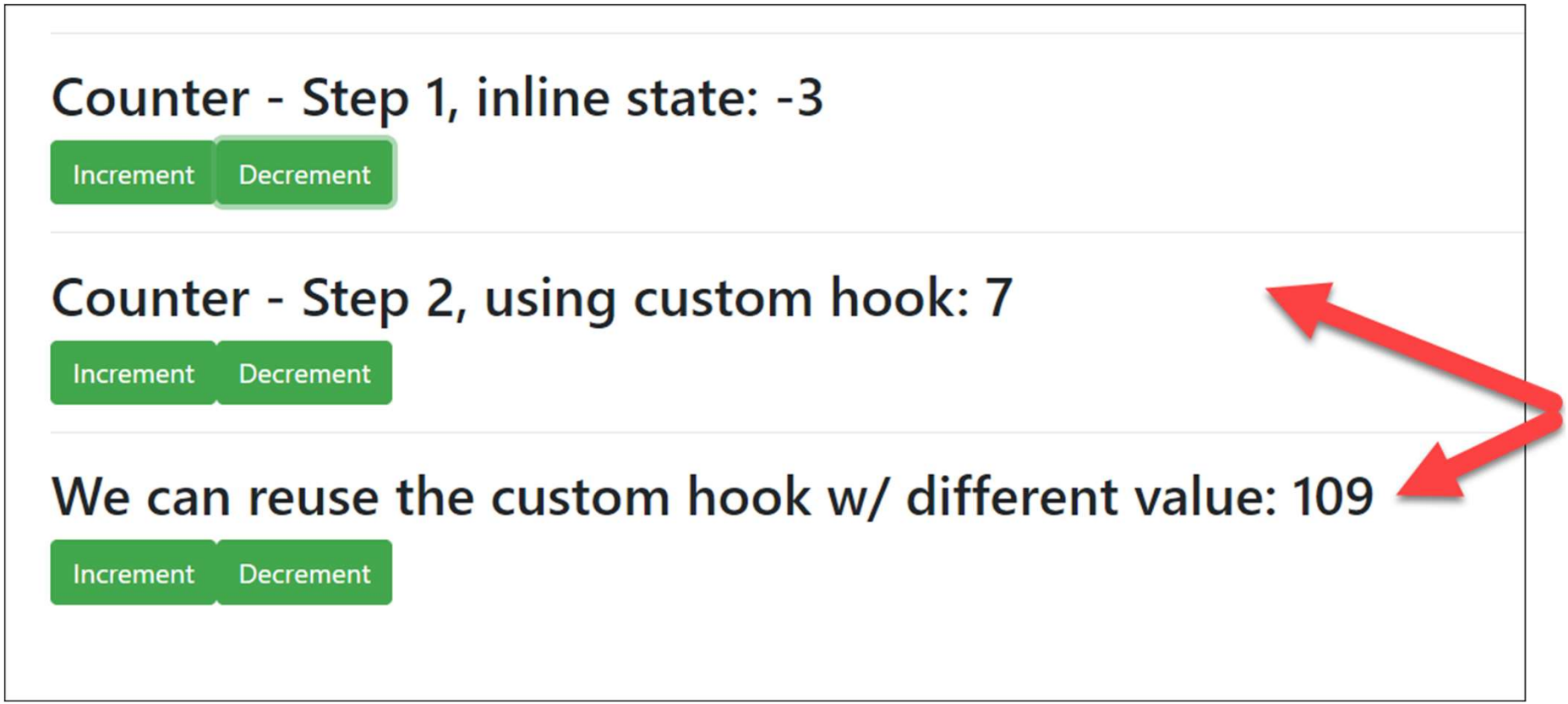
3. Using a hook

- `import` the hook in the component
- You may call a hook multiple times!
  - After all, it is just a function. Functions can be called multiple times
  - Initialize `counter` with different values
  - Each function call creates its own scope with a state of `counter`

```
const [count, increment, decrement] = useCounter(10);
const [count2, plus, minus] = useCounter(103);
```

…/counterStep2.js

4. Result



…/720-custom-hooks

Another hooks example

- Storing the value of `counter` in `localStorage`
- Requirements:
  - Get the value from local storage.
  - If it doesn’t exist set it to 0.
  - Update the value on every click in local storage.
  - Catch errors if bad value is passed in (i.e. `NaN`)

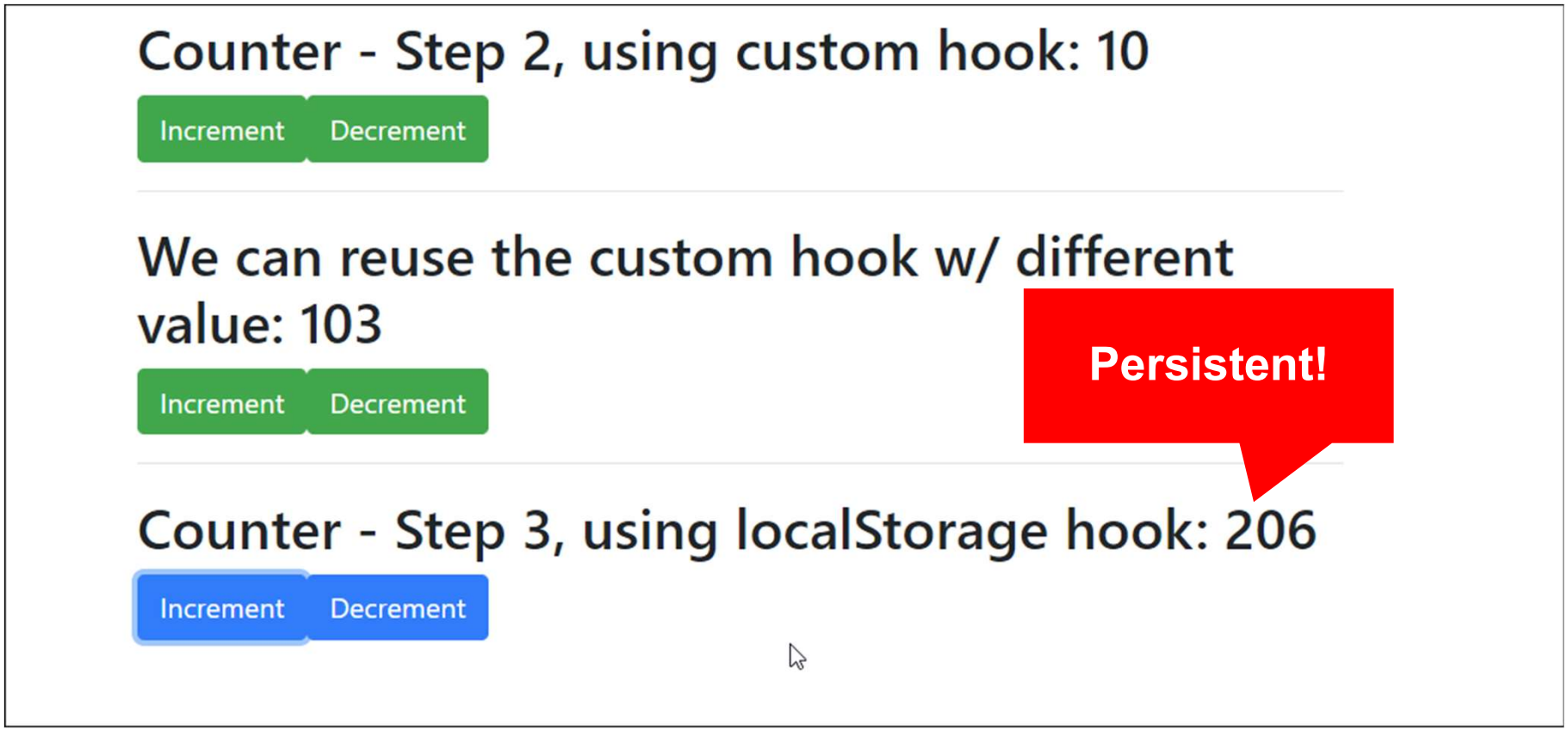
Name of the hook

Using a function in `useState()` to check the defaultValue

```
import {useSta useEffect} from 'react'

function useLocalStorage(key, defaultValue) {
    const [state, setState] = useState(() => {
        let value;
        try {
            value = JSON.parse(window.localStorage.getItem(key)) || Number(defaultValue)
        } catch (e) {
            value = defaultValue;
        }
        // fallback to 0 if an invalid value is passed in (i.e. NaN)
        return isNaN(+value) ? 0 : value;
    });

    // Update the item on every change of state.
    useEffect(() => {
        window.localStorage.setItem(key, state);
    }, [state]);

    // Return an array with the state and setState function
    return [state, setState]
}
export default useLocalStorage
```

This determines the value of `state`

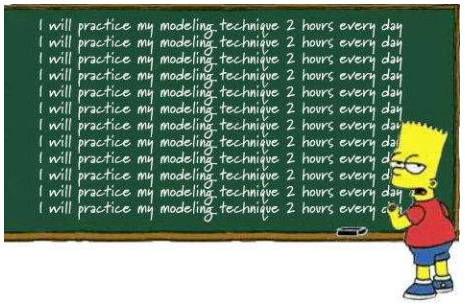
4. Result



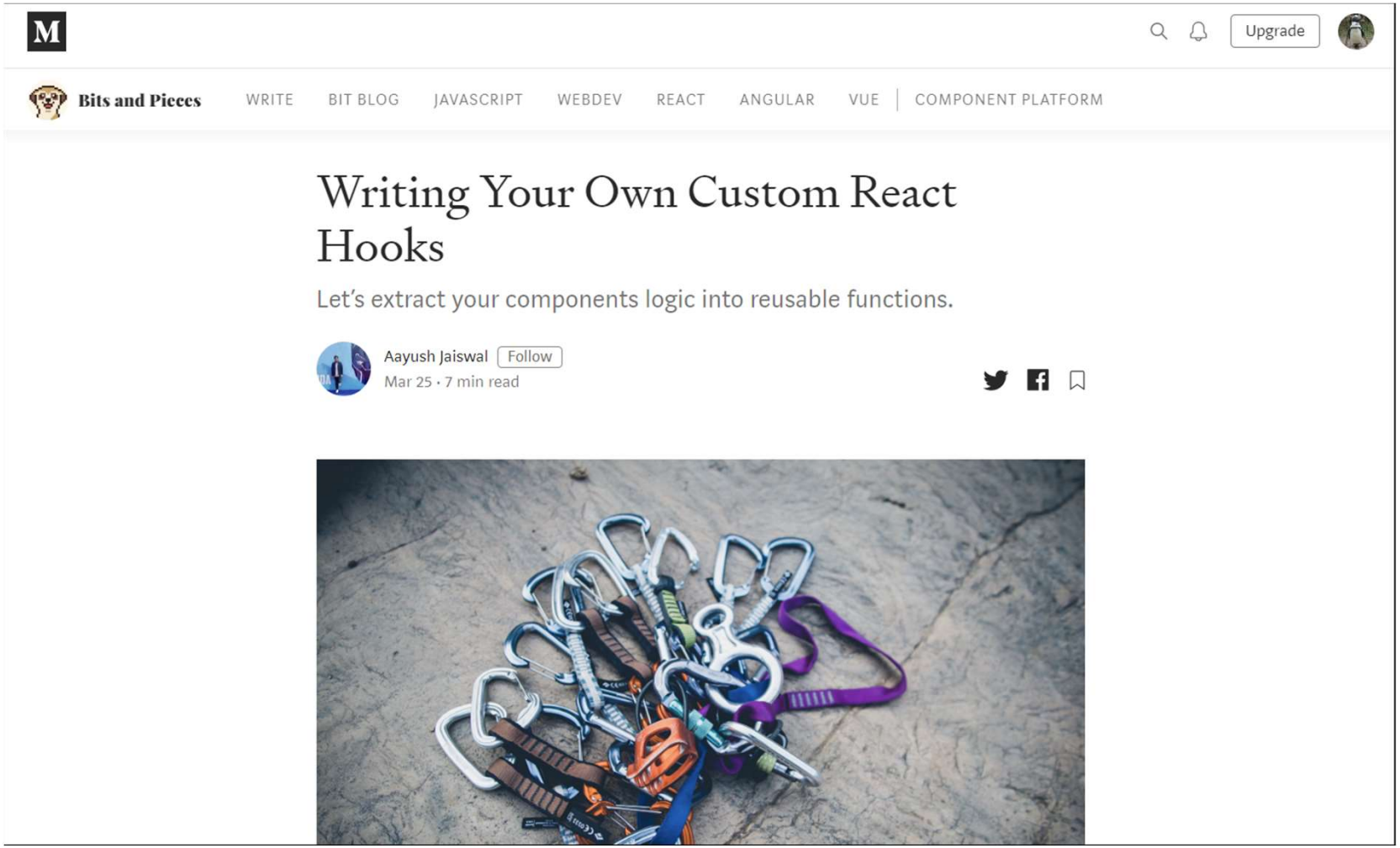
…/720-custom-hooks

Workshop

- Start with example `../720-custom-hooks` and
- Create a new hook that sets a name in localStorage.
- You can retrieve/update the name from different components
  - Use `../hooks/useLocalStorage.js` as an example
- Optional: read https://blog.bitsrc.io/writing-your-own-custom-hooks-4fbcf77e112e
  - Implement the `useUnSplashPhotos()` hook
  - Sign up for an API-key at www.unsplash.com

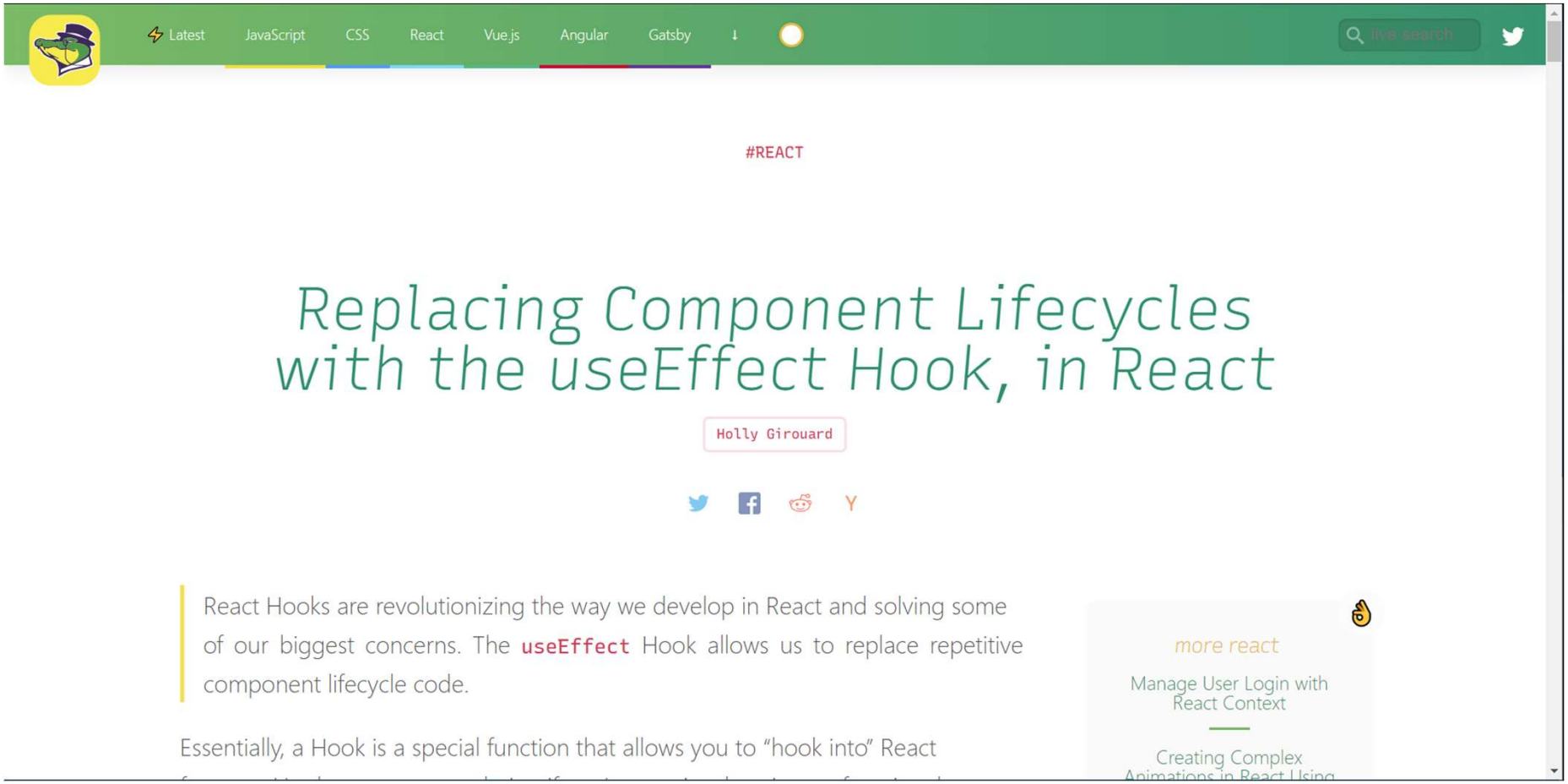


Writing Your Own Custom React Hooks



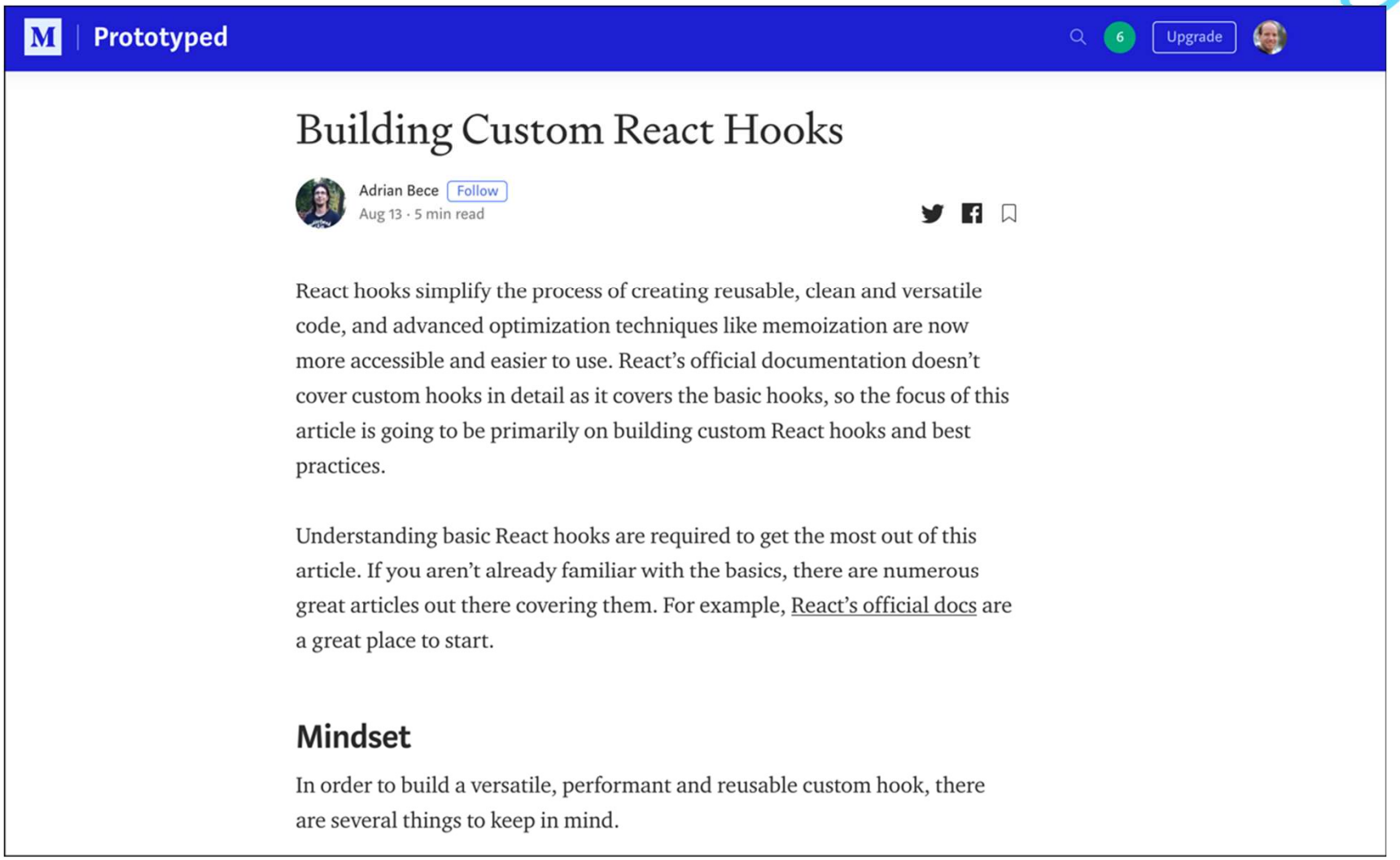
https://blog.bitsrc.io/writing-your-own-custom-hooks-4fbcf77e112e

Replacing Component Lifecycles with the useEffect Hook



https://alligator.io/react/replacing-component-lifecycles-with-useeffect/

Building custom react hooks



Prototyped

6 Upgrade

Building Custom React Hooks

Adrian Bece Follow
Aug 13 · 5 min read

React hooks simplify the process of creating reusable, clean and versatile code, and advanced optimization techniques like memoization are now more accessible and easier to use. React's official documentation doesn't cover custom hooks in detail as it covers the basic hooks, so the focus of this article is going to be primarily on building custom React hooks and best practices.

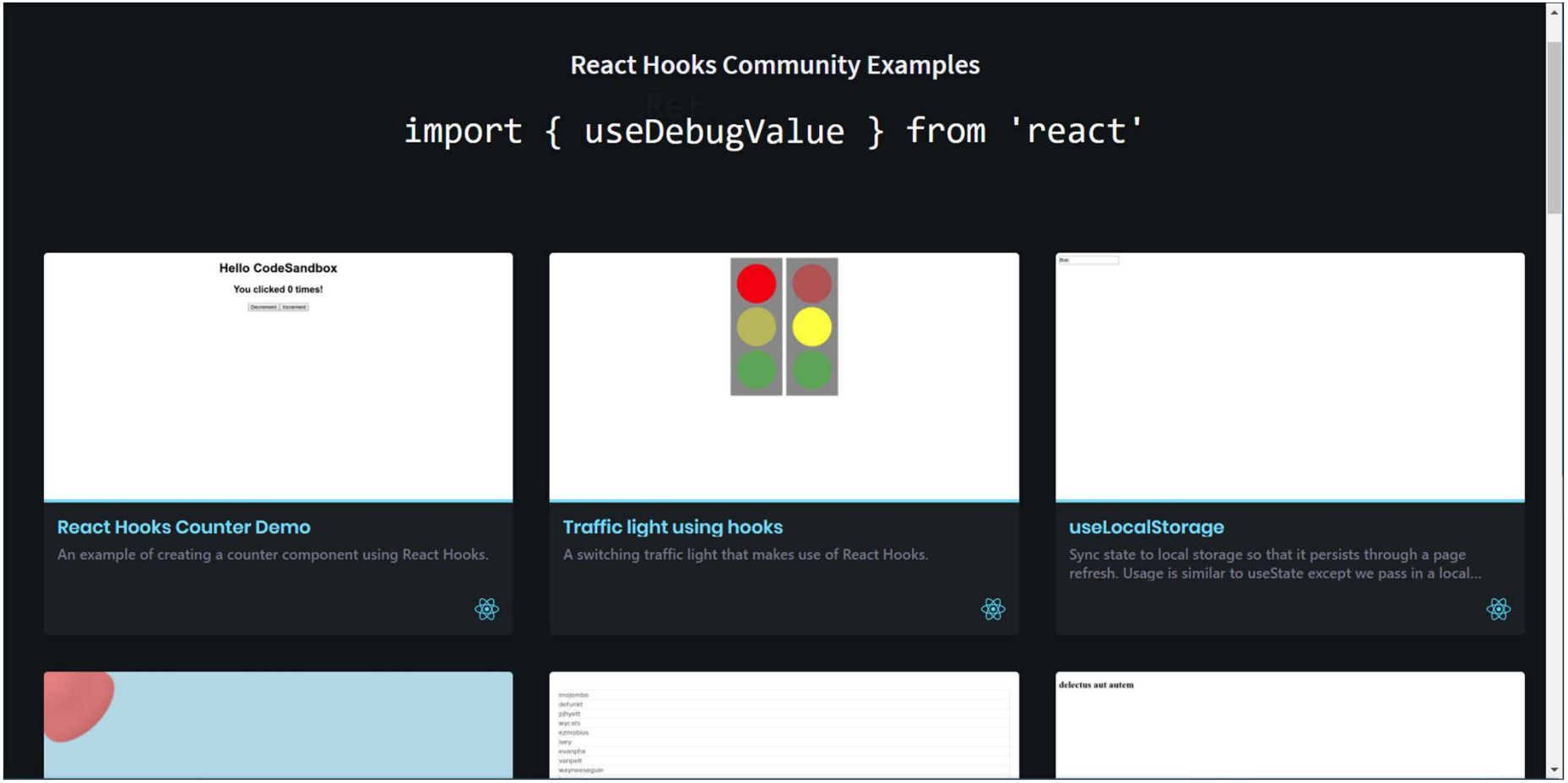
Understanding basic React hooks are required to get the most out of this article. If you aren't already familiar with the basics, there are numerous great articles out there covering them. For example, React's official docs are a great place to start.

Mindset

In order to build a versatile, performant and reusable custom hook, there are several things to keep in mind.

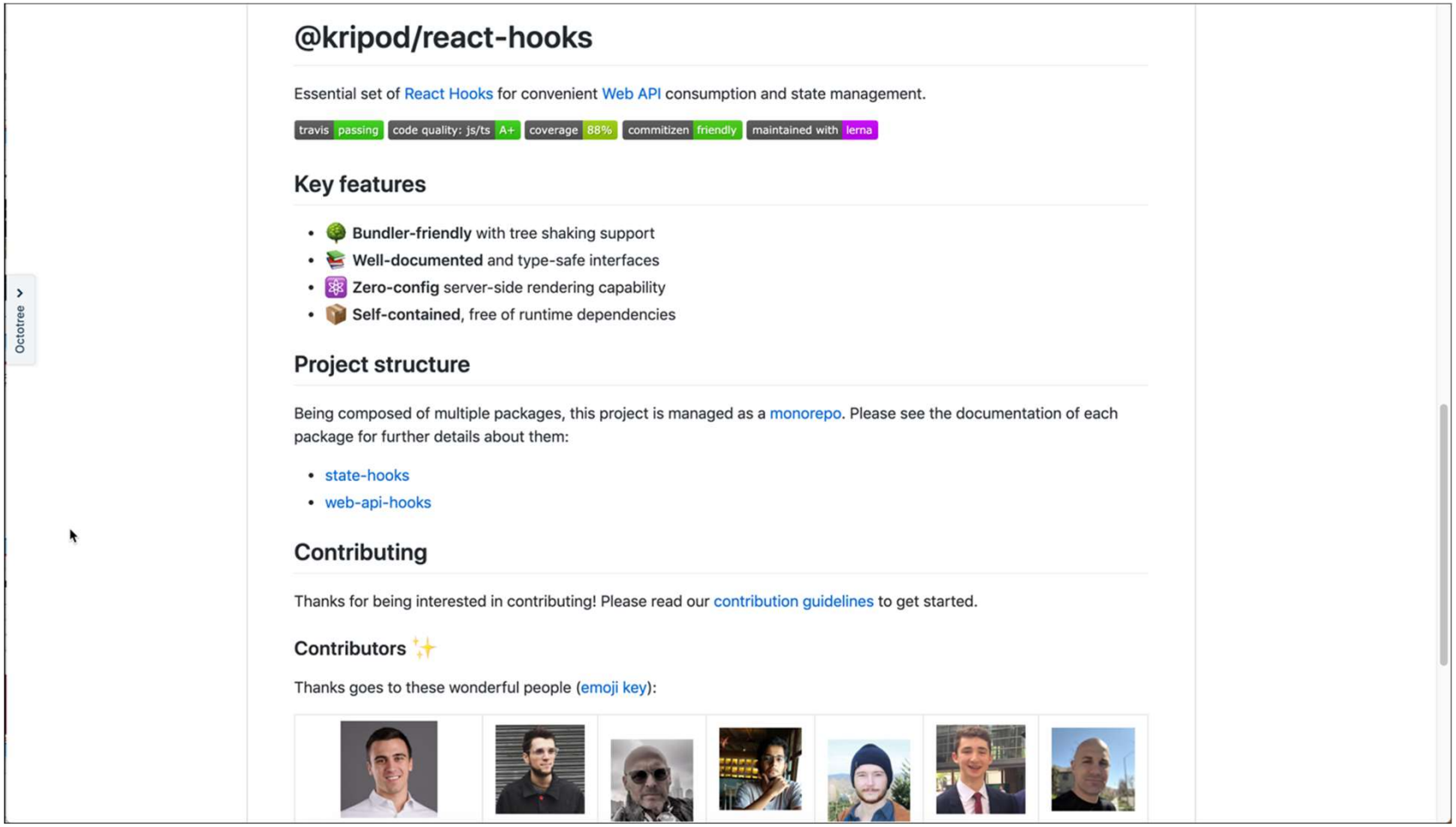
https://medium.com/prototyped/building-custom-react-hooks-f6aad8567825

React Hooks Community Examples



https://codesandbox.io/react-hooks

"Essential set of React Hooks for convenient Web API consumption and state management"



@kripod/react-hooks

Essential set of React Hooks for convenient Web API consumption and state management.

travis passing | code quality: js/ts A+ | coverage 88% | commitizen friendly | maintained with lerna

Key features

- **Bundler-friendly** with tree shaking support
- **Well-documented** and type-safe interfaces
- **Zero-config** server-side rendering capability
- **Self-contained**, free of runtime dependencies

Project structure

Being composed of multiple packages, this project is managed as a monorepo. Please see the documentation of each package for further details about them:

- state-hooks
- web-api-hooks

Contributing

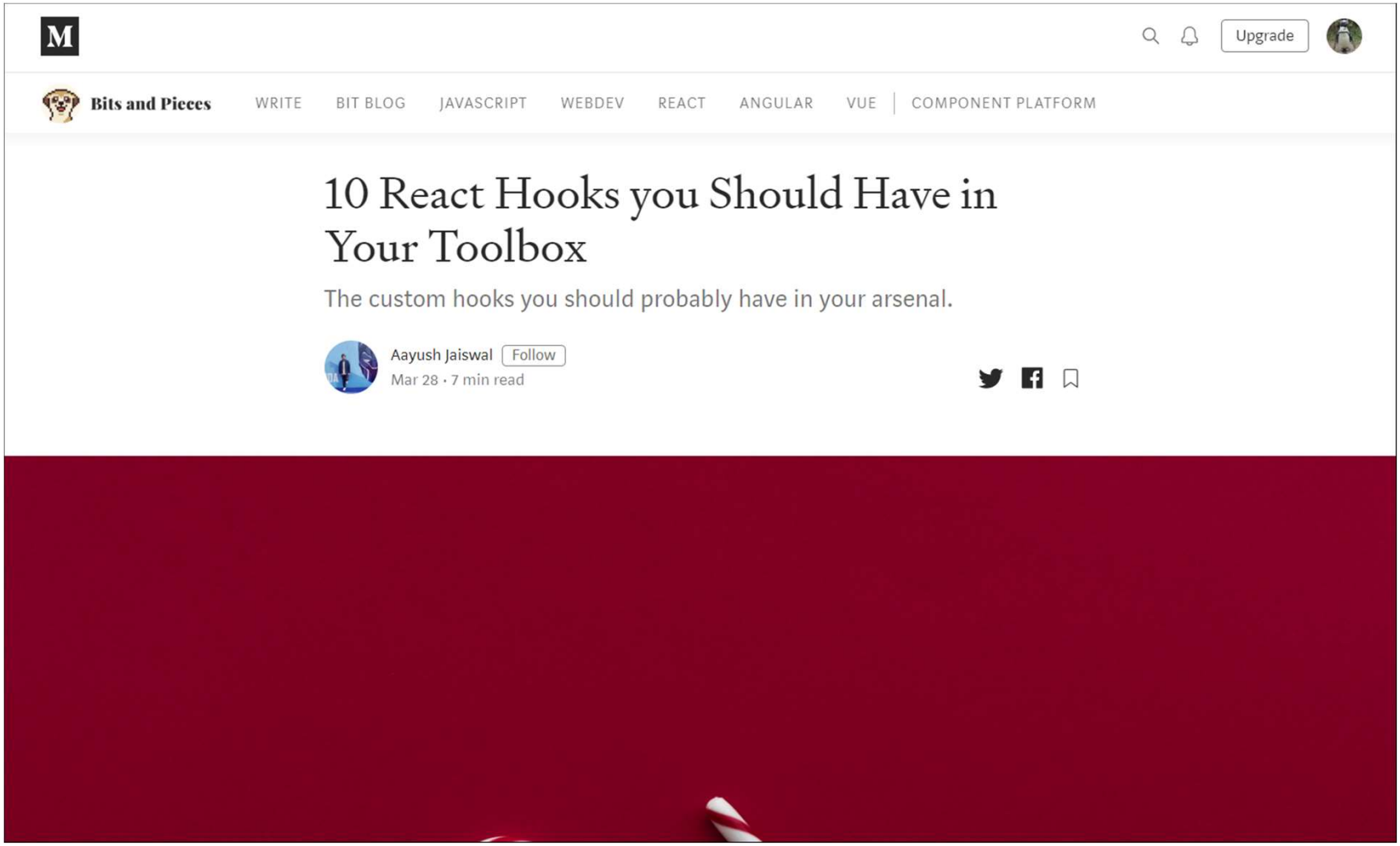
Thanks for being interested in contributing! Please read our contribution guidelines to get started.

Contributors ✨

Thanks goes to these wonderful people (emoji key):

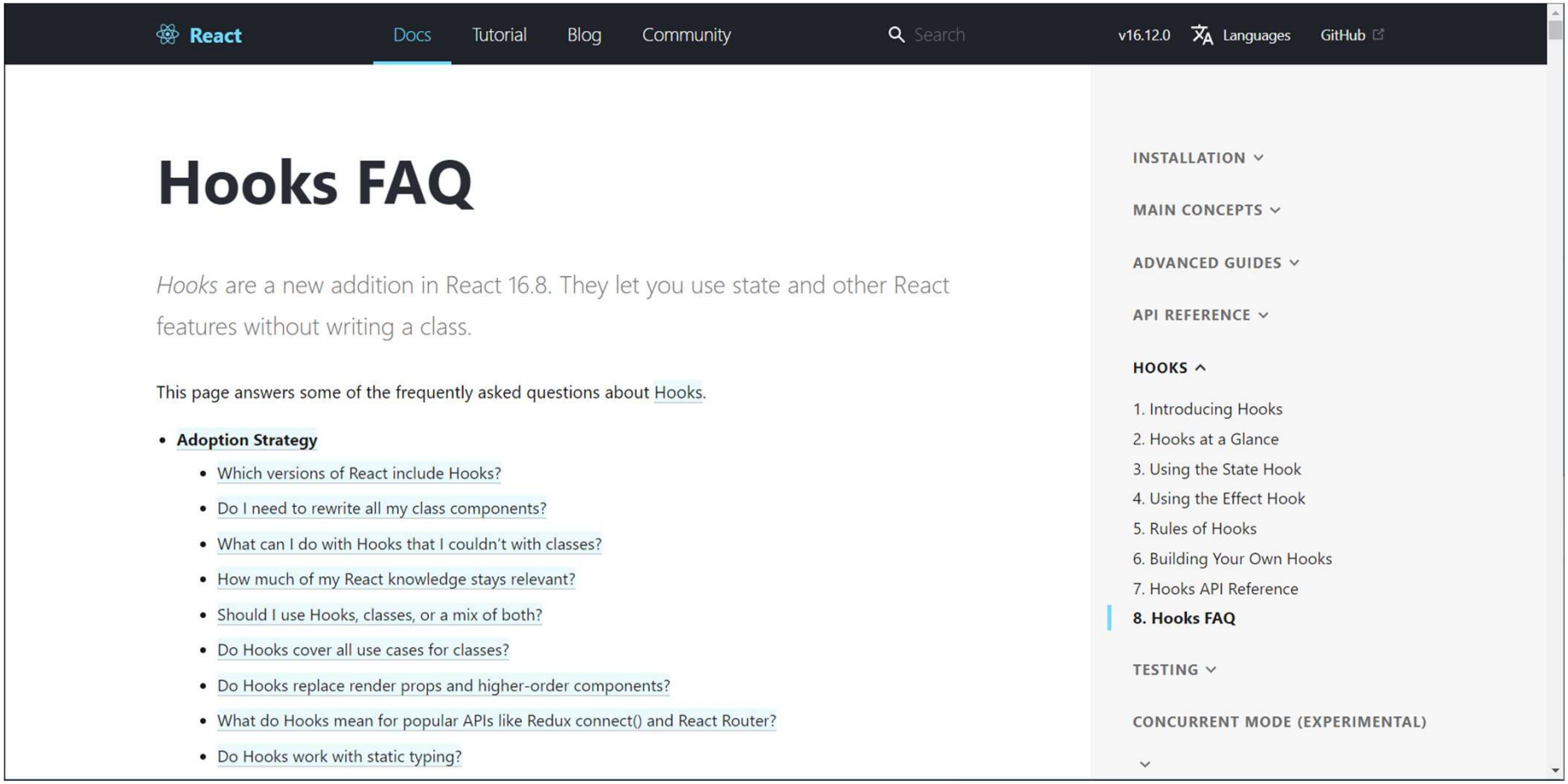
https://github.com/kripod/react-hooks

10 React Hooks you Should Have in Your Toolbox



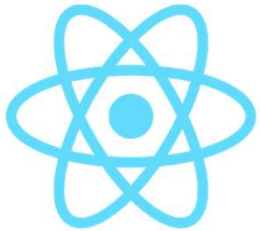
https://blog.bitsrc.io/10-react-custom-hooks-you-should-have-in-your-toolbox-aa27d3f5564d

Official Hooks FAQ



https://reactjs.org/docs/hooks-faq.html

Checkpoint



- You know what hooks are and how they are used in function based components
- You know 2 of the standard React hooks: `useState()` and `useEffect()`
- You can create your own, custom hooks
- You are familiar with some examples on how to use hooks and where to find them