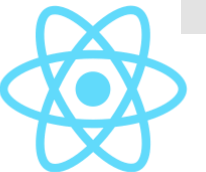


React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

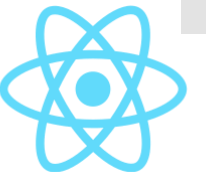


Course Goals



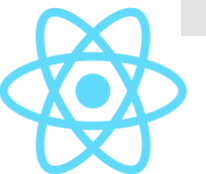
Course Goals

- **Getting started** with functional components and hooks
- Go **beyond** the basic use cases



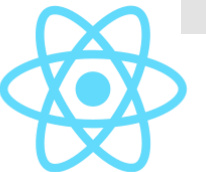
Why this course?

- React hooks are deceptively **simple**
 - But you can still go wrong with them



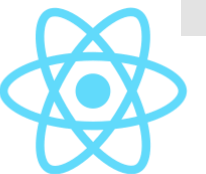
Basic Hooks

- We will cover **the basics** of React hooks first



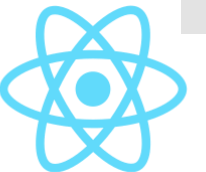
Custom Hooks

- Creating **custom hooks**
 - What can't be done with custom hooks



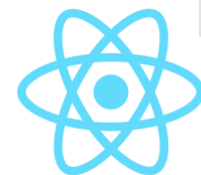
The Rules

- Why do you need to **follow the rules**?
 - Take a look under the covers of hooks and find out



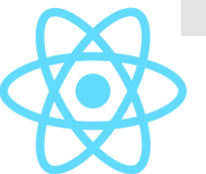
Beyond the Basics

- We are going **beyond** the basic hooks
 - When should you use the more advanced hooks?

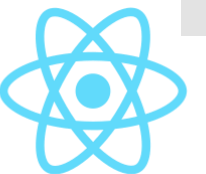


Putting it all together

- Build a more **complex example**
 - A Formik like forms over data library
 - Combining hooks with other React features



See you in the next video

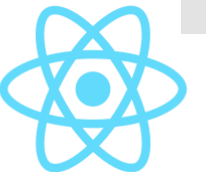


React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb



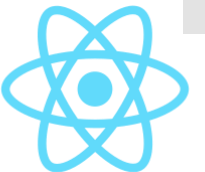
Personal introduction



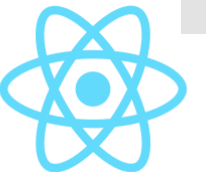


Maurice de Beijer

Independent software developer and trainer



The Netherlands

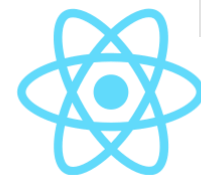








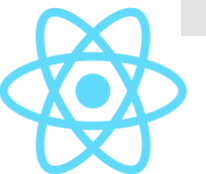
Happily married





Independent software developer & instructor

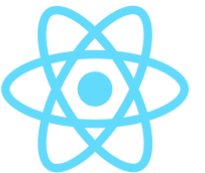
Since 1995



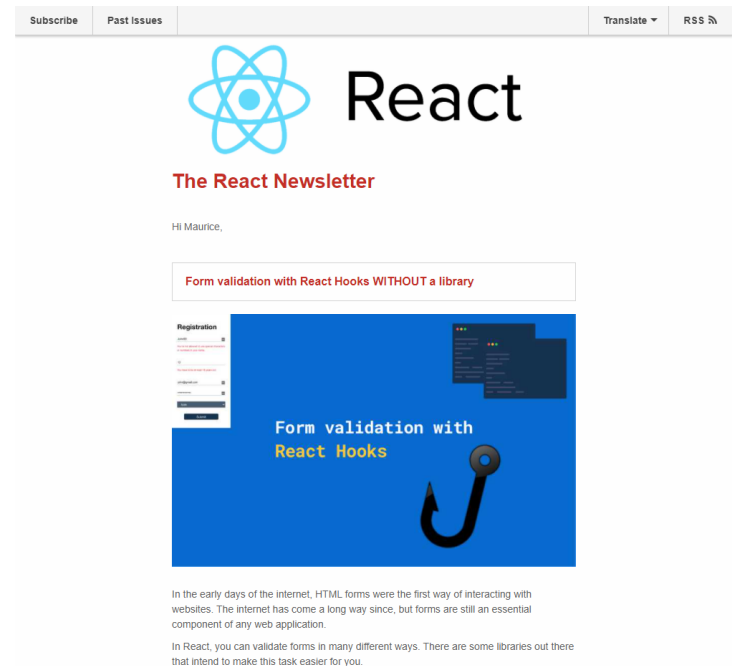
The MVP logo consists of a white diamond shape with a blue shadow, containing the letters 'MVP' in a bold, blue, sans-serif font.

MVP

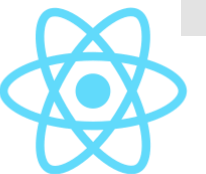
**Microsoft[®]
Most Valuable
Professional**



The React Newsletter



See you in the next video



React Hooks Tips Only the Pros Know

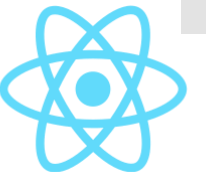
Maurice de Beijer - @mauricedb



Prerequisites

Install Node & NPM

Install the GitHub starter repository

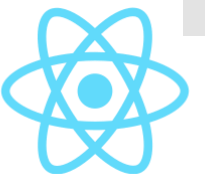


Following Along



```
12 export function usePerson(  
13   initialPerson: Person  
14 ): [Person | null, (person: Person | null) => void] {  
15   const [person, setPerson] = useState<Person | null>(null);  
16   const loaded = useRef(false);  
17  
18   useEffect(() => {  
19     loaded.current = true;  
20  
21     return () => {  
22       loaded.current = false;  
23     };  
24   });  
25  
26   useEffect(() => {  
27     const getPerson = async () => {  
28       await sleep(2500);  
29       const person = await localStorage.getItem<Person>("person");  
30       if (loaded.current) {  
31         setPerson(person ?? initialPerson);  
32       }  
33     };  
34     getPerson();  
35   }, [initialPerson]);
```

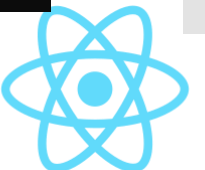
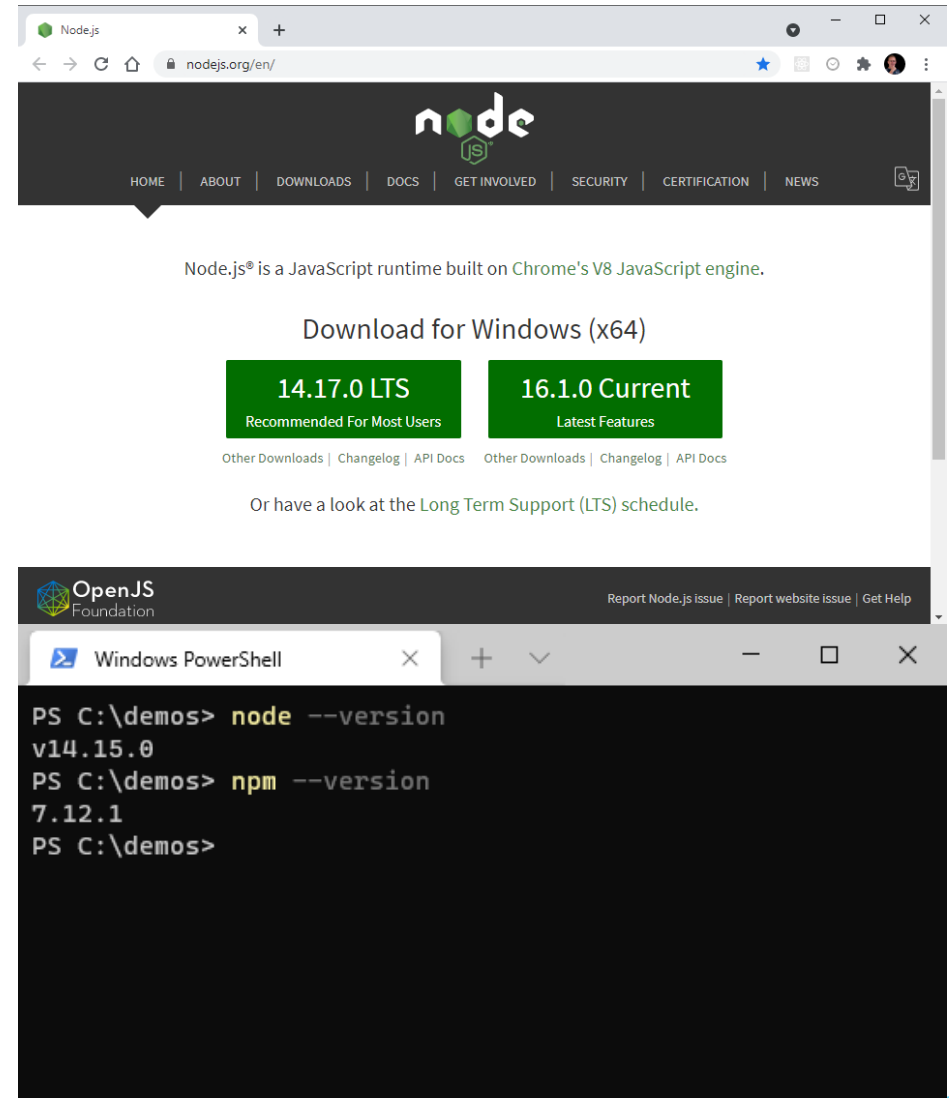
- Slides:
 - <http://theproblemsolver.nl/react-hooks-tips-only-the-pros-know-course.pdf>
- Starter repository:
 - <http://bit.ly/react-hooks-tips-only-the-pros-know-code>



Install Node.js & NPM



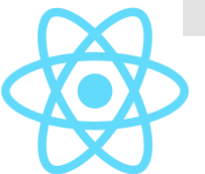
- Minimal:
 - Node version 12
 - NPM version 6



Clone the GitHub Repository



```
Windows PowerShell
PS C:\Temp> git clone git@github.com:mauricedb/react-hooks-tips-only-the-pros-know-course.git
Cloning into 'react-hooks-tips-only-the-pros-know-course'...
remote: Enumerating objects: 112, done.
remote: Counting objects: 100% (112/112), done.
remote: Compressing objects: 100% (72/72), done.
Receiving objects: 55% (62/112)used 107 (delta 32), pack-reused 0 eceiving objects: 38% (43/112)
Receiving objects: 100% (112/112), 91.52 KiB | 520.00 KiB/s, done.
Resolving deltas: 100% (34/34), done.
PS C:\Temp> |
```



Install NPM Packages



```
Windows PowerShell
PS C:\Temp\react-hooks-tips-only-the-pros-know-course> npm install
npm WARN deprecated @types/classnames@2.3.1: This is a stub types definition. classnames provides its own type definitions, so you do not need this installed.

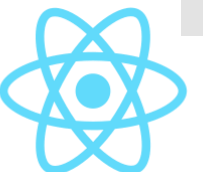
> react-hooks-tips-only-the-pros-know-course@0.0.0 prepare
> husky install

husky - Git hooks installed

added 370 packages, and audited 371 packages in 6s

78 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Temp\react-hooks-tips-only-the-pros-know-course> |
```



Start the application



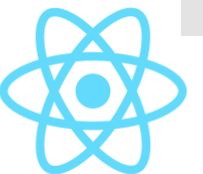
```
C:\WINDOWS\system32\cmd.exe X + v
PS C:\Temp\react-hooks-tips-only-the-pros-know-course> npm start

> react-hooks-tips-only-the-pros-know-course@0.0.0 start
> vite

vite v2.3.3 dev server running at:

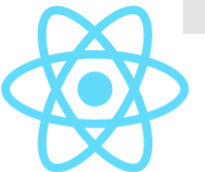
> Local: http://localhost:3000/
> Network: use '--host' to expose

ready in 421ms.
```



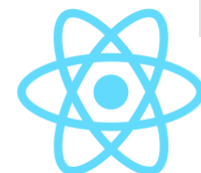
Type it out
by hand?

"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

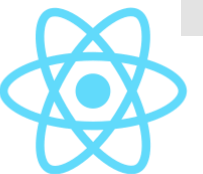


TypeScript

- All slides and exercises use **TypeScript**
 - It's highly recommended in any serious project
- Not used to TypeScript?
 - Just give it a try 😊
 - Or set ***strict*** to ***false*** in the ***tsconfig.json***



See you in the next video



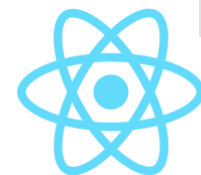
React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb



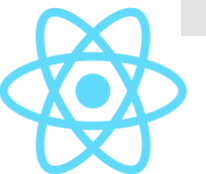
React hooks

The basics



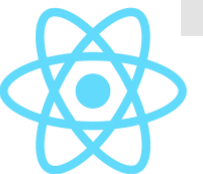
The basics of React hooks

- **Three basic React hooks**
 - `useState()`
 - `useEffect()`
 - `useContext()`
- Hooks can only be **used in functional components**



useState()

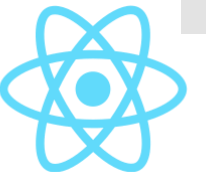
Returns a **stateful value**, and a function to update it



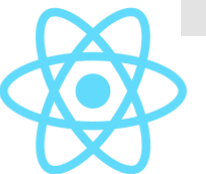
The useState()



```
TS PersonEditor.tsx M X
src > person-editor > TS PersonEditor.tsx > ...
1  import React, { ReactElement, useState } from "react"
2
3  import { LabeledInput } from "../components"
4  import { initialPerson } from "../utils"
5
6  export function PersonEditor(): ReactElement {
7    const [person, setPerson] = useState(initialPerson)
8
9    return (
10     <form>
11       <h2>Person Editor</h2>
12       <LabeledInput
13         label="Firstname:"
14         value={person.firstname}
15         onChange={(e) => {
16           const newPerson = {
17             ... person,
18             firstname: e.target.value,
19           }
20           setPerson(newPerson)
21         }}
22     />
```



See you in the next video



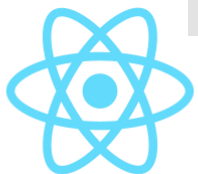
React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb



useState()

With callbacks

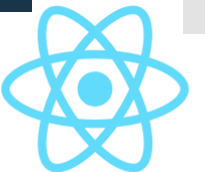


State hook with functions

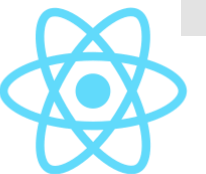


```
TS PersonEditor.tsx M X
src > person-editor > TS PersonEditor.tsx > ...

You, seconds ago | 1 author (You)
1 import React, { ReactElement, useState } from "react"
2
3 import { LabeledInput } from "../components"
4 import { initialPerson } from "../utils"
5
6 export function PersonEditor(): ReactElement {
7   const [person, setPerson] = useState(() => initialPerson)
8
9   return (
10     <form
11       className="person-editor"
12       onSubmit={(e) => {
13         e.preventDefault()
14         alert(`Submitting\n${JSON.stringify(person, null, 2)}`)
15       }}
16     >
17     <h2>Person Editor</h2>
18     <LabeledInput
19       label="Firstname:"
20       value={person.firstname}
21       onChange={(e) =>
22         setPerson((state) => ({ ...state, firstname: e.target.value })
23       )
24     />
```



See you in the next video



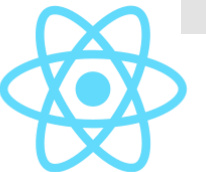
React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb



useEffect()

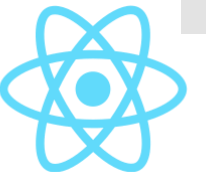
Accepts a function that contains imperative, possibly **effectful** code



Side effect



```
TS PersonEditor.tsx M x
src > person-editor > TS PersonEditor.tsx > PersonEditor > <function> > setPerson() callback > <unknown>
10 function savePerson(person: Person | null): void {
11   console.log("Saving", person)
12   localStorage.setItem("person", person)
13 }
14
15 export function PersonEditor(): ReactElement {
16   const [person, setPerson] = useState<Person | null>(null)
17
18   useEffect(() => {
19     const getPerson = async () => {
20       const person = await localStorage.getItem<Person>("person")
21       setPerson(person ?? initialPerson)
22     }
23
24     getPerson()
25   }, [])
26
27   useEffect(() => {
28     savePerson(person)
29   }, [person])
30
31   if (!person) {
32     return <Loading />
33   }
```

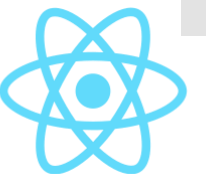


Side effect with cleanup

```
useEffect(() => {  
  const handle = setInterval(() => {  
    // Do something every second  
  }, 1000)  
  
  return () => clearInterval(handle)  
}, [])
```



See you in the next video



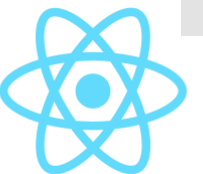
React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb



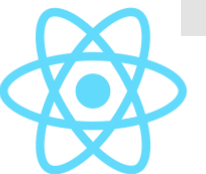
useContext()

Accepts a **context object** and returns it's current value



Context Provider

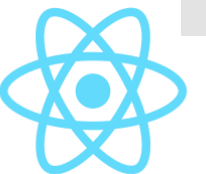
```
18 export const themeContext = createContext<ThemeContext>({
19   style: undefined,
20   setStyle: () => void null,
21 })
22
23 export function ThemeProvider({ children }: Props): ReactElement {
24   const [style, setStyle] = useState<CSSProperties>()
25
26   return (
27     <themeContext.Provider value={{ style, setStyle }}>
28       {children}
29     </themeContext.Provider>
30   )
31 }
```



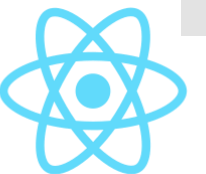
useContext()



```
9  export function App(): ReactElement {  
10    const { style } = useContext(themeContext)  
11  
12    return (  
13      <div className="container" style={style}>  
14        <BrowserRouter>  
15          <AppNavbar />  
16          <Routes />  
17        </BrowserRouter>  
18      </div>  
19    )  
20  }
```



See you in the next video

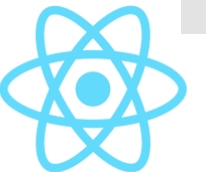


React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

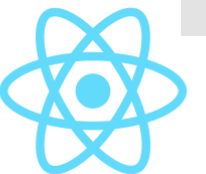


Custom hooks



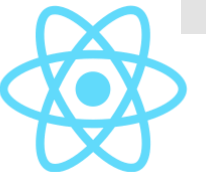
Custom hooks

- Make component code **reusable**
- Great for **extracting code** from components
 - Even when reuse is not a goal
- Custom hooks can **use other React hooks** as needed



Creating the usePerson() hook

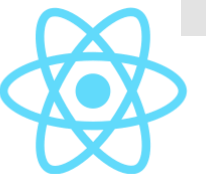
```
TS PersonEditor.tsx TS usePerson.ts X
src > person-editor > TS usePerson.ts > ...
You, 2 minutes ago | 1 author (You)
1 import localforage from "localforage";
2 import { useEffect, useState } from "react";
3 import { Person } from "../types/person";
4
5 function savePerson(person: Person | null): void {
6   console.log("Saving", person);
7
8   localforage.setItem("person", person);
9 }
10
11 export function usePerson(
12   initialPerson: Person
13 ): [Person | null, (person: Person | null) => void] {
14   const [person, setPerson] = useState<Person | null>(null);
15
16   useEffect(() => {
17     const getPerson = async () => {
18       const person = await localforage.getItem<Person>("person");
19       setPerson(person ?? initialPerson);
20     };
21     getPerson();
22   }, [initialPerson]);
23
24   useEffect(() => {
25     savePerson(person);
26   }, [person]);
27
28   return [person, setPerson];
29 }
```



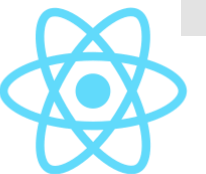
Using the
usePerson()
hook



```
TS PersonEditor.tsx × TS usePerson.ts
src > person-editor > TS PersonEditor.tsx > PersonEditor
5
6 import { LabeledInput, Loading } from "../components";
7 import { initialPerson } from "../utils";
8 import { usePerson } from "./usePerson";
9
10 export function PersonEditor(): ReactElement {
11   const [person, setPerson] = usePerson(initialPerson);
12
13   if (!person) {
14     return <Loading />;
15   }
16
17   return (
18     <form
19       className="person-editor"
```

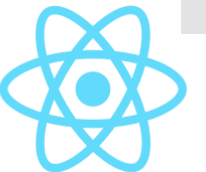


See you in the next video



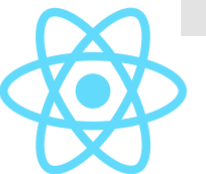
Rules of Hooks

Custom hooks names



Custom hooks names

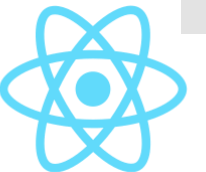
👉 Custom hooks must be functions named ``use....`` 👈





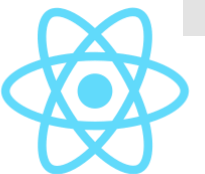
useLayoutEffect()

Like `useEffect()` but **fires synchronously** after all DOM mutations



useEffect() versus useLayoutEffect()

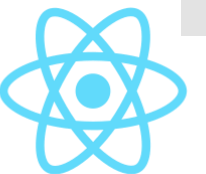
- **useLayoutEffect()** executes **synchronously**
 - After the render but before the component is painted
- While **useEffect()** executes **asynchronously**
 - After the component is painted on screen
- **Normally useEffect() is what you need**
 - Use useLayoutEffect() when you want the synchronous behavior



useLayoutEffect()
to track the
loaded state

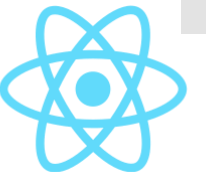


```
12 // Note: This is wrong as this value is shared between all instances
13 let loaded = false;
14
15 export function usePerson(
16   initialPerson: Person
17 ): [Person | null, (person: Person | null) => void] {
18   const [person, setPerson] = useState<Person | null>(null);
19
20   useLayoutEffect(() => {
21     loaded = true;
22
23     return () => {
24       loaded = false;
25     };
26   });
27
28   useEffect(() => {
29     const getPerson = async () => {
30       await sleep(2500);
31       const person = await localforage.getItem<Person>("person");
32       if (loaded) {
33         setPerson(person ?? initialPerson);
34       }
35     };
36     getPerson();
37   }, [initialPerson]);
```



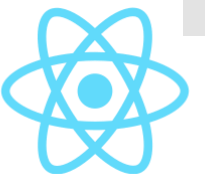
useRef()

useRef() returns a mutable ref object whose **.current property** is initialized to the passed argument



useRef()

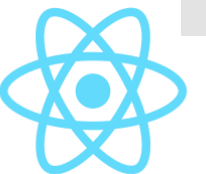
- One of it's main purposes is to get a **reference to a DOM element**
- But can be used to keep a **reference to any state**
 - Holds the same state with each render
 - Updating the state doesn't trigger a render
- useRef() is stable and **doesn't need to be added as a dependency**
 - Great way to share values between useEffect() functions



useRef() with a HTML element

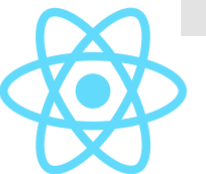


```
TS demo.tsx ×
src > components > TS demo.tsx > ...
1  import React, { ReactElement, useEffect, useRef } from "react";
2
3  export function Demo(): ReactElement {
4    const ref = useRef<HTMLInputElement>(null);
5
6    useEffect(() => {
7      if (ref.current) {
8        ref.current.value = "An unbound input";
9      }
10   }, []);
11
12   return (
13     <div>
14       <input type="text" ref={ref} />
15     </div>
16   );
17 }
```



useRef() with a component

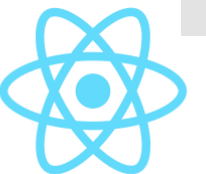
```
TS demo.tsx X TS InputField.tsx
src > components > TS demo.tsx > ...
1  import React, { ReactElement, useEffect, useRef } from "react";
2  import { InputField } from "../InputField";
3
4  export function Demo(): ReactElement {
5      const ref = useRef<HTMLInputElement>(null);
6
7      useEffect(() => {
8          if (ref.current) {
9              ref.current.value = "It's value";
10          }
11      }, []);
12
13      return (
14          <InputField
15              label={
16                  <div>
17                      An <b>unbound</b> input
18                  </div>
19              }
20              ref={ref}
21          >
22      );
23  }
```



Forwarding useRef()

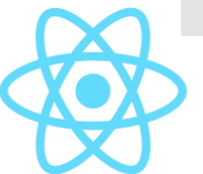


```
TS demo.tsx  TS InputField.tsx X
src > components > TS InputField.tsx > ...
1  import React, { InputHTMLAttributes, ReactElement, ReactNode } from "react";
2
3  interface Props extends InputHTMLAttributes<HTMLInputElement> {
4    label: ReactNode;
5  }
6
7  export const InputField = React.forwardRef<HTMLInputElement, Props>(
8    ({ label, ...props }, ref): ReactElement => (
9      <div>
10        <label>{label}</label>
11        <input type="text" ref={ref} { ... props } />
12      </div>
13    )
14  );
```



useRef() for generic state

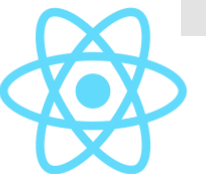
- With useRef() you can maintain **any state** for a component
 - When you don't want a render on updates



Using useRef() to maintain state

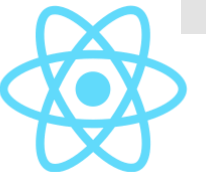


```
12 export function usePerson(  
13   initialPerson: Person  
14 ): [Person | null, (person: Person | null) => void] {  
15   const [person, setPerson] = useState<Person | null>(null);  
16   const loaded = useRef(false);  
17  
18   useEffect(() => {  
19     loaded.current = true;  
20  
21     return () => {  
22       loaded.current = false;  
23     };  
24   });  
25  
26   useEffect(() => {  
27     const getPerson = async () => {  
28       await sleep(2500);  
29       const person = await localforage.getItem<Person>("person");  
30       if (loaded.current) {  
31         setPerson(person ?? initialPerson);  
32       }  
33     };  
34     getPerson();  
35   }, [initialPerson]);
```



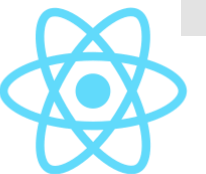
useRef() and dependencies

👉 The `useRef()` object is not required in dependency arrays 👉



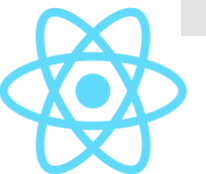


Debounce and useEffect()



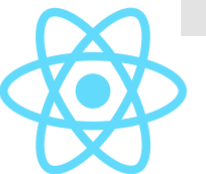
Debounce

- Creating a **debounce hook with useEffect()** is easy
 - Execute the code via a `setTimeout()`
 - Cancel the `setTimout()` in the cleanup




useDebounce()

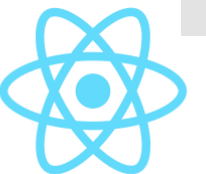
```
TS useDebounce.ts X
src > components > TS useDebounce.ts > ...
1  import { useEffect } from "react";
2
3  export function useDebounce(fn: () => void, timeout: number): void {
4    useEffect(() => {
5      const handle = setTimeout(() => fn(), timeout);
6
7      return () => clearTimeout(handle);
8    }, [fn, timeout]);
9  }
```



Using useDebounce()

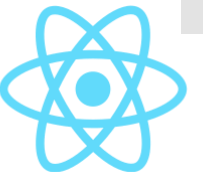


```
27   useEffect(() => {
28     const getPerson = async () => {
29       // await sleep(2500);
30       const person = await localforage.getItem<Person>("person");
31       if (loaded.current) {
32         setPerson(person ?? initialPerson);
33       }
34     };
35      getPerson();
36   }, [initialPerson]);
37
38   useDebounce(() => {
39     savePerson(person);
40   }, 1000);
41
42   return [person, setPerson];
```



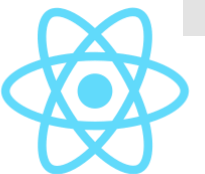
useCallback()

Returns a memoized callback



useCallback()

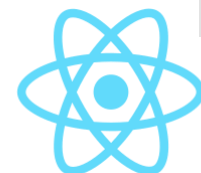
- **Pass a function and dependencies** to useCallback()
 - Returns the same function reference with the same dependencies
- Useful when **passing callbacks to child** components
 - Or other hooks
- **Note:** useMemo() can be used for values



useCallback()

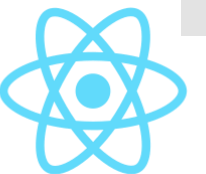


```
44   useDebounce(  
45     useCallback(() => {  
46       savePerson(person);  
47     }, [person]),  
48     1000  
49   );  
50  
51   return [person, setPerson];  
52 }
```

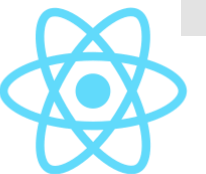




useMemo()

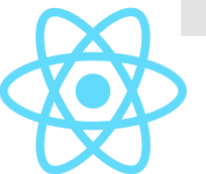


useDebugValue()





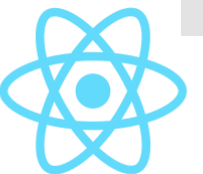
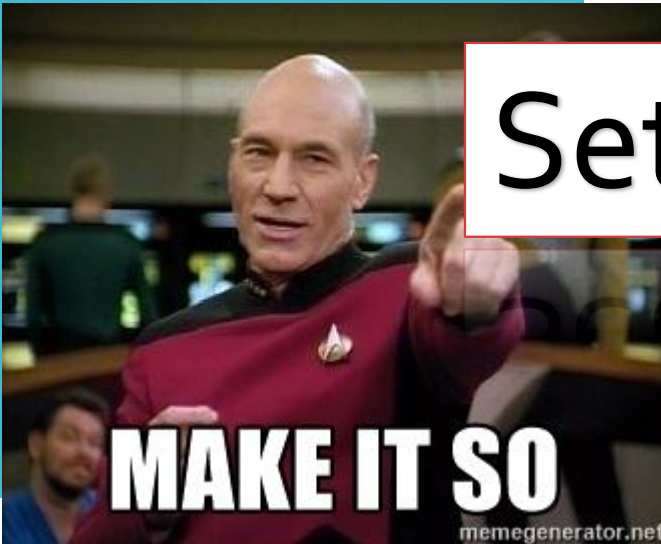
useThrottle()



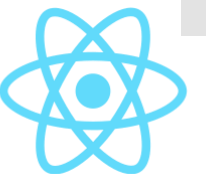
useThrottle()

```
1 import { useEffect, useRef } from "react";
2
3 export function useThrottle(fn: () => void, interval: number): void {
4   const ref = useRef(fn);
5   ref.current = fn;
6
7   useEffect(() => {
8     const handle = setInterval(() => {
9       ref.current();
10    }, interval);
```

Set ref.current to null!

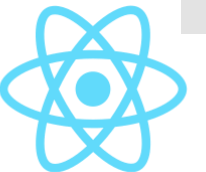


Unmount and useEffect()



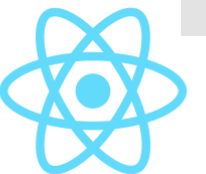
Unmount

- Creating an **unmount hook with useEffect()** is easy
 - Execute the passed code in the effect cleanup
 - Use a ref so there are no dependencies



useWillUnmount()

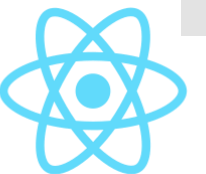
```
TS useWillUnmount.ts X TS demo.tsx
src > components > TS useWillUnmount.ts > ...
1  import { useEffect, useRef } from "react";
2
3  export function useWillUnmount(fn: () => void): void {
4    const functionRef = useRef(fn);
5    functionRef.current = fn;
6
7    useEffect(() => {
8      return () => functionRef.current();
9    }, []);
10 }
```



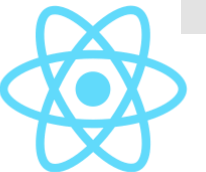
Save changes
when unmounting



```
39   useDebounce(() => {  
40     |   savePerson(person);  
41   }, 1000);  
42  
43   useWillUnmount(() => {  
44     |   savePerson(person);  
45   });  
46  
47   return [person, setPerson];  
48 }
```

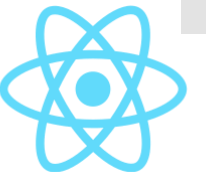


Rules of Hooks



Rules of hooks

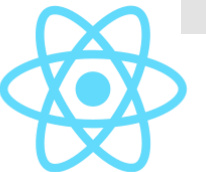
- **Only Call Hooks at the Top Level**
 - Don't call Hooks inside loops, conditions, or nested functions
- **Only Call Hooks from React functions**
 - Call Hooks from React function components
 - Call Hooks from custom Hooks





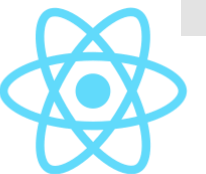
More complex state

With `useState()`



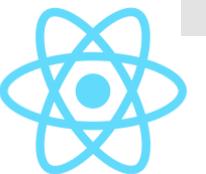
More complex state

- Use as **many useState() hooks** as needed
 - One for the person object
 - A second for the dirty and valid states



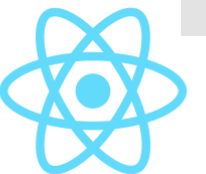
Two useState() hooks

```
14 interface FormState {  
15   isDirty: boolean;  
16   isValid: boolean;  
17 }  
18  
19 type UsePersonReturnType = [  
20   Person | null,  
21   (person: Person | null) => void,  
22   FormState  
23 ];  
24  
25 export function usePerson(initialPerson: Person): UsePersonReturnType {  
26   const [person, setPersonState] = useState<Person | null>(null);  
27   const [formState, setFormState] = useState({  
28     isDirty: false,  
29     isValid: true,  
30   });  
31  
32   const loaded = useRef(false);
```



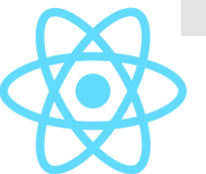
Multiple updates

```
42  useEffect(() => {
43    const getPerson = async () => {
44      // await sleep(2500);
45      const person = await localforage.getItem<Person>("person");
46      if (loaded.current) {
47        setPersonState(person ?? initialPerson);
48      }
49    };
50    getPerson();
51  }, [initialPerson]);
52
53  useDebounce(() => {
54    savePerson(person);
55  }, 1000);
56
57  useWillUnmount(() => {
58    savePerson(person);
59  });
60
61  const setPerson = (person: React.SetStateAction<Person | null>) => {
62    setPersonState(person);
63    setFormState((s) => ({ ...s, isDirty: true }));
64  };
65
66  return [person, setPerson, formState];
67 }
```



Exposing more data from a custom hook

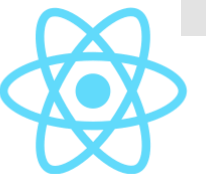
```
10 export function PersonEditor(): ReactElement {  
11   const [person, setPerson, { isDirty }] = usePerson(initialPerson);  
12  
13   if (!person) {  
14     return <Loading />;  
15   }
```



Disable save
button if there
are no changes

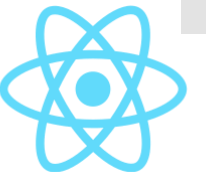


```
61     <button className="btn btn-primary" disabled={!isDirty}>  
62       Save  
63     </button>  
64     <hr />  
65     <pre>{JSON.stringify(person, null, 2)}</pre>  
66   </form>
```



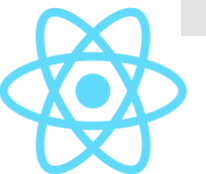
useReducer()

An alternative to useState()



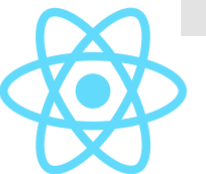
useReducer()

- **useReducer()** is the **more powerful** brother of `useState()`
 - Internally `useState()` is just a special `useReducer()`
- **Dispatch action objects** and use a reducer function to create state
 - Very similar to how Redux works
- The state is still **tied to a component**
 - Not global like with Redux



The personEditorReducer() 1/2

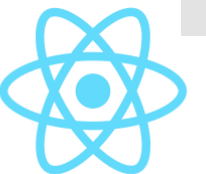
```
1  import { Person } from "../types/person";
2
3  export interface FormState {
4    isDirty: boolean;
5    isValid: boolean;
6  }
7
8  interface ReducerState {
9    person: Person | null;
10   formState: FormState;
11 }
12
13 interface SetPersonAction {
14   type: "set-initial-person";
15   payload: Person;
16 }
17
18 interface SetPropertyAction {
19   type: "set-property";
20   payload: { name: string; value: unknown };
21 }
22
23 type SomeAction = SetPersonAction | SetPropertyAction;
```



The personEditorReducer() 2/2

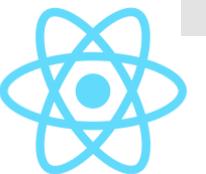


```
25 export function personEditorReducer(  
26   state: ReducerState,  
27   action: SomeAction  
28 ): ReducerState {  
29   switch (action.type) {  
30     case "set-initial-person":  
31       return { ...state, person: action.payload };  
32     case "set-property":  
33       return {  
34         ...state,  
35         formState: { ...state.formState, isDirty: true },  
36         person: {  
37           ...(state.person ?? {  
38             id: 0,  
39             firstname: "",  
40             surname: "",  
41             address: "",  
42             balance: 0,  
43             email: "",  
44             picture: "",  
45             phone: "",  
46           })),  
47           [action.payload.name]: action.payload.value,  
48         },  
49       };  
50     }  
51   }  
52   return state;  
53 }
```



The usePerson() hook 1/2

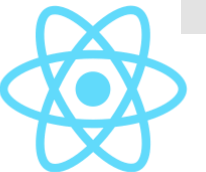
```
21 type UsePersonReturnType = [  
22   Person | null,  
23   (name: keyof Person, value: unknown) => void,  
24   FormState  
25 ];  
26  
27 export function usePerson(initialPerson: Person): UsePersonReturnType {  
28   const [state, dispatch] = useReducer(personEditorReducer, {  
29     person: null,  
30     formState: { isDirty: false, isValid: true },  
31   });  
32  
33   const loaded = useRef(false);  
34  
35   useEffect(() => {  
36     loaded.current = true;  
37  
38     return () => {  
39       loaded.current = false;  
40     };  
41   });
```



The usePerson() hook



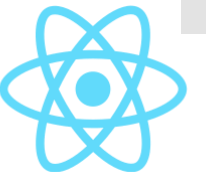
```
43   useEffect(() => {
44     const getPerson = async () => {
45       // await sleep(2500);
46       const person = await localforage.getItem<Person>("person");
47       if (loaded.current) {
48         dispatch({
49           type: "set-initial-person",
50           payload: person ?? initialPerson,
51         });
52       }
53     };
54     getPerson();
55   }, [initialPerson]);
56
57   useDebounce(() => {
58     savePerson(state.person);
59   }, 1000);
60
61   useWillUnmount(() => {
62     savePerson(state.person);
63   });
64
65   const setProperty = (name: keyof Person, value: unknown) => {
66     dispatch({ type: "set-property", payload: { name, value } });
67   };
68
69   return [state.person, setProperty, state.formState];
70 }
```



The Person Editor form

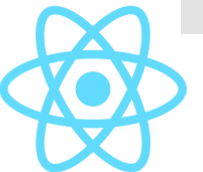


```
10 export function PersonEditor(): ReactElement {
11   const [person, setProperty, { isDirty }] = usePerson(initialPerson);
12
13   if (!person) {
14     return <Loading />;
15   }
16
17   return (
18     <form
19       className="person-editor"
20       onSubmit={(e) => {
21         e.preventDefault();
22         alert(`Submitting\n${JSON.stringify(person, null, 2)}`);
23       }}
24     >
25       <h2>Person Editor</h2>
26       <LabeledInput
27         label="Firstname:"
28         value={person.firstname}
29         onChange={(e) => {
30           setProperty("firstname", e.target.value);
31         }}
32       />
33       <LabeledInput
34         label="Surname:"
35         value={person.surname}
36         onChange={(e) => {
37           setProperty("surname", e.target.value);
38         }}
39     />
```

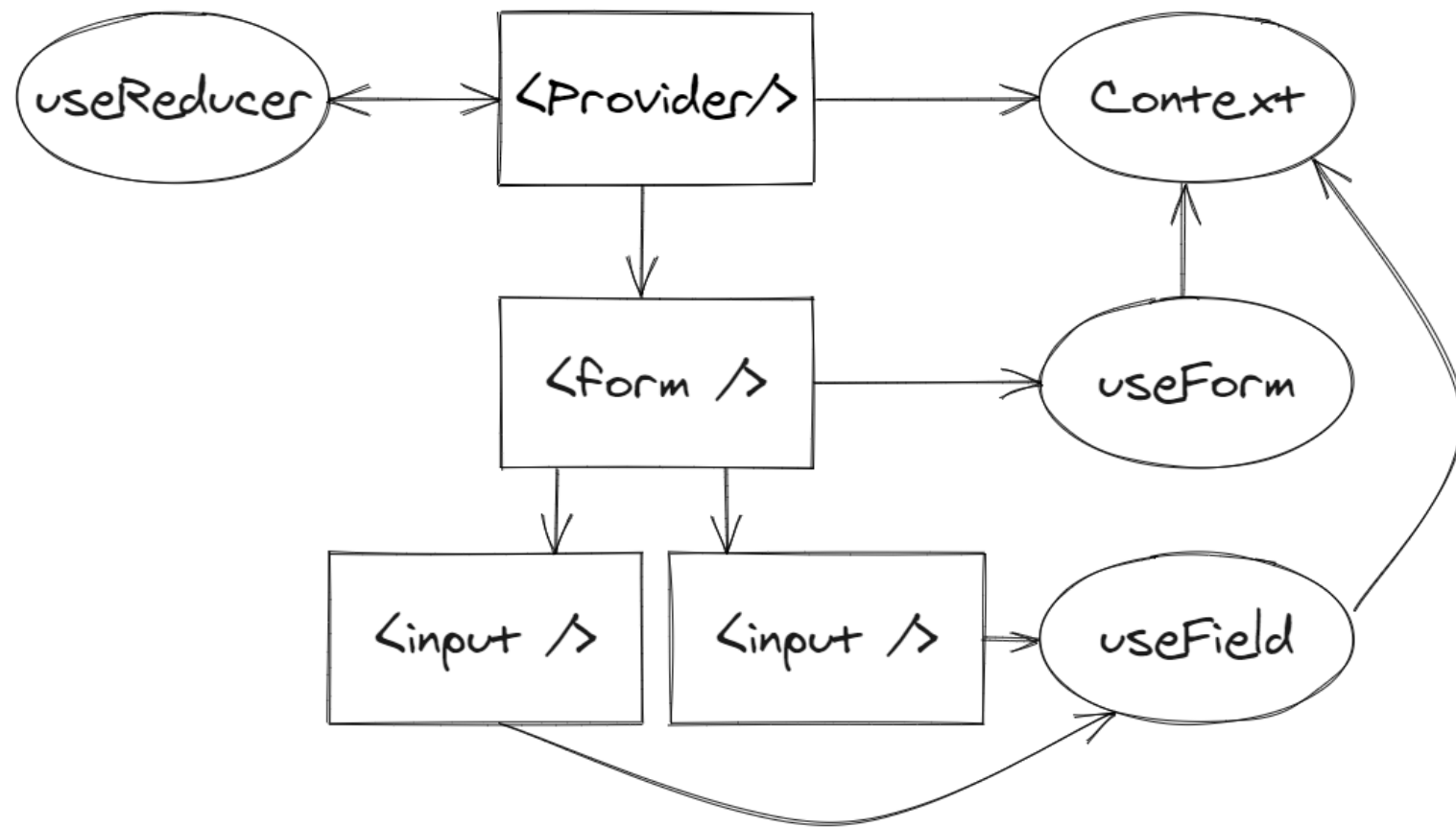


Kimrof

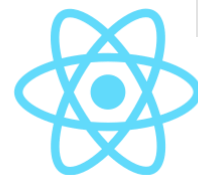
A Formik like forms utility



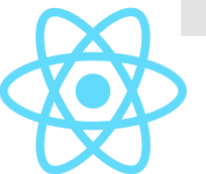
Kimrof



Made with Excalidraw

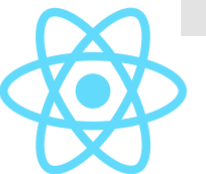


Context and Provider



Context and Provider

- The context will **manage the internal state**
 - The object to edit
 - Updates to it's properties
 - Validity/dirty states
 - Form submission
- The provider component will **make this available**
 - To the various hooks



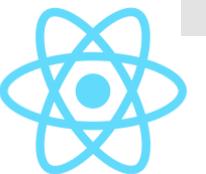
The kimrofContext

```
TS KimrofContext.ts X TS KimrofPersonEditor.tsx TS Kimrof.tsx
src > kimrof-person-editor > kimrof > TS KimrofContext.ts > ...
You, seconds ago | 1 author (You)
1 import React from "react";
2
3 import { KimrofObject, KimrofProperty } from "./Types";
4
You, seconds ago | 1 author (You)
5 export interface KimrofContext {
6   values: KimrofObject;
7 }
8
9 export const kimrofContext = React.createContext<KimrofContext>({
10   values: {},
11 });
```



The Kimrof provider

```
5 import { KimrofObject, KimrofProperty } from "../Types";
6 import { KimrofContext, kimrofContext } from "../KimrofContext";
7
8 You, seconds ago | 1 author (You)
9 interface Props<TData> {
10   children: ReactNode;
11   initialValues: TData;
12 }
13 export function Kimrof<TData extends KimrofObject>({
14   children,
15   initialValues,
16 }: Props<TData>): ReactElement {
17   const values = initialValues;
18
19   const context: KimrofContext = React.useMemo(
20     () => ({
21       values,
22     }),
23     [values]
24   );
25
26   return (
27     <kimrofContext.Provider value={context}>{children}</kimrofContext.Provider>
28   );
29 }
```



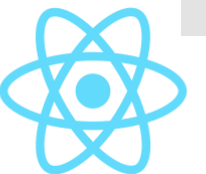
Adding the provider



```
7 import { PersonEditor } from "../PersonEditor";
8 import { Kimrof } from "../kimrof";
9
10 export function KimrofPersonEditor(): ReactElement {
11   return (
12     <Kimrof initialValues={initialPerson as IndexedPerson}>
13       <PersonEditor />
14     </Kimrof>
15   );
16 }
```

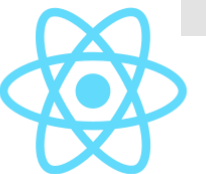


Displaying the values



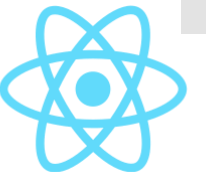
Displaying the form values

- The **useKimrofField()** hook allows **input components** to work
 - Returns an object with a value and onChange prop
- **Spread the result** into any HTML `<input />`
- **Note:** In this step we are only adding the value props



The useKimrofField() hook

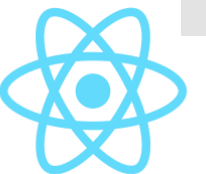
```
3 import { KimrofProperty } from "../Types";
4 import { kimrofContext } from "../KimrofContext";
5
6 You, seconds ago | 1 author (You)
7 interface UseKimrofField {
8   value: KimrofProperty;
9 }
10 export const useKimrofField = (name?: string): UseKimrofField => {
11   if (!name) {
12     throw new Error("The name prop is required");
13   }
14   const { values } = React.useContext(kimrofContext);
15
16   return {
17     value: values[name],
18   };
19 };
```



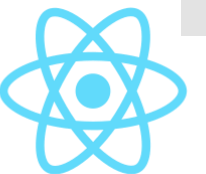
The KimrofLabeledField component



```
3 import { LabeledInput } from "../components";
4 import { useKimrofField } from "./useKimrofField";
5
6 You, a day ago | 1 author (You)
7 interface Props
8   extends Omit<
9     React.ComponentProps<typeof LabeledInput>,
10     "onChange" | "value"
11   > {
12     name: string;
13   }
14 export function KimrofLabeledField(props: Props): ReactElement {
15   const fieldProps = useKimrofField(props.name);
16
17   return <LabeledInput { ...props } { ...fieldProps } />;
18 }
```

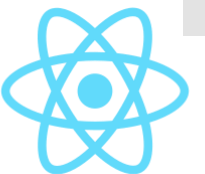


Adding the reducer



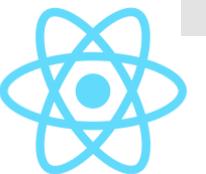
Adding the reducer

- The **Kimrof state** is maintained with a **useReducer() hook**
 - Supports a set-property action for when an edit is made
- **Note:** This reducer will support other actions like setting validity, resetting the form values etc. in a more complete solution



The kimrofReducer()

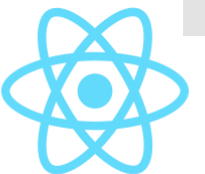
```
20 export function kimrofReducer(  
21   state: ReducerState,  
22   action: SomeAction  
23 ): ReducerState {  
24   switch (action.type) {  
25     case "set-property":  
26       return {  
27         ...state,  
28         formState: { ...state.formState, isDirty: true },  
29         values: {  
30           ...state.values,  
31           [action.payload.name]: action.payload.value,  
32         },  
33       };  
34     }  
35   }  
36   return state;  
37 }
```



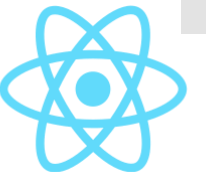
The Kimrof provider with useReducer()



```
14 export function Kimrof<TData extends KimrofObject>({
15   children,
16   initialValues,
17 }: Props<TData>): ReactElement {
18   const [
19     {
20       values,
21       formState: { isDirty },
22     },
23     dispatch,
24   ] = useReducer(kimrofReducer, {
25     values: initialValues,
26     formState: { isDirty: false, isValid: true },
27   });
28
29   const context: KimrofContext = React.useMemo(
30     () => ({
31       isDirty,
32       values,
33     }),
34     [isDirty, values]
35   );
36
37   return (
38     <kimrofContext.Provider value={context}>{children}</kimrofContext.Provider>
39   );
40 }
```

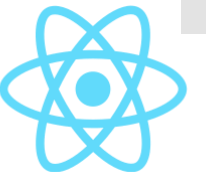


Editing data



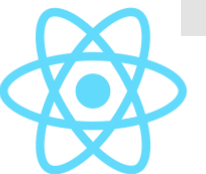
Editing data

- Add the **onChange handler** to the useKimrofField() hook
 - Dispatch the set-property action when a change is detected



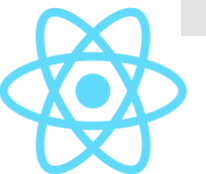
The updated kimrofContext

```
5  export interface KimrofContext {  
6  |   isDirty: boolean;  
7  |   values: KimrofObject;  
8  |   setFieldValue: (name: string, value: KimrofProperty) => void;  
9  | }  
10  
11 export const kimrofContext = React.createContext<KimrofContext>({  
12 |   isDirty: false,  
13 |   values: {},  
14 |   setFieldValue: () => void null,  
15 | });
```



The Kimrof provider with setFieldValue()

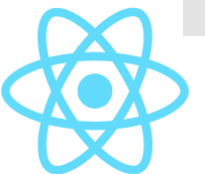
```
29  const context: KimrofContext = React.useMemo(  
30    () => ({  
31      isDirty,  
32      values,  
33      setFieldValue: (name: string, value: KimrofProperty) => {  
34        dispatch({ type: "set-property", payload: { name, value } });  
35      },  
36    }),  
37    [isDirty, values]  
38  );  
39  
40  return (  
41    <kimrofContext.Provider value={context}>{children}</kimrofContext.Provider>  
42  );  
43 }
```



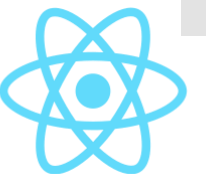
The useKimrofField() hook with setFieldValue()



```
6 interface UseKimrofField {
7   value: KimrofProperty;
8   onChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
9 }
10
11 export const useKimrofField = (name?: string): UseKimrofField => {
12   if (!name) {
13     throw new Error("The name prop is required");
14   }
15   const { values, setFieldValue } = React.useContext(kimrofContext);
16
17   return {
18     value: values[name],
19     onChange: (e: React.ChangeEvent<HTMLInputElement>) => {
20       setFieldValue(e.target.name, e.target.value);
21     },
22   };
23 };
```

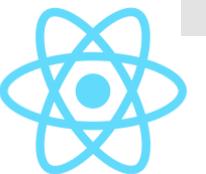


Submitting the form data



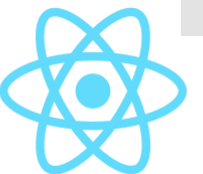
Submitting the form data

- The **useKimrofForm()** hook allows a form to be **submitted**
 - Returns an object with an onSubmit prop
- **Spread the result** into the HTML `<form />`
- **Note:** Not required but useful to have with an HTML `<form />`



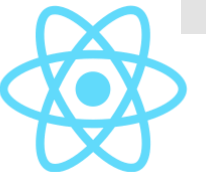
The kimrofContext

```
5  export interface KimrofContext {  
6    isDirty: boolean;  
7    values: KimrofObject;  
8    onSubmit: (e: React.FormEvent<HTMLFormElement>) => void;  
9    setFieldValue: (name: string, value: KimrofProperty) => void;  
10 }  
11  
12 export const kimrofContext = React.createContext<KimrofContext>({  
13   isDirty: false,  
14   values: {},  
15   onSubmit: () => void null,  
16   setFieldValue: () => void null,  
17 });
```



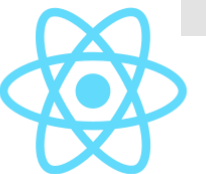
The Kimrof provider

```
9 interface Props<TData> {
10   children: ReactNode;
11   initialValues: TData;
12   onSubmit: (values: TData) => void;
13 }
14
15 export function Kimrof<TData extends KimrofObject>({
16   children,
17   initialValues,
18   onSubmit,
19 }: Props<TData>): ReactElement {
20   const [
21     {
22       values,
23       formState: { isDirty },
24     },
25     dispatch,
26   ] = useReducer(kimrofReducer, {
27     values: initialValues,
28     formState: { isDirty: false, isValid: true },
29   });
30
31   const context: KimrofContext = React.useMemo(
32     () => ({
33       isDirty,
34       values,
35       onSubmit: (e: React.FormEvent) => {
36         e.preventDefault();
37         onSubmit(values as TData);
38       },
39       setFieldValue: (name: string, value: KimrofProperty) => {
40         dispatch({ type: "set-property", payload: { name, value } });
41       },
42     }),
43     [isDirty, onSubmit, values]
44   );
```



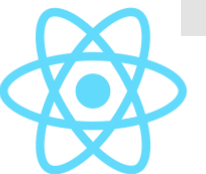
The useKimrofForm() hook

```
1 import React from "react";
2
3 import { kimrofContext } from "../KimrofContext";
4
5 You, 2 minutes ago | 1 author (You)
6 interface UseKimrofForm {
7   onSubmit: (e: React.FormEvent<HTMLFormElement>) => void;
8 }
9
10 export const useKimrofForm = (): UseKimrofForm => {
11   const { onSubmit } = React.useContext(kimrofContext);
12
13   return {
14     onSubmit,
15   };
16 }
```



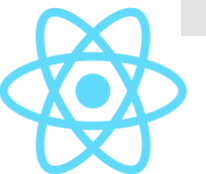
The useKimrof() hook

```
3 import { KimrofObject } from "../Types";
4 import { kimrofContext } from "../KimrofContext";
5
6 You, seconds ago | 1 author (You)
7 interface UseKimrof {
8   isDirty: boolean;
9   values: KimrofObject;
10 }
11 export const useKimrof = (): UseKimrof => {
12   const { values, isDirty } = React.useContext(kimrofContext);
13
14   return {
15     isDirty,
16     values,
17   };
18 };
```



Updating the PersonEditor

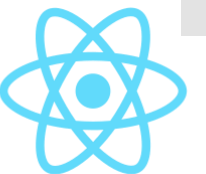
```
2 | import { KimrofLabeledField, useKimrof, useKimrofForm } from "../kimrof";
3 |
4 | export function PersonEditor(): ReactElement {
5 |   const { values, isDirty } = useKimrof();
6 |   const formProps = useKimrofForm();
7 |
8 |   return (
9 |     <form className="person-editor" { ... formProps}>
10 |       <h2>Kimrof Person Editor</h2>
11 |       <KimrofLabeledField label="Firstname:" name="firstname" />
12 |       <KimrofLabeledField label="Surname:" name="surname" />
13 |       <KimrofLabeledField label="Email:" name="email" />
14 |       <KimrofLabeledField label="Address:" name="address" />
15 |       <KimrofLabeledField label="Phone:" name="phone" />
16 |       <button className="btn btn-primary" disabled={!isDirty}>
17 |         Save
18 |       </button>
19 |       <hr />
20 |       <pre>{JSON.stringify(values, null, 2)}</pre>
21 |     </form>
22 |   );
23 | }
```



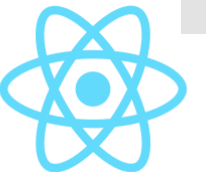
Adding the onSubmit handler to KimrofPersonEditor



```
10 export function KimrofPersonEditor(): ReactElement {
11   return (
12     <Kimrof
13       initialValues={initialPerson as IndexedPerson}
14       onSubmit={(person) => {
15         alert(`Submitting\n${JSON.stringify(person, null, 2)}`);
16       }}
17     >
18     <PersonEditor />
19   </Kimrof>
20 );
21 }
```



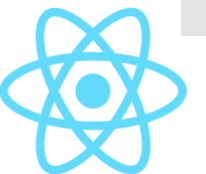
Form Validation



Adding the onSubmit
handler to
KimrofPersonEditor

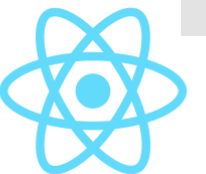


```
10 export function KimrofPersonEditor(): ReactElement {
11   return (
12     <Kimrof
13       initialValues={initialPerson as IndexedPerson}
14       onSubmit={(person) => {
15         alert(`Submitting\n${JSON.stringify(person, null, 2)}`);
16       }}
17     >
18     <PersonEditor />
19   </Kimrof>
20 );
21 }
```



Conclusion

- **Hooks are simple** but stick to the rules
 - They exist for a good reason
- The implementation of hooks is **simple and elegant**
- **Custom hooks** are extremely useful
 - Keep them small and focused and use hook composition
- Hooks are a great way to **provide non UI functionality**
 - When building non UI React libraries
 - Regardless of inhouse or published to NPM



Maurice de Beijer

@mauricedb

maurice.de.beijer
@gmail.com

