# 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

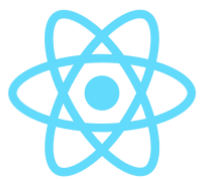
# Happily married


IN SEARCH OF
SAILOR SPIRIT
SOUL
FOLLOW
TWINLIFE
SONG

# Independent software developer & instructor

Since 1995


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

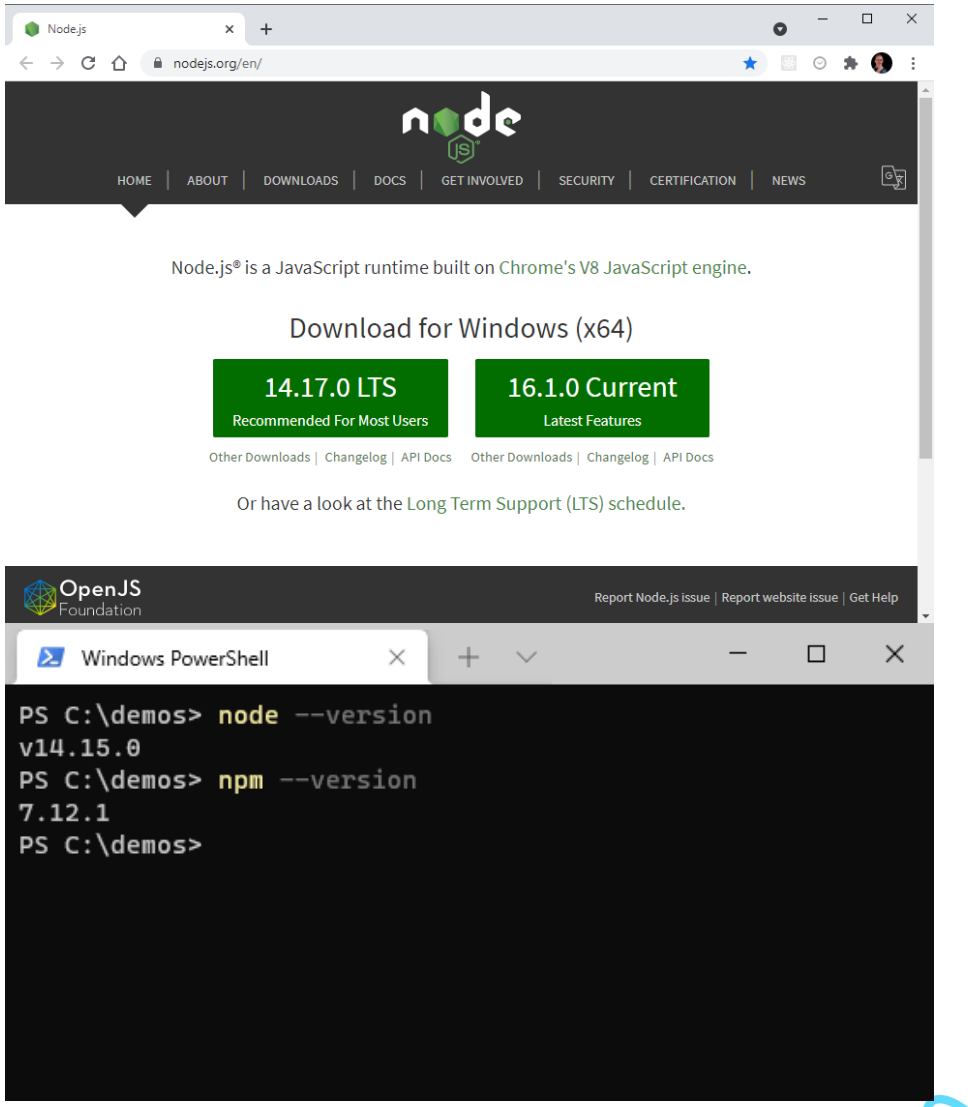
# Following Along


```tsx
export function usePerson(
  initialPerson: Person
): [Person | null, (person: Person | null) => void] {
  const [person, setPerson] = useState<Person | null>(null);
  const loaded = useRef(false);

  useLayoutEffect(() => {
    loaded.current = true;

    return () => {
      loaded.current = false;
    };
  });

  useEffect(() => {
    const getPerson = async () => {
      await sleep(2500);
      const person = await localforage.getItem<Person>("person");
      if (loaded.current) {
        setPerson(person ?? initialPerson);
      }
    };
    getPerson();
  }, [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.ex
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() H


PersonEditor.tsx M

src > person-editor > PersonEditor.tsx > ...

```tsx
import React, { ReactElement, useState } from "react"

import { LabeledInput } from "../components"
import { initialPerson } from "../utils"

export function PersonEditor(): ReactElement {
  const [person, setPerson] = useState(initialPerson)

  return (
    <form>
      <h2>Person Editor</h2>
      <LabeledInput
        label="Firstname:"
        value={person.firstname}
        onChange={(e) => {
          const newPerson = {
            ...person,
            firstname: e.target.value,
          }
          setPerson(newPerson)
        }}
      />
```

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

src > person-editor > TS PersonEditor.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, { ReactElement, useState } from "react"
import { LabeledInput } from "../components"
import { initialPerson } from "../utils"

export function PersonEditor(): ReactElement {
  const [person, setPerson] = useState(() => initialPerson)

  return (
    <form
      className="person-editor"
      onSubmit={(e) => {
        e.preventDefault()
        alert(`Submitting\n${JSON.stringify(person, null, 2)}`)
      }}
    >
      <h2>Person Editor</h2>
      <LabeledInput
        label="Firstname:"
        value={person.firstname}
        onChange={(e) =>
          setPerson((state) => ({ ...state, firstname: e.target.value }))
        }
      />
```

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


PersonEditor.tsx M

src > person-editor > PersonEditor.tsx > PersonEditor > <function> > setPerson() callback > <unknown>

```tsx
function savePerson(person: Person | null): void {
  console.log("Saving", person)
  localforage.setItem("person", person)
}

export function PersonEditor(): ReactElement {
  const [person, setPerson] = useState<Person | null>(null)

  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      setPerson(person ?? initialPerson)
    }

    getPerson()
  }, [])

  useEffect(() => {
    savePerson(person)
  }, [person])

  if (!person) {
    return <Loading />
  }
```

# 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

```tsx
export const themeContext = createContext<ThemeContext>({
  style: undefined,
  setStyle: () => void null,
})

export function ThemeProvider({ children }: Props): ReactElement {
  const [style, setStyle] = useState<CSSProperties>()

  return (
    <themeContext.Provider value={{ style, setStyle }}>
      {children}
    </themeContext.Provider>
  )
}
```

# 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 usePerson.ts U × TS PersonEditor.tsx M

src > person-editor > TS usePerson.ts > ...

```ts
import { useState, useEffect } from "react"
import localforage from "localforage"

import type { Person } from "../types/person"

function savePerson(person: Person | null): void {
  console.log("Saving", person)
  localforage.setItem("person", person)
}

export function usePerson(initialPerson: Person) {
  const [person, setPerson] = useState<Person | null>(null)

  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      setPerson(person ?? initialPerson)
    }

    getPerson()
  }, [initialPerson])

  useEffect(() => {
    savePerson(person)
  }, [person])

  return [person, setPerson] as const
}
```

# Using the usePerson() ho


TS PersonEditor.tsx M × TS usePerson.ts U

src > person-editor > TS PersonEditor.tsx > ...

You, seconds ago | 1 author (You)

```tsx
/* eslint-disable @typescript-eslint/no-non-null-assertion */
import React, { ReactElement } from "react"

import { LabeledInput, Loading } from "../components"
import { initialPerson } from "../utils"
import { usePerson } from "./usePerson"

export function PersonEditor(): ReactElement {
  const [person, setPerson] = usePerson(initialPerson)

  if (!person) {
    return <Loading />
  }
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# Rules of Hooks

Custom hook names

# Custom Hook Names

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

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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


Counter.tsx M

src > rules-of-hooks > Counter.tsx > ...

You, a minute ago | 1 author (You)

```tsx
import React, { ReactElement, useEffect, useRef, useState } from "react"

export function Counter(): ReactElement {
  const [counter, setCounter] = useState(0)
  const button = useRef<HTMLButtonElement>(null)

  useEffect(() => {
    setTimeout(() => {
      button.current?.focus()
    }, 1000)
  }, [])

  return (
    <div>
      <div>Count: {counter}</div>
      <div>
        <button
          ref={button}
          className="btn btn-primary"
          onClick={() => setCounter(counter + 1)}
        >
          Increment
        </button>
      </div>
    </div>
  )
}
```

# useRef() with a component

PersonEditor.tsx 2, M

src > person-editor > PersonEditor.tsx > ...

```tsx
import React, { ReactElement, useEffect, useRef } from "react"

import { LabeledInput, Loading } from "../components"
import { initialPerson } from "../utils"
import { usePerson } from "./usePerson"

export function PersonEditor(): ReactElement {
  const [person, setPerson] = usePerson(initialPerson)
  const input = useRef<HTMLInputElement>(null)

  useEffect(() => {
    setTimeout(() => {
      input.current?.focus()
    }, 1000)
  }, [])

  if (!person) {
    return <Loading />
  }

  return (
    <form>
      <h2>Person Editor</h2>
      <LabeledInput
        ref={input}
        label="Firstname:"
        value={person.firstname}
        onChange={(e) => {
          setPerson((person) => ({
```

# Forwarding useRef()


TS LabeledInput.tsx M

src > components > TS LabeledInput.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, { forwardRef, InputHTMLAttributes, ReactElement } from "react"
import classNames from "classnames"

interface Props extends InputHTMLAttributes<HTMLInputElement> {
  label: string
}

export const LabeledInput = forwardRef<HTMLInputElement, Props>(
  ({ id, label, className, ...props }, ref): ReactElement => {
    return (
      <div className={classNames("form-group", className)}>
        <label htmlFor={id} className="form-label">
          {label}
        </label>

        <input {...props} id={id} className="form-control" ref={ref} />
      </div>
    )
  }
)

LabeledInput.displayName = "LabeledInput"
```

You, a week ago | 1 author (You)

# useRef() for generic state

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

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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

# Synchronous DOM Mutations


Counter.tsx M

src > rules-of-hooks > Counter.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, {      You, seconds ago • Uncommitted c
  ReactElement,
  useEffect,
  useLayoutEffect,
  useRef,
  useState,
} from "react"

export function Counter(): ReactElement {
  const [counter, setCounter] = useState(0)
  const button = useRef<HTMLButtonElement>(null)

  useEffect(() => {
    setTimeout(() => {
      button.current?.focus()
    }, 1000)
  }, [])

  useLayoutEffect(() => {
    if (button.current) {
      button.current.style.backgroundColor = "green"
    }
  }, [])
```

# useIsMounted()

useIsMounted.ts U

src > hooks > useIsMounted.ts > ...

```ts
import { MutableRefObject, useRef, useLayoutEffect } from "react"

export function useIsMounted(): Readonly<MutableRefObject<boolean>> {
  const isMounted = useRef(false)

  useLayoutEffect(() => {
    isMounted.current = true

    return () => {
      isMounted.current = false
    }
  }, [])

  return isMounted
}
```

# Using useRef() to maintain state


usePerson.ts M

src > person-editor > usePerson.ts > ...

```ts
import { useState, useEffect } from "react"        You, 3 days ago
import localforage from "localforage"

import type { Person } from "../types/person"
import { sleep } from "../utils"
import { useIsMounted } from "../hooks/useIsMounted"

function savePerson(person: Person | null): void {
  console.log("Saving", person)
  localforage.setItem("person", person)
}

export function usePerson(initialPerson: Person) {
  const [person, setPerson] = useState<Person | null>(null)
  const isMounted = useIsMounted()

  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      await sleep(2500)
      if (isMounted.current) {
        setPerson(person ?? initialPerson)
      }
    }

    getPerson()
  }, [initialPerson, isMounted])
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# useRef(), useState() and dependencies

👉 **The useRef() object and the useState() updater function are not required in dependency arrays** 👈

# No dependency

PersonEditor.tsx M

src > person-editor > PersonEditor.tsx > PersonEditor > <function> > setPerson() callback > <unknown>

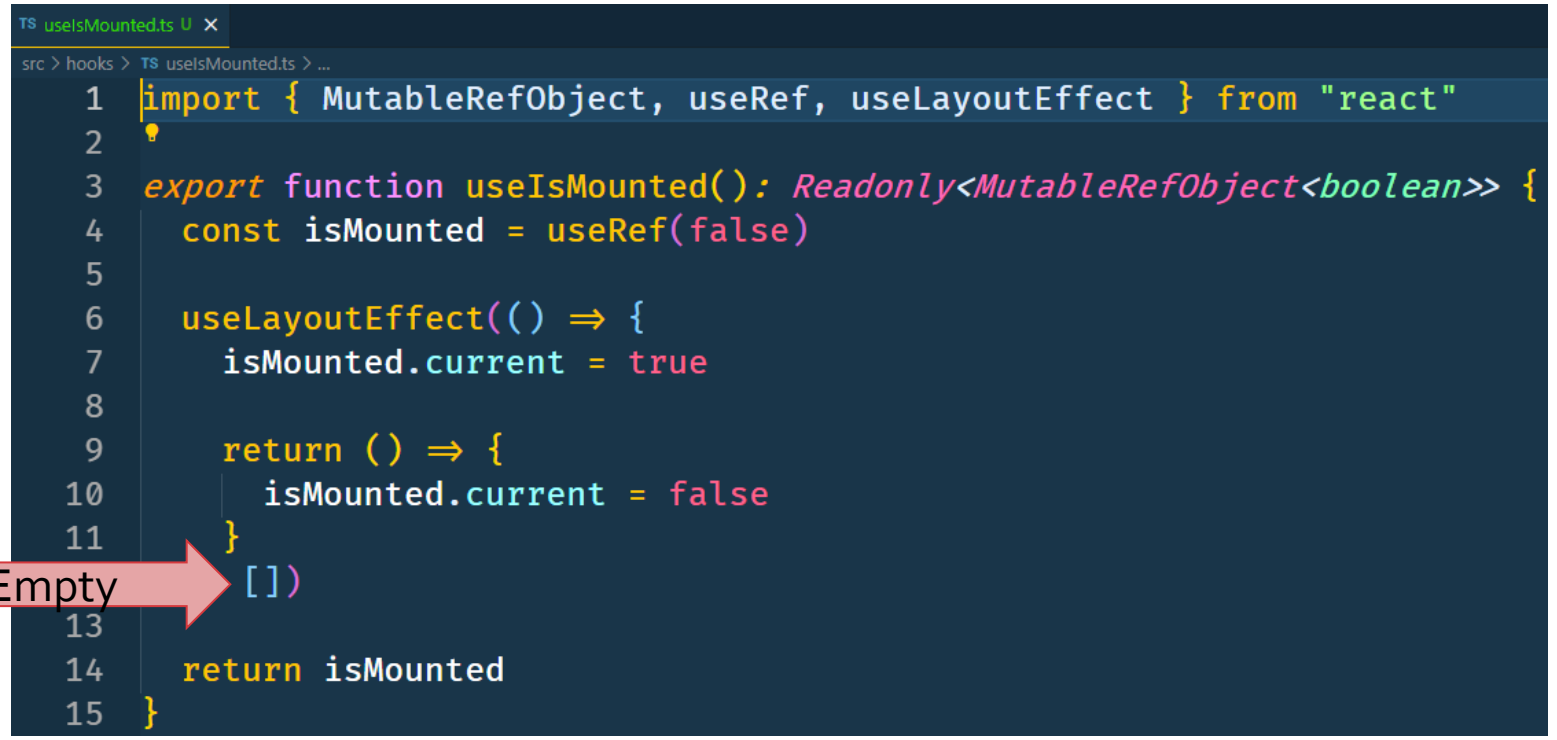
```tsx
function savePerson(person: Person | null): void {
  console.log("Saving", person)
  localforage.setItem("person", person)
}

export function PersonEditor(): ReactElement {
  const [person, setPerson] = useState<Person | null>(null)

  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      setPerson(person ?? initialPerson)
    }

    getPerson()
  }, [])

  useEffect(() => {
    savePerson(person)
  }, [person])

  if (!person) {
    return <Loading />
  }
```

Empty

Not empty

# No dependency

Empty




# With Dependency

usePerson.ts M

src > person-editor > usePerson.ts > ...

```ts
import { useState, useEffect } from "react"        You, 3 days ago
import localforage from "localforage"

import type { Person } from "../types/person"
import { sleep } from "../utils"
import { useIsMounted } from "../hooks/useIsMounted"

function savePerson(person: Person | null): void {
  console.log("Saving", person)
  localforage.setItem("person", person)
}

export function usePerson(initialPerson: Person) {
  const [person, setPerson] = useState<Person | null>(null)
  const isMounted = useIsMounted()

  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      await sleep(2500)
      if (isMounted.current) {
        setPerson(person ?? initialPerson)
      }
    }

    getPerson()
  }, [initialPerson, isMounted])
```

Not empty

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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 U ×

src > hooks > TS useDebounce.ts > ...

```ts
import { useEffect } from "react"

export function useDebounce(fn: () => void, timeout: number): void {
  useEffect(() => {
    const handle = setTimeout(fn, timeout)

    return () => clearTimeout(handle)
  }, [fn, timeout])
}
```

# Using useDebounce()


usePerson.ts 1, M

src > person-editor > usePerson.ts > ...

```tsx
  useEffect(() => {
    const getPerson = async () => {
      const person = await localforage.getItem<Person>("person")
      // await sleep(2500)
      if (isMounted.current) {
        setPerson(person ?? initialPerson)
      }
    }

    getPerson()
  }, [initialPerson, isMounted])

  useDebounce(() => {
    savePerson(person)
  }, 1000)

  return [person, setPerson] as const
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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()


```
36    const saveFn = useCallback(() ⇒ {
37      savePerson(person)
38    }, [person])
39
40    useDebounce(saveFn, 1000)
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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 U ×

src > hooks > TS useWillUnmount.ts > ...

```ts
import { useEffect, useRef } from "react"

export function useWillUnmount(fn: () => void): void {
  const functionRef = useRef(fn)
  functionRef.current = fn

  useEffect(() => {
    return () => functionRef.current()
  }, [])
}
```

# Save changes when unmounting


TS usePerson.ts 1, M

src > person-editor > TS usePerson.ts > ...

```
36
37    const saveFn = useCallback(() => {
38      savePerson(person)
39    }, [person])
40
41    useDebounce(saveFn, 10000)
42    useWillUnmount(saveFn)
43
44    return [person, setPerson] as const
45  }
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# useThrottle()

# Throttle versus Debouce

- A debounced functions is only called after it has not been called for a specified duration
- A throttled function is called once after ever timeout

# useThrottle()

useThrottle.ts U

src > hooks > useThrottle.ts > ...

```ts
import { useEffect, useRef } from "react"

export function useThrottle(fn: () => void, timeout: number): void {
  const previousRef = useRef<(() => void) | null>(null)
  const currentRef = useRef<(() => void) | null>(fn)
  if (previousRef.current !== fn) {
    currentRef.current = fn
  }

  useEffect(() => {
    const handle = setInterval(() => {
      if (currentRef.current) {
        currentRef.current()
        previousRef.current = currentRef.current
        currentRef.current = null
      }
    }, timeout)

    return () => clearInterval(handle)
  }, [timeout])
}
```

# usePerson()


TS usePerson.ts 2, M

src > person-editor > TS usePerson.ts > ...

```
38    const saveFn = useCallback(() => {
39      savePerson(person)
40    }, [person])
41
42    useThrottle(saveFn, 1000)
43    useWillUnmount(saveFn)
44
45    return [person, setPerson] as const
46  }
```

See you in the next video

# React Hooks Tips Only the Pros Know

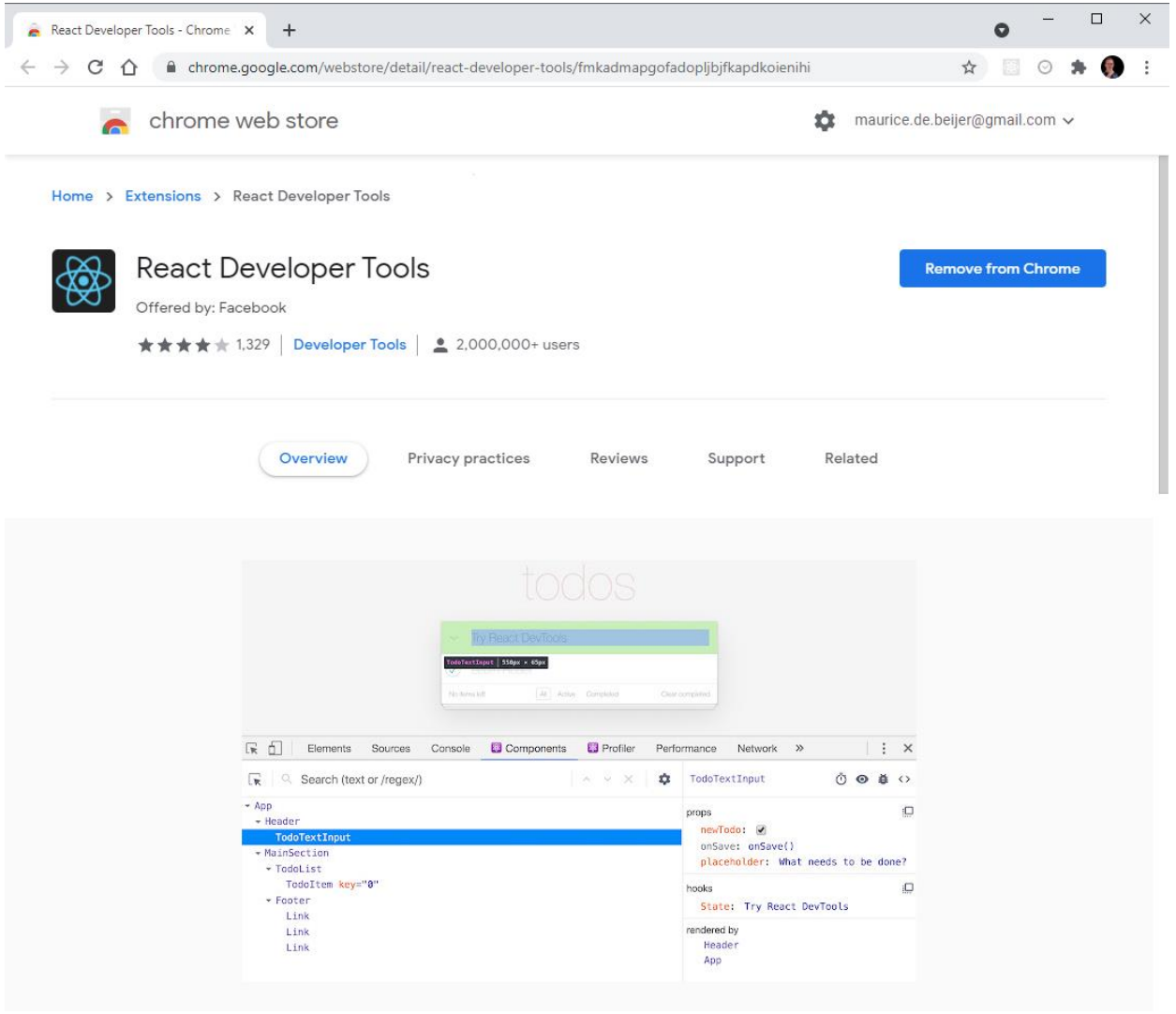
Maurice de Beijer - @mauricedb

# useDebugValue()

# useDebugValue()

- Used to display a label for custom hooks in React DevTools
- Only recommended for reusable hooks

# React Developer Tools



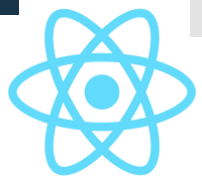
# useDebugValue()


useThrottle.ts M

src > hooks > useThrottle.ts > ...

You, 2 minutes ago | 1 author (You)

```ts
import { useEffect, useRef, useDebugValue } from "react"

export function useThrottle(fn: () => void, timeout: number): void {
  const previousRef = useRef<(() => void) | null>(null)
  const currentRef = useRef<(() => void) | null>(fn)
  if (previousRef.current !== fn) {
    currentRef.current = fn
  }

  useDebugValue(currentRef.current, (fn) => fn?.toString())

  useEffect(() => {
    const handle = setInterval(() => {
      if (currentRef.current) {
        previousRef.current = currentRef.current
        currentRef.current()
        currentRef.current = null
      }
    }, timeout)

    return () => clearInterval(handle)
  }, [timeout])
}
```



See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# More complex state

With useState()

# More complex state

- Track multiple state items
  - The form data as well as meta data about the form
- Use as **many useState() hooks** as needed
  - One for the person object
  - A second for the dirty and valid states

# Two useState() hooks

usePerson.ts 2, M

src > person-editor > usePerson.ts > ...

```tsx
interface Metadata {
  isDirty: boolean
  isValid: boolean
}
      You, seconds ago • Uncommitted changes
export function usePerson(initialPerson: Person) {
  const [person, setPerson] = useState<Person | null>(null)
  const [metadata, setMetadata] = useState<Metadata>({
    isDirty: false,
    isValid: true,
  })
  const isMounted = useIsMounted()
```

# Multiple updates

usePerson.ts 2, M

src > person-editor > usePerson.ts > ...

```ts
  useThrottle(saveFn, 1000)
  useWillUnmount(saveFn)

  const setPersonAndMeta = (value: SetStateAction<Person | null>) => {
    setPerson(value)
    setMetadata((m) => ({ ...m, isDirty: true }))
  }

  return [person, setPersonAndMeta, metadata] as const
}
```

# Exposing more data from a custom hook

PersonEditor.tsx M

src > person-editor > PersonEditor.tsx > PersonEditor

```tsx
export function PersonEditor(): ReactElement {
  const [person, setPerson, { isDirty, isValid }] = usePerson(initialPerson)
  const input = useRef<HTMLInputElement>(null)

  useEffect(() => {
    setTimeout(() => {
      input.current?.focus()
    }, 1000)
  }, [])

  if (!person) {
    return <Loading />
  }
```

# Disable save button if there are no changes


TS PersonEditor.tsx M ×

src > person-editor > TS PersonEditor.tsx > PersonEditor

```tsx
      <div className="btn-group">
        <button
          type="submit"
          className="btn btn-primary"
          disabled={!isDirty || !isValid}
        >
          Submit
        </button>
      </div>
      <hr />
      <pre>{JSON.stringify(person, null, 2)}</pre>
    </form>
  )
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# 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

# personEditorReducer()

personEditorReducer.ts U

src > person-editor > personEditorReducer.ts > ...

```ts
interface Metadata {
  isDirty: boolean
  isValid: boolean
}

interface ReducerState {
  person: Person | null
  metadata: Metadata
}

interface SetPersonAction {
  type: "set-initial-person"
  payload: Person
}

type SomeAction = SetPersonAction

export function personEditorReducer(
  state: ReducerState,
  action: SomeAction
): ReducerState {
  switch (action.type) {
    case "set-initial-person":
      return { ...state, person: action.payload }
    default:
      return state
  }
}
```

# usePerson()

usePerson.ts 4, M

src > person-editor > usePerson.ts > usePerson

```ts
useEffect(() => {
  const getPerson = async () => {
    const person = await localforage.getItem<Person>("person")
    // await sleep(2500)
    if (isMounted.current) {
      // setPerson(person ?? initialPerson)
      dispatch({
        type: "set-initial-person",
        payload: person ?? initialPerson,
      })
    }
  }

  getPerson()
}, [initialPerson, isMounted])
```

# personEditorReducer()

personEditorReducer.ts U

src > person-editor > personEditorReducer.ts > ...

```ts
type SomeAction = SetPersonAction | SetPropertyAction

export function personEditorReducer(
  state: ReducerState,
  action: SomeAction
): ReducerState {
  switch (action.type) {
    case "set-initial-person":
      return { ...state, person: action.payload }
    case "set-property":
      return {
        ...state,
        metadata: { ...state.metadata, isDirty: true },
        person: {
          ...state.person!,
          [action.payload.name]: action.payload.value,
        },
      }

    default:
      return state
  }
}
```

# usePerson()

TS usePerson.ts 5, M

src > person-editor > TS usePerson.ts > ...

```ts
  function setProperty(name: keyof Person, value: unknown) {
    dispatch({ type: "set-property", payload: { name, value } })
  }

  return [person, setProperty, metadata] as const
}
```

# PersonEditor

TS PersonEditor.tsx M ×

src > person-editor > TS PersonEditor.tsx > PersonEditor

```tsx
30          <h2>Person Editor</h2>
31          <LabeledInput
32            ref={input}
33            label="Firstname:"
34            value={person.firstname}
35            onChange={(e) => {
36              // setPerson((person) => ({
37              //    ...person!,
38              //    firstname: e.target.value,
39              // }))
40              setProperty("firstname", e.target.value)
```

# personEditorReducer()

personEditorReducer.ts U

src > person-editor > personEditorReducer.ts > ...

```ts
export function personEditorReducer(
  state: ReducerState,
  action: SomeAction
): ReducerState {
  switch (action.type) {
    case "set-initial-person":
      return { ...state, person: action.payload }
    case "set-property":
      return {
        ...state,
        metadata: { ...state.metadata, isDirty: true },
        person: {
          ...state.person!,
          [action.payload.name]: action.payload.value,
        },
      }
    case "set-properties":
      return {
        ...state,
        metadata: { ...state.metadata, isDirty: true },
        person: {
          ...state.person!,
          ...action.payload,
        },
      }
```

# usePerson()

usePerson.ts 5, M

src > person-editor > usePerson.ts > ...

```ts
  function setProperty(name: keyof Person, value: unknown) {
    dispatch({ type: "set-property", payload: { name, value } })
  }

  function setProperties(payload: Partial<Person>) {
    dispatch({ type: "set-properties", payload })
  }

  return [person, setProperty, setProperties, metadata] as const
}
```

# PersonEditor


TS PersonEditor.tsx M

src > person-editor > TS PersonEditor.tsx > PersonEditor

```tsx
<LabeledInput
  ref={input}
  label="Firstname:"
  value={person.firstname}
  onChange={(e) => {
    setProperty("firstname", e.target.value)

    if (e.target.value === "Ford") {
      setProperties({
        surname: "Prefect",
        address: "Outer space",
        email: "",
        phone: "",
      })
    }
  }}
/>
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# useMemo()

Returns a memoized value

# useMemo()

- Uses the creator function to recompute a value
  - Only when the dependencies change
- useMemo() is a performance optimization
  - Not a semantic guarantee
- 👉 **The API allows for memoized values to be forgotten** 👈
  - In React 17 this doesn't happen

# useMemo()


TS usePerson.ts 3, M

src > person-editor > TS usePerson.ts > usePerson > useEffect() callback

```
38    const firstAndSurName = useMemo(
39      () => ({
40        firstname: person?.firstname,
41        surname: person?.surname,
42      }),
43      [person?.firstname, person?.surname]
44    )
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# Kimrof

A Formik like forms utility

# Kimrof

- Putting it all together
  - Combining hooks, context and components
- Kimrof, the demo library
  - A Formik like forms library

# Kimrof Example

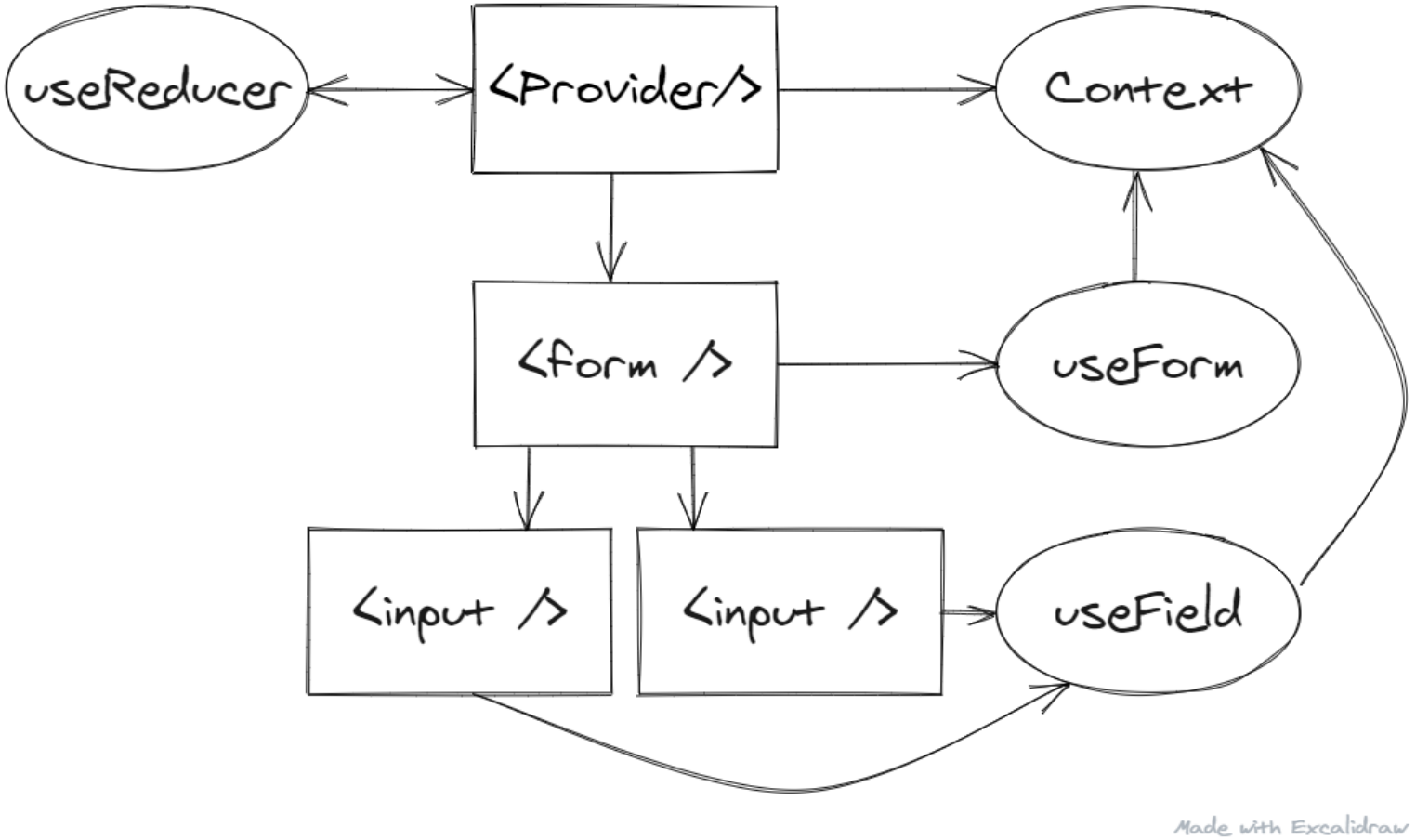
TS UserEditor.tsx M ×

src > kimrof-user-editor > TS UserEditor.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, { ReactElement } from "react"
import { KimrofLabeledField, useKimrof, useKimrofForm } from "./kimrof"

export function UserEditor(): ReactElement {
  const {
    metadata: { isDirty, isValid },
  } = useKimrof()
  const formProps = useKimrofForm()

  return (
    <form className="person-editor" {...formProps}>
      <h2>Kimrof User Editor</h2>
      <KimrofLabeledField label="Firstname:" name="firstname" />
      <KimrofLabeledField label="Surname:" name="surname" />
      <KimrofLabeledField label="Email:" name="email" />
      <KimrofLabeledField label="Address:" name="address" />
      <KimrofLabeledField label="Phone:" name="phone" />
      <button className="btn btn-primary" disabled={!isDirty || !isValid}>
        Save
      </button>
    </form>
  )
}
```

# Kimrof Structure



See you in the next video

# React Hooks Tips Only the Pros Know

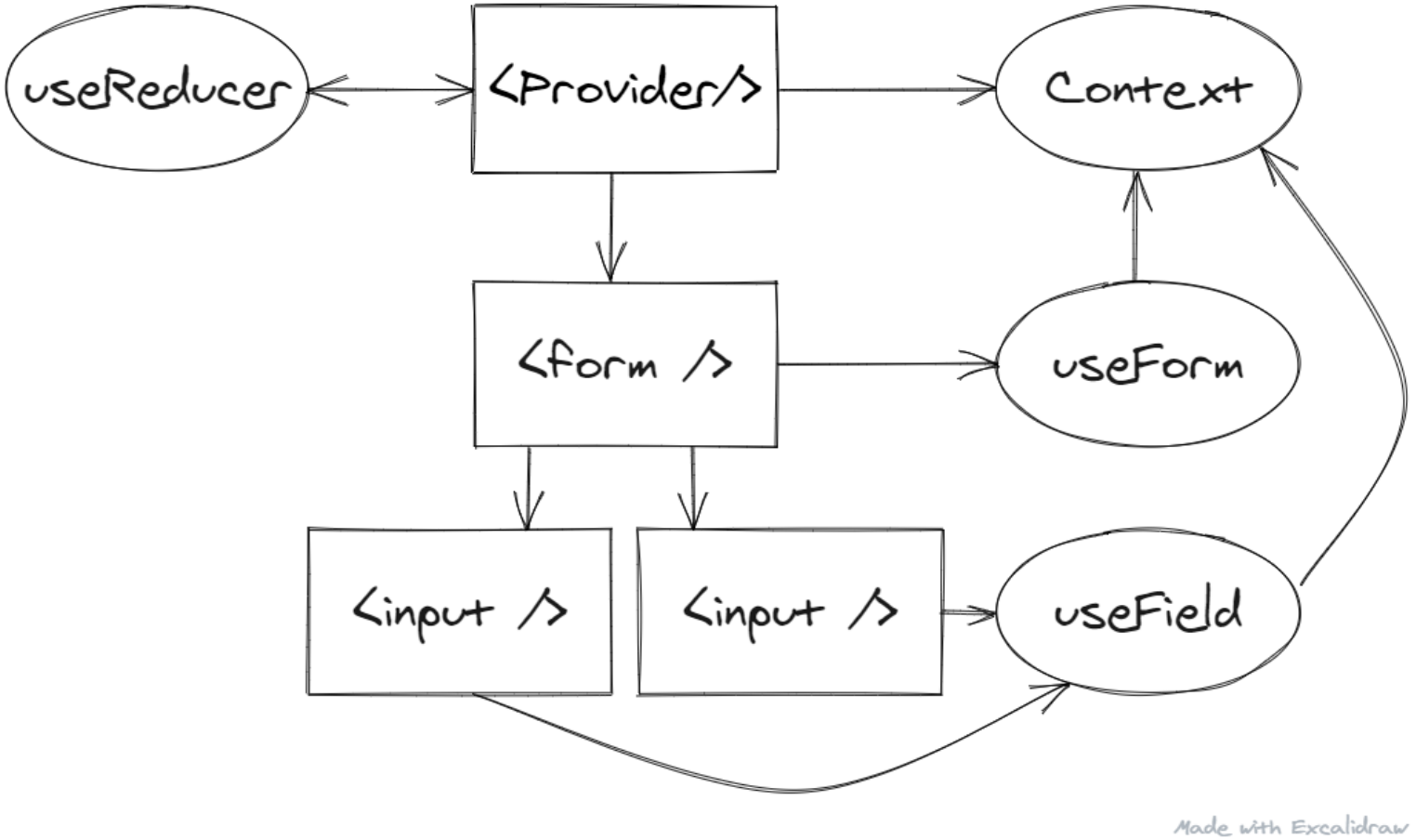
Maurice de Beijer - @mauricedb

# Context and Provider

# Context and Provider

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

# Kimrof Structure



# The kimrofContext

TS KimrofContext.ts 2, M ×

src > kimrof-user-editor > kimrof > TS KimrofContext.ts > ...

```
You, 2 minutes ago | 1 author (You)
1  import React, { createContext } from "react"
2
3  import { KimrofObject, KimrofProperty } from "./Types"
4
   You, 2 minutes ago | 1 author (You)
5  export interface KimrofContext {
6    values: KimrofObject
7  }
8
9  export const kimrofContext = createContext<KimrofContext>({
10   values: {},
11 })
```

# The Kimrof provider

Kimrof.tsx 1, M

src > kimrof-user-editor > kimrof > Kimrof.tsx > ...

You, seconds ago | 1 author (You)

```tsx
// Kimrof = Formik reversed :-)

import React, { ReactElement, ReactNode, useMemo } from "react"

import { KimrofObject, KimrofProperty } from "./Types"
import { KimrofContext, kimrofContext } from "./KimrofContext"

```

You, seconds ago | 1 author (You)

```tsx
interface Props<TData> {
  children: ReactNode
  initialValues: TData
}

export function Kimrof<TData extends KimrofObject>({
  children,
  initialValues,
}: Props<TData>): ReactElement {
  const values = initialValues

  const context: KimrofContext = useMemo(() => ({ values }), [values])

  return (
    <kimrofContext.Provider value={context}>{children}</kimrofContext.Provider>
  )
}
```

# Adding the provider


TS KimrofUserEditor.tsx M ×

src > kimrof-user-editor > TS KimrofUserEditor.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, { ReactElement } from "react"

import { initialPerson } from "../utils"
import { IndexedPerson } from "../types/IndexedPerson"

// Kimrof = Formik reversed :-)
import { UserEditor } from "./UserEditor"
import { Kimrof } from "./kimrof"

export function KimrofUserEditor(): ReactElement {
  return (
    <Kimrof initialValues={initialPerson as IndexedPerson}>
      <UserEditor />
    </Kimrof>
  )
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

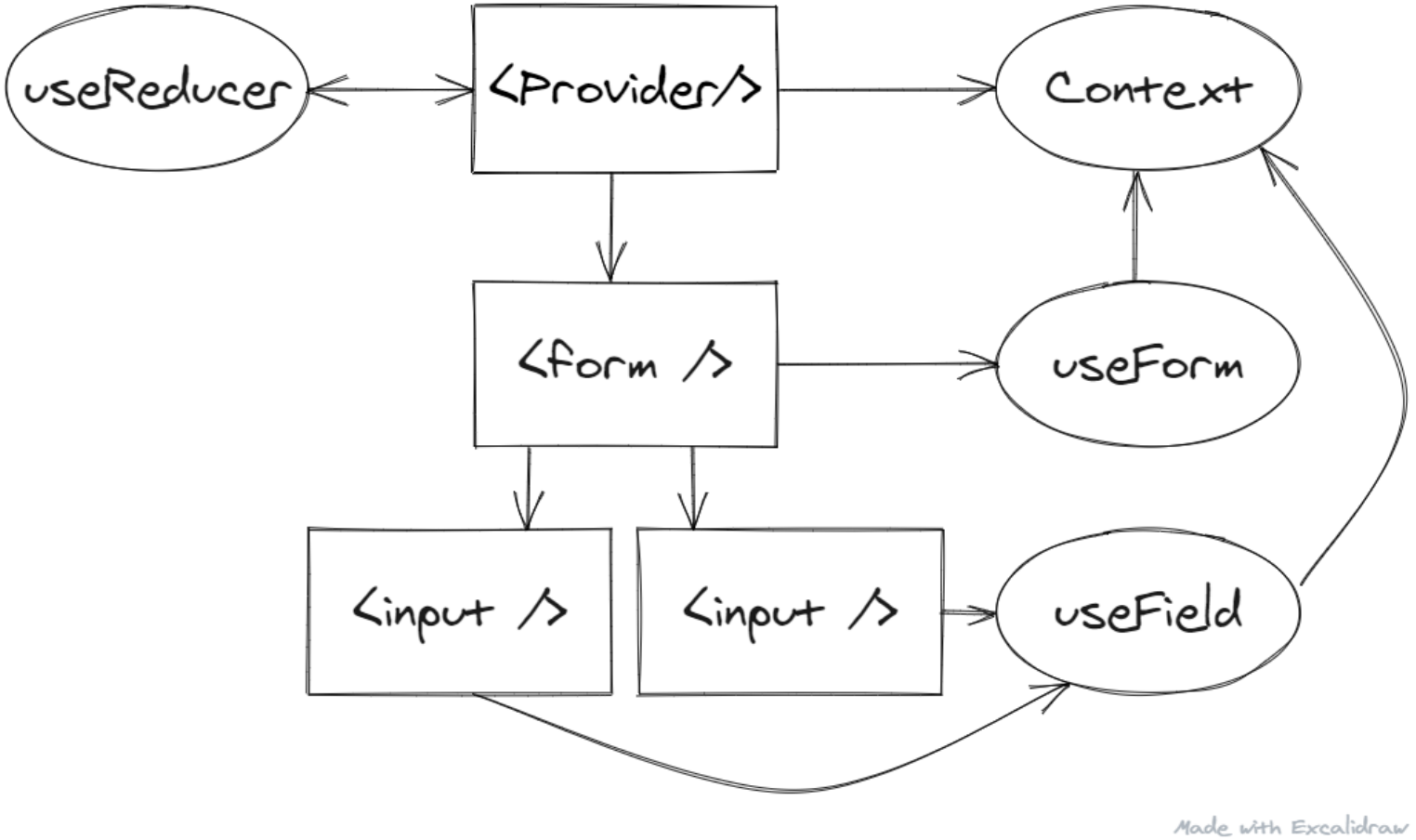
Maurice de Beijer - @mauricedb

# Displaying the values

# Displaying the form values

- The **useKimrofField() hook** allows **input components** to work
  - Returns an object with a value and onChanged prop
- **Spread the result** into any HTML <input />

# Kimrof Structure



# The useKimrofField() hook

TS useKimrofField.ts 1, M ×

src > kimrof-user-editor > kimrof > TS useKimrofField.ts > ...

You, a minute ago | 1 author (You)

```ts
import { ChangeEvent, useCallback, useContext } from "react"

import { kimrofContext } from "./KimrofContext"

export function useKimrofField(name: string) {
  const { values } = useContext(kimrofContext)

  const onChange = useCallback((e: ChangeEvent<HTMLInputElement>) => {
    // TODO
  }, [])

  return {
    value: values[name],
    onChange,
  } as const
}
```

# The KimrofLabeledField component


TS KimrofLabeledField.tsx M

src > kimrof-user-editor > kimrof > TS KimrofLabeledField.tsx > ...

You, seconds ago | 1 author (You)

```tsx
import React, { ComponentProps, ReactElement } from "react"

import { LabeledInput } from "../../components"
import { useKimrofField } from "./useKimrofField"

type LabeledInputProps = ComponentProps<typeof LabeledInput>

```

You, seconds ago | 1 author (You)

```tsx
interface Props extends Omit<LabeledInputProps, "onChange" | "value"> {
  name: string
}

export function KimrofLabeledField(props: Props): ReactElement {
  const fieldProps = useKimrofField(props.name)

  return <LabeledInput {...props} {...fieldProps} />
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# Editing data

# Editing data

- The **Kimrof state** is maintained with a **useReducer() hook**
  - Supports a set-property action for when an edit is made
- Add the **onChange handler** to the useKimrofField() hook
  - Dispatch the set-property action when a change is detected

# The kimrofReducer()

TS kimrofReducer.ts M ×

src > kimrof-user-editor > kimrof > TS kimrofReducer.ts > ...

```ts
export function kimrofReducer(
  state: ReducerState,
  action: SomeAction
): ReducerState {
  switch (action.type) {
    case "set-property":
      return {
        ...state,
        metadata: { ...state.metadata, isDirty: true },
        values: {
          ...state.values,
          [action.payload.name]: action.payload.value,
        },
      }
  }

  return state
}
```

# The KimrofContext

TS KimrofContext.ts 1, M ×

src > kimrof-user-editor > kimrof > TS KimrofContext.ts > ...

```
You, seconds ago | 1 author (You)
1  import React, { createContext } from "react"
2
3  import { KimrofObject, KimrofProperty } from "./Types"
4
   You, seconds ago | 1 author (You)
5  export interface KimrofContext {
6    values: KimrofObject
7    setFieldValue: (name: string, value: KimrofProperty) => void
8  }
9
10 export const kimrofContext = createContext<KimrofContext>({
11   values: {},
12   setFieldValue: () => void null,
13 })
```

# The Kimrof Provider

Kimrof.tsx M

src > kimrof-user-editor > kimrof > Kimrof.tsx > ...

```tsx
export function Kimrof<TData extends KimrofObject>({
  children,
  initialValues,
}: Props<TData>): ReactElement {
  const [{ values }, dispatch] = useReducer(kimrofReducer, {
    values: initialValues,
    metadata: { isDirty: false, isValid: true },
  })

  const context: KimrofContext = useMemo(
    () => ({
      values,
      setFieldValue: (name: string, value: KimrofProperty) => {
        dispatch({ type: "set-property", payload: { name, value } })
      },
    }),
    [values]
  )

  return (
    <kimrofContext.Provider value={context}>{children}</kimrofContext.Provider>
  )
}
```

# The useKimrofField() hook


useKimrofField.ts M

src > kimrof-user-editor > kimrof > useKimrofField.ts > ...

```ts
export function useKimrofField(name: string) {
  const { values, setFieldValue } = useContext(kimrofContext)

  const onChange = useCallback(
    (e: ChangeEvent<HTMLInputElement>) => {
      setFieldValue(e.target.name, e.target.value)
    },
    [setFieldValue]
  )

  return {
    value: values[name],
    onChange,
  } as const
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

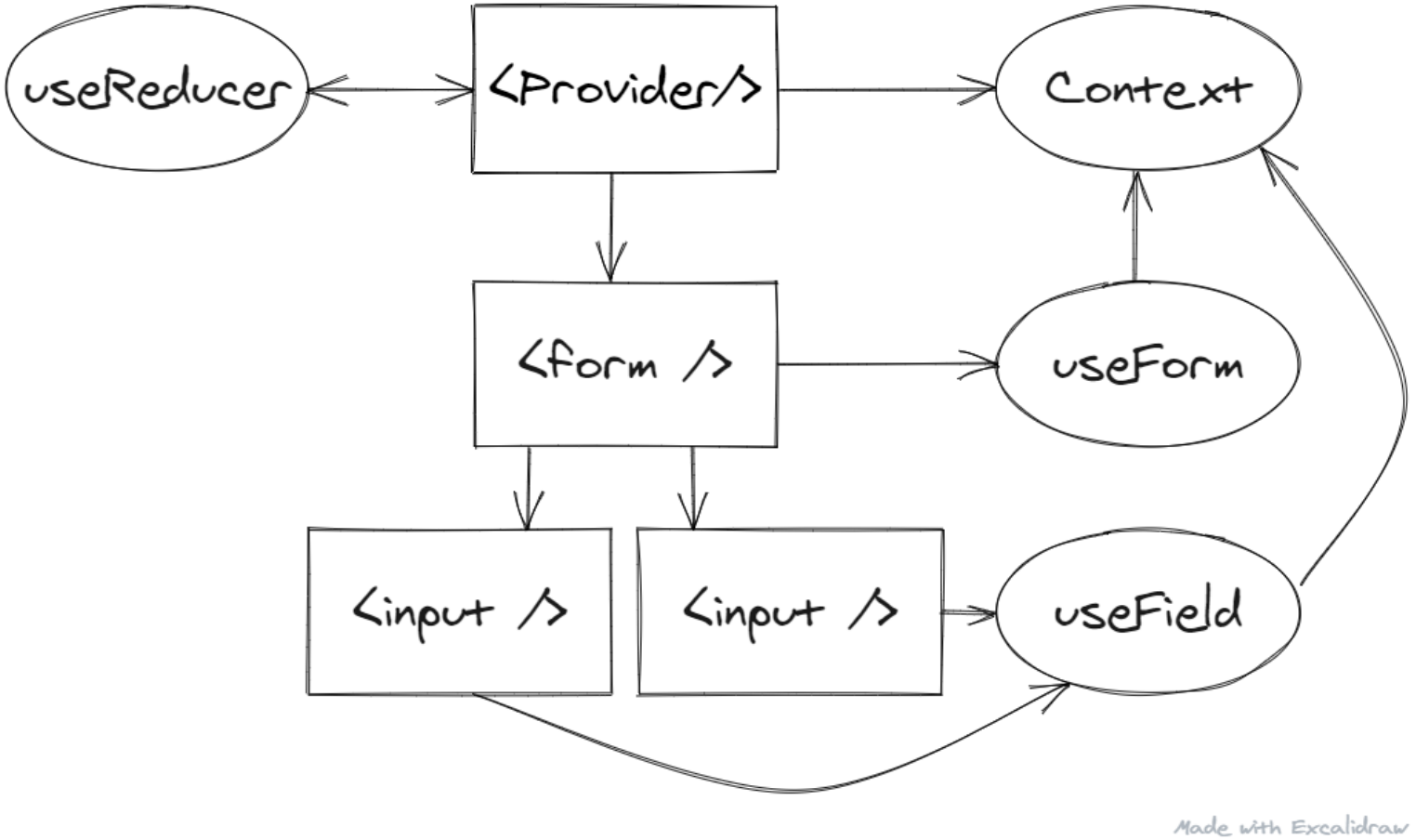
Maurice de Beijer - @mauricedb

# 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 />
- The **useKimrof() hook** returns values and metadata
  - The dirty and valid flags

# Kimrof Structure



# The kimrofContext

TS KimrofContext.ts 1, M

src > kimrof-user-editor > kimrof > TS KimrofContext.ts > ...

```ts
export interface KimrofContext {
  values: KimrofObject
  metadata: Metadata
  submitForm: () => void
  setFieldValue: (name: string, value: KimrofProperty) => void
}

export const kimrofContext = createContext<KimrofContext>({
  values: {},
  metadata: { isDirty: false, isValid: true },
  submitForm: () => void null,
  setFieldValue: () => void null,
})
```

# The Kimrof provider

Kimrof.tsx M

src > kimrof-user-editor > kimrof > Kimrof.tsx > Kimrof

```tsx
interface Props<TData> {
  children: ReactNode
  initialValues: TData
  onSubmit: (values: TData) => void
}

export function Kimrof<TData extends KimrofObject>({
  children,
  initialValues,
  onSubmit,
}: Props<TData>): ReactElement {
  const [{ values, metadata }, dispatch] = useReducer(kimrofReducer, {
    values: initialValues,
    metadata: { isDirty: false, isValid: true },
  })

  const context: KimrofContext = useMemo(
    () => ({
      values,
      metadata,
      submitForm: () => onSubmit(values as TData),
      setFieldValue: (name: string, value: KimrofProperty) => {
        dispatch({ type: "set-property", payload: { name, value } })
      },
    }),
    [values, metadata, onSubmit]
  )
```

# Adding the onSubmit handler to KimrofUserEditor

TS KimrofUserEditor.tsx M

src > kimrof-user-editor > TS KimrofUserEditor.tsx > ...

```tsx
export function KimrofUserEditor(): ReactElement {
  return (
    <Kimrof
      initialValues={initialPerson as IndexedPerson}
      onSubmit={(person) => alert(JSON.stringify(person, null, 2))}
    >
      <UserEditor />
    </Kimrof>
  )
}
```

# The useKimrofForm() hook

TS useKimrofForm.ts M ×

src > kimrof-user-editor > kimrof > TS useKimrofForm.ts > ...

You, seconds ago | 1 author (You)

```ts
import { useCallback, useContext } from "react"

import { kimrofContext } from "./KimrofContext"

export function useKimrofForm() {
  const { submitForm } = useContext(kimrofContext)

  const onSubmit = useCallback(
    (e: React.FormEvent) => {
      e.preventDefault()
      submitForm()
    },
    [submitForm]
  )

  return {
    onSubmit,
  } as const
}
```

# The useKimrof() hook

TS useKimrof.ts M ×

src > kimrof-user-editor > kimrof > TS useKimrof.ts > ...

You, seconds ago | 1 author (You)

```ts
import { useContext } from "react"
import { kimrofContext } from "./KimrofContext"

export function useKimrof() {
  const { values, metadata } = useContext(kimrofContext)

  return { values, metadata } as const
}
```

# Updating the UserEditor


TS UserEditor.tsx M

src > kimrof-user-editor > TS UserEditor.tsx > ...

```tsx
import React, { ReactElement } from "react"
import { KimrofLabeledField, useKimrof, useKimrofForm } from "./kimrof"

export function UserEditor(): ReactElement {
  const formProps = useKimrofForm()
  const {
    values,
    metadata: { isDirty },
  } = useKimrof()

  return (
    <form className="person-editor" {...formProps}>
      <h2>Kimrof User Editor</h2>
      <KimrofLabeledField label="Firstname:" name="firstname" />
      <KimrofLabeledField label="Surname:" name="surname" />
      <KimrofLabeledField label="Email:" name="email" />
      <KimrofLabeledField label="Address:" name="address" />
      <KimrofLabeledField label="Phone:" name="phone" />
      <button className="btn btn-primary" disabled={!isDirty}>
        Save
      </button>
      <hr />
      <pre>{JSON.stringify(values, null, 2)}</pre>
    </form>
  )
}
```

See you in the next video

# React Hooks Tips Only the Pros Know

Maurice de Beijer - @mauricedb

# Form Validation

# Adding validation to the KimrofPersonEditor


```tsx
export function KimrofPersonEditor(): ReactElement {
  return (
    <Kimrof
      initialValues={initialPerson as IndexedPerson}
      onSubmit={(person) => {
        alert(`Submitting\n${JSON.stringify(person, null, 2)}`);
      }}
    >
      <PersonEditor />
    </Kimrof>
  );
}
```

# 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