React Alicante

Hooked on React Hooks

Workshop

Basefactor Team

September 2019

Thanks to our sponsors!

devcenter
Advanced Technology

eventbrite

letgo

niche

taxfix

marketgoo

DEISER

Interacso

FRONTASTIC

BASEFACTOR
{ CONSULTANCY BY LEMONCODE }

mongoDB

wework

XING
For a better working life

Limenius

Concepts

State, Functional Components,
Side Effects …

Destructuring in a nutshell

Destructuring is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into different variables:

Destructuring Object

```
const myObject = {
  id: '1',
  name: 'John',
  lastname: 'Doe',
  country: 'U.S.A.'
}

const {name, country} =
myObject;

console.log(name);
// John
console.log(country);
// U.S.A.
```

Destructuring Array

```
const countries = ['Spain',
'Belgium', 'France'];

const [first, second] =
countries;

console.log(first);
// ‘Spain’
console.log(second);
// ‘Belgium’
```

Spread & Rest operators in a nutshell

Spread operator (...) can be used for several purposes, the most common use is to make a copy of an existing object or to create a new object with more properties. Rest operator (...) is mainly used for capturing and grouping parameters into an array:

Spread

```
const defaultClient = {
  id: -1,
  name: '',
  country: 'France',
}

const createNewClient = (name) => ({
  ...defaultClient,
  name,
});

console.log(createNewClient('John’));
/*{
  id: -1
  name: 'John',
  country: 'France'
}*/
```

Rest

```
const sum = (...args) => args.reduce(
  (value, total) => total + value
);

console.log(sum(1, 1));
// 2
console.log(sum(1, 1, 1, 1));
// 4
console.log(sum(1, 2, 3));
// 6
```

Props

TL;DR If a Component needs to alter one of its attributes at some point in time, that attribute should be part of its state, otherwise it should just be a prop for that Component.

Props

- Components configuration
- Parent component is responsible of feeding them
- Props are read only / immutable
- Props can be data or callbacks

```
import * as React from "react";

interface Props {
 fullname: string;
}

const DisplayNameComponent = (props: Props) =>
<h2>{props.fullname}</h2>;

const App = () => (
 <DisplayNameComponent fullname="John Doe" />
);
```

Source: https://stackoverflow.com/questions/27991366/what-is-the-difference-between-state-and-props-in-react

State

- State is a data structure that starts with a default value when a Component mounts.
- It may be mutated across time, mostly as a result of user events.
- A component manages its own state internally. Besides settings an initial state, it has no business fiddling with the state of its children.
- You might conceptualize state as private to that component.

```
interface Props {}

interface State {
 fullname: string;
}

export class NameEdit extends React.Component<Props, State>
{
 constructor(props: Props) {
  super(props);
  this.state = { fullname: "John Doe" };
 }

 render() {
  return (
   <>
    <h1>{this.state.fullname}</h1>
    <input
      value={this.state.fullname}
      onChange={e =>
       this.setState({ fullname: e.target.value })
      }
    />
   </>
  );
 }
}
```

Source: https://stackoverflow.com/questions/27991366/what-is-the-difference-between-state-and-props-in-react

Side Effects

A "**side effect**" is anything that affects something outside the scope of the function being executed**.**

Examples

- Network Request, which has your code communicating with a third party and thus making the request, causing logs to be recorded, caches to be saved or updated and all sorts of effects that are outside the function.
- Pushing a new item into an array that was passed in as an argument is a side effect as well.

Functions that execute without side effects are called "pure" functions:

- They take in arguments
- They return values

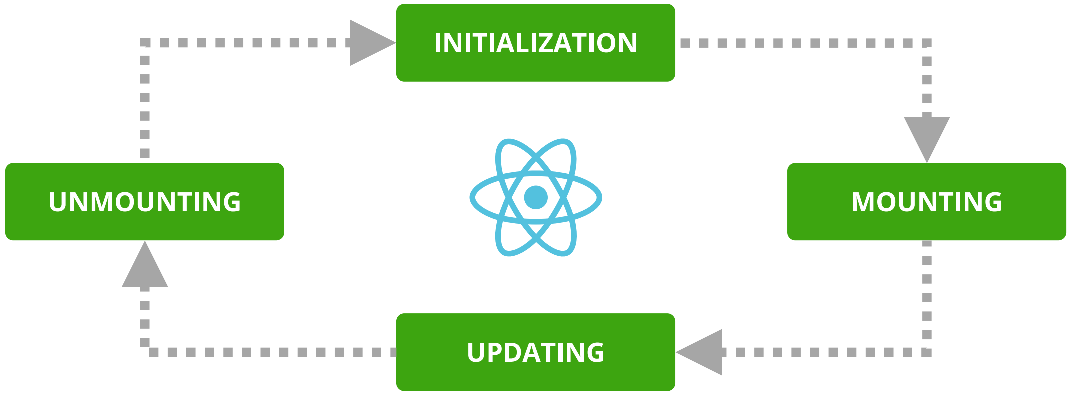
Nothing else happens upon executing the function.

Source: https://www.reddit.com/r/reactjs/comments/8avfej/what_does_side_effects_mean_in_react/

Component Lifecycle

As we know, everything in this world follows a cycle, say humans or trees. We are born, grow, and then die. Almost everything follows this cycle in its life, and React components do as well.

Components are created (mounted on the DOM), grow by updating, and then die (unmount on DOM). This is referred to as a component lifecycle.



Source: https://www.freecodecamp.org/news/how-to-understand-a-components-lifecycle-methods-in-reactjs-e1a609840630/

Class vs Functional Components

Class

A class component offers:

- State
- Lifecycle

Functional

A function based component:

- It is stateless
- It does not have lifecycle events

NOT ANYMORE!

Hooks

What the hell is that?

What are React Hooks?

React Hooks are **functions** that let us hook into the React state and add lifecycle features from function components.

“They add super powers to a functional component"

STATE

LIFECYCLE LIKE

“Functional components become first citizen class"

Source: https://hackernoon.com/react-hooks-usestate-using-the-state-hook-89ec55b84f8c

Basic hooks `useState`

UseState is a built-in React hook that lets you add state in your functional components.

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

Basic hooks `useState`

Clone repo: https://github.com/Lemoncode/workshop-boiler-plate

Codesandbox: https://codesandbox.io/s/github/lemoncode/workshop-boiler-plate

Use state

Use state object

Demo Time

Basic hooks `useEffect`

UseEffect is a hook for encapsulating code that has 'side effects'. It is like a combination of *componentDidMount , componentDidUpdate , and componentWillUnmount* . Previously, functional components didn't have access to the component lifecycle, but with useEffect you can tap into it.

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

const App = () => {
  const [name, setName] = useState("Ceci");

  useEffect(() => {
    document.title = name;
  }, [name]);

  return (
    <input
     value={name}
     onChange={event => setName(event.target.value)}
    />
  );
}
```

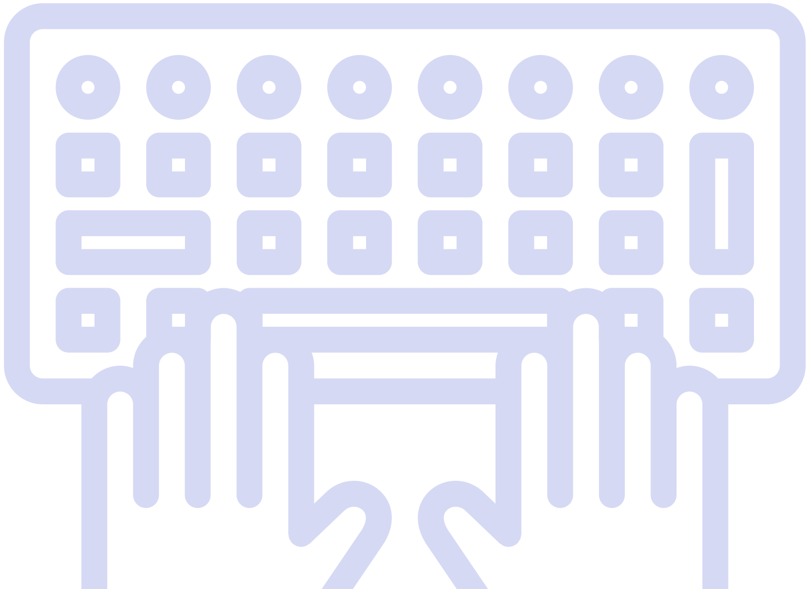
Basic hooks `useEffect`

Component did mount + did update

Component unmount

Ajax field change

Demo Time



Rules of hooks

✓ Only call Hooks at the Top Level

Don’t call hooks inside loops, conditions, or nested functions.
You must ensure that hooks are called in the same order each time a component renders.

✓ Only call Hooks from React Functions

Don’t call hooks from regular Javascript functions. Instead:

- ✓ Call Hooks from React function components
- ✓ Call Hooks from custom Hooks

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

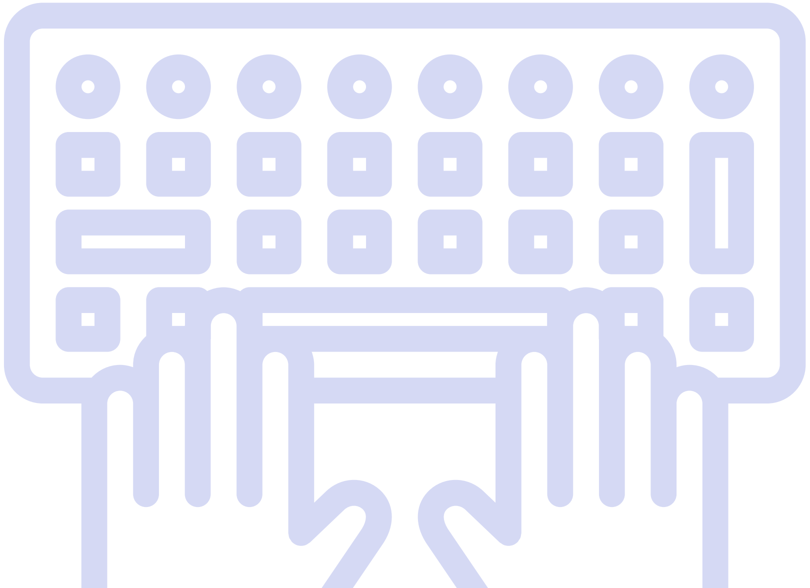
Why Hooks?

Let’s code!

Class component pitfalls

- The “this” nightmare
- Monolyth state plus hard to extract functionallity
- Managing related concerns in separate event handlers
- High Order Components usage noise
- Swap from functional to class and vice versa
- Mixed approach functional vs classes

Demo Time



Block 1

Custom Hook

Pure Component

Pure Component Callback

Demo Time

Block 2

useReducer

useContext

Demo Time

Block 3

Async Closure

Promise Unmounted

Should Update

Demo Time

Exercises

Challenges

Challenge 1

Starting Material

In our application we need to load tons of collections from servers (usually lookups), and the server rest-api comes from a third party, for the sake of performance:

- We don't want to load all the lookups at application startup
- We dont' want to load a given lookup on every request
- We want to be lazy, load one first request, and keep it globally in context

Startup Repository: https://github.com/Lemoncode/react-alicante-hooks-workshop/tree/master/06%20Excercise%201/00-start

Tips

- **Make it global:**
  - Move the lookup data to context.
  - Create a hook that has access to this context and expose a loadMethod.
  - Use the hook on PageA and PageB.
- **Make it lazy:** Just add a state on the context (or check if the lookup is null or empty), and on the custom hook you have created check if the lookup has been already loaded, and depending on the result just return the promise resolved with the data if not fire the async request.
- **Extra:** What if we can launch in parallel several request at the same for the same lookup?
  Could we just store the first promise launched and return that?

Challenge 2

Starting Material

In a previous initial sample we just learnt how to filter a list of users from a rest-api, listening to an input onChange event.

This is not optimal, shouldn't it be cool to wait a bit until the user is not typing for some milliseconds in order to fire the server request?

Startup Repository: https://github.com/Lemoncode/react-alicante-hooks-workshop/tree/master/03-block-1/00-custom-hook

Tips

- You can try to build something like that from scratch, your custom debounce (nice excercise).

- Read from this blog post and check if the solution was similar to your proposal: https://dev.to/gabe_ragland/debouncing-with-react-hooks-jci

- Use an already-made hook: https://github.com/xnimorz/use-debounce

Challenge 3

Starting Material

CSS media queries are great when we are creating a responsive web design but sometimes, we need to apply same media queries in the Javascript and execute code like:

"isDesktop ? use Header component else use HamburgerMenu"

Startup Repository: https://github.com/Lemoncode/react-alicante-hooks-workshop/tree/master/08%20Exercise%203/00-start

Tips

- Play with component feeding *showMenu* props equals true / false.

- Create `useMediaQuery` hook. You can use matchMedia (https://developer.mozilla.org/en-US/docs/Web/API/Window/matchMedia) Javascript method to check if the document matches the media query.

BASEFACTOR

{ CONSULTANCY BY LEMONCODE }

basefactor.com

github.com/lemoncode

@basefactorteam

facebook.com/lemoncoders