# React coding

### Voorbeeld

### Zie ook

* ZIE: SAMENVATTING react (folder)
* rebrand.ly/reactbootcamp

https://www.youtube.com/watch?v=BYbgopx44vo

### Te gebruiken bij:

-

### Eigenschappen:

#### What you'll learn

* Components
* JSX
* Styling components
* Props & state
* Event handling
* Lifecycle methods
* HTTP
* Forms
* More!

### Info

Afbeelding met tekst, schermopname, Webpagina, software

Automatisch gegenereerde beschrijving

**Update from 2019: I wrote this article a long time ago and my views have since evolved. In particular, I don’t suggest splitting your components like this anymore. If you find it natural in your codebase, this pattern can be handy. But I’ve seen it enforced without any necessity and with almost dogmatic fervor far too many times. The main reason I found it useful was because it let me separate complex stateful logic from other aspects of the component.** [**Hooks**](https://reactjs.org/docs/hooks-custom.html) **let me do the same thing without an arbitrary division. This text is left intact for historical reasons but don’t take it too seriously.**

There’s a simple pattern I find immensely useful when writing React applications. If [you’ve been doing React for a while](http://facebook.github.io/react/blog/2015/03/19/building-the-facebook-news-feed-with-relay.html), you have probably already discovered it. [This article explains it well](https://medium.com/@learnreact/container-components-c0e67432e005), but I want to add a few more points.

You’ll find your components much easier to reuse and reason about if you **divide them into two categories.** I call them Container and Presentational components\* but I also heard Fat and Skinny, Smart and Dumb, Stateful and Pure, Screens and Components, etc. These all are not exactly the same, but the core idea is similar.

My **presentational** components:

* Are concerned with how things look.
* May contain both presentational and container components\*\* inside, and usually have some DOM markup and styles of their own.
* Often allow containment via this.props.children.
* Have no dependencies on the rest of the app, such as Flux actions or stores.
* Don’t specify how the data is loaded or mutated.
* Receive data and callbacks exclusively via props.
* Rarely have their own state (when they do, it’s UI state rather than data).
* Are written as [functional components](https://facebook.github.io/react/blog/2015/10/07/react-v0.14.html#stateless-functional-components) unless they need state, lifecycle hooks, or performance optimizations.
* Examples: Page, Sidebar, Story, UserInfo, List.

My **container** components:

* Are concerned with how things work.
* May contain both presentational and container components\*\* inside but usually don’t have any DOM markup of their own except for some wrapping divs, and never have any styles.
* Provide the data and behavior to presentational or other container components.
* Call Flux actions and provide these as callbacks to the presentational components.
* Are often stateful, as they tend to serve as data sources.
* Are usually generated using [higher order components](https://medium.com/@dan_abramov/mixins-are-dead-long-live-higher-order-components-94a0d2f9e750) such as connect() from React Redux, createContainer() from Relay, or Container.create() from Flux Utils, rather than written by hand.
* Examples: UserPage, FollowersSidebar, StoryContainer, FollowedUserList.

I put them in different folders to make this distinction clear.

### Benefits of This Approach

* Better separation of concerns. You understand your app and your UI better by writing components this way.
* Better reusability. You can use the same presentational component with completely different state sources, and turn those into separate container components that can be further reused.
* Presentational components are essentially your app’s “palette”. You can put them on a single page and let the designer tweak all their variations without touching the app’s logic. You can run screenshot regression tests on that page.
* This forces you to extract “layout components” such as Sidebar, Page, ContextMenu and use this.props.children instead of duplicating the same markup and layout in several container components.

https://medium.com/@dan\_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Afbeelding met tekst, schermopname, software, Computerpictogram

Automatisch gegenereerde beschrijving

Afbeelding met tekst, elektronica, software, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

Afbeelding met tekst, software, schermopname, computer

Automatisch gegenereerde beschrijving

### Standard opbouw

Index.html

<html>

<head>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div id="root"></div>

<script src="index.pack.js"></script>

</body>

</html>

Index.js

import React from "react"

import ReactDOM from "react-dom"

\\jsx

ReactDOM.render<div><h1>Hello world!</h1><p>This is a paragraph</p></div>

, document.getElementById("root"))