

***React***

**Ein Erfahrungsbericht**

# Disclaimer

- Kein React vs. X vs. Y Hate
- Ich bin kein React Experte™
- Vieles kann man sicherlich schöner lösen

H V Kehiaian	8,929
--------------	-------

# Carlo.js



# Dimensions

## Backend-Stack

- Python
- Pyramid Framework
- Jinja Templates

# Dimensions

## Frontend-Stack

- Sass
- Natives JS Framework
- React

# Dimensions

## v1 (bis Januar 2018)

- Ajax Calls (async HTML Einbettung)
- Bootstrap
- Exzessive jQuery-Suppe

# Dimensions

v2 (ab Januar 2018)

- Ajax JSON Calls
- Natives JS
- React



# React

## Am Anfang...



# JSX - XML in JS

Example [1](#):

```
<div id="app">  
  <p>Foo</p>  
  <p>Bar</p>  
</div>
```

- 
1. <https://lernen.react-js.dev/die-grundlagen/jsx-eine-einfuehrung> ↩

# JSX - XML in JS

Example [1](#):

```
React.createElement(  
  'div',  
  { id: 'app' },  
  React.createElement('p', null, 'Foo'),  
  React.createElement('p', null, 'Bar')  
);
```

- 
1. <https://lernen.react-js.dev/die-grundlagen/jsx-eine-einfuehrung> ↩

**Komponenten...**

## ... als Funktion:

```
function ProfilePage(props) {  
  const handleClick = () => {  
    alert('Followed ' + props.user);  
  };  
  
  return (  
    <button onClick={handleClick}>Follow</button>  
  );  
}
```

[Demo](#)

## ... als Klasse:

```
class ProfilePage extends React.Component {  
  handleClick = () => {  
    alert('Followed ' + props.user);  
  };  
  
  render() {  
    return <button onClick={this.handleClick}>Follow</button>;  
  }  
}
```

[Demo](#)

# Klasse vs. Funktion (React < 16.8!)

Functional vs class-based React components	
Functional	Class-based
<b>1</b> Functional programming style	<b>1</b> Object-oriented programming style
<b>2</b> Minimal boilerplate Clean and simple	<b>2</b> Can have state
	<b>3</b> Can have lifecycle methods  perform actions when the component is mounted, unmounted, about to be updated, etc.
	<b>4</b> Can have refs Reference and manipulate underlying DOM elements
	<b>5</b> Performance optimisation With <code>shouldComponentUpdate</code> and <code>PureComponent</code> <b>Use with caution!</b>
<a href="https://ozmoro.com">https://ozmoro.com</a>	

# Local state management

- Lesen: `this.state`
- Schreiben: `this.setState()`

[Documentation](#)



# "Pure" Components

Rendering, nur wenn es sein muss!

<https://codesandbox.io/s/n55r8v94n0>

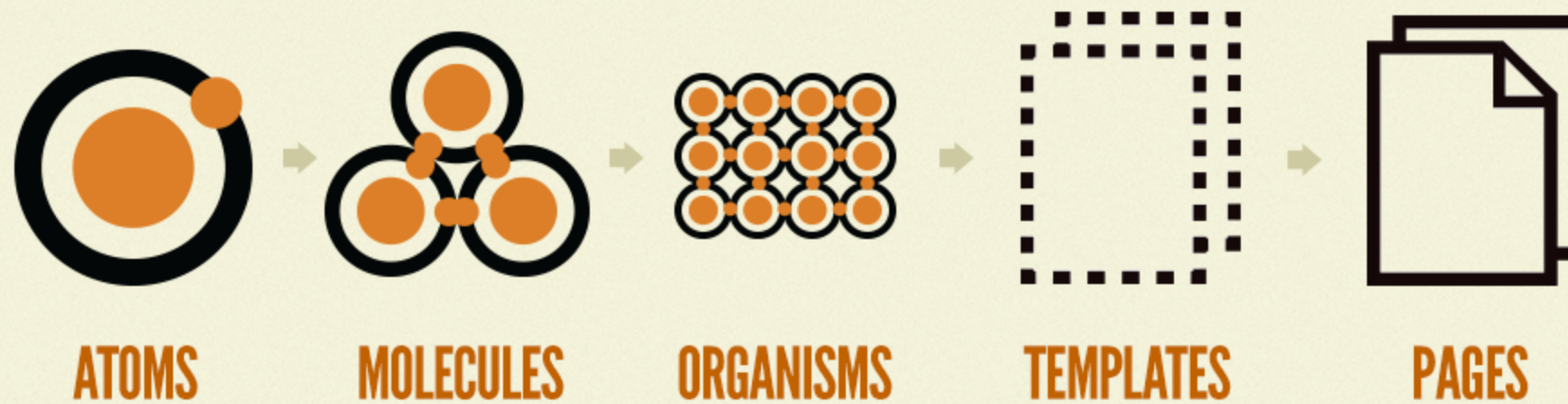
# React Komponenten

- Properties sind read-only, immutable
- stateful vs. stateless
- uncontrolled vs. controlled
- Wasserfall vs. Inverse Data Flow

# Thinking in React

<https://reactjs.org/docs/thinking-in-react.html>

# Wie strukturieren?



# React Hooks

[Hooks Documentation](#)

# Klasse vs. Funktion (mit Hooks!)

Was hat sich geändert?

Functional vs class-based React components	
Functional	Class-based
1 Functional programming style	1 Object-oriented programming style
2 Minimal boilerplate Clean and simple	2 Can have state
	3 Can have lifecycle methods perform actions when the component is mounted, unmounted, about to be updated, etc.
	4 Can have refs Reference and manipulate underlying DOM elements
	5 Performance optimisation With <code>shouldComponentUpdate</code> and <code>PureComponent</code> <b>Use with caution!</b>

<https://ozmoroz.com>

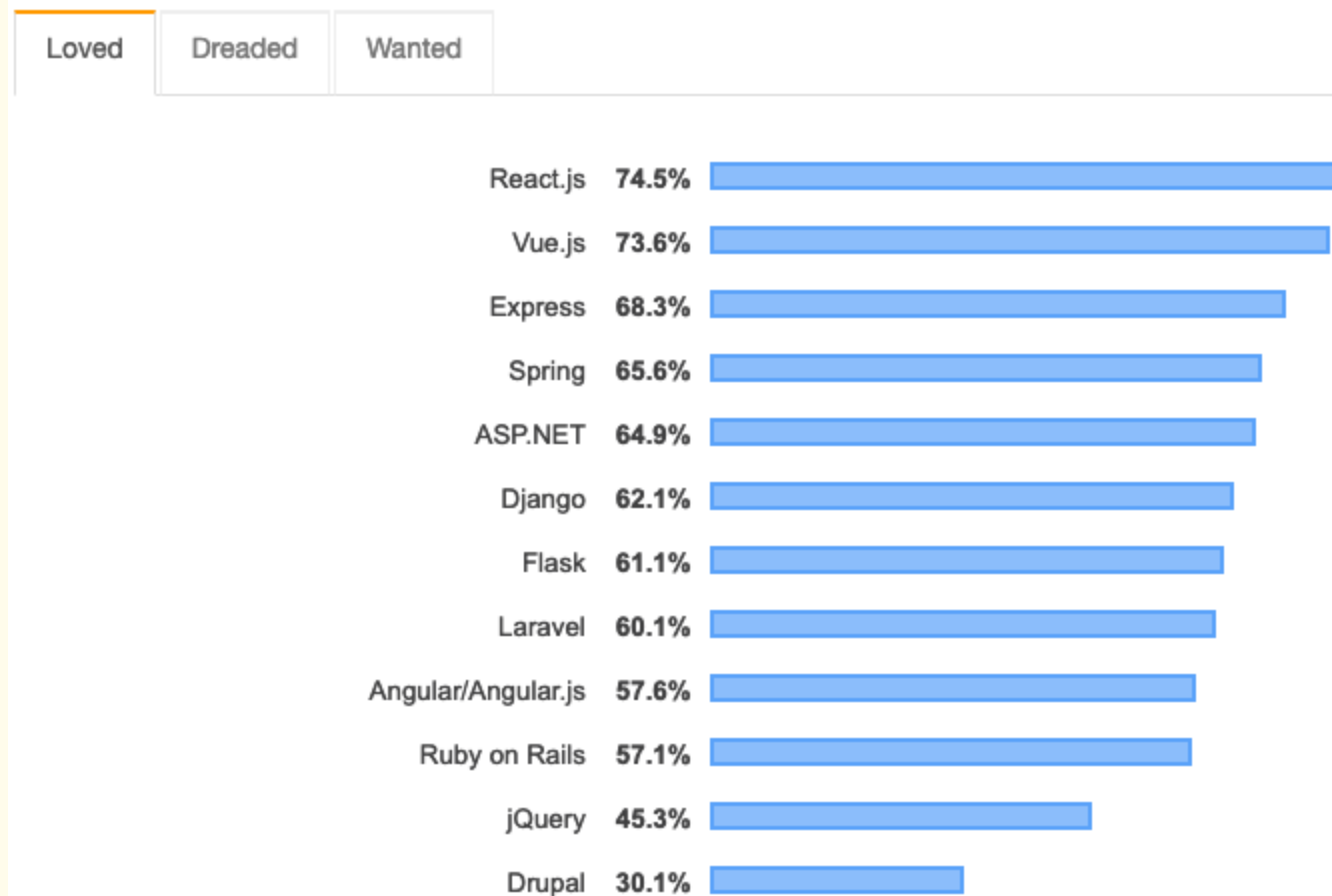
# Zusammenfassung

React ist eine Library zur Implementierung von UI Komponenten.

# React und Vue.js

## Meist geliebt und gewünscht

### Most Loved, Dreaded, and Wanted Web Frameworks



*% of developers who are developing with the language or technology and have expressed interest in continuing to develop with it*

React.js and Vue.js are both the most loved and most wanted web frameworks by developers, while Drupal and jQuery are most dreaded.

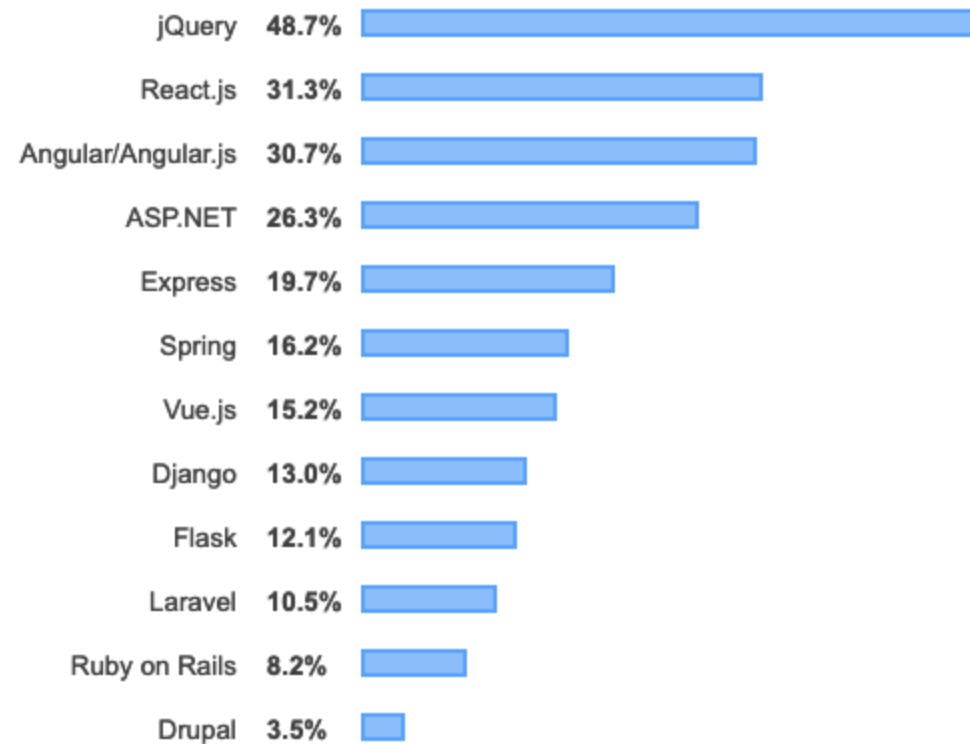


# Große Community

## Web Frameworks

All Respondents

Professional Developers



63,585 responses; select all that apply

This year, we asked about frameworks for the web separately from other frameworks and libraries. jQuery is the most broadly used of these web frameworks, and this year more developers say they use React.js than Angular, a switch from last year.

# Tipp 1

Nutzt Storybook o.Ä. um mit dem Design-Team  
an der Code-Basis zu arbeiten

# Tipp 2

Extended nur von `React.Component` oder  
`React.PureComponent`, nicht von eigenen Komponenten.

# Tipp 3

Nutzt [PropTypes](#) für Typencheck (Linting).

# Tipp 4

Definiert eine konsistente Ordnerstruktur.

# Tipp 5

"Make each component do one thing well." (UNIX Philosophie)

# Tipp 6

Schreibt Tests für eure Komponenten.

# Tipp 7

Lernt JavaScript. :D



# Artikel

- [Writing Resilient Components](#)
- [How are function components different from classes](#)

# Tools

- [DevTools](#)
- [Codesandbox.io](#)
- [Create React App Boilerplate](#)
- [Lifecycle Diagram](#)

# Tools

- [Storybook](#)
- [PropTypes](#)
- [Styled Components](#)

# Bücher

- [React lernen](#)

# Personen

- [Dan Abramov](#) (React Core Team)
- [Kent C. Dodds](#) (Paypal)
- [Twitter-Thread](#)

# Podcasts

- [Reactive](#)
- [React Podcast](#)