React Best Practices

2 - Komponenten strukturieren

Timo Mämecke TH Köln // MI Master // Weaving the Web 25. Juni 2019

Inhalt

Grundlegendes zu Komponenten.

Typische Fehler vermeiden.

Patterns zum Strukturieren.

Grundlegendes zu Komponenten

Neue Komponenten schreiben ist euer täglich Brot.



Nicht blocken lassen

Neue Komponenten anlegen soll kein Aufwand sein.

Komponenten anlegen, refactoren, löschen, ist ganz normal.

Werdet schnell darin, neue Komponenten anzulegen.

Beispiel: Code Snippet anlegen

Jede Komponente entsteht bei mir so:

```
import React from 'react'
import PropTypes from 'prop-types'

function SomeComponent(props) {
  return <div />
}

SomeComponent.propTypes = {}

export default SomeComponent
```

```
import React from 'react'
      import PropTypes from 'prop-types'
      function SomeComponent(props) {
        return <div />
      SomeComponent.propTypes = {}
      export default SomeComponent
10
```

Video: vom Snippet zur Component

Functional Components vs Class Components

```
function Greeting({ name }) {
  return <div>Hello, {name}!</div>
}
```

```
class Greeting extends React.Component {
  render() {
    return <div>Hello, {this.props.name}!</div>
  }
}
```

Vor React 16.8: Keinen State, keinen Lifecycle.

Haben State & Lifecycle.

Functional Components vs Class Components

```
function Greeting() {
  const [name, setName] = useState('...')
  useEffect(() => {
    fetch('name-api.tld')
        .then(res => res.json())
        .then(json => setName(json.name)
    })

  return <div>Hello, {name}!</div>
}
```

Seit React 16.8:
Hooks erlauben State & Lifecycle.

```
class Greeting extends React.Component {
 constructor(props) {
    super(props)
    this.state = {
     name: '...'
 componentDidMount() {
    fetch('name-api.tld')
      .then(res => res.json())
      .then(json => this.setState({ name: json.name })
 render() {
   return <div>Hello, {this.state.name}!</div>
```

Functional Components vs Class Components

Seit Hooks haben Class Components keine Vorzüge mehr.

Class Components sind nicht tot und müssen nicht umgebaut werden.

Functional Components mit Hooks sind schlanker und verständlicher.



Unübersichtlicher Code

```
render() {
  let sets = []
  for (let step = 0; step < this.state.setCount; step++) {</pre>
   sets.push(<Score key={step} index={step} score={this.setScore.bind(this)}/>)
  return (
    <d1v>
      {this.props.league && <HeaderWrapper><PlayerSelectAndShow preSelect={this.state.team1.concat(this.state.team2)} break={
     </HeaderWrapper> }
      {this.state.completed ፟፟፟፟፟፟ <div>
        <h4 className='aHeadline asSmall'>Result</h4>
       { sets }
       <AddSet className='headlineFont' onClick={()=>this.setState({setCount: this.state.setCount + 1})}>+</AddSet>
      </div>}
      { this.state.completed && this.state.sets.size > 0 && <Button onClick={()=> this.saveMatch()}>
        Save Match
      </Button> }
    </div>
```

Inkonsistente Einrückung und Whitespaces, kein Zeichen-Limit pro Zeile, ...

Zu große Components

```
Limport Frantispes from "proc-types
 import ( css ) from 'glamor'
   Bespons (veContainer
   BarChart,
   Maxin.
    "oaltip,
   LabelList
 import ( injectint) from 'meact-intl'
 import Bycharticottip from "./Hycharticottig"
  import ByChartRoundedEar from './HyChartRoundedBur'
 import ( colors, RESOLUTIONS ) from '../../constants'
If Move bars lov up because otherwise they overlay the year's
 caseglobal(", recharts-layer, recharts-ber", 4
  transform: "translate(6, =los)"
 css, global(", recharts-layer, recharts-bar-rectangle path", [
  transitions 'fill 130ms'
 caseglobal(".recharis-layer.recharts-bur-rectangle:hover path", (
   '1tt: cotors.darktrey
  issiglobal(".recharts-layer.recharts-bar-rectangle:hover path.unverified". (
 issiglocation recharts-surface, (
 class HerizontalBarthart extends React.PureComponent (
   static prooTypes - (
     data: PropTypes.arrayOf(
       Progrypes, shape of
         pedestrians(ount: FropTroes, number, isRequired.
         timestamps PropTypes.pneOfType()
           PropTypes.string.
           PrapTypes.instanceOf(Bate)
         1). isRequired,
         unverified: PropTypes_bool
      reselution: PropTypes.oneDf(['hoar', 'day', 'week', 'month']).isRequired,
       latestMeasurement: PropTypes.string.isRequired,
    formatYAxisTick = tick => {
     const { formatNumber } = this.preps.intl
      return farnationher(tirk)
    formatXAxisTick = dayDiff == timestamp == {
     const { resolution } = this.props
     const ( formatMessage ) = this.procs.intl
     const date = moment.unix(timestamp)
     let id = "shart.x_f{resolut.on}"
       // If multiple days are shown, each new day should be visible
       const shouldMeutay = dayD:ff = 8 && date.diff(startOfDay) --- 8
       id - cheuAsNewCay ? "_day" : "_time"
      return date.format(formatMessage({ id }))
    // Transferm timestamp to unit time (seconds), necessary for recharts.js
    setTransformedOsts - data -- (
     const { parlicattonsyrement } = this,props
     return data.maplobj -> (
       const issefore - moment(obj.timestamp).isBefore
         moment lear [!estMeasurement]
       const incomplete - isBefore LG obj.perestriansCount - 0
         incomplete: incomplete,
          timestamps moment(db).timestamp).unix()
```

```
// Get max value of y-Axis
setMacY = [] => {
 const maxPedestriansCount = Math.max(...pedestriansCounts)
 return maxmedes1r3an;Count
// De'ine step size of y-Axis. depending on max walse of y-Axis.
If Always keeps amount of ticks between 4 and 6 by much calculations
// and human readable step sizes.
ectYAcinStrpSize = maxPedentriamsCount +> (
 # maxPbd(striinscount > Seemes) (
   return Math, floor(naxPedestriansCount / 5000000: + 1200000
 of (maxPedestriansCount = 1000000) {
   return Meth. floor(nexPedestriansCount / Lecesse) + 25esee
   return Mith. floor(naxPedestriansCount / 200000) + 50000
 of |maxPodestriensCount > 50000) (
   return Moth. Floor (maxPedestriansCount / 25000) * 12500
 # maxPedestriansCount > 10000) {
   return Mith, floor(naxPedestriansCount / 18088) + 2888
 of (maxPodes triansCount > 2500) (
   return Mith, Floor (nawnegest rianstount / 2500) + 500
 # maxPedestriansCount > 500) {
   return Mith. floor(nonPedectriansCount / 100) + 100
 If maxPtd(striinscount > 100) (
   return Mith. floor(naxPedestriansCount / 188) + 25
// Return an array of ticks for y-locis
getYAsisTicks = () - (
 const maxPedestriansCount = this.getMoxY()
  const stapSize = this.getVArisStapSize(mayPedestriansCount)
 const tickeLength = Math.ce:l(maxPedestrians(ount / steplize) + 1
 retern [...Array(ticks)ergth)].map((value, index) => index = stepSize)
eetXTickInterval = dayDiff +> {
 switch (dayDiff) (
     netwrn 1
   case 2:
   case 3:
   default:
 const ( sala ) = this, grops
 return moment(data[data.length - 1].timestamp).diff(
   data[8].timestamp,
rempertooltigContent = props -> (
 const { active, ...passProps } = props
 of (lactive) return null
  return whythartToeltip resolutionwithis props resolution) (. passProps) /w
 const { resolution, data, earliestheasusement } = this.props
 const transformedCata = this.getTransformedDuta(duta)
```

const yTicks a this.getYArisTicks()

```
compt yourier = (yTicks[1] - yTicks[0]) / t
        const dayDiff = this.octoby91ff()
        // interval of x axis is dynamically calculated when hour resolution
          resolution .... SESMITTING. HOUR
           * this getHTickInterval(dayBiff)
           preserveStart
          «ResponsiveContainer width="189%" height={558}»
            <BarChart
                 domain=(["dataMin", "dataMax"])
                minTickSag-{10}
                interval=(sInterval)
                tickromatter=(trus.formatxAxisiic((dayDiff))
                tickLine=(false)
                tick-[[
                  fill: '4761738'.
                  fentSizes 14,
                  fontWeight: 700,
                  transform: 'trans'ate(4, 7)'
                stroke="#EFC4C8"
                padding={{ left: 20, right: 20 ]}
                strokeWidth=(2)
                 domain=([0, 'datafar + $(ybuffer)'])
                 ticks={yTick;}
                  14 U. 147617781,
                  fontSize: 14,
                  fontWeight: 200,
                  transform: 'translate(-7, 0)'
                tickFormatter=(this.format\AxisTick)
                strokes"#BEC4CE"
                strokeWidth-(2)
                offset=i0)
                position=(( y: 0 ))
                content-{mi.renderTcoltipContent}
                 manAmount=(yManTick = yBufter)
                dataKey="pedestrian:Count"
                 fill=(colers.baseColor)
                 shape={«HyChartRoundedBar /»}
                 minPointSize=(4)
                [transfermedbata.map(entry -> {
                  return 6
                     key*(entry.timestamp)
                       entry unversified ? colors accentCalor : rolars baseColor
                      unverified feator, unverified)
                      incomplete (estry, incomplete)
           «Responsive Container»
276 export default injection (barizantalitae(bart)
```

Props falsch angewendet

Keine PropTypes. 😀

Große Objects in einer Prop. 😀

Typische Fehler vermeiden

Konsistenter Code-Style:

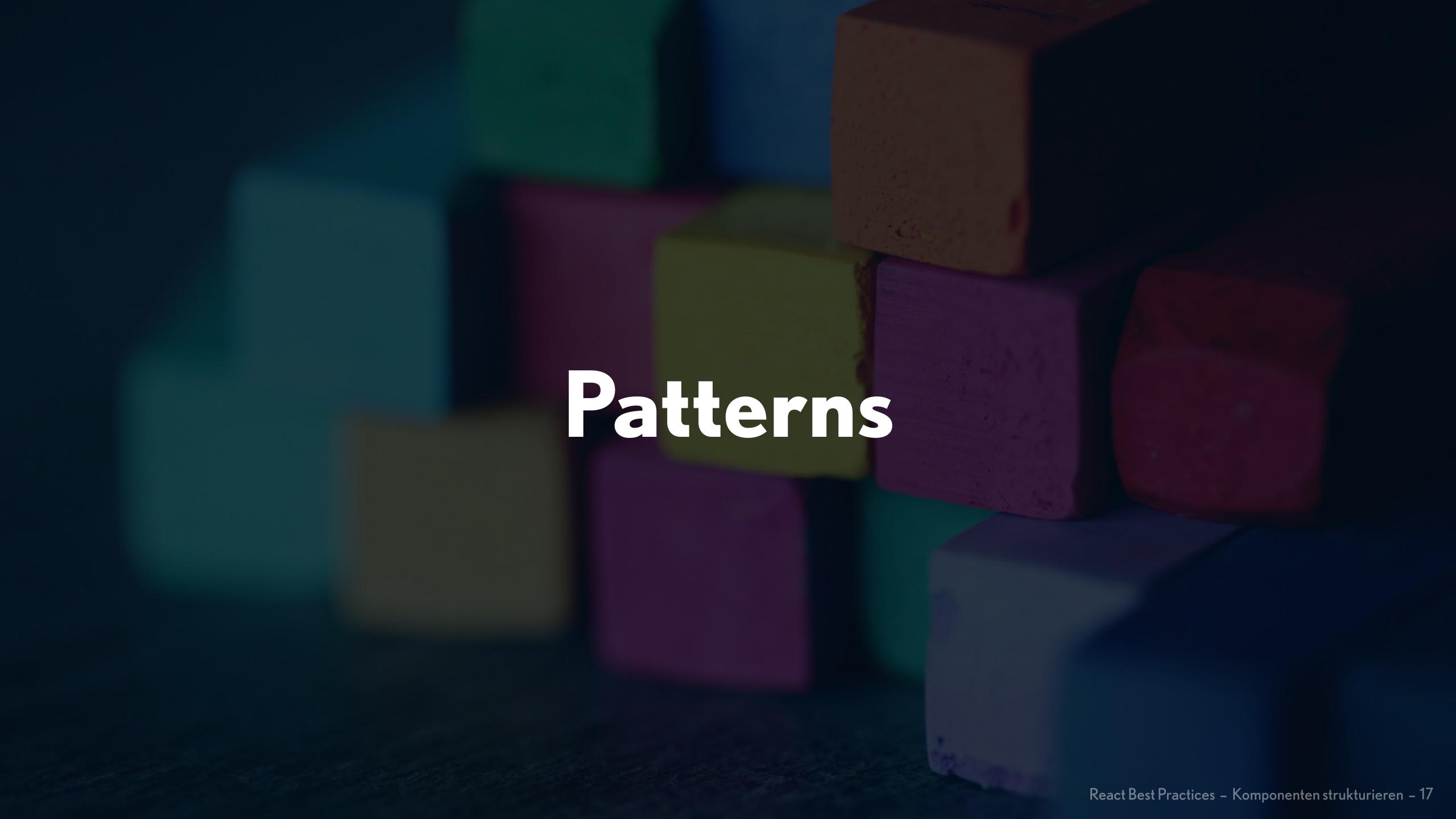
- Single Quotes vs Double Quotes
- Einrückung
- mehr dazu in "5 Code Quality & Testing"

Übersichtliche Komponenten:

- Eine Component hat eine bestenfalls nur eine Aufgabe
- Große Komponenten in mehrere kleine aufbrechen
- Stateful Components vs. UI Components
- PropTypes auch als Dokumentation nutzen

Wie ich Features baue

- 1 Großes Feature in kleine Teile aufbrechen.
- 2 Feature-Teil in einer großen Komponente runterschreiben.
- 3 Alle Concerns erkennen, um die sich diese Komponente kümmert. (State, Lifecycle, asynchrone Daten laden, Ul darstellen, ...)
- 4 Komponente in ihre Concerns refactoren.



Lifting the State

Problem:

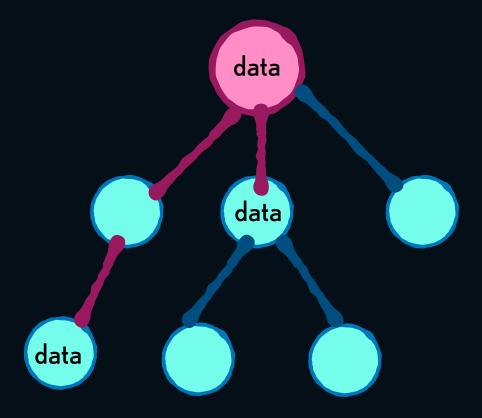
mehrere Components basieren auf gleichem State.

Falsch:

State in mehreren Components synchron halten.

Richtig:

State in gemeinsame Eltern-Komponente verschieben. State via Props von oben nach unten durchreichen.



^{*} Siehe: https://reactjs.org/docs/lifting-state-up.html

Render Props

Problem:

- Components werden immer tiefer verschachtelt.
- Props werden immer weiter durchgereicht. "Prop Drilling"

Lösung: Render Props¹
Komponenten nicht direkt voneinander abhängig machen, sondern dynamisch rendern.

```
<Dashboard renderTitle={user => (
    <Headline>Hello {user.name}!</Headline>
)} />
```

¹ https://reactjs.org/docs/render-props.html

Context

Problem:

- Props werden immer weiter durchgereicht. "Prop Drilling"
- Gleicher State wird an sehr vielen unterschiedlichen Stellen benötigt

Lösung: Context¹
Gemeinsame Daten in Context schieben

```
function Dashboard() {
  const user = useContext(UserContext)
  return <Headline>Hello {user.name}!</Headline>
}
```

¹ https://reactjs.org/docs/context.html#when-to-use-context

Regeln für Props

Props "flach halten".

- Große Objects in einer Prop vermeiden.
- Lieber mehrere Props mit kleinen oder gar keinen Objects.

Immer PropTypes nutzen.

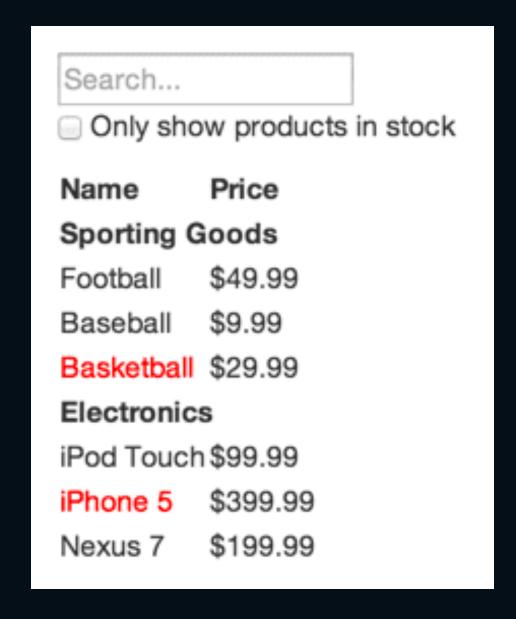
PropTypes mit Objects¹:

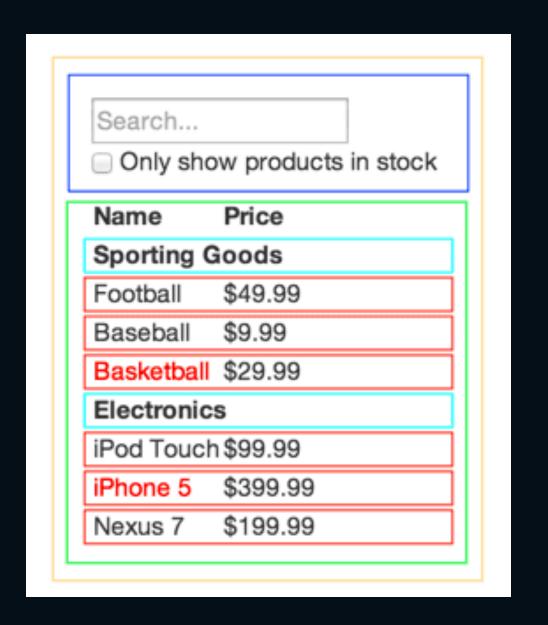
- Alle Werte von Objekten mit shape () definieren, die verwendet werden.
- Nur Werte von Objekten definieren, die auch verwendet werden.

¹ Gleiches gilt für Arrays.

Thinking in React

Vor dem Coden analysieren, wie die Komponenten aufgebaut sind. Dabei direkt Namen definieren.





^{*} Siehe: https://reactjs.org/docs/thinking-in-react.html

tl;dr

Neue Komponenten erstellen sollte schnell von der Hand gehen.

Functional Components und Hooks nutzen.

PropTypes nutzen.

Komponenten nicht zu groß werden lassen. Ständig refactoren und Patterns für Composition anwenden.