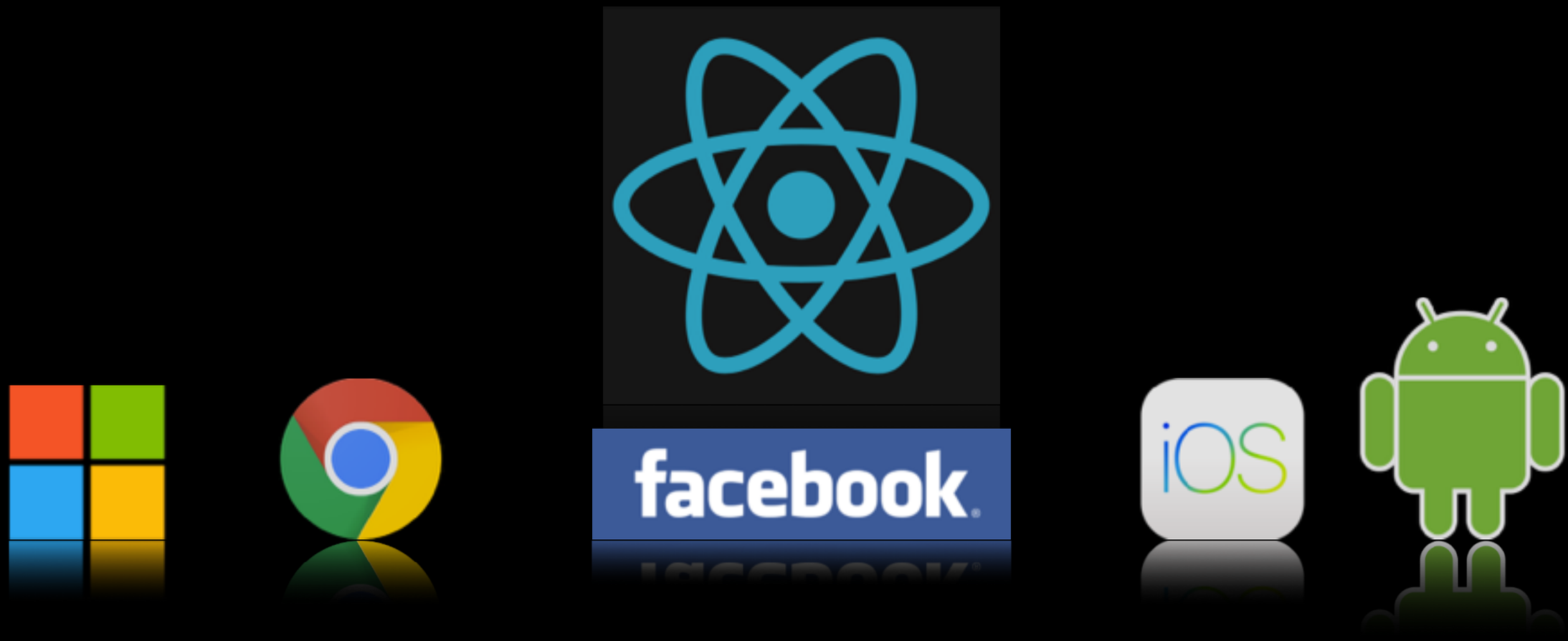


REACT NATIVE UND REDUX IM PRAXISTEST

NATIVE APP ENTWICKLUNG AUF PILZEN

MOTIVATION

- ~~Weltherrschaft~~
- ~~Profit~~
- Spaß, Interesse, Neugierig
- React + Redux ist anders, sagt man
- Challenge yourself! Lernen fetzt!

REACT NATIVE

NATIVE APPS CROSSPLATFORM ENTWICKELN: PUR, FUNKTIONAL, REAKTIV, HIPP, FETZIG

- vorgestellt auf der F8 in 2015
- Native Apps Crossplattform mit bekannten Web-Mitteln
- JS Engine getrieben (JavaScriptCore, V8 on Debugging via Chrome)
- Stylesheets, Flexbox, CSS3-Zeugs, SVG
- React Components auf native UI-Elemente gemappt
- NPM Packages (einige, „react-native-*“)
- alles ist JSX, it just works, Babel transpillt alles von ES2015 auf ES5

Quelle: [RN Doku JavaScript Environment]

SYNTAX TRANSFORMER: BABEL

ES6

- Arrow functions: `<C onPress={() => this.setState({pressed: true})}`
- Block scoping: `let greeting = 'hi';`
- Call spread: `Math.max(...array);`
- Classes: `class C extends React.Component { render() { return <View />; } }`
- Constants: `const answer = 42;`
- Destructuring: `var {isActive, style} = this.props;`
- for...of: `for (var num of [1, 2, 3]) {}`
- Modules: `import React, { Component } from 'react';`
- Computed Properties: `var key = 'abc'; var obj = {[key]: 10};`
- Object Consise Method: `var obj = { method() { return 10; } };`
- Object Short Notation: `var name = 'vjeux'; var obj = { name };`
- Rest Params: `function(type, ...args) { }`
- Template Literals: `var who = 'world'; var str = \`Hello ${who}\`;`

ES7

- Object Spread: `var extended = { ...obj, a: 10 };`
- Function Trailing Comma: `function f(a, b, c,) { }`
- Async Functions: `async function doStuffAsync() { const foo = await doOtherStuffAsync(); }`;

Specific

- JSX: `<View style={{color: 'red'}} />`
- Flow: `function foo(x: ?number): string {}`

REACT

- nur das V in MVC
- veröffentlicht in 2013 (!)

```
render () {
  return (
    <Text style={{color:'red'}}
          placeholder='Name?'
          value={this.state.name} />
  )
}
```

- Template in Code statt Code in Template
- props = Initial-Config und Einstellungen
- state = aktueller Zustand der Komponente
- neue Komponenten sind so einfach wie Prozesse in Elixir
- JSX = JS + inline XML

REDUX

- das M in MVC
- veröffentlicht in 2015
- ein vorhersagbarer State-Container
- Build on the shoulders of Giants:
  Flux, Elm, Immutable, Baobab (?), Rx
- Unidirektionales Databinding

REDUX

- Datenfluss von oben nach unten durch die Komponenten
- Komponenten brauchen keinen State mehr
- …
- Anderungen als Action per Dispatch an Store gemeldet, dort via Reducer die Daten „geändert“, landen als geänderter Props wieder in Komponente
- Actions und Reducer sind Pure Funktionen

```
export default function searchReducer (term = "", action = {}) {
  switch (action.type) {
    case actions.DO_SEARCH:
      return action.term;
    case actions.CLEAR_SEARCH:
      return null;
    default:
      return term;
  }
}
```

```
export function doSearch(term) {
  return {
    type: actions.DO_SEARCH,
    term: term
  };
}
```

IM WEB: WARUM REAKTIV? WARUM IMMUTABLE?

- *Applications, especially on the web have changed over the years from being a simple static page, to DHTML with animations, to the Ajax revolution.*

- *Each time, we're adding more complexity, more data, and asynchronous behavior to our applications. How do we manage it all? How do we scale it?*

- *By moving towards "Reactive Architectures" which are event-driven, resilient and responsive.*

Quelle: GitHub RxJS README

LET'S TRY A NEW FRAMEWORK

Quelle: devopsreactions

DAS PROJEKT

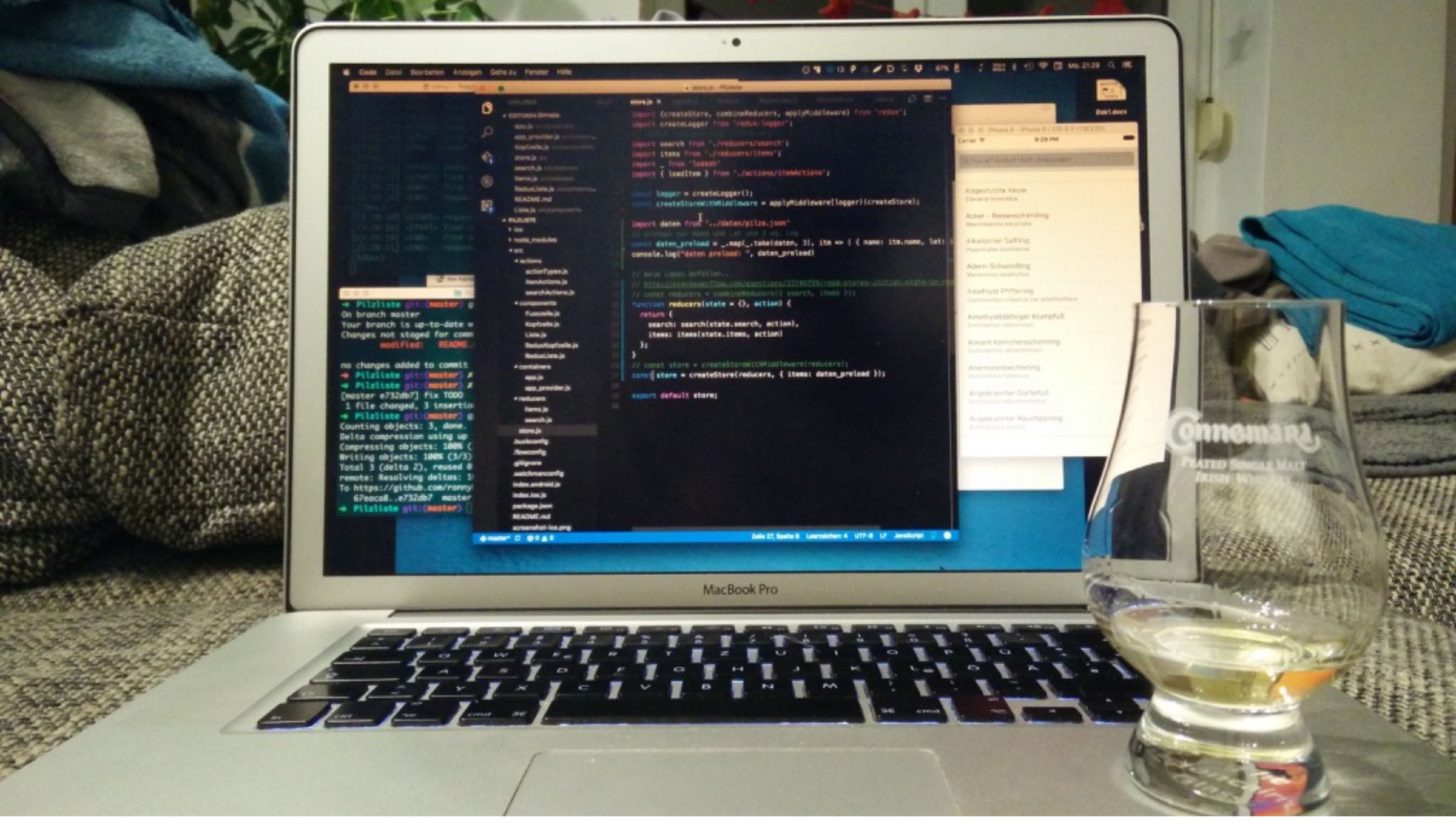
PILZLISTE

GEGEBEN

- Daten von ~1000 Pilzen inkl. Bilder (~180 MB) von Papa
- viel Interesse, etwas Zeit

GESUCHT

- Native App mit einer Listend
- Bilder nachladen + cachen
- Einträge sternen, Galeriedars



PAPA TIME! WENN DER JUNG SCHLÄFT..

CHALLENGES

- Listendarstellung, „viele“ Daten
- Details nach Tap auf Eintrag
- Bilder + Liste Lazyload wg. Performance
- Schnelle Suche
- Redux zur Datenhaltung
- React State vs. Props
- Ladebalken für großes Bild in Details
- Bilder in Liste offline verfügbar
- ES2015/ES6

IRRELEVANT

- CRUD der Daten
- Forms
- Readonly…

DEMO-TIME!

- iOS-Simulator, Android Emulator
- Debugger in App
- Chrome Dev Tools + console.log(..)
- VS Code Integration bei Stack-Traces
- React Komponenten

VIEW

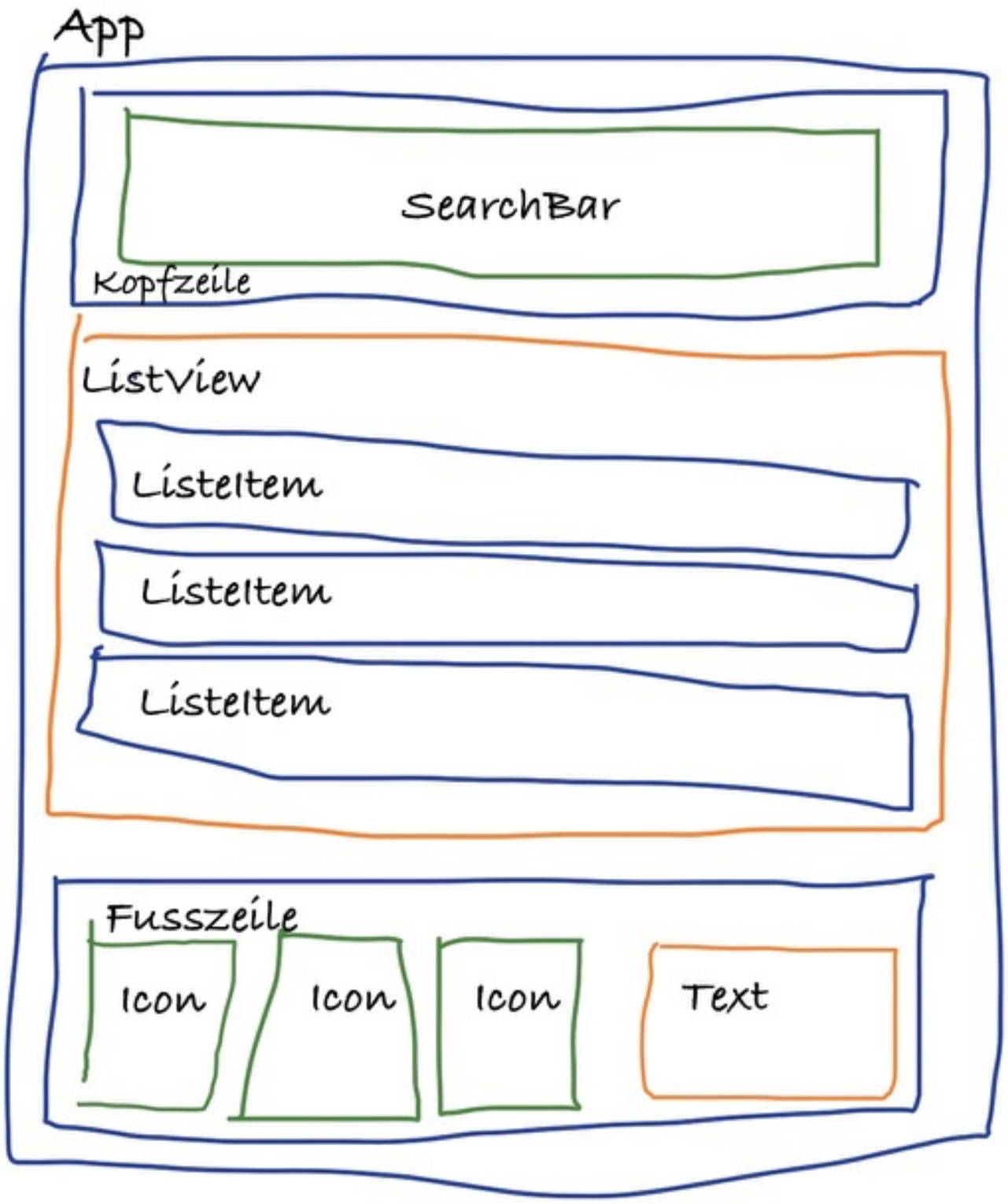
REACT

KOMPONENTEN BAUM

```
<View style={styles.container}>
    <ReduxKopfzeile />
    <ReduxListe />
    <ReduxFusszeile />
</View>
```

```
<View style={styles.kopfzeile}>
    <SearchBar
      lightTheme
      onChangeText={(term) => this.onChange(ter
      placeholder='Name? Farbe? Hut? Unterseite
      inputStyle={styles.inputText}
      containerStyle={styles.inputCont}
      value={this.state.searchterm} />
</View>
```

```
<View style={styles.container}>
  <View style={styles.iconRow}>
    <Icon
      iconStyle={styleIcon}
      name='view-list'
      color={selectedTab === 'liste' ? colActi
      onPress={() => this.changeTab('liste')} /
    <Icon
      iconStyle={styleIcon}
      name='view-module'
```



Quelle: RN Doku: Component Specs and Lifecycle

LEBENSZYKLUS EINER KOMPONENTE

- Mounting: componentWillMount
- Mounting: componentDidMount
- Updating: componentWillReceiveProps
- Updating: shouldComponentUpdate
- Updating: componentWillUpdate
- Updating: componentDidUpdate
- Unmounting: componentWillUnmount

STYLESHEETS IN JS

- quasi CSS3, aber inline per JS-Objekt; mächtig wie SASS
- sehr viele bekannte CSS Sachen möglich: padding, margin, boxShadow, border, color, transform etc.
- bei Hot-Replace sogar ohne Neuladen - wie bei Meteor

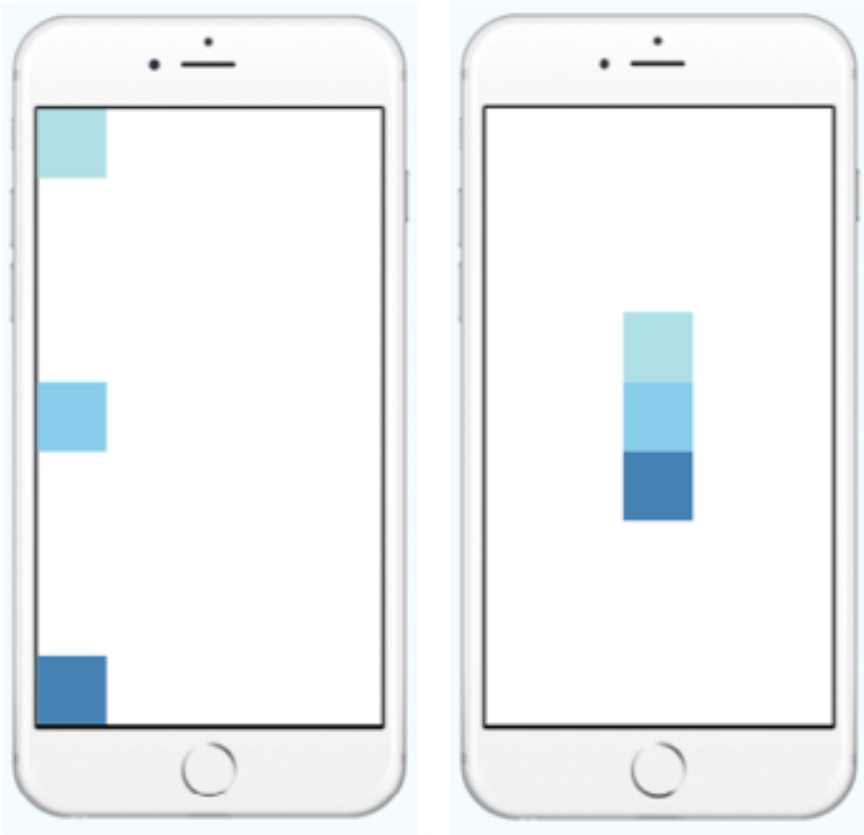
```
const styles = StyleSheet.create({
  kopfzeile: {
    height: 55,
    marginTop: Platform.OS === 'ios' ? 25 : 0,
    backgroundColor: 'white'
  },
  inputText: {
    height: 35,
    color: 'black',
    backgroundColor: 'white'
  },
  inputCont: {
    height: 55,
    backgroundColor: 'oldlace'
  }
});
```

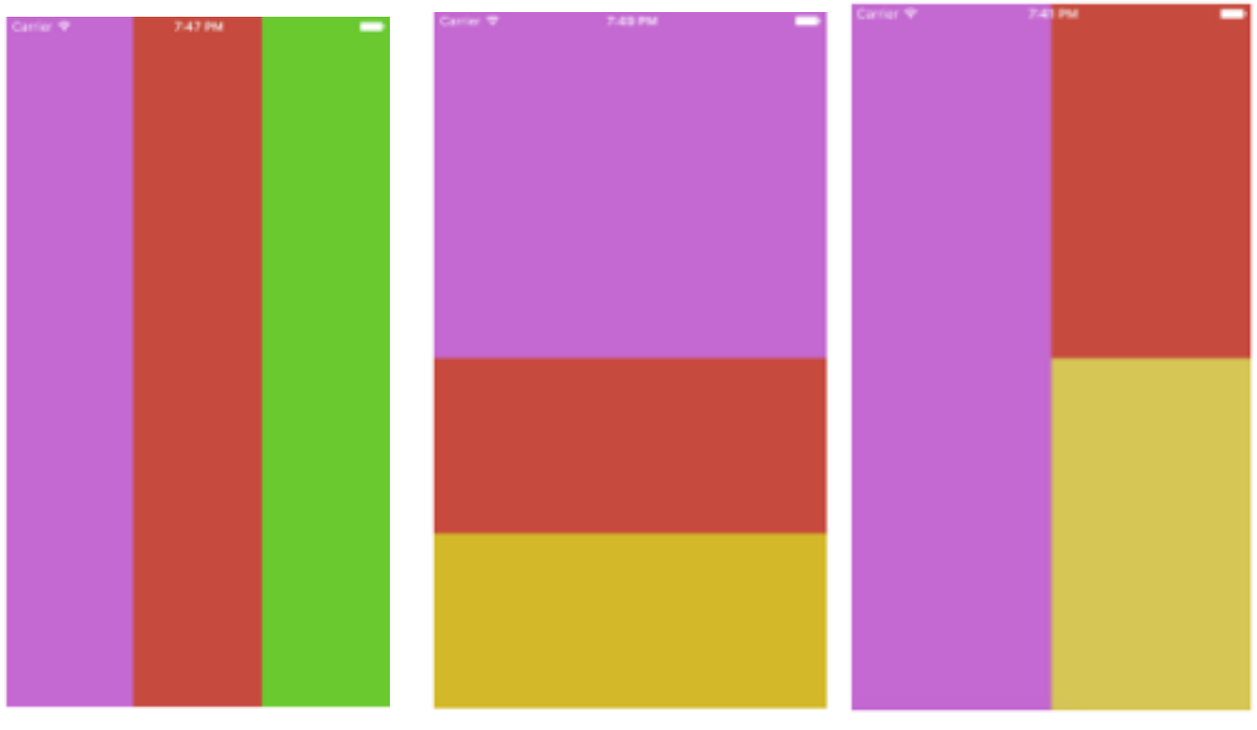
JSS | RADIUM

APHRODITE

FLEXBOX

- offizieller Guide zu RN
- Guide auf css-tricks.com
- einfacher: GeekyAnts/ react-native-easy-grid

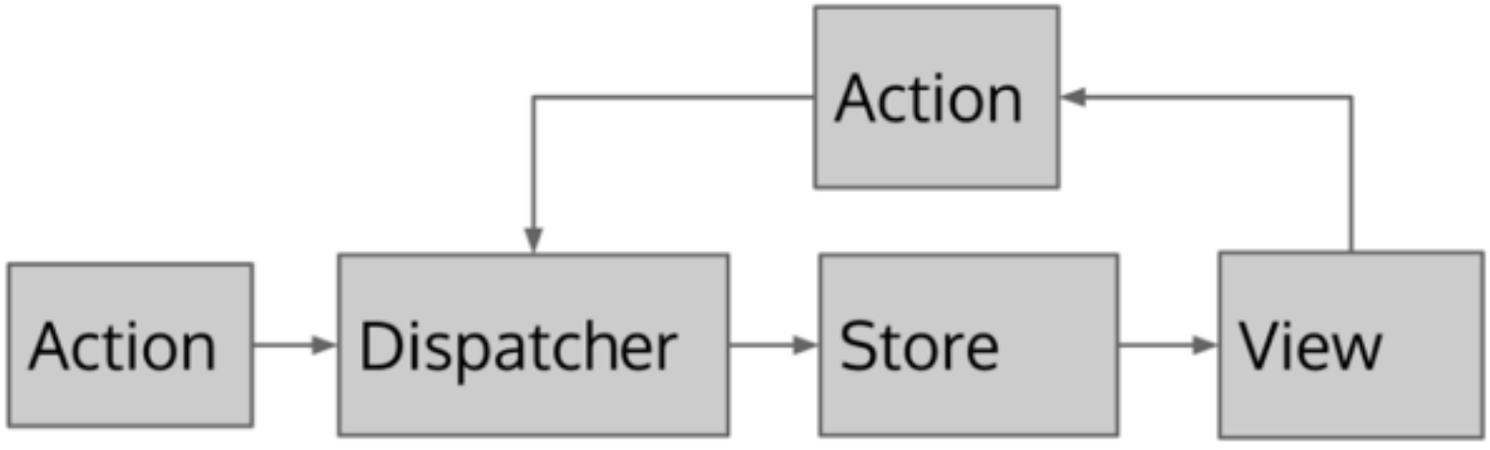




STATE

REDUX

REDUX ABLAUF



http://facebook.github.io/flux/docs/overview.html

- Action trägt die Payload, wird dispatcht,
  über Reducer die Daten im Store geändert (immutable!),
  dann die betreffenden Komponenten neu gerendert

- Komponente: onChange -> Redux Connect-Nubsi: dispatch(doSearch(term))

- ```
  doSearch(term) {
   return {
    type: actions.DO_SEARCH,
     term: term
   };
  }
  ```
  ```

CONNECT

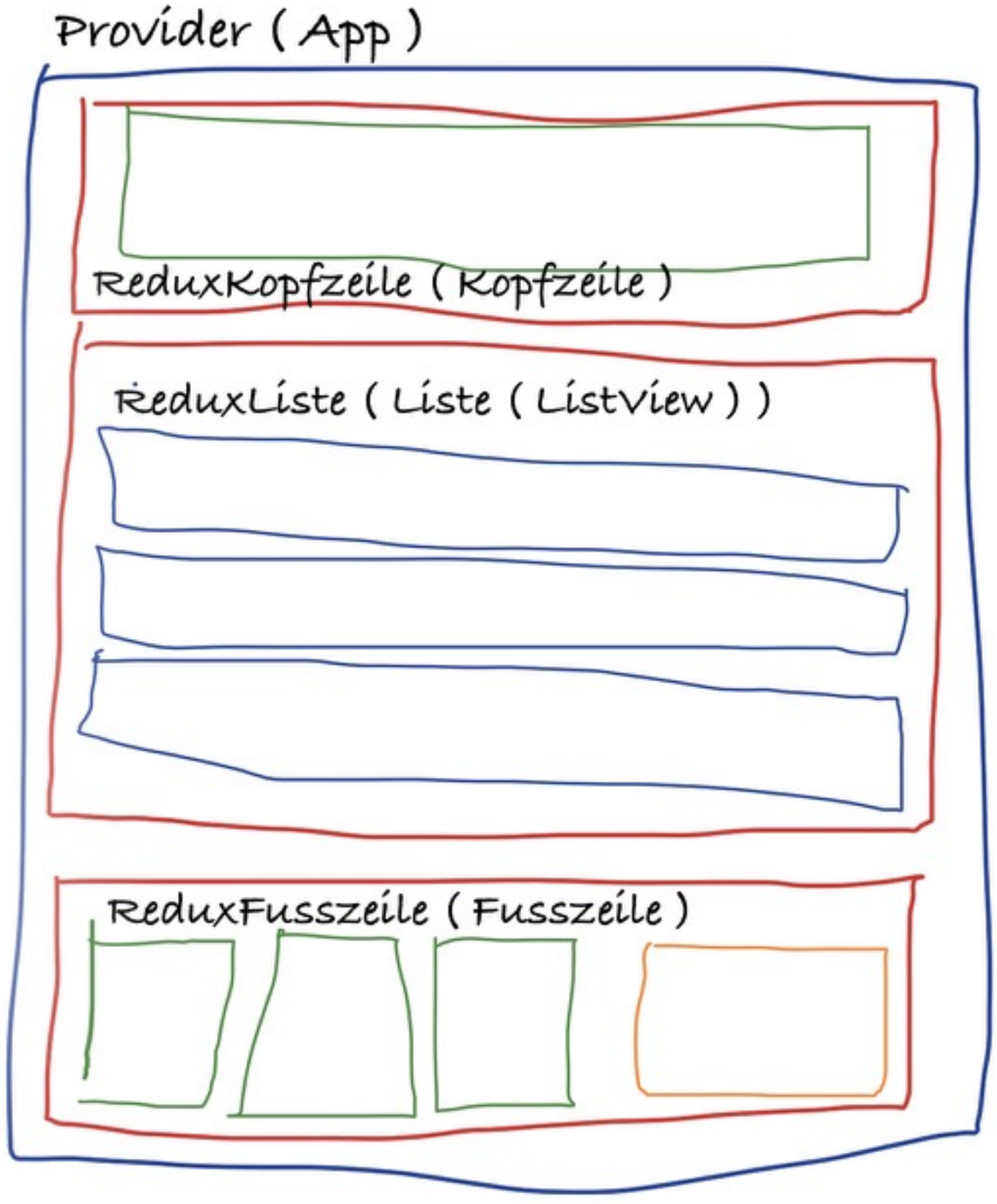
```
<Provider store={store}>
  <App />
</Provider>
```

```
<View style={styles.container}>
```

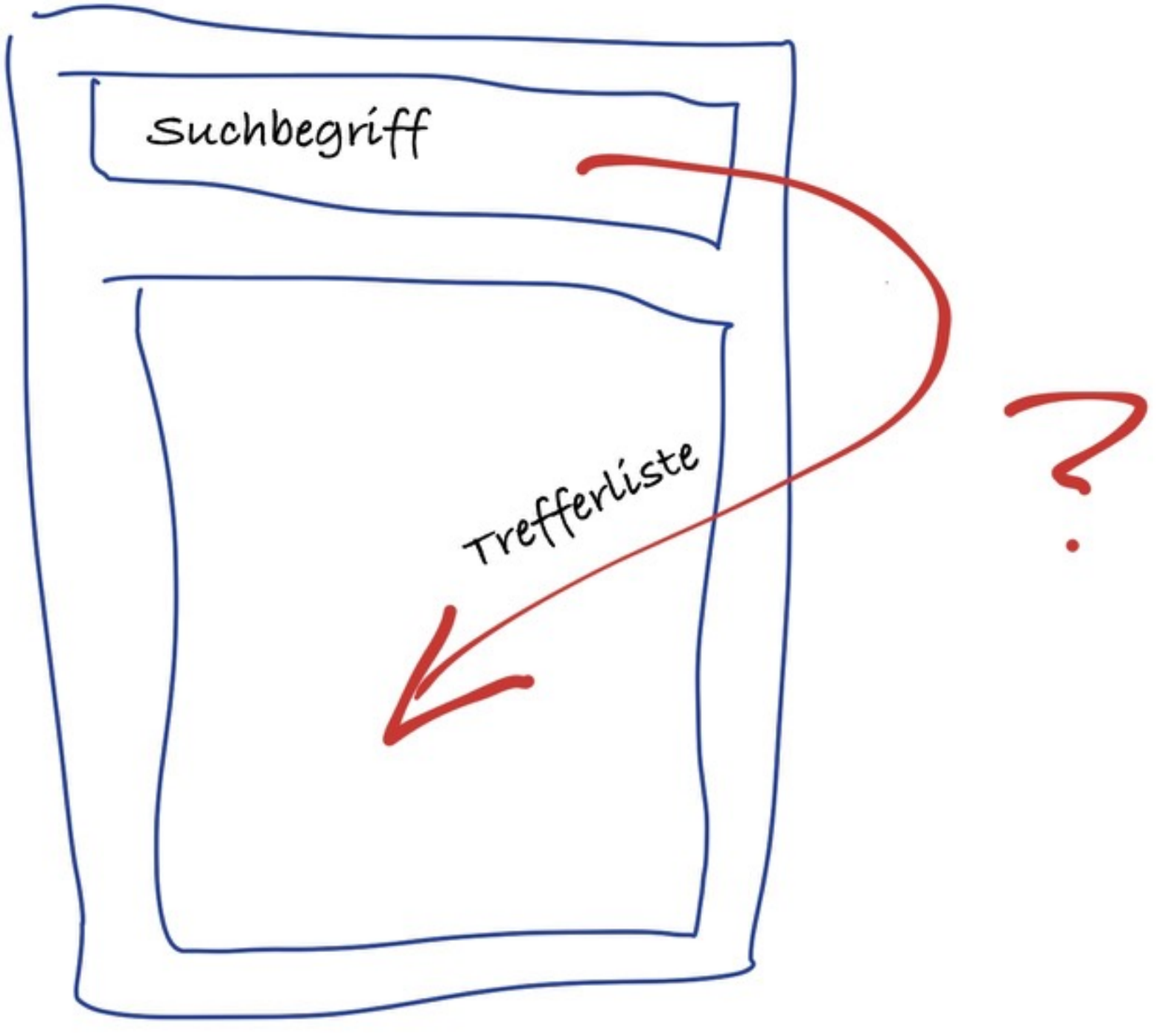
```
export default class Kopfzeile extends Component {
  constructor(props) {
    super(props);
    this.state = {
      activeSearch: this.props.activeSearch
    }
  }
  // ..
  render() {
    return (
      <View style={styles.kopfzeile}>
          <SearchBar
            lightTheme
            onChangeText={(term) => this.onChange(term)}
            placeholder='Name? Farbe? Hut? Unterseite? (m
            inputStyle={styles.inputText}
            containerStyle={styles.inputCont}
            value={this.state.searchterm} />
      </View>
    )
  }
}
```

```
  mapDispatchToProps
)(Kopfzeile)

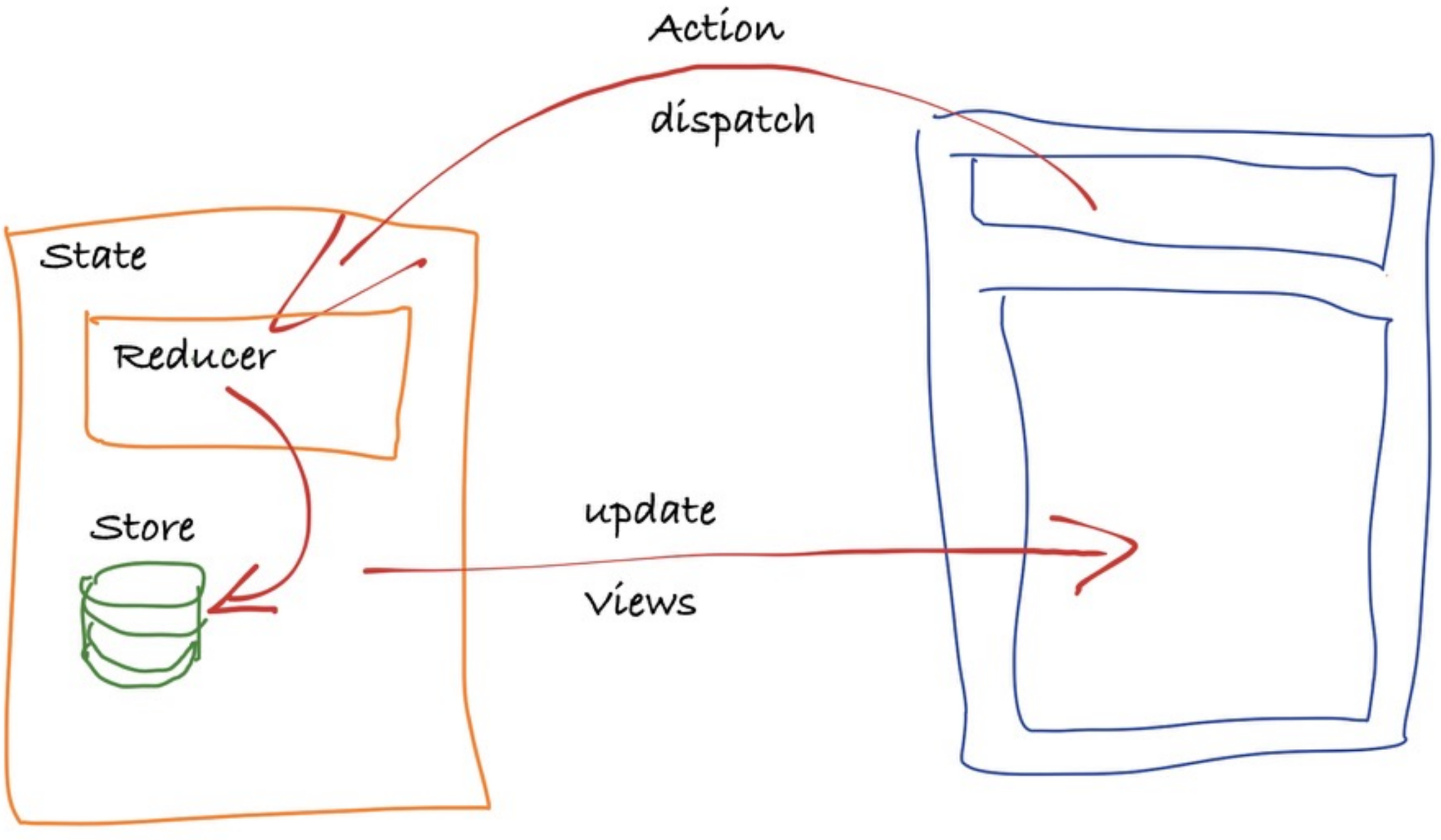
export default ReduxKopfzeile
```



SUCHE?



SUCHE!



SUCHBEGRIFF IN STATE, TREFFER IN LISTE

```
onChange(term) {
  this.setState({ activeSearch: term }
  this.doSearch(term)
}
doSearch(term) {
  this.props.doSearch(term);
}
render() {
  return (
    <View style={styles.kopfzeile}>
        <SearchBar
          lightTheme
          onChangeText={(term) => this
          placeholder='Name? Farbe? Hu
```

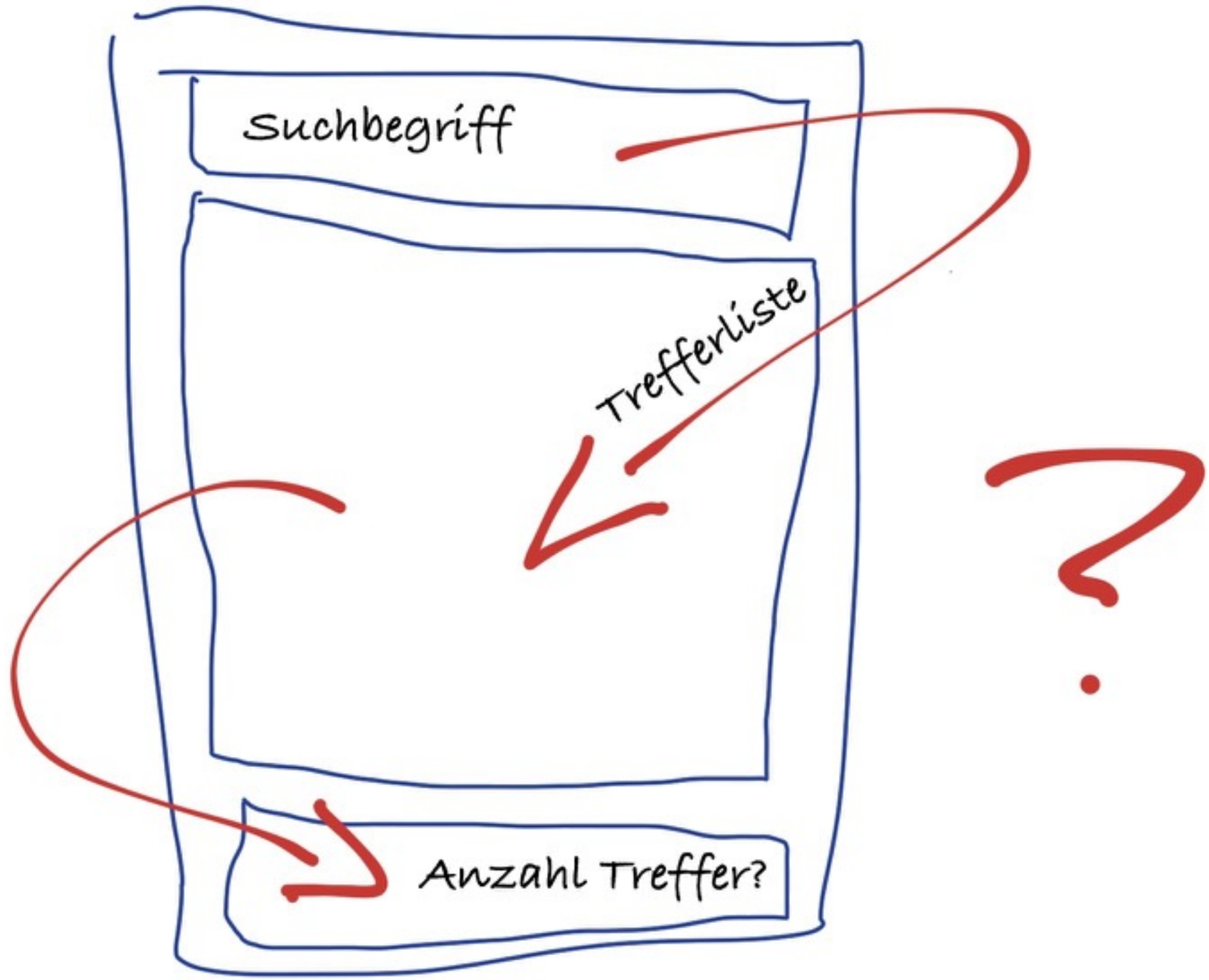
```
export function doSearch(term) {
  return {
    type: actions.DO_SEARCH,
    term: term
  };
}

export default function searchReducer (
  switch (action.type) {
    case actions.DO_SEARCH:
      return action.term;
    case actions.CLEAR_SEARCH:
```

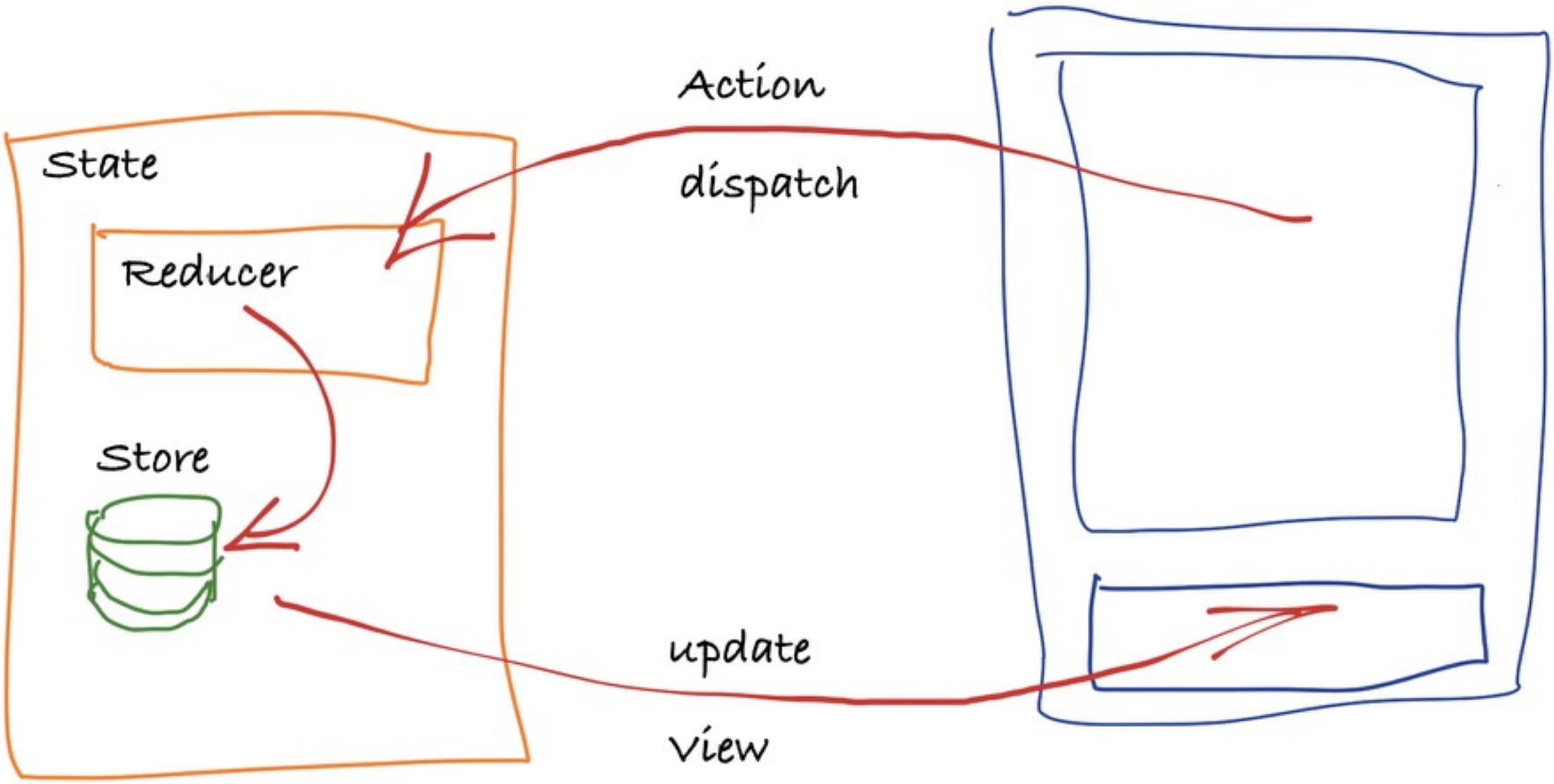
```
const getVisibleItems = createSelector(
  [ getSearch, getItems ],
  (search, items) => {
    if (_.isEmpty(search) || search.length < 3) {
      return items
    } else {
      // Suchbegriffe mit Leerzeichen getrennt..
      const terms = search.trim().split(' ')
      const startTime = new Date()
      const result = _.filter(items, itm => {
          const haystack = _.values(itm).join(' ')
          return terms.map(term => ( haystack.indexOf(term) !== -1 ))
                      .reduce((prev, curr) => ( prev && curr ), true)
      })
      console.log(`Suche: ${result.length} Treffer in ${new Date() - st
      return result
    }
  }
)

const mapStateToProps = (state) => {
  return {
    filteredItems: getVisibleItems(state),
    activeTab: state.tab
  }
}
```

TREFFERANZAHL?



TREFFERANZAHL!



LISTE MELDET AN STATE, FUSSZEILE ABONNIERT

```
export function updateNumberItems(number) {
  return {
    type: actions.NUMBER_ITEMS,
    number
  };
}

const mapDispatchToProps = (dispatc
  return {
    updateNumberItems: (number) => {
      dispatch(updateNumberItems(number))
    },
    setStar: (item) => {
```

```
export default function numberItemsReducer (number = 0, action = {}) {
  switch (action.type) {
    case actions.NUMBER_ITEMS:
      return action.number;
    default:
```

```
const mapStateToProps = (state) => {
  return {
    numberItems: state.numberItems
  }
```

```
componentWillMount () {
  this.updateNumberItems()
}
componentDidUpdate () {
  this.updateNumberItems()
}
updateNumberItems() {
  this.props.updateNumberItems(this.state.items.length)
}
```

```
render() {
  const { numberItems, selectedTab } = this.state
  //..
  return (
        //..
        <Text style={styles.treffer}>{numberItems} Pilze</Text>
        //..
  )
```

JUST ONE MORE THING — UNDO!

- mit klassischen MVC eine Herausforderung
- mit Redux ein Kinderspiel -> redux-undo
- Daten sind **nicht veränderlich (Immutable)**
- keine multiplen Modelle,
- es genügt ein Zustand und die nachfolgend angewandten Actions
- http://redux.js.org/docs/recipes/ImplementingUndoHistory.html#using-redux-undo


THAT'S ALL?
YOU JUST KIDDING ME
imgflip.com

DER SPASS BEGINNT

CHALLENGES

REDUX SUCHE

```
const terms = search.split(' ')
const startTime = new Date()
const result = _.filter(items, itm => {
    const haystack = _.values(itm).join(' ')
    return terms.map(term => ( haystack.indexOf(term) !== -1 )) // Treffer je Suchbegriff?
                .reduce((prev, curr) => ( prev && curr ), true) // alle Suchbegriffe true?
})
console.log(`Suche: ${result.length} Treffer in ${new Date() - startTime}ms`)
return result
```

- Fuzzy: https://github.com/mattyork/fuzzy
  - versucht, ist aber zu langsam
- Redux-search: https://github.com/treasure-data/redux-search
  - spannend, wenn CRUD

REDUX SUCHE

- Debounced Sucheingabe

```
constructor(props) {
  super(props);
  this.state = {
    activeSearch: this.props.activeSearch
  }
}
componentWillMount() {
  this.doSearch = _.debounce(this.doSearch, 300)
}
onChange(term) {
  this.setState({ activeSearch: term })
  this.doSearch(term)
}
doSearch(term) {
  this.props.doSearch(term);
}
```

- Memoized Selectors

```
const getSearch = (state) => state.search
const getItems = (state) => state.items
// der Memoized Selectors
const getVisibleItems = createSelector(
  [ getSearch, getItems ],
  (search, items) => {
    if (_.isEmpty(search) || search.length < 3) {
      return items
    } else {
      // .. Suche ..
    }
  }
)
const mapStateToProps = (state) => {
  return {
    filteredItems: getVisibleItems(state)
  }
}
```

REDUX UPDATE REACT

- kein Neubau der Komponente bei Änderung sondern per Props-Änderung in React Komponente

```
constructor(props) {
  super(props)
  this.state = {
    items: this.props.filteredItems
  }
}
componentWillReceiveProps (nextProps) {
  if (nextProps.filteredItems !== this.props.filteredItems) {
    this.setState({
      items: nextProps.filteredItems
    })
  }
}
```

REDUX PREFILL

```
import daten from '../daten/pilze.json'
// erstmal nur Name und Lat und 3 wg. Log
const daten_preload = daten
// const daten_preload = _.take(daten, 50)
// const daten_preload = _.filter(daten, itm => /rasling/.test(itm.name))
// console.log("daten preload: ", daten_preload)
```

```
// beim Laden befüllen..
// http://stackoverflow.com/questions/33749759/read-stores-initial-state-in-redux-reducer#33791942
function reducers(state = {}, action) {
  return {
    search: search(state.search, action),
    items: items(state.items, action)
  };
}
const store = createStore(reducers, { items: daten_preload });
```

REACT LAZYLOAD LISTE

```
render() {
  return (
    <LazyloadScrollView
        style={styles.container}
        contentContainerStyle={styles.content}
        name="lazyload-list"
    >
      { this.state.items.map((item, i) => (
        <ListeItem key={i} item={item} />
      )) }
    </LazyloadScrollView>
  );
}
```

REACT LAZYLOAD ITEM

```
return (
    <View style={viewStyles}>
    <TouchableOpacity onPress={() => this.onPressItem()}>
        <LazyloadView
            host="lazyload-list"
        >
            <View style={styles.item}>
                <Image style={styles.image} source={{uri: image_uri}} />
                <View style={styles.name}>
                    <Text style={styles.nameText}>{item.name}</Text>
                    <Text style={styles.latText}>{item.lat}</Text>
                </View>
            </View>
            <View>
                {this.state.details
                  ? <ListeItemDetails item={item} show={this.state.details} />
                  : <Text/>}
            </View>
        </LazyloadView>
    </TouchableOpacity>
    </View>
)
```

LISTVIEW – ANDERS LAZY

- mächtige Kernkomponente zur Darstellung vertikal scrollender Listen mit ändernden Daten
- automagisches Verhalten, wiederverwendete Zeilen-Views
- nur geänderte Zeilen werden neu gerendert
- nur bestimmte Anzahl Zeilen außerhalb des Sichtfeldes werden vorgeredet
- Sections mit Sticky Headers
- Header+Footer Support für nachladende Listen
- https://facebook.github.io/react-native/docs/listview.html
  https://facebook.github.io/react-native/docs/using-a-listview.html

LISTVIEW VS. SCROLLVIEW

- **ScrollView**: alle Elemente werden vorgerendert - LazyScrollView macht das ondemand, trotzdem werden bei 1000 Einträgen 1000 Views vorgehalten
  - YouTube: ScrollViews vs ListViews (0:51)
- **ListView:** behält nur eine (einstellbar) bestimmte Menge Einträge/Views im Speicher und verwendet aus dem Sichtfeld verschwundene wieder - und: unterstützt Footer für Nachladen, Sections, etc.
  - YouTube: ListView and Recycling (1:40)


SAY "WELL, IT' EASY"
ONE MORE TIME
imgflip.com

KONKURRENZ: WEEX + VUE

- **Vue:** V in MVC, gleiche Kernkonzepte wie React, aber schlanker
  https://vuejs.org/
- **Weex:** Crosscompilierendes JS zu Nativ von Alibaba
  http://alibaba.github.io/weex/
- Alibaba: der kleine Milliarden-Konzern, der Amazon und Ebay in Asien das Leben schwer macht. Dort wo man den Chinakram herbekommt..
- aber React hat Jahre Vorsprung..

INSPIRATION

- Offizielle Doku: https://facebook.github.io/react-native/docs/
- Awesome-Liste: https://github.com/jondot/awesome-react-native
- GitHub, Blogposts, Medium.com
- Reddit: https://www.reddit.com/r/reactnative


ABOVE ALL ELSE
NEVER QUIT, AND NEVER GIVE UP
memegenerator.net

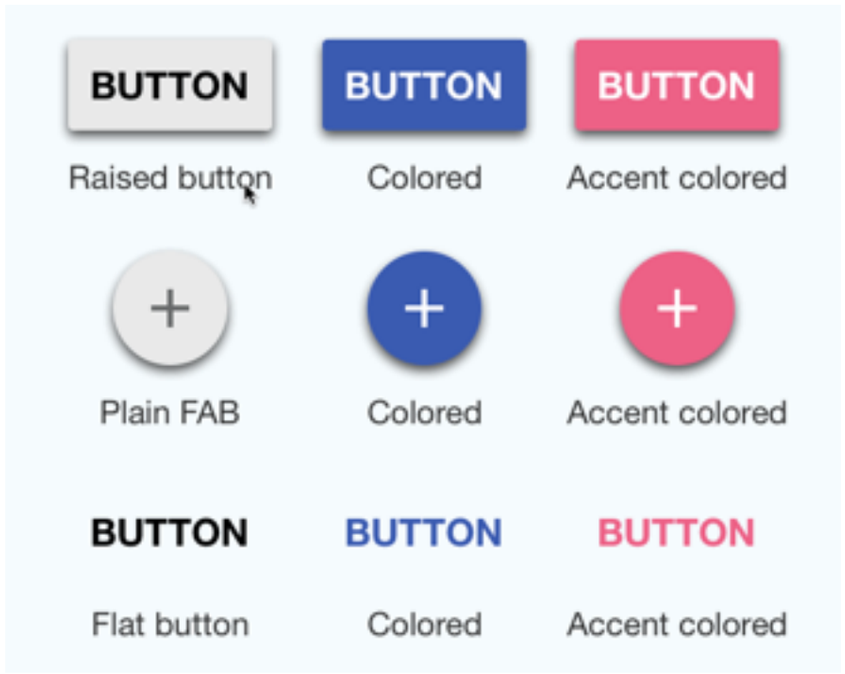
FRAGEN?

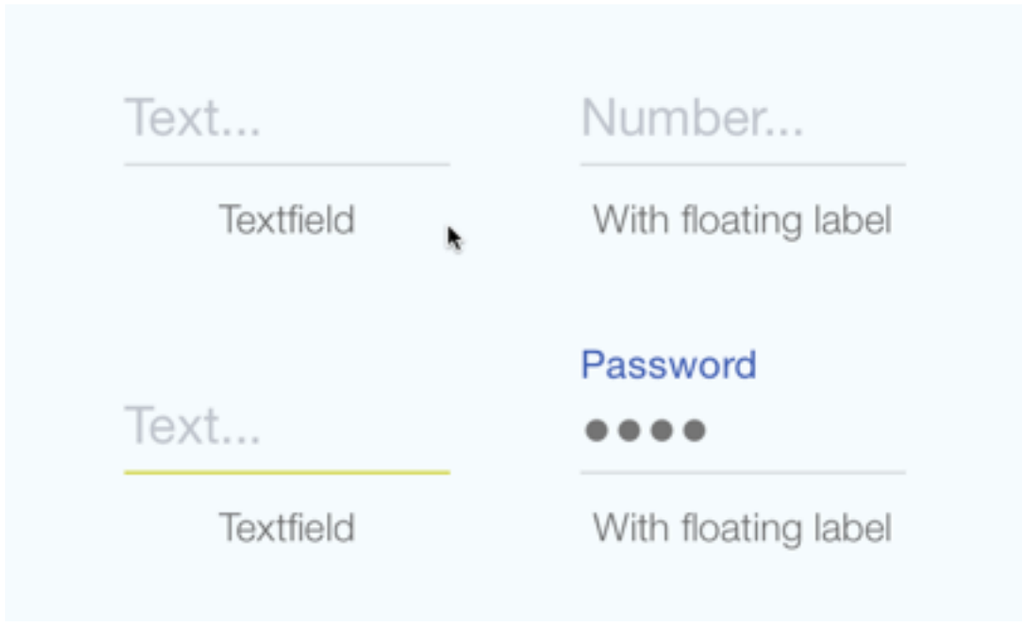
- Ronny Hartenstein
- Twitter: @rhflow_de
- GitHub: ronnyhartenstein/pilzliste-react-native-redux
- Webseite: http://blog.rh-flow.de/pilzliste-react-native-redux

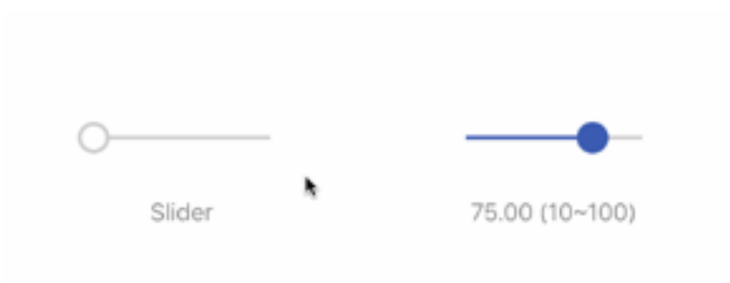
DAS ÖKOSYSTEM

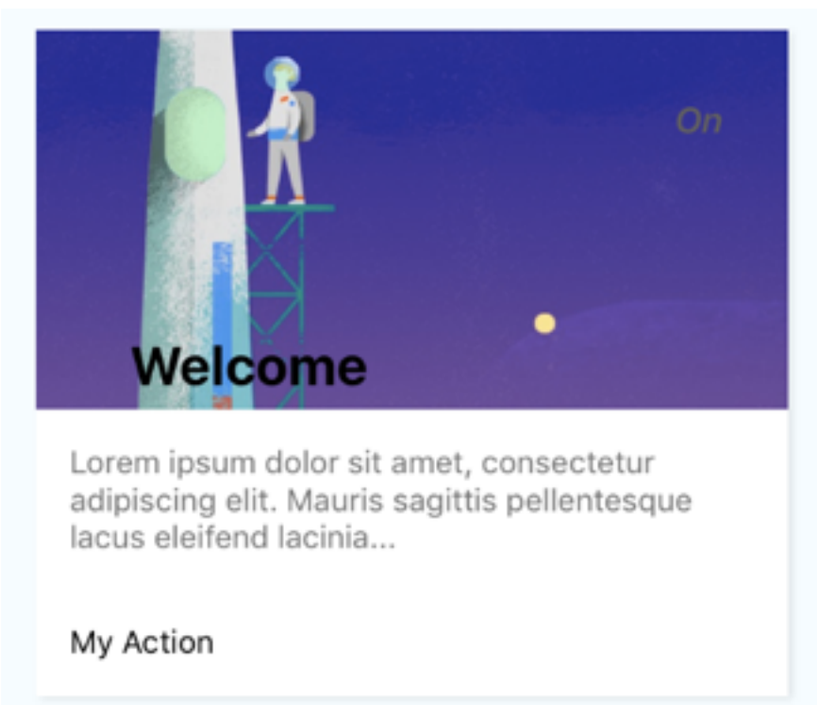
xinthink/react-native-material-kit

MATERIAL DESIGN KIT





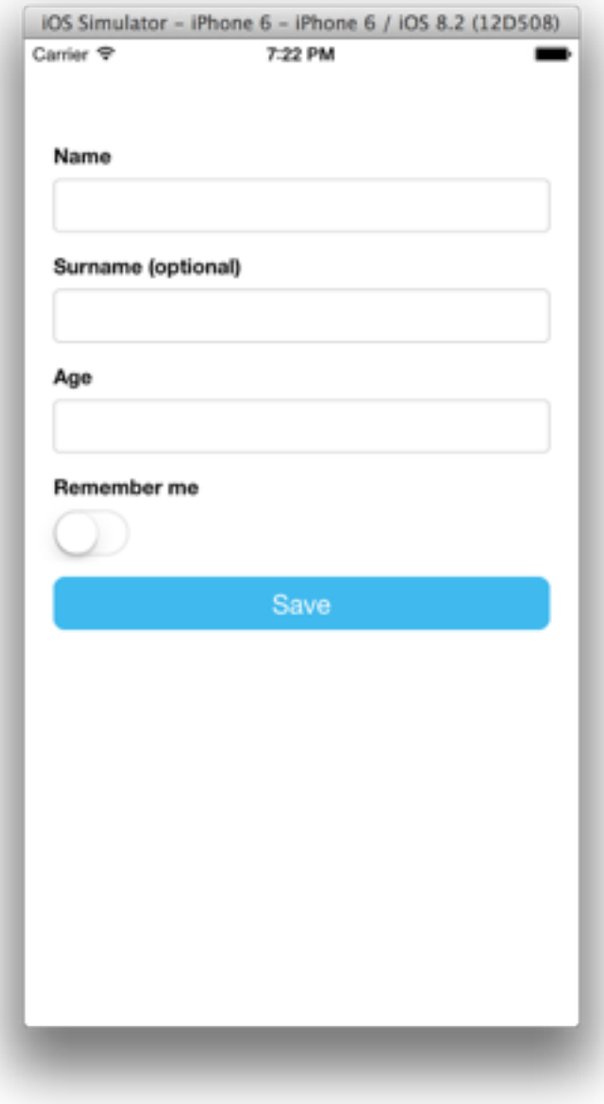




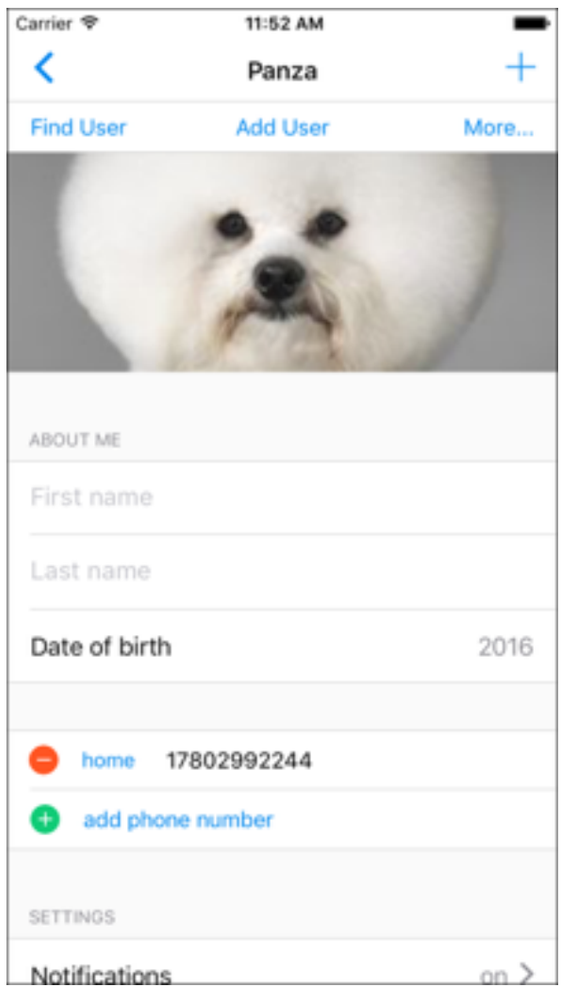
DESIGN PAKETE

- NativeBase ($50)
- Flat App Theme ($100)
- Login Logout Animation ($35)
- Shoutem UI Toolkit (BSD)
- Material Design Kit (MIT)
- **Bootstrap**? So einfach ist es leider nicht

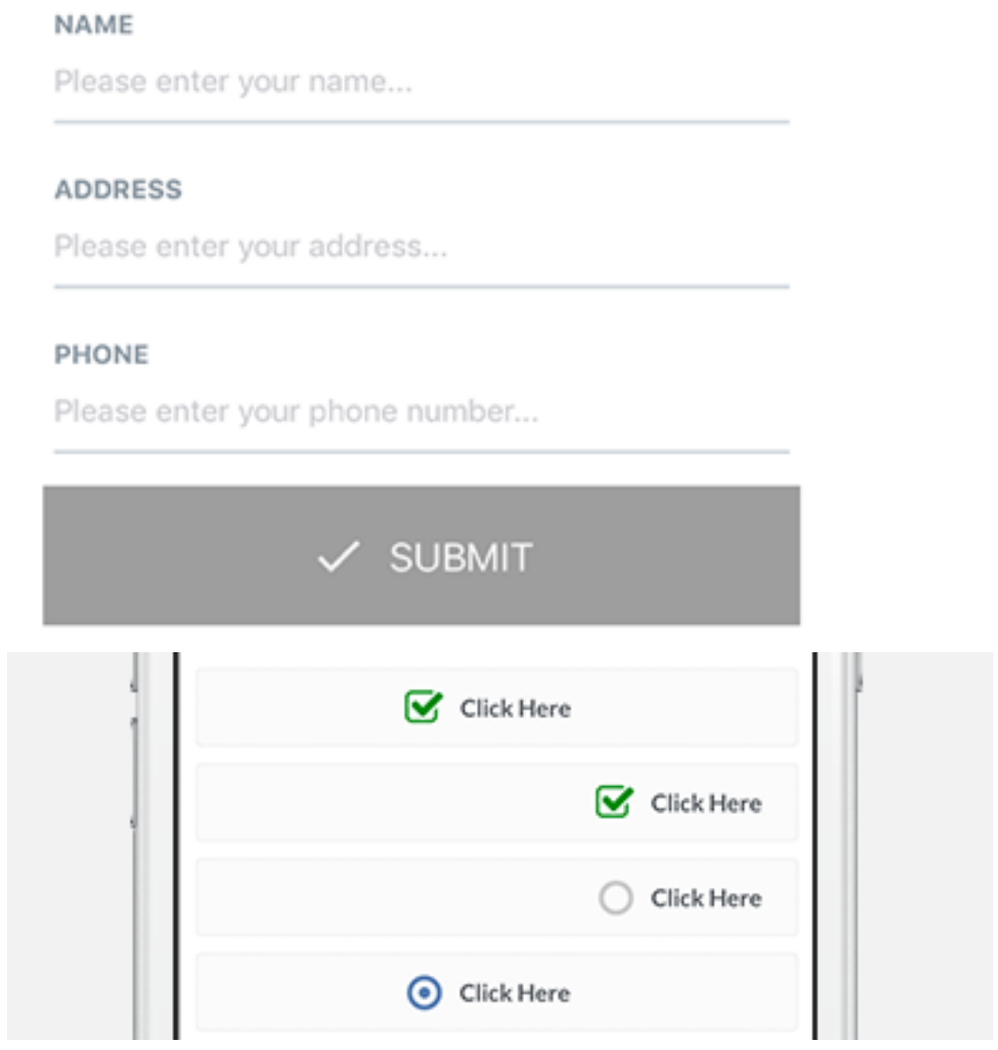
FORMS



[gcanti/tcomb-form-native](#)



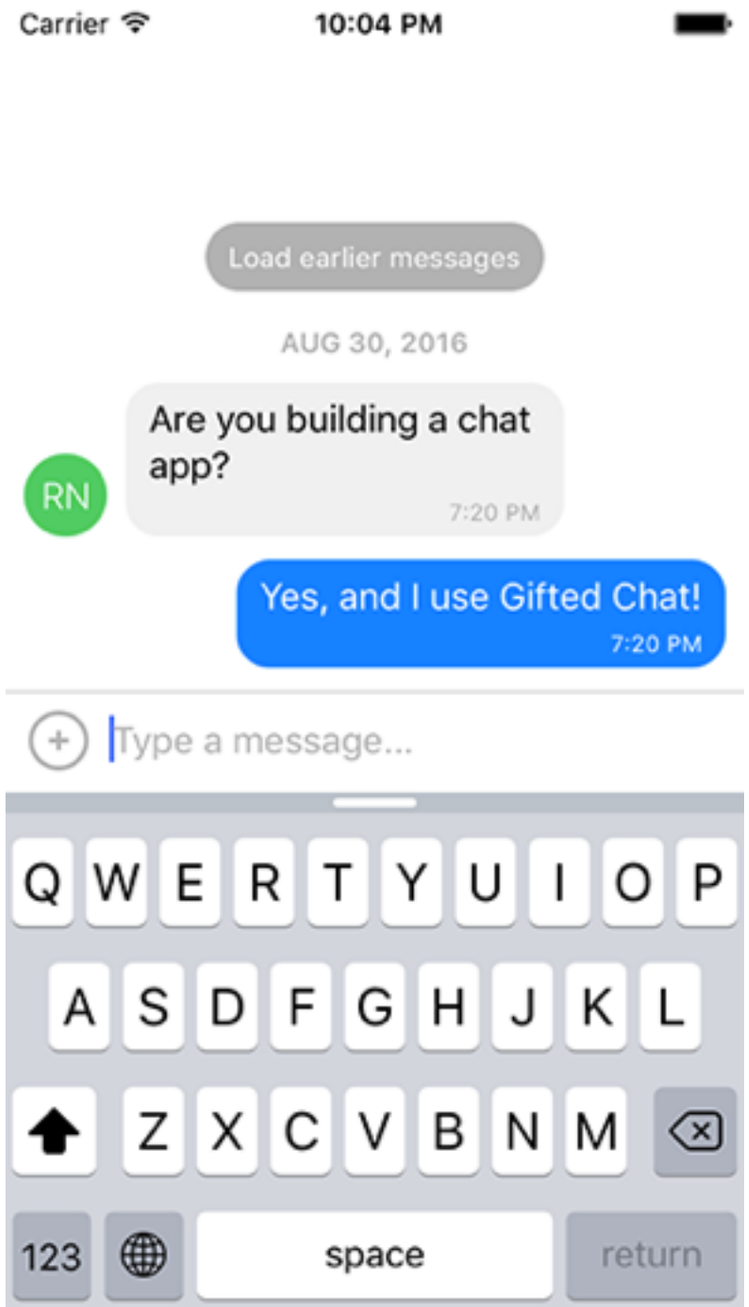
[panza-org/panza](#)

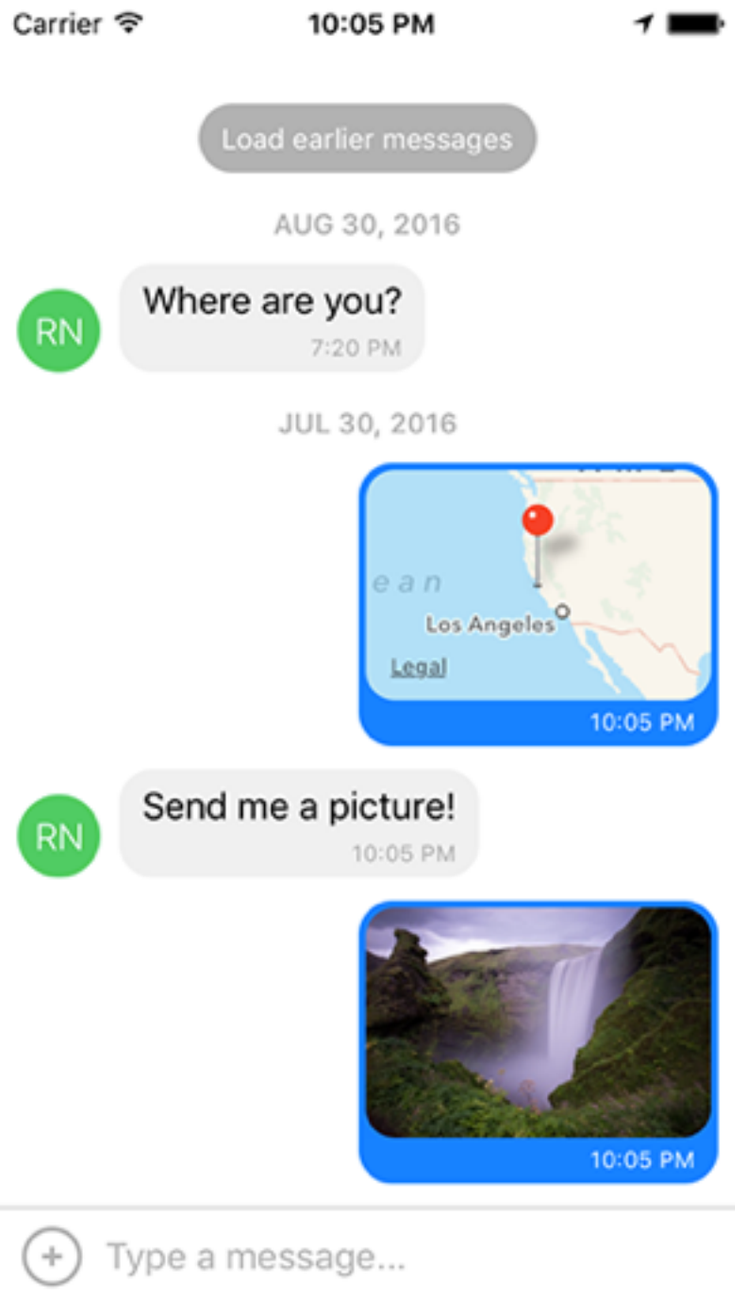


[react-native-community/react-native-elements](#)

FaridSafi/react-native-gifted-chat

CHAT



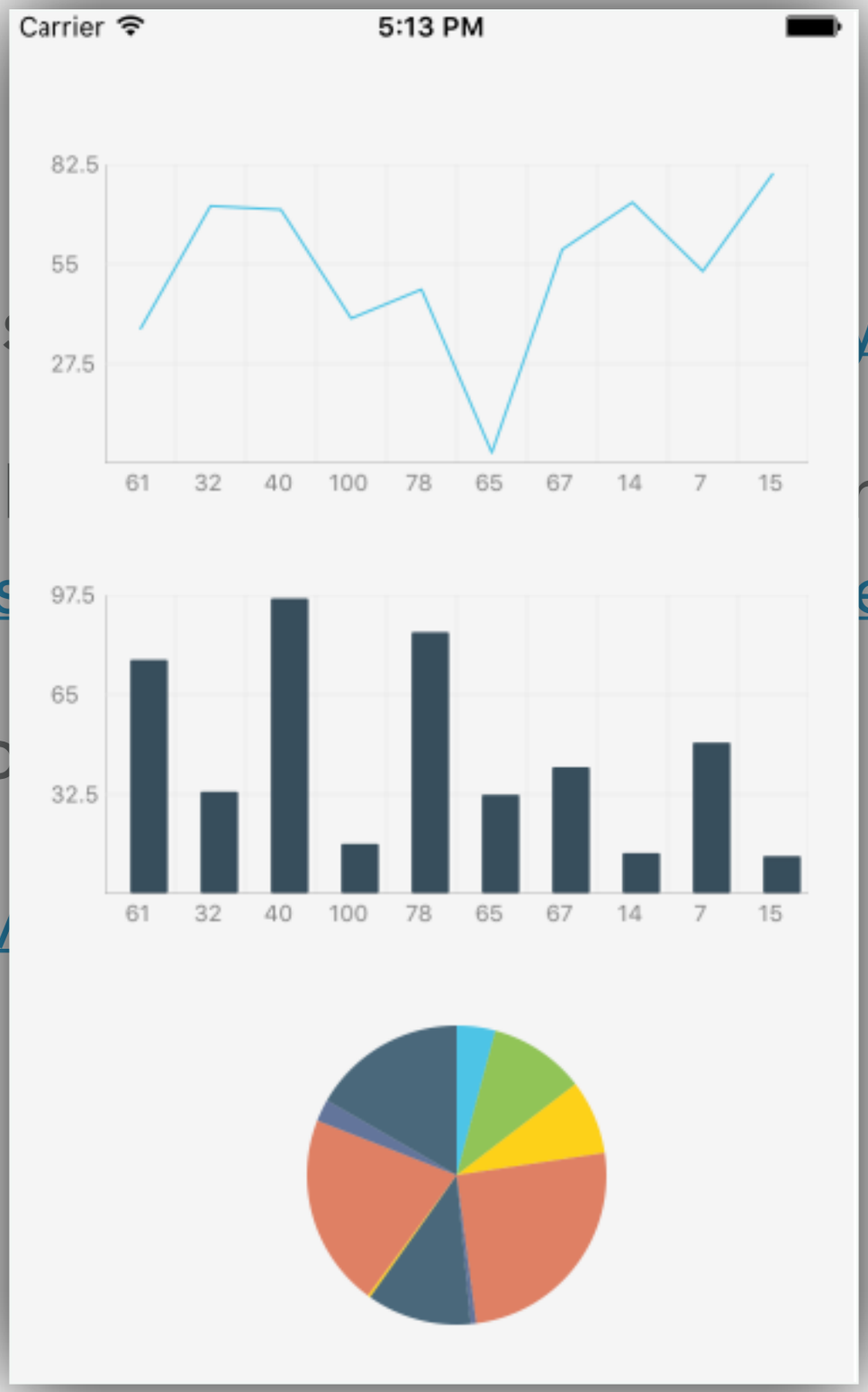


oblador/react-native-lightbox

LIGHTBOX

SVG & GRAPHEN

- Lib react-native-s /react-native-svg
- React ART kann n
  Blogpost: http://hs e-art-and-d3/
- aber schlecht do
- Charts: tomauty/



REACT KOMPONENTEN

- Karten: react-native-maps
- Textblock Handling: react-native-parsed-text
- **Privatsphäre** wenn App in den Hintergrund geht: kayla-tech/react-native-privacy-snapshot
- Splashscreen: /react-native-splashscreen
- On-Screen-Meldungen (Toasts): react-native-toast
- Vector Icons (tausende): react-native-vector-icons

UTILS, INFRASTRUKTUR, SYSTEM

- Verschlüsselung: react-native-aes, react-native-crypto, react-native-des, react-native-rsa
- Workers: react-native-workers
- Mehrsprachigkeit: react-native-i18n
- Barcode-Scanner: react-native-barcodescanner
- Touch-ID: react-native-touch-id + jariz/react-native-fingerprint-android
- Kamera: react-native-camera, react-native-uploader
- Push Notifications: zo0r/react-native-push-notification

IMAGE PREFETCH

- offizieller Weg momentan: `Image.prefetch(uri)`
- ist eigentlich innerhalb der Komponenten gedacht,
  ich möchte es bei Redux-Store-Init machen
- unter Android Fehler "ImagePipelineFactory was not initialized!"

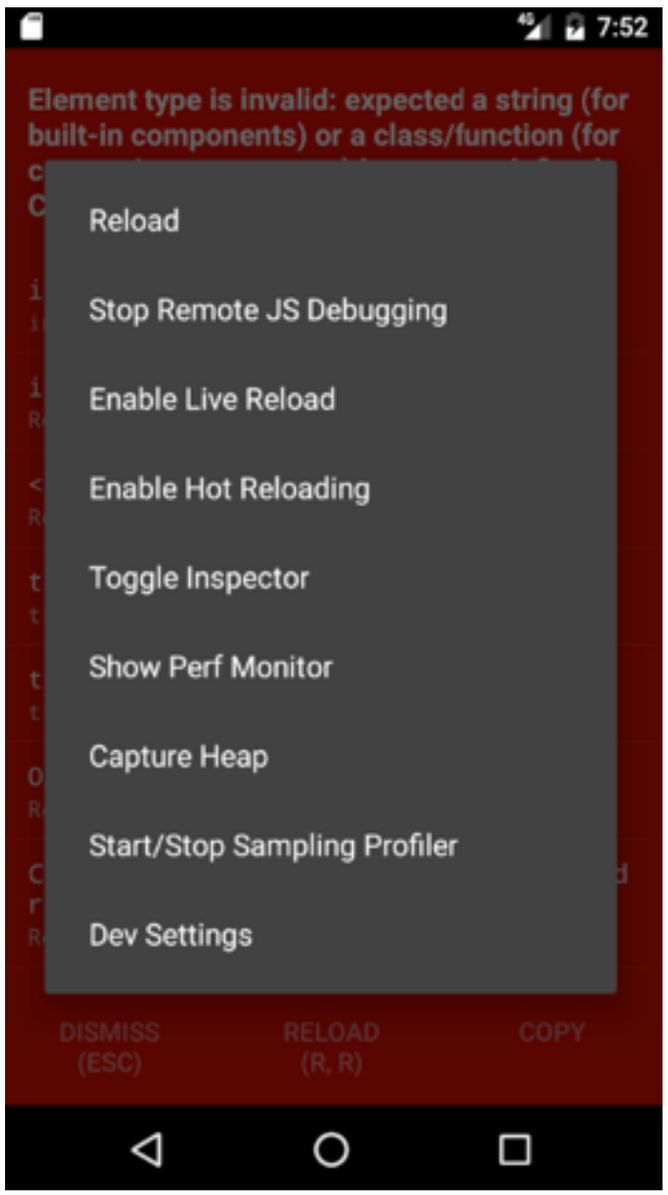
```
const prefetch = function(uris, i) {
    const uri = uris[i]
    let startTime = new Date()
    Image.prefetch(uri).then(() => {
        // console.log(`✔ Prefetch OK (+${new Date() - startTime}ms) from ${uri}`)
    }, error => {
        console.log(`✘ Prefetch failed (+${new Date() - startTime}ms) from ${uri}`)
    }).then(() => {
        if (i+2 <= uris.length) {
            prefetch(uris, i+1)
        }
    })
}
```

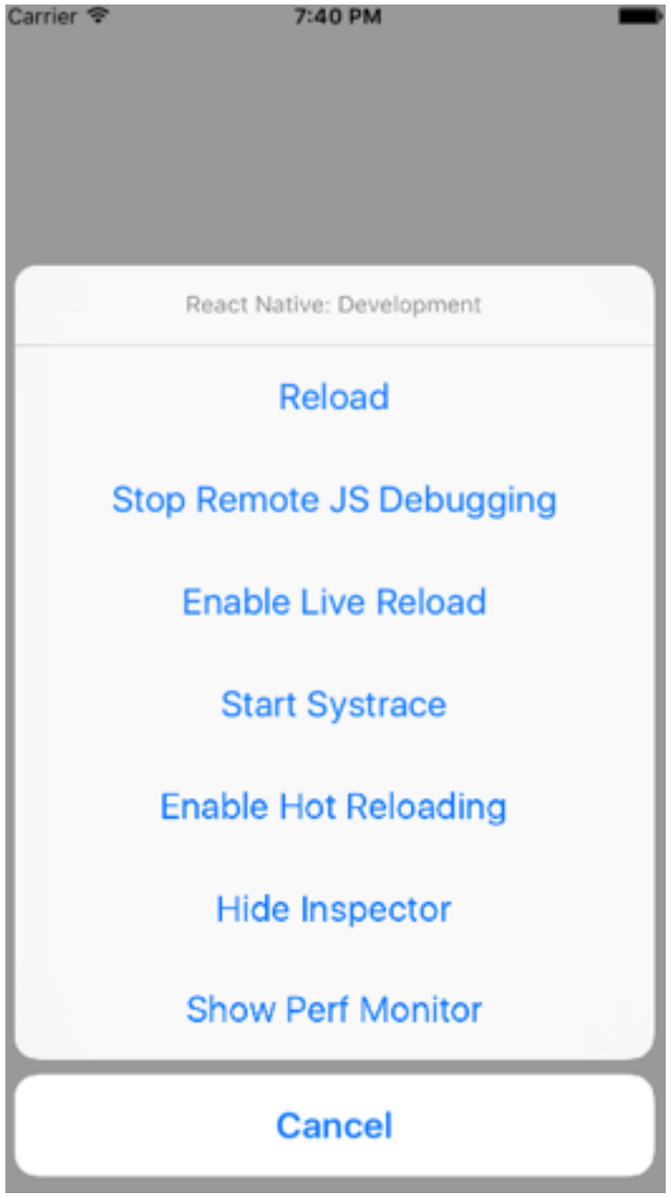
IMAGE CACHING

- Ziel: alle Thumbnails beim Start einmalig cachen
- transparent zur <Image /> Komponente
- **durchprobiert:**
  - react-native-http-cache (GitHub) –> schwierig aber allgemein verwendbar
  - react-native-realm-cache-image (GitHub) –> Realm + react-native-fs –> Dependency-Hell.. + FS ist 2.0.0RC
  - react-native-ximage (GitHub) –> deprecated
- **Ausblick:**
  - selber was KISS bauen mit `fetch()` und react-native-fs
  - dann als lokale Datei in `<Image source=„./bilder/Bla.jpg“ />`

DAS BESTE ZUM SCHLUSS: DEBUGGING

IOS SIMULATOR, ANDROID EMULATOR





- Reload mit Cmd+R
- Live Reload
  (Auto Cmd+R)
- Hot Code Reloading
  (toll bei Style-Tuning)
- Inspector
  (No shit, Watson!)

- Android: Cmd+M
- iOS: Cmd+D

OFFLINE-FIRST TESTEN MIT IOS SIMULATOR?

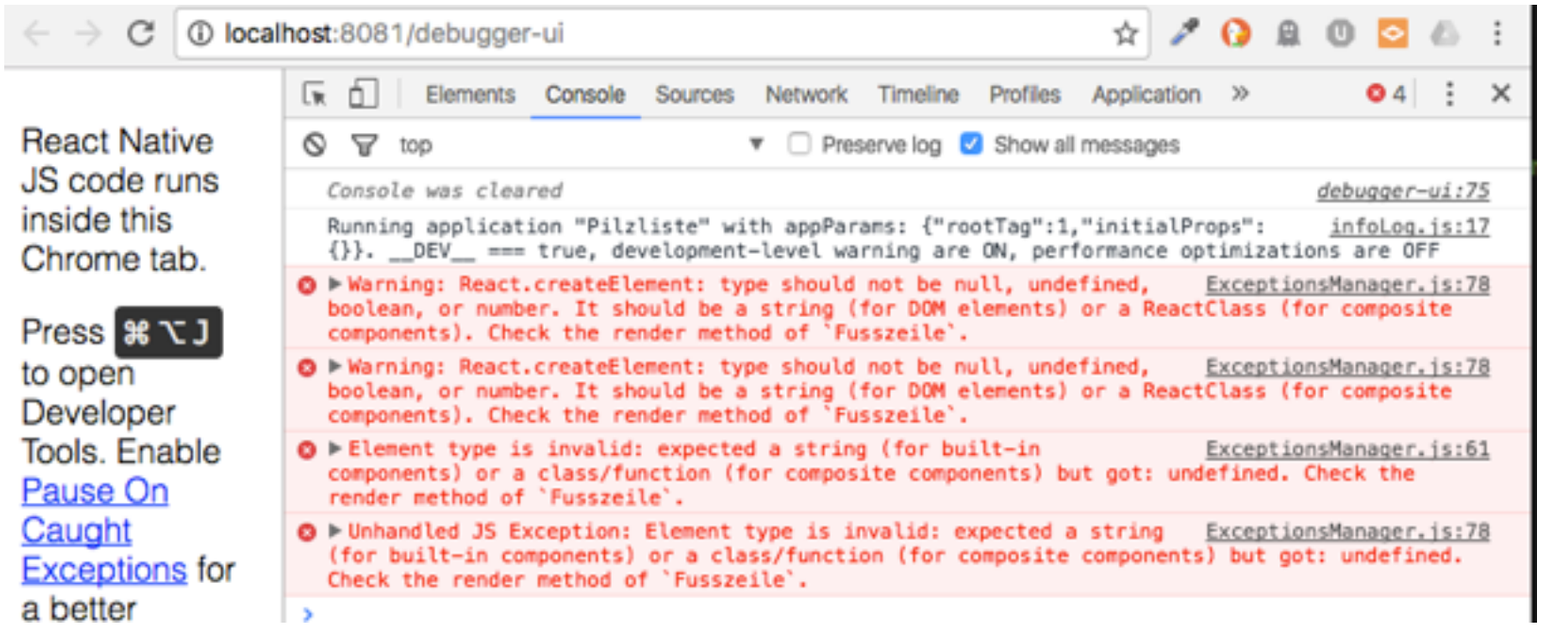
- Überraschung! Flight-Mode ist nicht vorgesehen
- einfache Lösung: **MacBook WLAN deaktivieren**
- etwas schwieriger: Bilder-Domain in `/etc/hosts` auf localhost umbiegen
- oder: Application-Firewall
- but wait, there is more: Apple's Network Link Conditioner
- aber so richtig offline geht der Simulator damit nicht 😞

DEBUGGING TOOLS

Element type is invalid: expected a string (for built-in components) or a class/function (for composite components) but got: undefined. Check the render method of `Fusszeile`.

instantiateReactComponent
instantiateReactComponent.js:70

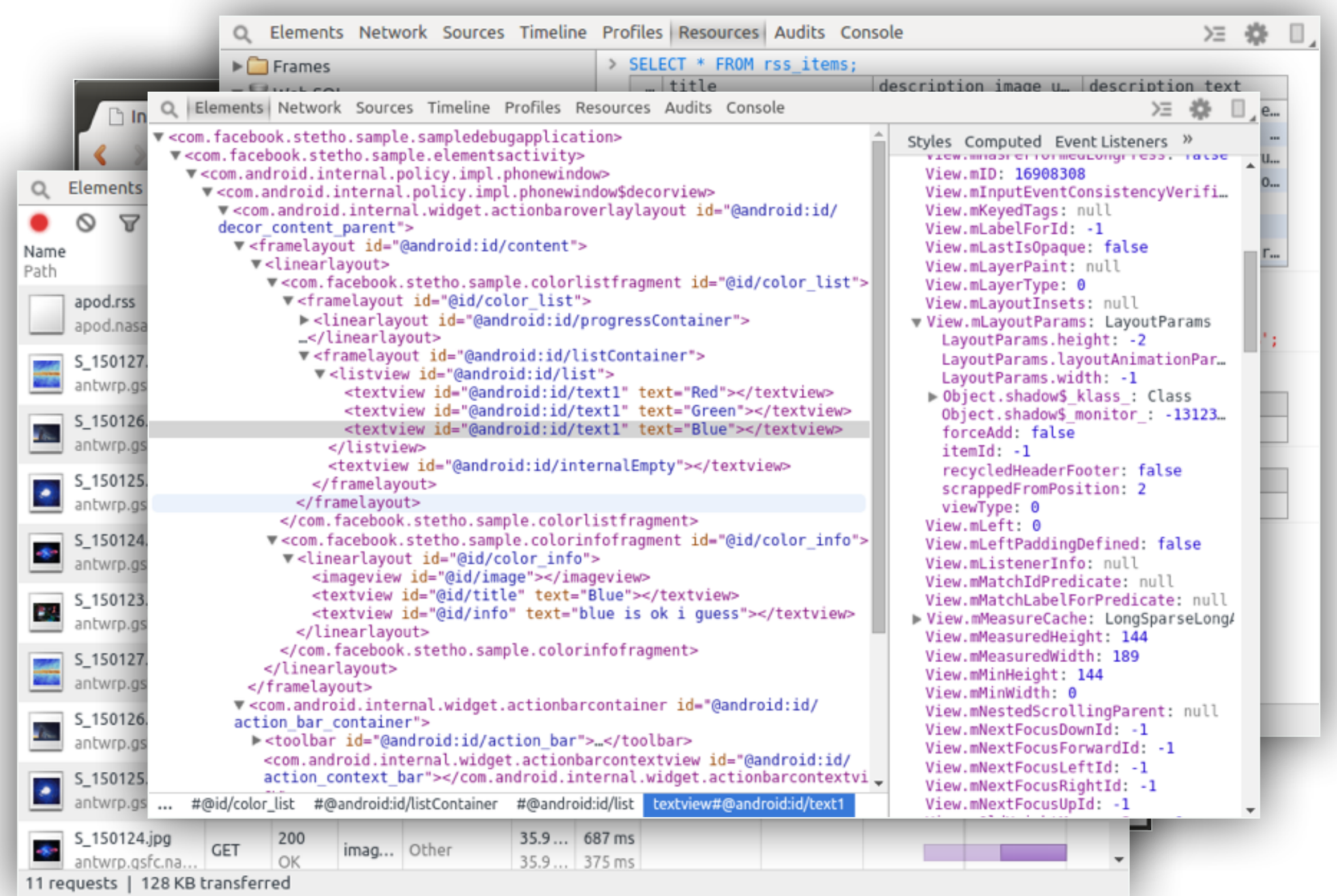
instantiateChild
ReactChildReconciler.js:45



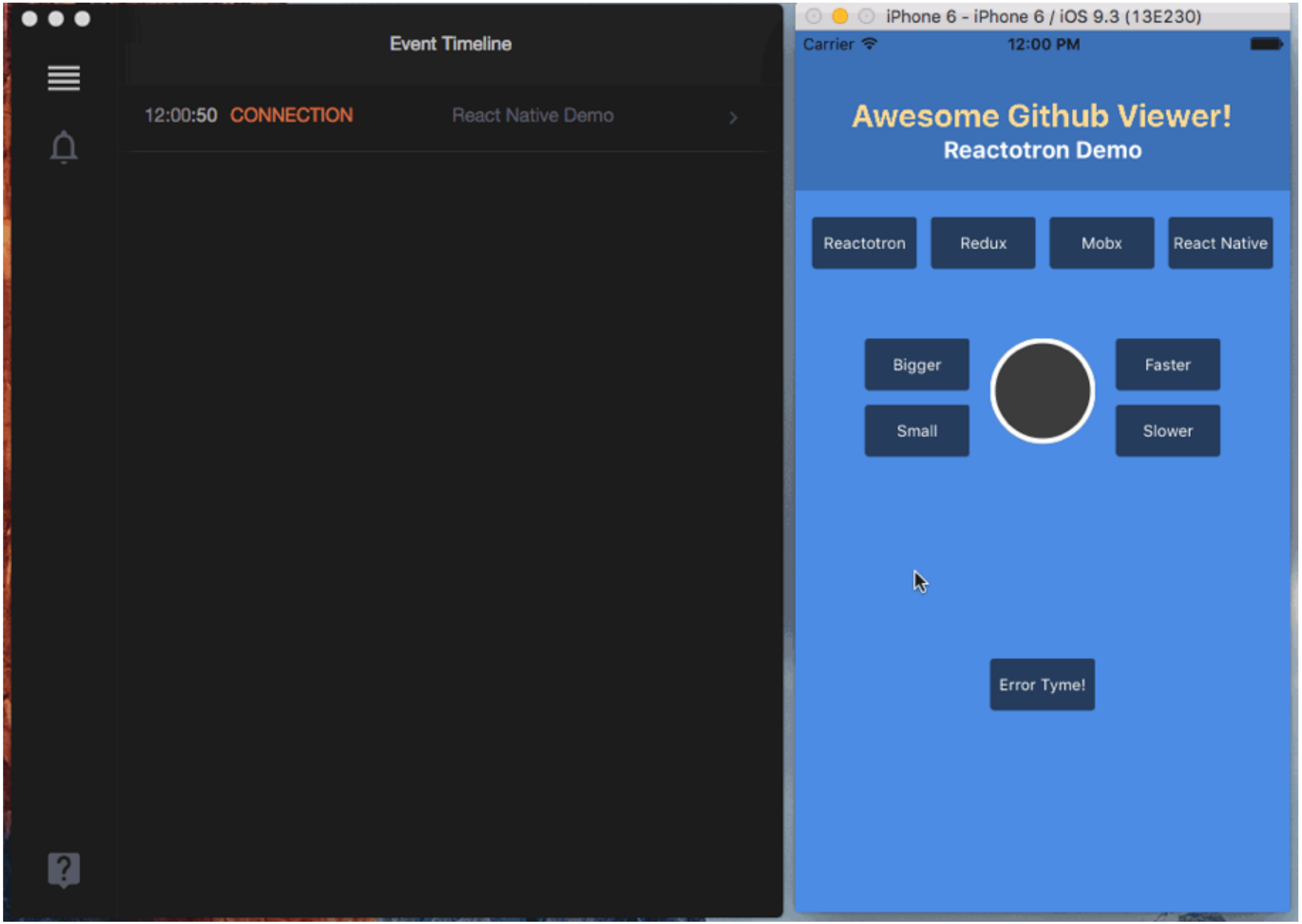
- **Chrome Developer Tools**
- console.log(..) (und auch .warn(), .error() etc..)
- react-native log-[ios|android]
- Android Debugger Bridge von FB: Stetho

STETHO

facebook.github.io/stetho/

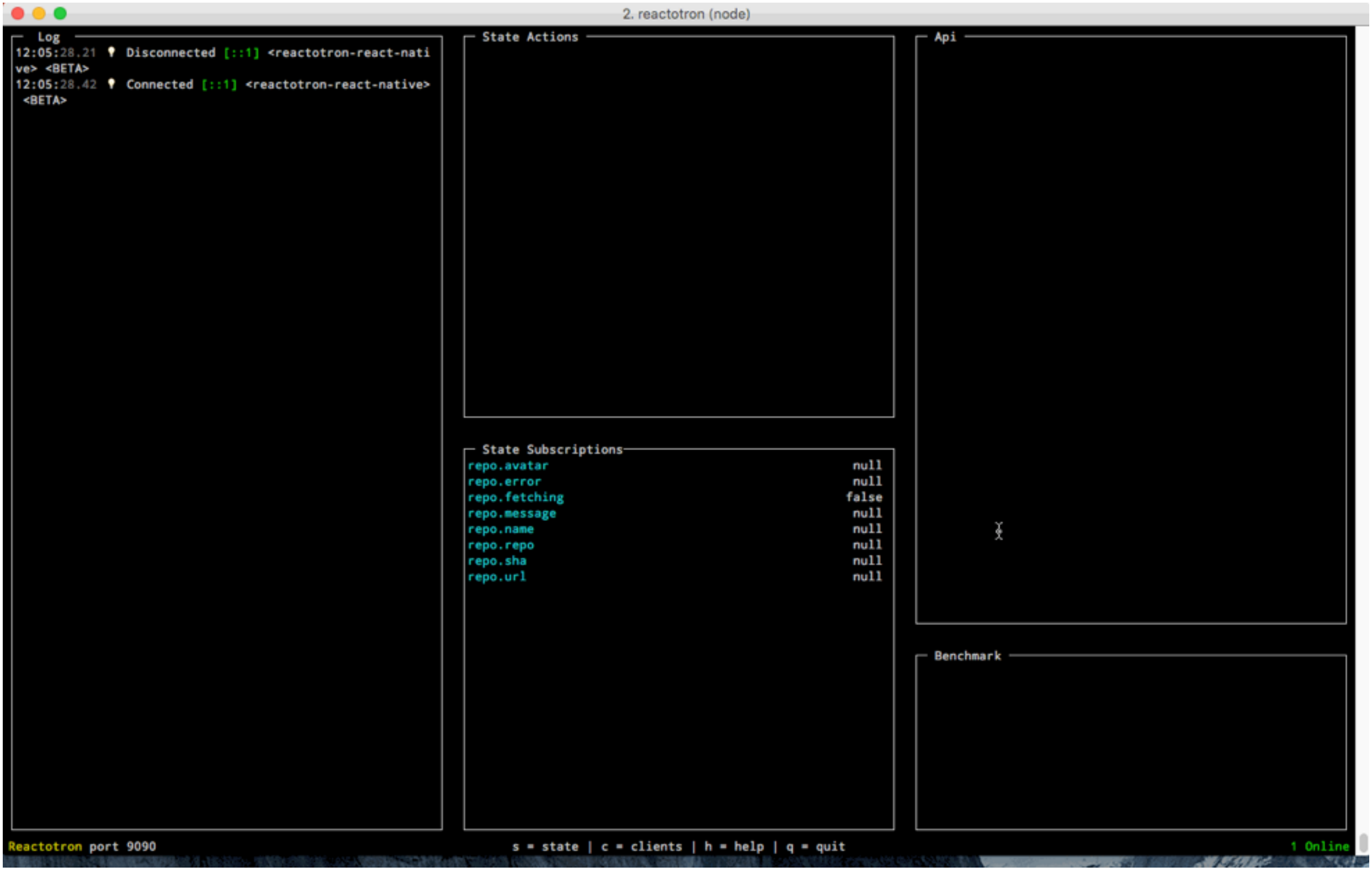


REACTORON GUI



REACTORON CLI

reactotron/reactotron



SCHLUSS!

- Ronny Hartenstein
- Twitter: @rhflow_de
- GitHub: ronnyhartenstein/pilzliste-react-native-redux
- Webseite: http://blog.rh-flow.de/pilzliste-react-native-redux