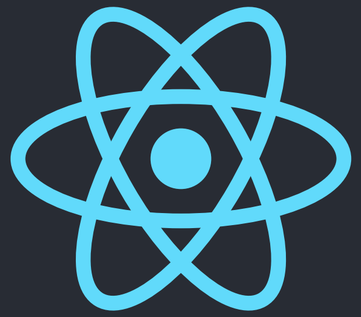


React Native

Tim Kuhlmann, Lukas Obermann,
Julia Schmekel, Maja Weiblen

Wer kennt React oder React Native?

1 Aktuelle Verwendung

Wer nutzt React Native?

- Facebook (Android & iOS)
- Instagram (Android & iOS)
- Oculus (Android & iOS)
- Shopify (Android & iOS)
- Discord (iOS)
- Skype (Android & iOS)
- Pinterest (Android & iOS)

Warum?

- Kann auf mehreren Plattformen (Android, IOS, Windows und Web) eingesetzt werden
- Oberflächen, Animationen und aufwändige Berechnungen werden nativ ausgeführt
- Es ist Open Source
- Zweitbeliebtestes Cross-Platform-Framework

2

Thinking in React Part 1

React

- Seit 29. Mai 2013 (Web) bzw. 26. März 2015 (Native)
- Meta
- Deklarative UI-Programmierung
- JavaScript

Komponenten

- Functional Components mit Hooks
- Class Components

```
const Hello = () ⇒ …

class Hello extends React.Component {
  …
}
```

Elemente

`createElement`-API

```
import React from "react"

const Hello = () ⇒ React.createElement(
  elementType,
  props,
   ... children
)
```

Elemente

`createElement`-API

```
import React from "react"

const Hello = () ⇒ React.createElement(
  "section",            // Element Typ
  null,                 // Attribute/Props
  React.createElement(  // Children
    "p",                // …
    null,
    "Hello World!"
  )
)
```

Entry Point

```
<body>
  <div id="root">
    <section>
      <p>
        Hello World!
      </p>
    </section>
  </div>
  …
</body>
```

```
import React from "react"
import { createRoot } from "react-dom"

const Hello = () ⇒ React.createElement(
  "section",
  null,
  React.createElement(
    "p",
    null,
    "Hello World!"
  )
)

const root = createRoot(document.querySelector("#root"))
root.render(React.createElement(Hello))
```

Komposition

Komponenten verschachteln

```
import React from "react"

const HelloParagraph = () ⇒ React.createElement(
  "p",
  null,
  "Hello World!"
)

const Hello = () ⇒ React.createElement(
  "section",
  null,
  React.createElement(
    HelloParagraph
  )
)
```

Attribute

Attribute (`Props`) übergeben

```
import React from "react"

const HelloParagraph = () ⇒ React.createElement(
  "p",
  null,
  "Hello World!"
)

const Hello = () ⇒ React.createElement(
  "section",
  { id: "my-section" },
  React.createElement(
    HelloParagraph
  )
)
```

Attribute

Attribute (`Props`) übergeben

›Pur‹ im Bezug auf die Props

```
import React from "react"

const HelloParagraph = props ⇒ React.createElement(
  "p",
  null,
  props.text
)

const Hello = () ⇒ React.createElement(
  "section",
  { id: "my-section" },
  React.createElement(
    HelloParagraph,
    { text: "Hello React!" }
  )
)
```

Attribute

Attribute (`Props`) übergeben

›Pur‹ im Bezug auf die Props

```
import React from "react"

const HelloParagraph = props ⇒ React.createElement(
  "p",
  null,
  props.children
)

const Hello = () ⇒ React.createElement(
  "section",
  { id: "my-section" },
  React.createElement(
    HelloParagraph,
    null,
    React.createElement(
      "span",
      null,
      "Hello World!"
    ),
    " ",
    React.createElement(
      "span",
      null,
      "Hello React!"
    )
  )
)
```

JSX

- XML-artige Syntax in JavaScript
- Babel/TypeScript

```
const HelloParagraph = props ⇒ (
  <p>
    {props.text}
  </p>
)

const Hello = () ⇒ (
  <section id="my-section">
    <HelloParagraph text="Hello World!" />
  </section>
)
```

```
import React from "react"

const HelloParagraph = props => React.createElement(
  "p",
  null,
  props.text
)

const Hello = () => React.createElement(
  "section",
  { id: "my-section" },
  React.createElement(
    HelloParagraph,
    { text: "Hello React!" }
  )
)
```

```
const HelloParagraph = props => (
  <p>
    {props.text}
  </p>
)

const Hello = () => (
  <section id="my-section">
    <HelloParagraph text="Hello World!" />
  </section>
)
```

```
import React from "react"

const HelloParagraph = props => React.createElement(
  "p",
  null,
  props.children
)

const Hello = () => React.createElement(
  "section",
  { id: "my-section" },
  React.createElement(
    HelloParagraph,
    null,
    React.createElement(
      "span",
      null,
      "Hello World!"
    ),
    " ",
    React.createElement(
      "span",
      null,
      "Hello React!"
    )
  )
)
```

```
const HelloParagraph = props => (
  <p>
    {props.children}
  </p>
)

const Hello = () => (
  <section id="my-section">
    <HelloParagraph>
      <span>Hello World!</span>
      {" "}
      <span>Hello React!</span>
    </HelloParagraph>
  </section>
)
```

React Basics

Hooks

Interaktivität bei Functional Components

`useState`, `useEffect`, `use…`

```
const HelloParagraph = props ⇒ (
  <p>
    {props.text}
  </p>
)

const Hello = () ⇒ (
  <section id="my-section">
    <HelloParagraph text="Hello World!" />
  </section>
)
```

State

```
const HelloParagraph = props ⇒ (
  <p>
    {props.text}
  </p>
)

const Hello = () ⇒ (
  <section id="my-section">
    <HelloParagraph text="Hello World!" />
  </section>
)
```

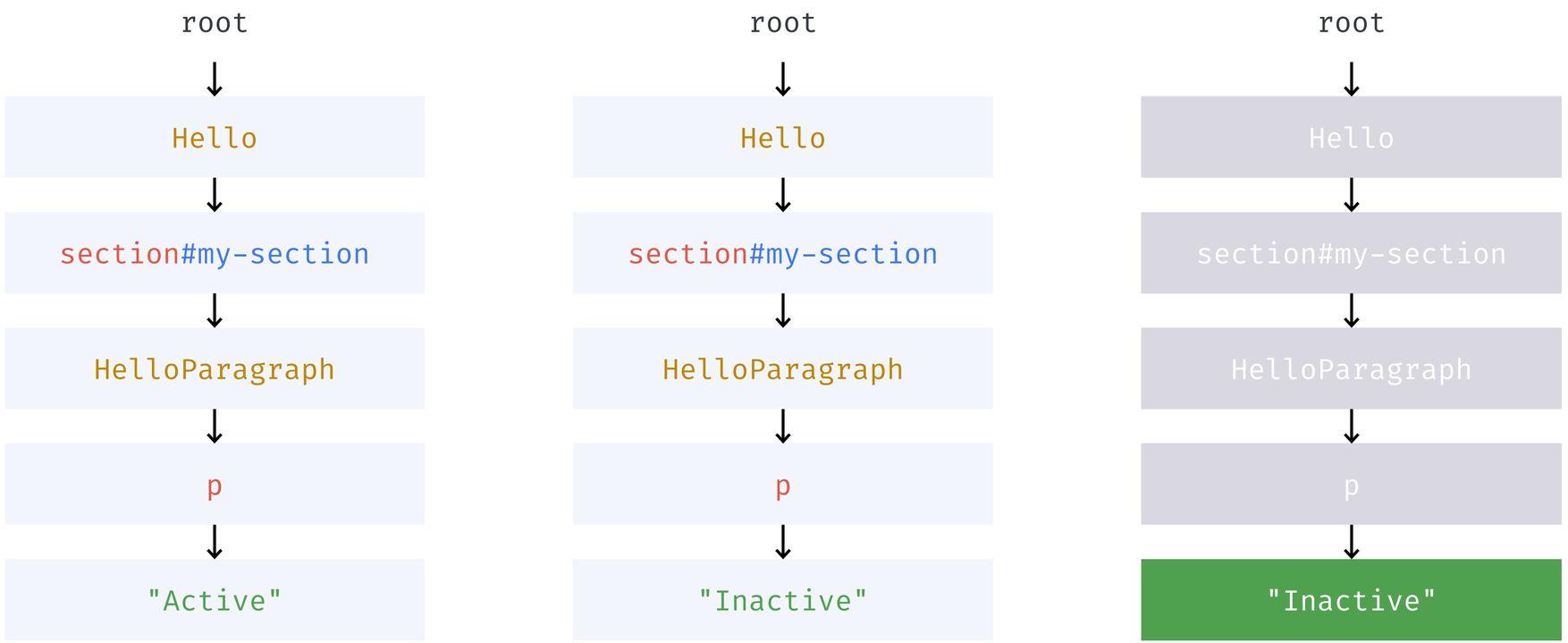
State

Ein `useState` pro Wert

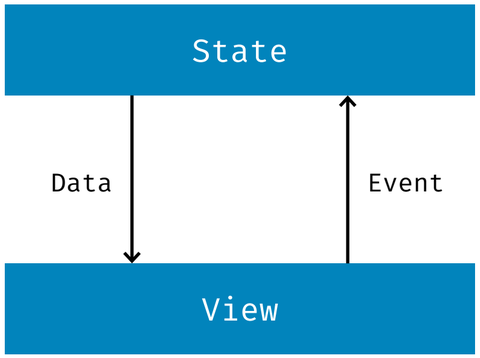
Bei jedem Rerender werden mindestens alle Children erneut durchlaufen

```
import { useState } from "react"

const HelloParagraph = props ⇒ …

const Hello = () ⇒ {
  const [isActive, setIsActive] = useState(false)

  return (
    <section id="my-section">
      <HelloParagraph
        text={isActive ? "Active" : "Inactive"}
        />
    </section>
  )
}
```

Exkurs: Virtual DOM



Events




```
import { useState } from "react"

const HelloParagraph = props ⇒ …

const Hello = () ⇒ {
  const [isActive, setIsActive] = useState(false)

  const handleOnClick = () ⇒
    setIsActive(oldValue ⇒ !oldValue)

  return (
    <section id="my-section">
      <HelloParagraph
        text={isActive ? "Active" : "Inactive"}
        />
      <button onClick={handleOnClick}>Toggle</button>
    </section>
  )
}
```

Side Effects

`useEffect`

- File I/O oder Web-API-Zugriffe
- DOM-Manipulation
- Timer

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

const HelloParagraph = props ⇒ …

const Hello = () ⇒ {
  …

  useEffect(
    () ⇒ {
      document.title =
        `Hello is ${isActive ? "Active" : "Inactive"}`
    }
  )

  return (
    <section id="my-section">
      <HelloParagraph
        text={isActive ? "Active" : "Inactive"}
        />
      <button onClick={handleOnClick}>Toggle</button>
    </section>
  )
}
```

Side Effects

`useEffect`

- File I/O oder Web-API-Zugriffe
- DOM-Manipulation
- Timer

```
import { useCallback, useEffect, useState } from "react"

const HelloParagraph = props ⇒ …

const Hello = () ⇒ {
  …

  useEffect(
    () ⇒ {
      document.title =
        `Hello is ${isActive ? "Active" : "Inactive"}`
    },
    [isActive] // Dependency Array
  )

  return (
    <section id="my-section">
      <HelloParagraph
        text={isActive ? "Active" : "Inactive"}
        />
      <button onClick={handleOnClick}>Toggle</button>
    </section>
  )
}
```

Side Effects

`useEffect`

- File I/O oder Web-API-Zugriffe
- DOM manipulation
- Timers

```
import { useCallback, useEffect, useState } from "react"

const HelloParagraph = props => …

const Hello = () => {
  …

  useEffect(
    () => {
      document.title =
        `Hello is ${isActive ? "Active" : "Inactive"}`
    },
    [isActive]
  )

  return (
    <section id="my-section">
      <HelloParagraph
        text={isActive ? "Active" : "Inactive"}
        />
      <button onClick={handleOnClick}>Toggle</button>
    </section>
  )
}
```

Memoization

useMemo

» Komplexe Berechnungen

```
import { …, useMemo, useState } from "react"

const HelloParagraph = props ⇒ …

const Hello = () ⇒ {
  …

  const activeText = useMemo(
    () ⇒ generateExpensiveString(isActive),
    [isActive]
  )

  useEffect(
    () ⇒ {
      document.title = `Hello is ${activeText}`
    },
    [activeText]
  )

  return (
    <section id="my-section">
      <HelloParagraph text={activeText} />
      <button onClick={handleOnClick}>Toggle</button>
    </section>
  )
}
```

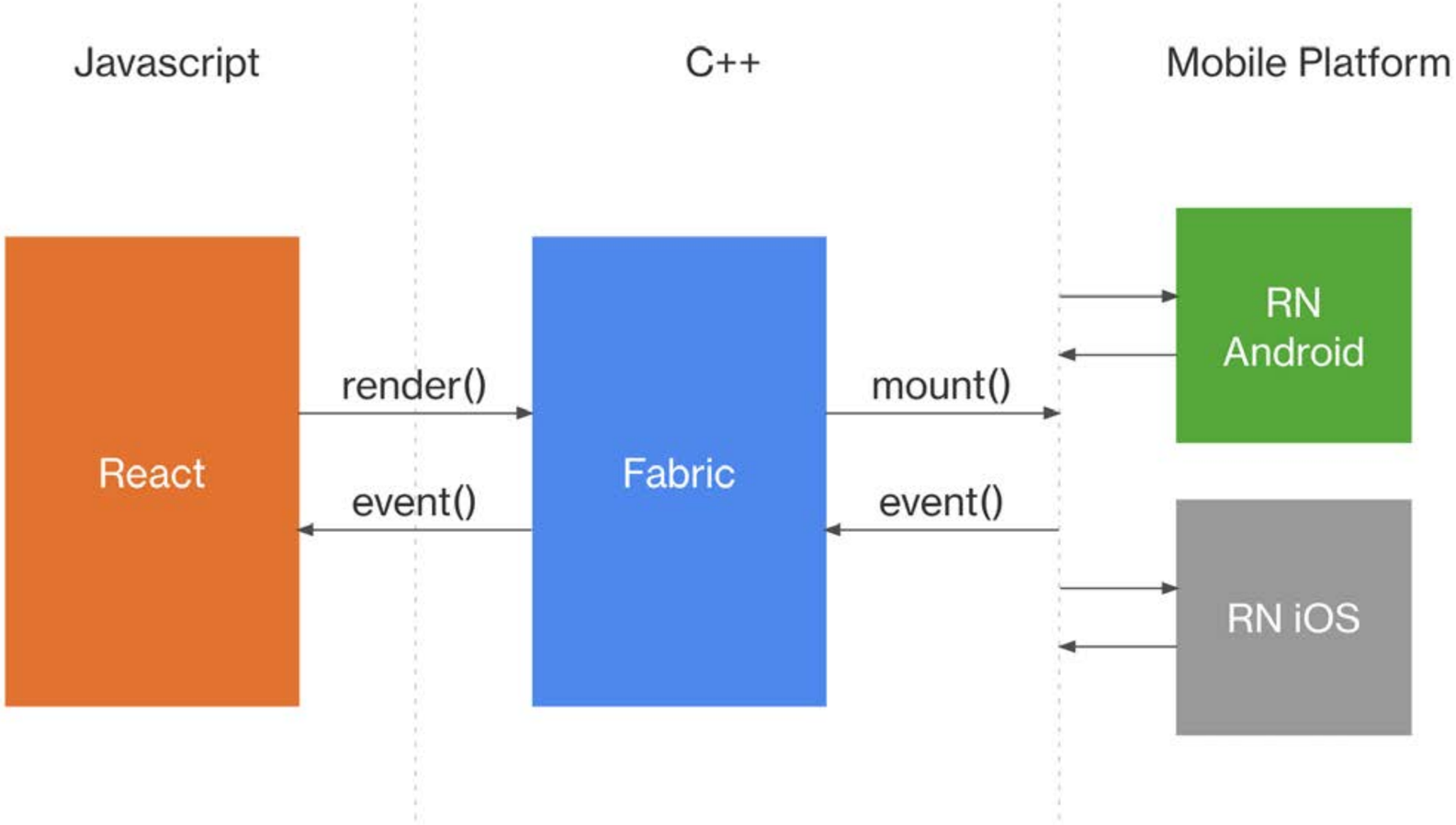
3

React Native

```jsx
import React from "react"


export default () => (
  <div className="container">
    <MyWidget />
  </div>
)

const MyWidget = () => (
  <h4>
    Hello, World!
  </h4>
)

// CSS

.container {
  justify-content: center;
  background-color: #ecf0f1;
  padding: 8px;
}

h4 {
  text-align: center;
}
```

```jsx
import React from 'react'
import { Text, View, StyleSheet } from 'react-native'

export default () => (
  <View style={styles.container}>
    <MyWidget />
  </View>
)

const MyWidget = () => (
  <Text style={ styles.h4 }>
    Hello, World!
  </Text>
)

const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  h4: {
    textAlign: 'center',
  }
})
```

React Native Architektur



Fabric Renderer

- evolutionäre Weiterentwicklung des alten Renderers
  - Entwicklung 2018, Einführung 2021 (Facebook-App)
- mehr Renderlogik in C++ umgesetzt
- verbesserte Interoperabilität mit Host-Plattformen
- reduzierte Startzeit und verbesserte Performance

Fabric Renderer

- Type Safety
  - aus JavaScript generierte C++-Structs werden gegen Host-Props abgeglichen
  - Typ-Probleme führen zu Build-Fehlern – nicht zu Laufzeitfehlern
- Renderer-Kern wird zwischen Host-Plattformen geteilt (C++)
  - erhöht die Konsistens zwischen den Plattformen
- Weniger (De-)Serialisierung von Daten zwischen JavaScript und C++

Render Pipeline

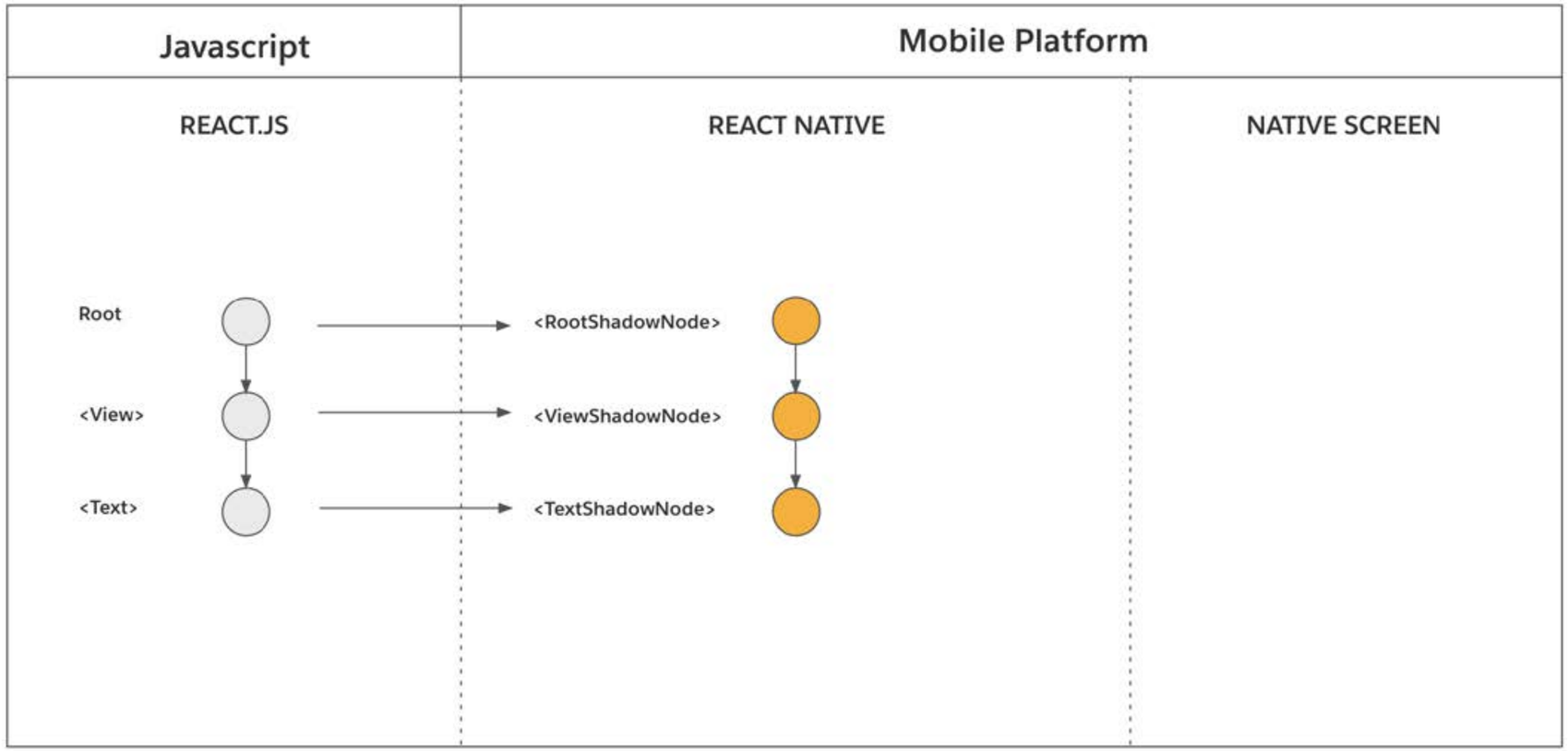
1. Render-Phase
   - » Erstellen des *React-Element-Trees* in JavaScript
     - » Repräsentation der UI-Elemente mit Eigenschaften, Styles und Kindelementen
   - » Erzeugen des *React-Shadow-Trees* in C++
     - » Repräsentation der nativen UI-Komponenten
2. Commit-Phase
   - » die vollständig aufgebauten *Trees* werden für die Darstellung eingereiht
   - » die Berechnung der Layoutinformationen wird angestoßen

Render Pipeline

3. Mount-Phase
   - » der vollständig ge*layout*ete *Shadow Tree* wird in den nativen *Host-View-Tree* überführt
   - » der aktuelle und der neue *Tree* werden verglichen (C++)
   - » nur Änderungen werden im *Host-View-Tree* angepasst

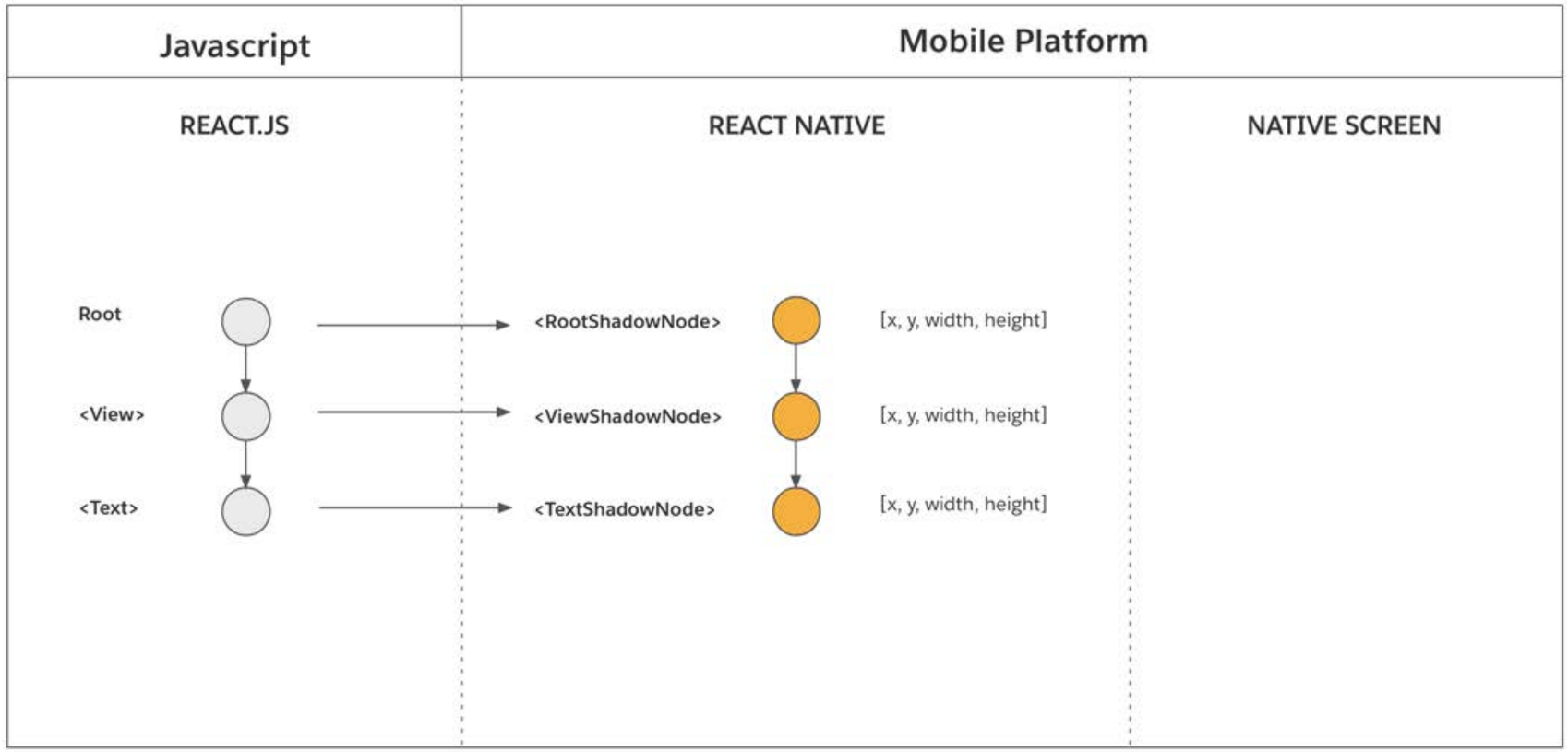
Render-Phase

```
function MyComponent() {
  return (
    <View>
      <Text>Hello, World</Text>
    </View>
  );
}
```



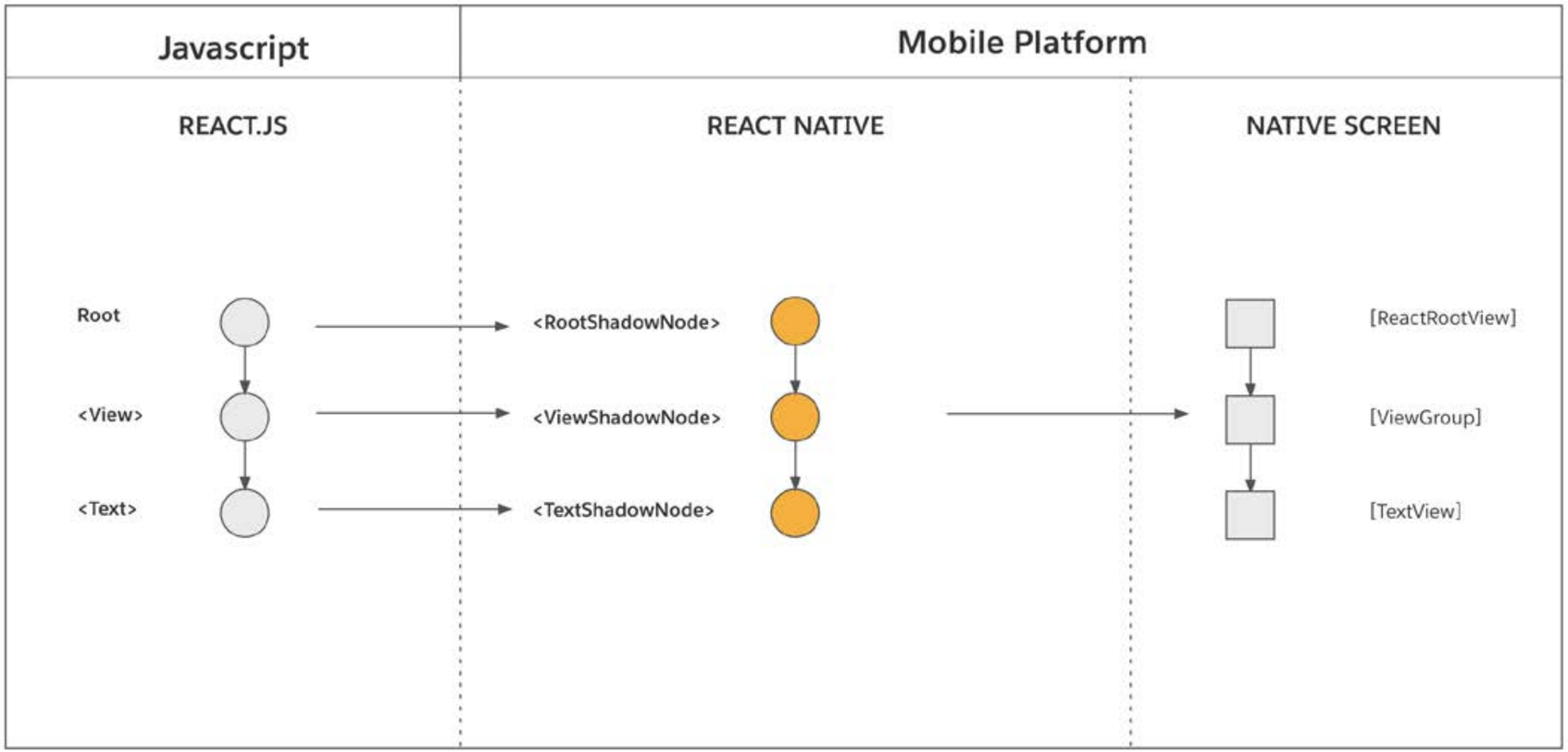
Commit-Phase

```
function MyComponent() {
  return (
    <View>
      <Text>Hello, World</Text>
    </View>
  );
}
```



Mount-Phase

```
function MyComponent() {
  return (
    <View>
      <Text>Hello, World</Text>
    </View>
  );
}
```



Expo 4

Warum Expo?

- Schnellster Weg, React Native Apps zu entwickeln
- Apps aktualisieren über die „Luft“ anstatt
  über den Play Store/App Store („Publish Over The Air“)
- Eingebauter Zugang zu Native APIs
- Es ist kostenlos und Open Source

5

Übung 1

github.com/elyukai/me-react-native

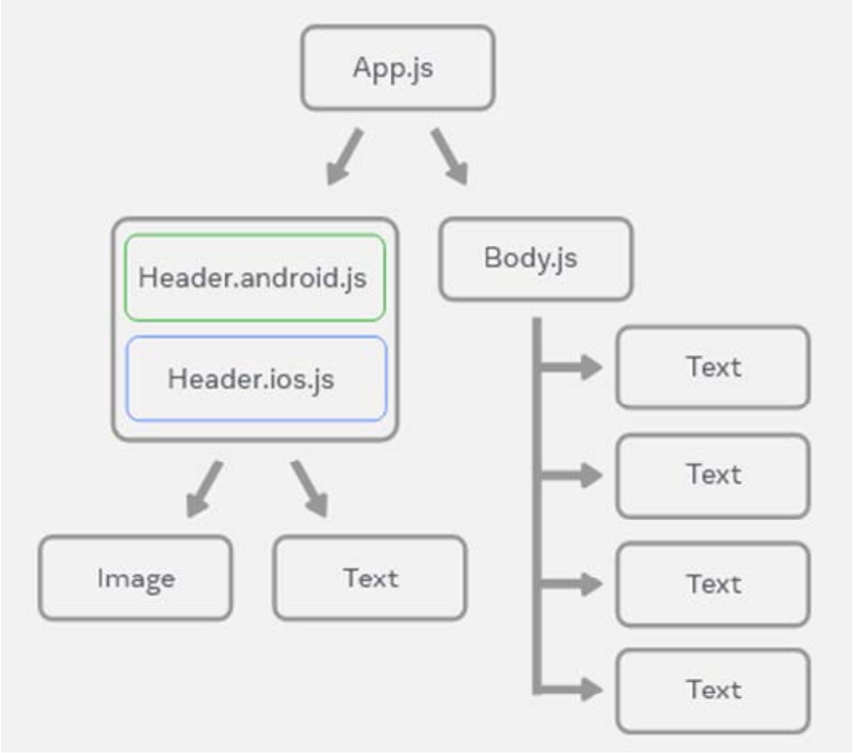
exercise-1/README.md

6

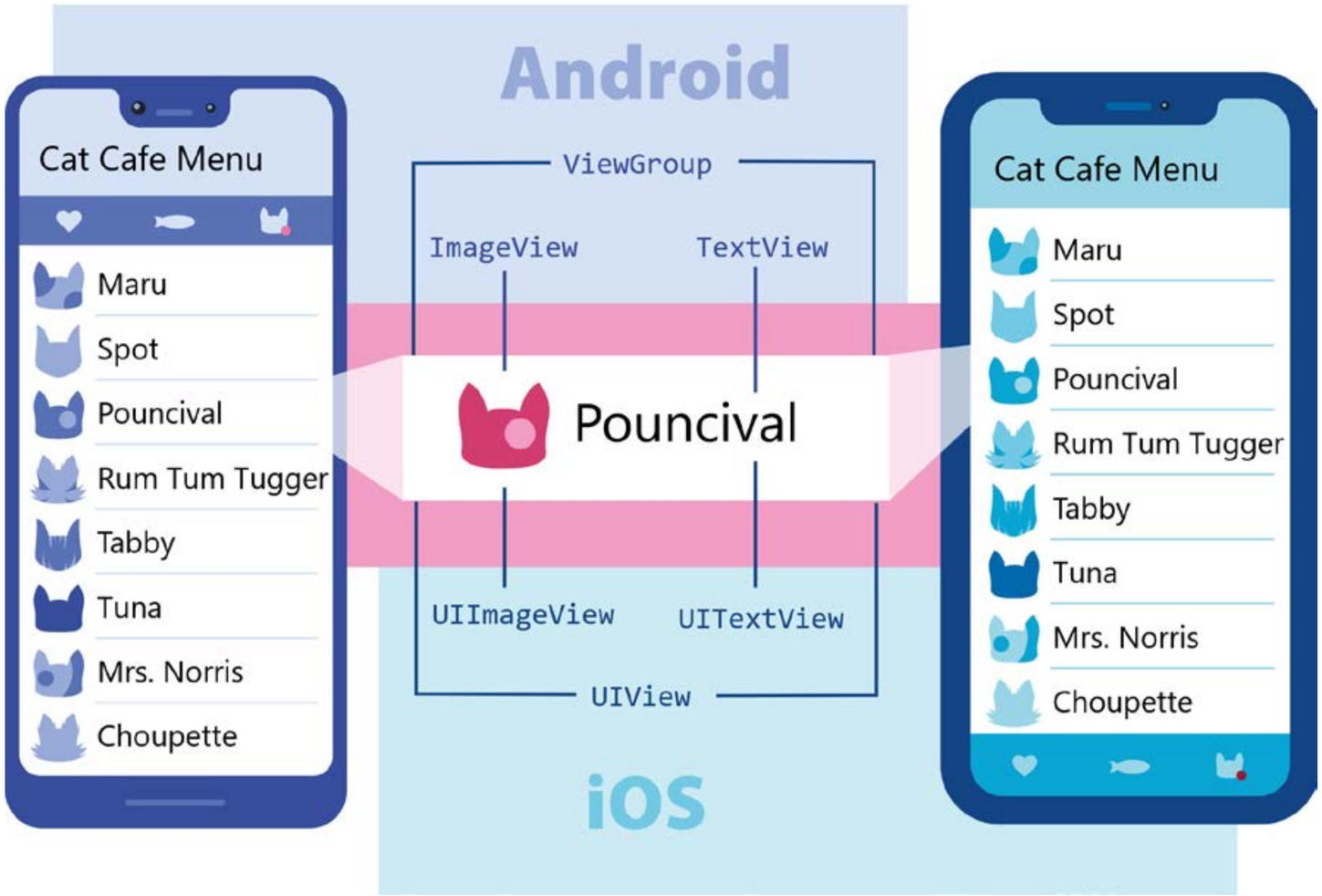
React Native als Cross-Platform-Umgebung

RN für verschiedene Plattformen

- Einmal JavaScript Code für verschiedene Plattformen:
  Android, iOS, Web
- Unterschiede in den Funktionen
  (z. B. Anmeldeprozesse etc.)
- React Native erstellt beim Ausführen die plattformspezifischen Anzeigen
- Entwicklung mit Mac empfohlen
- Unterschiede im Release Prozess



Views


Android
Cat Cafe Menu
Maru
Spot
Pouncival
Rum Tum Tugger
Tabby
Tuna
Mrs. Norris
Choupette
ViewGroup
ImageView
TextView
Pouncival
UIImageView
UITextView
UIView
iOS
Cat Cafe Menu
Maru
Spot
Pouncival
Rum Tum Tugger
Tabby
Tuna
Mrs. Norris
Choupette

Views

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrollling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images

7 Styling

Design

- Verschiedene Designrichtlinien je nach Platform
  - Android: Material Design
  - iOS: Human Interface Guidelines
- Die gleichen Components sehen auf den unterschiedlichen Plattformen verschieden aus
- Z. B. bei Buttons, Eingabefeldern, Date-/Timepicker, Listen, etc.

```
<Button title="Click Me" color="#ff0000" />

// Android
```


```
// iOS
```

StyleSheets

- Ähnlich wie CSS in der Webentwicklung
- Camel case
  - `borderRadius` statt `border-radius`


```jsx
import { StyleSheet, Text, View } from "react-native";

const App = () => (
  <View style={styles.container}>
    <Text style={styles.title}>React Native</Text>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    backgroundColor: "#eaeaea"
  },
  title: {
    marginTop: 16,
    paddingVertical: 8,
    backgroundColor: "#61dafb",
    textAlign: "center",
    fontSize: 30,
    fontWeight: "bold",
    …
  }
});

export default App;
```

Platform Module

» Plattformspezifische Styles für Android und iOS

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    ...Platform.select({
      ios: {
        backgroundColor: 'red'
      },
      android: {
        backgroundColor: 'green'
      },
      default: {
        // other platforms, web for example
        backgroundColor: 'blue'
      }
    })
  }
});
```

Übung 2

github.com/elyukai/me-react-native

exercise-2/README.md

9

Thinking in React Part 2

State Reducer

Pure Funktionen

```
[1, 2, 3].reduce((acc, x) ⇒ acc + x, 0) // 6

                  (0  , 1) ⇒ 0   + 1     // 1
                  (1  , 2) ⇒ 1   + 2     // 3
                  (3  , 3) ⇒ 3   + 3     // 6
```

State Reducer

- Komplexere, voneinander abhängige Werte
- Immutability

```
const reducer = (oldState, action) ⇒ {
  switch (action.type) {
    case "add":
      return [ ... oldState, action.payload.text]

    case "remove":
      return oldState.filter(
        element ⇒ element !== action.payload.text
      )

    case "reset":
      return []

    default:
      return oldState
  }
}

const exampleAction = {
  type: "add",
  payload: {
    text: "New List Item"
  }
}
```

React Basics

State Reducer

useReducer

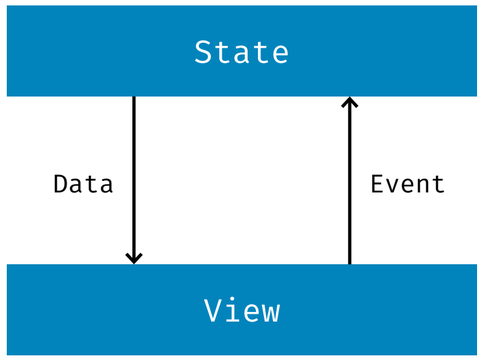
Redux

```
const App = () ⇒ {
  const [state, dispatch] = useReducer(reducer, [])

  …
}
```

Input Handling

›Controlled Input‹

React Native: onChangeText




```
const App = () => {
  const [state, dispatch] = useReducer(reducer, [])

  const [input, setInput] = useState("")

  return (
    <section className="list">
      <input
        value={input}
        onChange={event => setInput(event.target.value)}
        />
    </section>
  )
}
```

Input Handling

```
const App = () => {
  const [state, dispatch] = useReducer(reducer, [])

  …

  const handleAdd = () =>
    dispatch({ type: "add", payload: input })

  const handleReset = () =>
    dispatch({ type: "reset" })

  return (
    <section className="list">
      <input
        value={event => setInput(event.target.value)}
        onChange={handleInput}
        />
      <button
        onClick={handleAdd}
        disabled={input.length === 0}
        >
        Add
      </button>
      <button onClick={handleReset}>Reset</button>
    </section>
  )
}
```

Input Handling

```
const App = () => {
  …

  const handleRemove = value =>
    dispatch({ type: "remove", payload: value })

  return (
    <section className="list">
      <List list={state} onRemove={handleRemove} />
      <input
        value={event => setInput(event.target.value)}
        onChange={handleInput}
        />
      <button
        onClick={handleAdd}
        disabled={input.length === 0}
        >
        Add
      </button>
      <button onClick={handleReset}>Reset</button>
    </section>
  )
}
```

Arrays

```
const List = props ⇒ (
  <ul>
    {props.list.map((element, index) ⇒ (
      …
    ))}
  </ul>
)
```

Arrays

Unique Key erlaubt Optimierung

Registrierung bei oberster Komponente

```
const List = props => (
  <ul>
    {props.list.map((element, index) => (
      <ListItem
        key={index}
        value={element}
        onRemove={props.onRemove}
        />
    ))}
  </ul>
)
```

Arrays

Hier **nicht** den Key registrieren!

```
const ListItem = props ⇒ (
  <li>
    {props.value}
    <button onClick={() ⇒ props.onRemove(props.value)}>
      Remove
    </button>
  </li>
)
```

10

Übung 3

github.com/elyukai/me-react-native

exercise-3/README.md

11

Bewertung & React Native im Vergleich

Plattformen

- Android
- iOS
- macOS (3rd-Party)
- Windows/UWP (3rd-Party)
- weitere inoffiziell (Web, tvOS, Qt, …)

Sensor-Zugriff

Per `react-native-sensors`

- Beschleunigung (Accelerometer)
- Neigung (Gyroscope)
- Kompass (Magnetometer)
- Luftdruck, Höhenänderungen (Barometer)

Weitere Features

- Key-Value-Store (`async-storage`)
- Datenbank (`react-native-sqlite-storage`)
- Networking & Security
- Kamera (`react-native-vision-camera`)
- 3D-Modelle (`react-native-3d-model-view`)

React Native vs. Flutter

Backing

	React Native	Flutter
Backing	Facebook	Google

Plattformen

	React Native	Flutter
Mobile	Android, iOS	Android, iOS
Desktop	macOS, UWP (Partner)	macOS, Linux, Windows
Web	Preview (expo) + Community	Canvas-based PWA

Architektur

	React Native	Flutter
Programmieren	JavaScript (dyn. typisiert)	Dart (st. typisiert, null-safe)
Ausführen	JS (↑) + Native (Engine)	Native
Oberfläche	Native Elemente	Nachbau (OpenGL)
Look & Feel	Bedingt nativ	Nativ

```jsx
import React from 'react'
import { Text, View, StyleSheet } from 'react-native'

export default () => (
  <View style={styles.container}>
    <MyWidget />
  </View>
)

const MyWidget = () => (
  <Text style={ styles.h4 }>
    Hello, World!
  </Text>
)

const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  h4: {
    textAlign: 'center',
  }
})
```

```dart
import 'package:flutter/material.dart';

const Color backgroundColor = Color.fromARGB(…);

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: backgroundColor,
        body: Center(
          child: MyWidget()
        ),
      )
    );
  }
}

class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text(
      'Hello, World!',
      style: Theme.of(context).textTheme.headline4,
    );
  }
}
```

12

Fazit

Fazit

Pro

- Beschleunigt den Entwicklungsprozess
- Niedrigere Kosten bei der App-Entwicklung
- Stabiles Wachstum der App
- Zugang zu vielen Librarys und Tools
- Kann in eine native Anwendung integriert werden
- Wird von externen Tools unterstützt

Contra

- Ist keine native Lösung
- Schwer zu debuggen
- Beschleunigt den Testprozess nicht
- App testen ist komplizierter
- Nicht so guter Android-Support
- Schwieriger ein plattform-übergreifendes Team aufzustellen

13 Literatur

React

» https://reactjs.org/docs/introducing-jsx.html

» https://reactjs.org/docs/components-and-props.html

» https://kentcdodds.com/blog/usememo-and-usecallback

» https://reactjs.org/docs/hooks-intro.html

» https://reactjs.org/docs/hooks-overview.html

React Native

» https://www.codecademy.com/courses/learn-react-native/

» https://reactnative.dev/docs

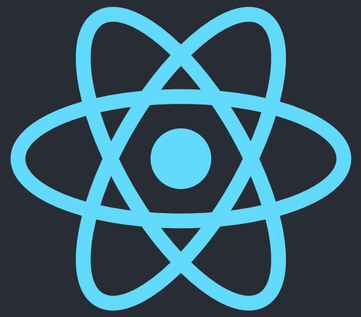
» https://reactnative.dev/architecture/overview

» https://reactnative.dev/architecture/xplat-implementation

- https://reactnative.dev/architecture/fabric-renderer
- https://reactnative.dev/architecture/render-pipeline
- https://programmingwithmosh.com/react-native/expo-or-not-building-react-native-apps/

Bewertung & Vergleich

- https://www.thedroidsonroids.com/blog/react-native-pros-and-cons
- https://programmingwithmosh.com/react-native/differences-in-building-ios-and-android-apps-using-react-native/
- https://terasol.medium.com/react-native-top-7-differences-between-ios-and-android-app-development-fb0ee45db5d5
- https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/



React Native

Tim Kuhlmann, Lukas Obermann,
Julia Schmekel, Maja Weiblen