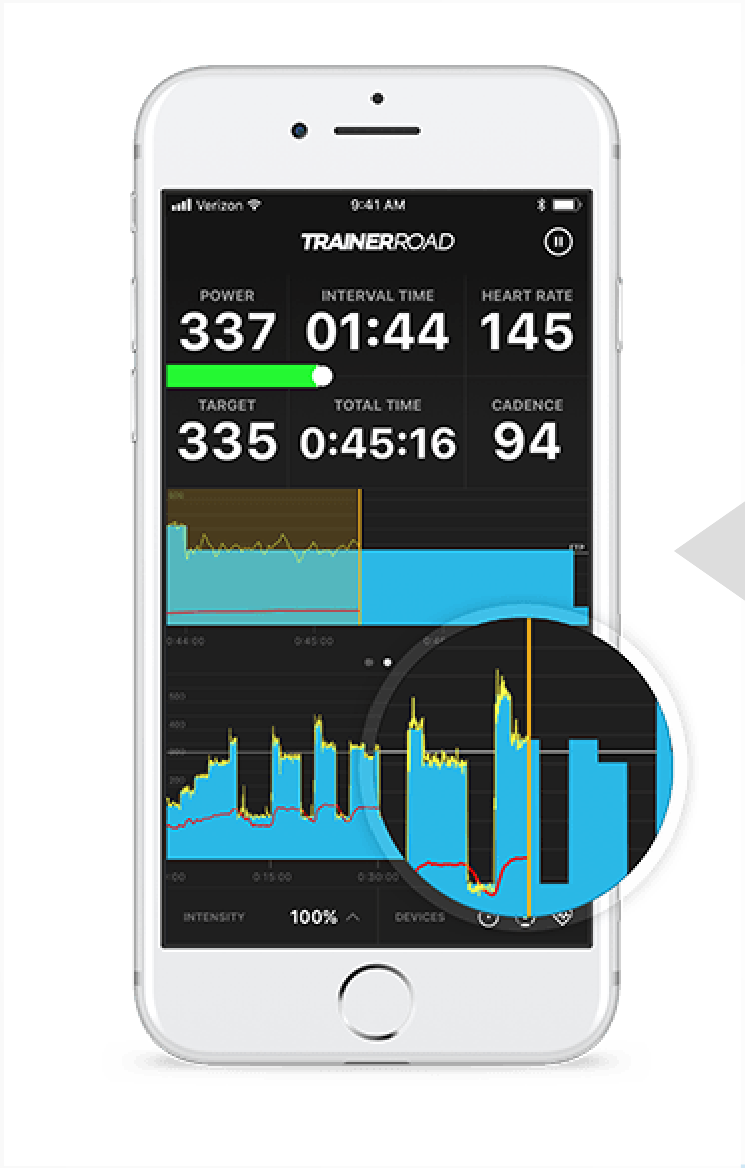
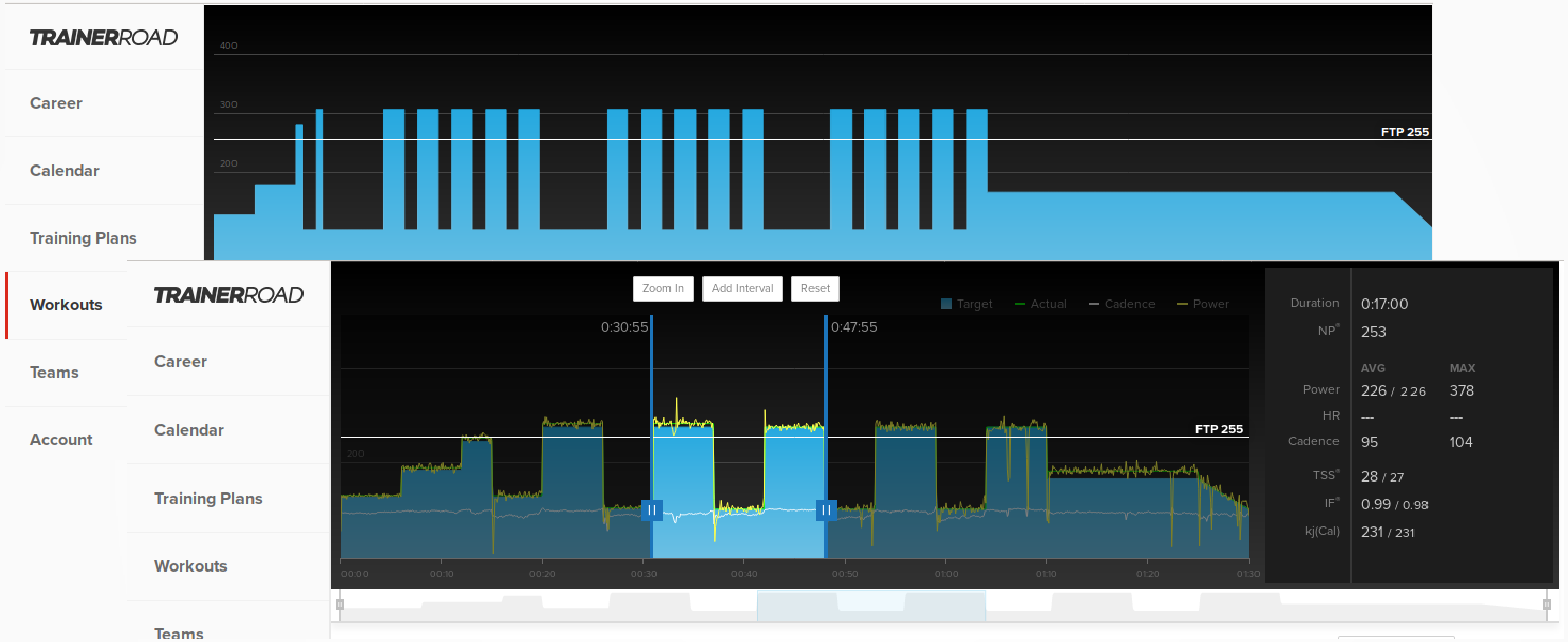
React / React Native

Programming Models for Emerging Platforms

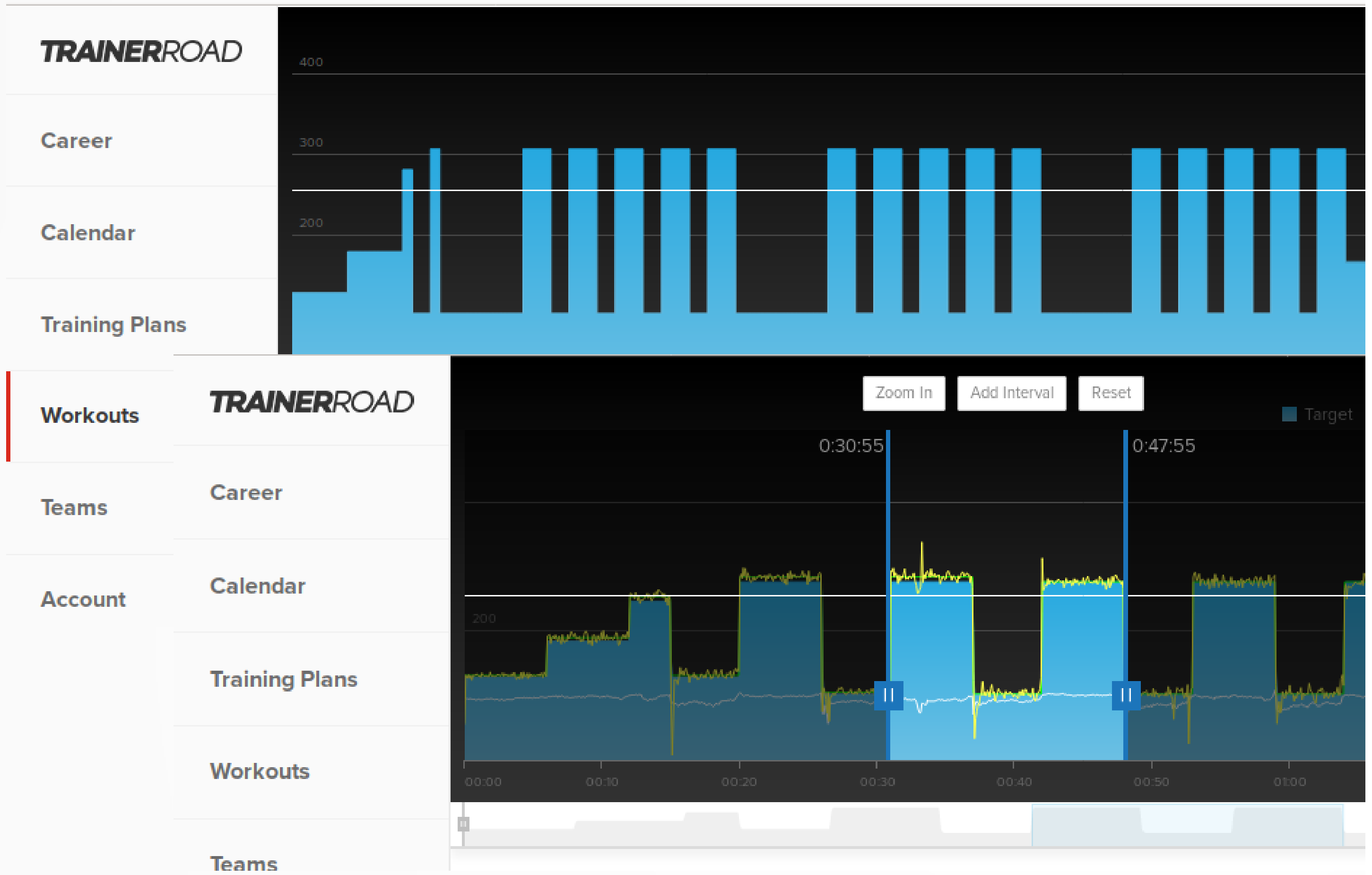


Trainerroad
Cycling Workout / Training App

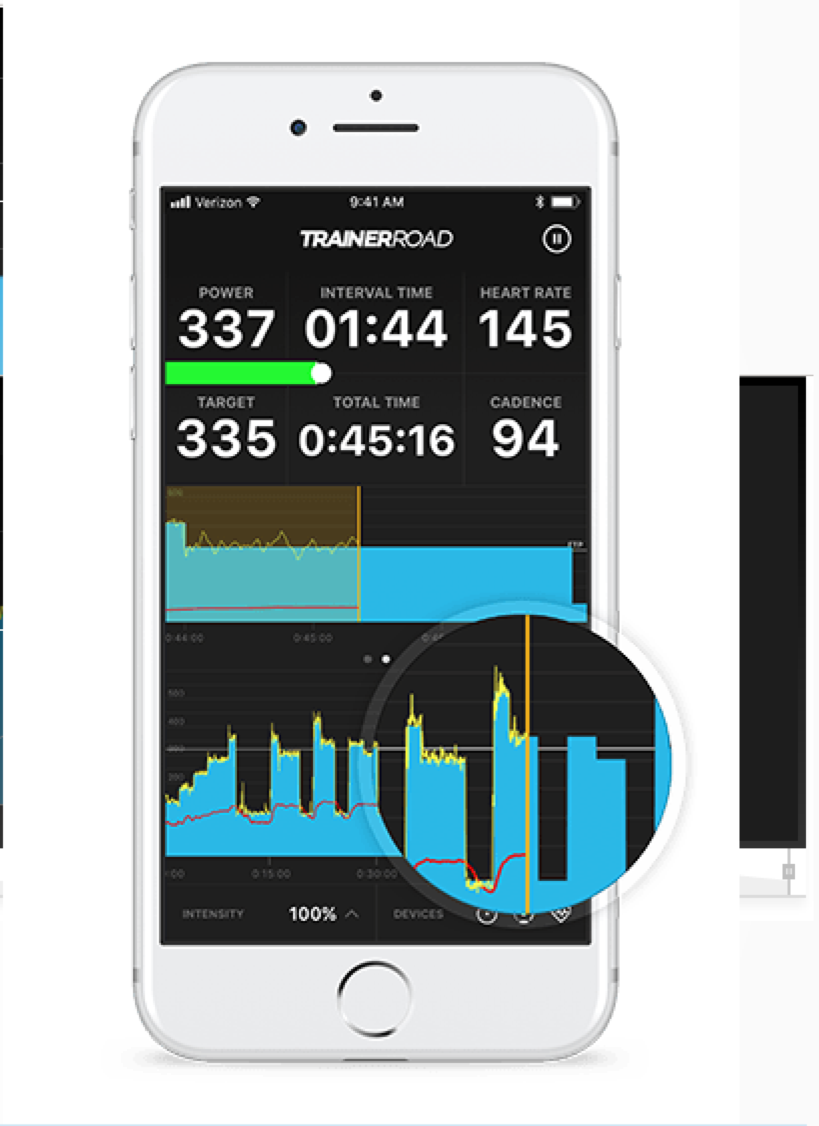


Duration	0:17:00	
NP*	253	
	AVG	MAX
Power	226 / 226	378
HR	---	---
Cadence	95	104
TSS*	28 / 27	
IF*	0.99 / 0.98	
kj(Cal)	231 / 231	

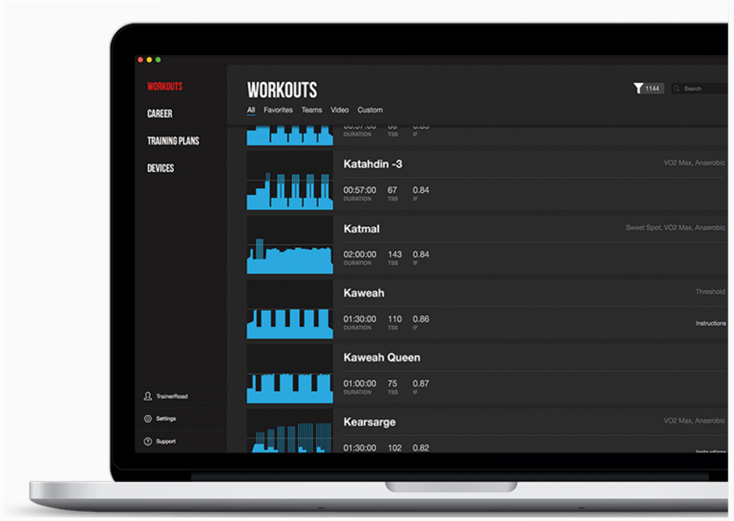
Web provides the “platform”



Web provides the "platform"



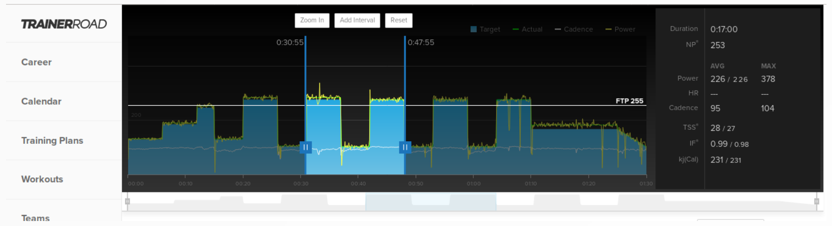
Phone interfaces to device



Challenge 1: Multiple Devices

Challenge 2: Half of platform web app anyways

Solution: Work with a
platform for web and native, if possible

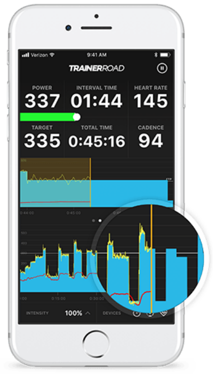


Duration 0:17:00
NP 253
AVG MAX
Power 226 / 226 378
HR — —
Cadence 95 104
TSS 28 / 27
IF 0.99 / 0.98
kJ(Cal) 231 / 231

Web Based GUI

Core UI framework in JS

Reuse framework with Native interface



POWER 337
INTERVAL TIME 01:44
HEART RATE 145
TARGET 335
TOTAL TIME 0:45:16
CADENCE 94

The Emerging Platform

- We care about **React + React Native**
- But we need to understand React first
- In isolation, React is a fun framework. If it interests you, I encourage you to pursue it further.

React Hello World

React Element

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

Container

Displays “Hello, word!” in the browser

ReactDOM is a link between React and the outer world

React Hello World

React Element

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

Container

Displays “Hello, word!” in the browser

ReactDOM is a link between React and the outer world

JSX

Embed JS in { }

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Pass JSX around like any other value

JSX syntax extension for javascript

Produces React elements from HTML-like syntax

JSX

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

```
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

JSX compiles down to React Elements for ease of use

React Components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Components are reusable UI pieces

Build up React elements for rendering

React Components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Take properties (props) as input, return JSX

Write them as functions, use them as HTML-like

React Components

- React / JS has a nice side effect of live coding
- https://codepen.io/acanino1/pen/bJKPRa?editors=0010

Exercise

- https://codepen.io/acanino1/pen/BEPmwd
- Make the following changes to Person list
  - 1. Create a Person component that encapsulates the <li> for a single person, add **age**
  - 2. Refactor people array into a array of JSON objects that represents people
  - 3. Refactor PersonList to create <Person/> for each person in the JSON object
  - 4. Create an a Pet component which will have a **name** and a **kind**
  - 5. Refactor JSON to include an array of pets for each person
  - 6. Refactor Person to render a sublist for each pet, per person

React Components

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  } }
```

Create a class component by extending React.Component

render() method drives the rendering

React Components

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  } }
```

Done mostly for **state**

React Components

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
    this.timerID = setInterval(() => this.tick(), 1000);
  }

  tick() { this.setState({ date: new Date() }) };

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  } }
```

Fired every 1s,
triggers re-render

Update state with **setState**

React State

- Props and State drive rendering of the app
- Changes to props and state cause component to get re-rendered, and has implication performances
- As such, state updates have semantics optimized for UI
  - 1. Updates may be asynchronous
  - 2. Independent updates are merged
  - 3. State local to component (top-down flow)

React State (Async Updates)

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
});}
```

Not guaranteed both are “current”

```
// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

Function will receive previous state, plus a snapshot of props

Note on Syntax

```
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

```
this.setState(function(state, props) {
  return {
    counter: state.counter + props.increment
  };
});
```

React State (Independent Updates)

```
fetchPosts().then(response => {
  this.setState({
    posts: response.posts
  });
});

fetchComments().then(response => {
  this.setState({
    comments: response.comments
  });
});
```

Updates merged back into state, but do not affect one another

Could be any of...
- {old.posts,old.comments}
- {old.posts,response.comments}
- {response.posts,old.comments}
- {response.posts,response.comments}

If state has independent variables, update independently

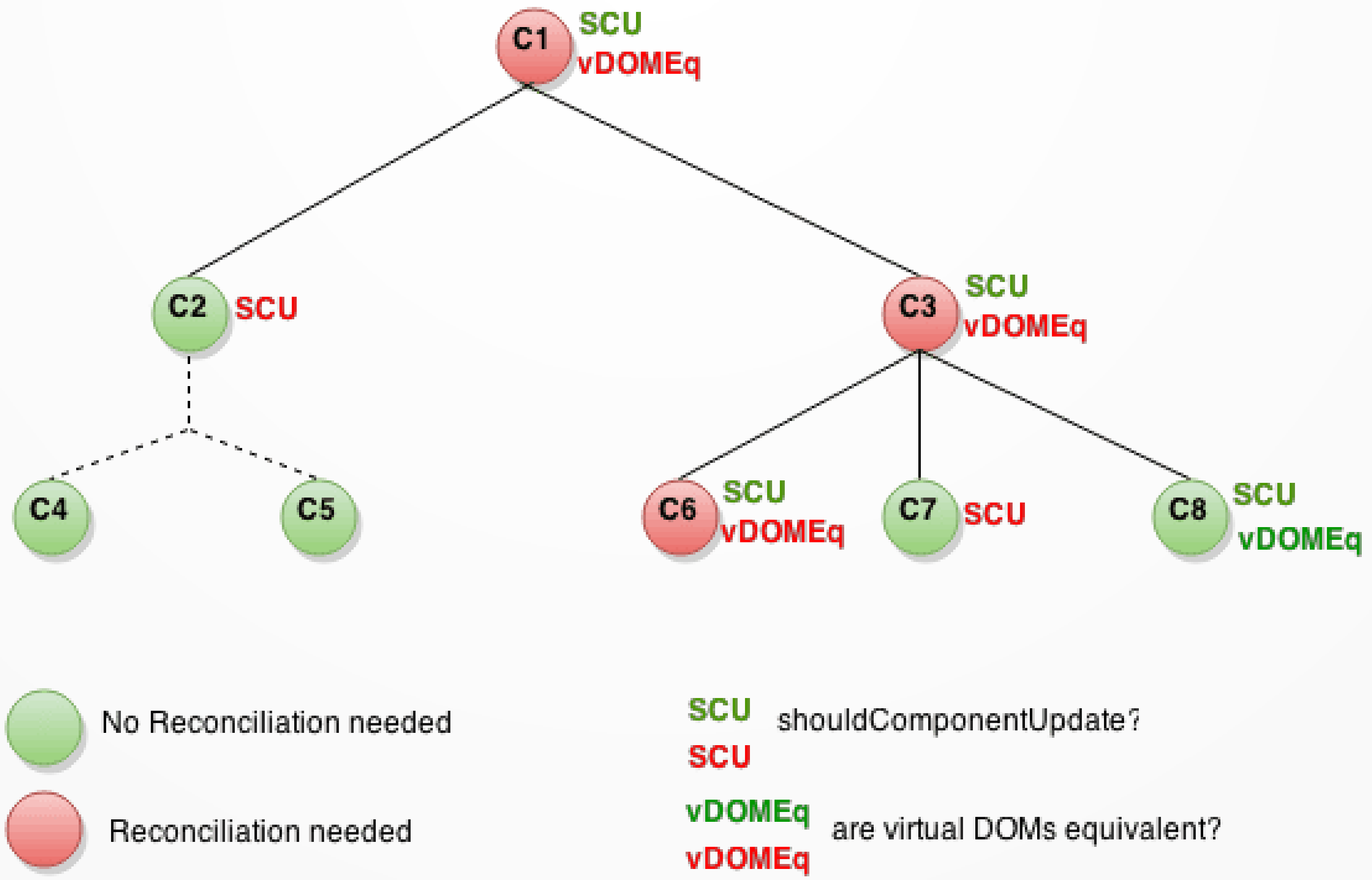
React State

- https://codepen.io/acanino1/pen/bJKPRa?editors=0010

Exercise

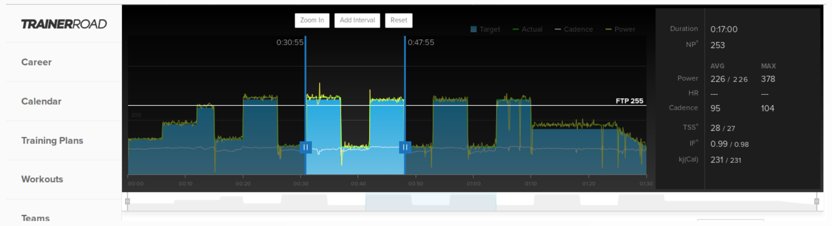
- https://codepen.io/acanino1/pen/VNBrze
- Try and complete the todo app
  - State is held in TodoApp
  - Create a TodoList, similar to lists demonstrated
    - Use JSON to build an item, which consists of text, and a date
    - Render both
  - Update **items** in handleSubmit
    - Date.now()
    - items.concat(...)

Why state matters?

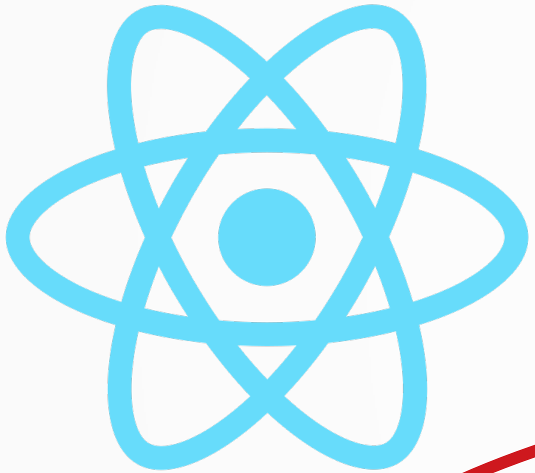


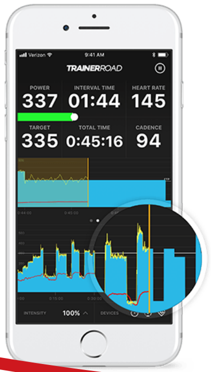
Some more fun

- https://codepen.io/acanino1/pen/gydYbq



Duration 0:17:00
NP 253
Power 226 / 226 378
HR — —
Cadence 95 104
TSS 28 / 27
IF 0.99 / 0.98
kJ(Cal) 231 / 231





Web Based GUI

Core UI framework in JS

Reuse framework with Native interface

Facebook

Facebook Ads Manager

Facebook Analytics

Instagram

F8

Bloomberg

Pinterest

Skype

React Hello World Native

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center",
                     alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

React Hello World Native

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Text>Hello {this.props.name}!</Text>
      </View>
    );
  }
}

export default class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', top: 50}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}
```

Components and props still apply

React Hello World Native

```
class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = { isShowingText: true };

    // Toggle the state every second
    setInterval(() => (
      this.setState(previousState => (
        { isShowingText: !previousState.isShowingText }
      ))
    ), 1000);
  }

  render() {
    if (!this.state.isShowingText) {
      return null;
    }

    return (
      <Text>{this.props.text}</Text>
    );
  }
}
```

As does state...

Style

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center",
                     alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

Import attribute for defining how application and components look.

Other Topics

- Style, layout, user experience, touch input, view types etc., all invaluable knowledge for building apps
  - However, very little depth involved. That is not to say these topics are “easy”, but that teaching each involves little more than showing how to use them.
- We will focus this lecture on some more interesting points
  - Networking (Asynchronous Events)
  - Native Boundary

Asynchronous Events

- In Javascript, and in React and React Native as a result, once something becomes **asynchronous**, it is **asynchronous** forever (no “unwrapping” and waiting)
- As such, you have to think and program asynchronously

Networking (Fetch API)

- JS interface for accessing pieces of the HTTP pipeline
- Useful for interfacing with JSON REST API
- Built on **Promises**

Promises

- Attach a series of callbacks to an asynchronous event
  - If the event has not completed, callbacks will get called upon completion
  - If the event has been completed, newely attached callbacks execute immediately
  - In this way, *promised* to always execute attached logic

Promises

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Old callback style. Tedious.

Promises

```
doSomething()
.then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log(
  'Got the final result: ' + finalResult);
})
.catch(failureCallback)
```

```
doSomething()
.then(result => doSomethingElse(result))
.then(newResult => doThirdThing(newResult))
.then(finalResult => {
  console.log(
  'Got the final result: ' + finalResult);
})
.catch(failureCallback)
```

Output of previous then input of next then

Promises + Chaining

Promises

```
new Promise((resolve, reject) => {
  console.log('Initial');
  resolve();
})
.then(() => {
  throw new Error('Something failed');
  console.log('Do this');
})
.catch(() => {
  console.log('Do that');
})
.then(() => {
  console.log('Do this, no matter what happened before');
});
```

What would this print?

Promises

```
new Promise((resolve, reject) => {
  console.log('Initial');
  resolve();
})
.then(() => {
  throw new Error('Something failed');
  console.log('Do this');
})
.catch(() => {
  console.log('Do that');
})
.then(() => {
  console.log('Do this, no matter what happened before');
});
```

What would this print?

```
Initial
Do that
Do this, no matter what happened before
```

Fetch API

```
fetch('https://facebook.github.io/react-native/movies.json')
.then((response) => response.json())
.then((responseJson) => {
  return responseJson.movies;
})
.catch((error) => {
  console.error(error);
});
```

```
{
  "title": "The Basics - Networking",
  "movies": [
    {
      "id": "1",
      "title": "Star Wars",
      "releaseYear": "1977"
    },
    {
      "id": "2",
      "title": "Back to the Future",
      "releaseYear": "1985"
    },
  ]
}
```

What does the chain do?

Make a ‘GET’ request
at supplied URL

Fetch API

```
fetch('https://facebook.github.io/react-native/movies.json')
.then((response) => response.json())
.then((responseJson) => {
  return responseJson.movies;
})
.catch((error) => {
  console.error(error);
});
```

JSON string into JS Objects

Access “movies” field

```
{
  "title": "The Basics - Networking",
  "movies": [
    {
      "id": "1",
      "title": "Star Wars",
      "releaseYear": "1977"
    },
    {
      "id": "2",
      "title": "Back to the Future",
      "releaseYear": "1985"
    },
  ]
}
```

Make a ‘GET’ request
at supplied URL

Promises

- What drawbacks do we face when using promises?

Promises

- What drawbacks do we face when using promises?
  - More complex logic gets tricky to encode using **then** style

Promises

```
fetch('https://facebook.github.io/react-native/movies.json')
.then((response) => response.json())
.then((responseJson) => {
  if (responseJson.forwardUrl) {
     return fetch(responseJson.forwardUrl)
       .then((response) => response.json())
       .then((responseJson) => {
         return responseJson.movies;
       });
  } else {
    return responseJson.movies;
  }
})
.catch((error) => {
  console.error(error);
});
```

Conditional processes gets ugly fast

We want to “think” synchronously

Networking (Async / Await)

- ES8 (ECMA Script 2017) adds two new keywords, **async** and **await**
- Define asynchronous function but use a syntax that looks like synchronous processing

async / await

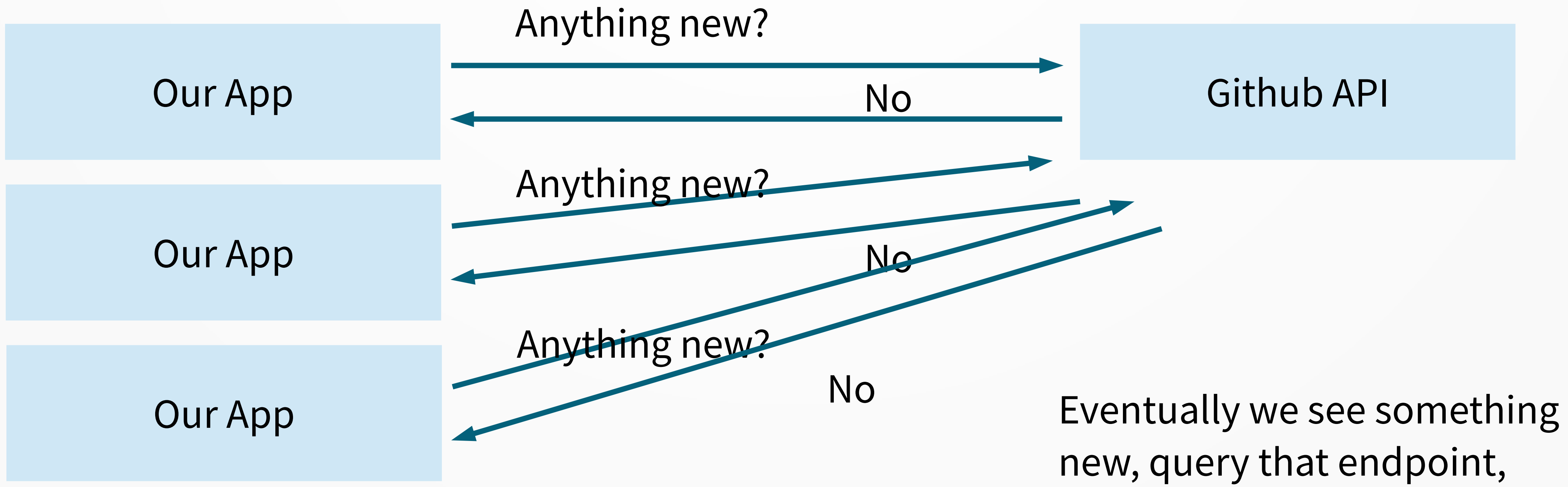
```
async function getMovies() {
  try {
    let response = await fetch('https://facebook.github.io/react-native/movies.json')
    let responseJson = await response.json();
    if (responseJson.forwardUrl) {
      let response2 = await fetch(responseJson.forwardUrl);
      let responseJson2 = await response2.json();
      return responseJson.movies;
    }
    return responseJson.movies;
  } catch (e) {
    console.log(e);
  }
}
```

async implicitly builds a promise from function

Inside async block, await waits for result of promise

Networking (HTTP REST)

- Given an endpoint that represents an API, we must continuously poll it with GET, POST, etc...



Eventually we see something new, query that endpoint, then repeat

Networking (Websockets)

- Websockets allow for full duplex communication over HTTP between a browser and server
  - React Native provides a Websocket API
- The *server* can send messages to the browser

Websockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
  // connection opened
  ws.send('something'); // send a message
};

ws.onmessage = (e) => {
  // a message was received
  console.log(e.data);
};
```

React Native API for Websockets is simple

Inside async block, await waits for result of promise

Networking (Websockets)

- Let’s actually build something

Other Topics

- Style, layout, user experience, touch input, view types etc., all invaluable knowledge for building apps
  - However, very little depth involved. That is not to say these topics are “easy”, but that teaching each involves little more than showing how to use them.
- We will focus this lecture on some more interesting points
  - Networking (Asynchronous Events)
  - **Native Boundary**

React Native (Boundary)

- How is it that React Native can work with *native* components?
  - From a high level, it seems like elegant magic
  - In reality, an interface wraps much of the calls to underlying modules, i.e., lots of tedious, hand written code

Native Modules

- A binding between React framework and Native code

Geolocation

```
 Geolocation.getCurrentPosition(
   position => {
     const initialPosition = JSON.stringify(position);
     this.setState({initialPosition});
   },
   error => Alert.alert('Error', JSON.stringify(error)),
   {enableHighAccuracy: true,
    timeout: 20000,
    maximumAge: 1000},
);
```

on success

on error

options

Return a promise for
current location of device

Geolocation (Android Java)

```
@ReactModule(name = GeolocationModule.NAME)
public class GeolocationModule extends ReactContextBaseJavaModule {

  public static final String NAME = "RNCGeolocation";
  private static final float RCT_DEFAULT_LOCATION_ACCURACY = 100;

  private final LocationListener mLocationListener = new LocationListener() {
    // implement LoctionListener for React interface
  };

  @ReactMethod
  public void getCurrentPosition(
      final ReadableMap options,
      final Callback success,
      final Callback error) {
    // ...
  }
}
```

Annotations provide hints to React

Native to bridge between JS and Java

Geolocation (Android JS)

```
const Geolocation = {
  getCurrentPosition: async function(
    geo_success: Function,
    geo_error?: Function,
    geo_options?: GeoOptions,
  ) {
    invariant(
      typeof geo_success === 'function',
      'Must provide a valid geo_success callback.',
    );

    // Permission checks/requests are done on the native side
    RNCGeolocation.getCurrentPosition(
      geo_options || {},
      geo_success,
      geo_error || logError,
    );
  },
  // ...
}
```

```
public static final String NAME = "RNCGeolocation";
```

Bound Native Module

Geolocation Code

- Dig through the full implementation

Integrating with Existing Apps

- React Native apps do not need to be built entirely in React
- Part of React Native flexibility is that we can *gradually* adopt React Native in an existing application

React Native into Android

```
public class MyReactActivity extends Activity implements
DefaultHardwareBackBtnHandler {
    private ReactRootView mReactRootView;
    private ReactInstanceManager mReactInstanceManager;
    // …
}
```

ReactRootView and
ReactInstanceManager provide way to
use React Native inside Android Activity

React Native into Android

```
public class MyReactActivity extends Activity implements
DefaultHardwareBackBtnHandler {
  private ReactRootView mReactRootView;
  private ReactInstanceManager mReactInstanceManager;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mReactRootView = new ReactRootView(this);
    mReactInstanceManager = ReactInstanceManager.builder()
            .setApplication(getApplication())
            .setCurrentActivity(this)
            .setBundleAssetName("index.android.bundle")
            .setJSMainModulePath("index")
            .addPackage(new MainReactPackage())
            .setUseDeveloperSupport(BuildConfig.DEBUG)
            .setInitialLifecycleState(LifecycleState.RESUMED)
            .build();
    mReactRootView.startReactApplication(
      mReactInstanceManager, "MyReactNativeApp", null);

    setContentView(mReactRootView);
}
```

What is going on here?

React Native into Android

```
public class MyReactActivity extends Activity implements
DefaultHardwareBackBtnHandler {
  private ReactRootView mReactRootView;
  private ReactInstanceManager mReactInstanceManager;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mReactRootView = new ReactRootView(this);
    mReactInstanceManager = ReactInstanceManager.builder()
            .setApplication(getApplication())
            .setCurrentActivity(this)
            .setBundleAssetName("index.android.bundle")
            .setJSMainModulePath("index")
            .addPackage(new MainReactPackage())
            .setUseDeveloperSupport(BuildConfig.DEBUG)
            .setInitialLifecycleState(LifecycleState.RESUMED)
            .build();
    mReactRootView.startReactApplication(
      mReactInstanceManager, "MyReactNativeApp", null);

    setContentView(mReactRootView);
}
```

Firing up a JS VM inside Activity

React Native into Android

```
public class MyReactActivity extends Activity implements
DefaultHardwareBackBtnHandler {
  private ReactRootView mReactRootView;
  private ReactInstanceManager mReactInstanceManager;

  // …

  @Override
   protected void onPause() {
    super.onPause();
    if (mReactInstanceManager != null) {
        mReactInstanceManager.onHostPause(this);
    }
  }

  @Override
  protected void onResume() {
    super.onResume();
    if (mReactInstanceManager != null) {
      mReactInstanceManager.onHostResume(this, this);
    }
  }
}
```

The rest is just event forwarding

Acknowledgments

- https://facebook.github.io/react-native/
- https://developer.mozilla.org/
- https://medium.com/@martin.sikora/node-js-websocket-simple-chat-tutorial-2def3a841b61
- https://hackernoon.com/6-reasons-why-javascripts-async-await-blows-promises-away-tutorial-c7ec10518dd9