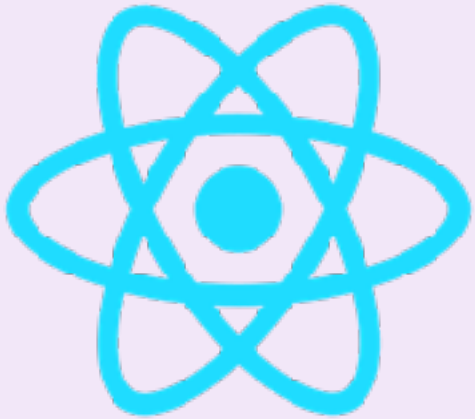
# React Native



Andrew Smith
OptumRx
December 7, 2020

# Objectives

- Set up a new React Native project w/ expo
- Writing React / Redux to interactive UIs
- Work with layouts, navigation, and others unique to mobile
- Build a mobile app with confidence
- Have fun 🙃

# Out of scope

Advanced React Native

Intermediate / Advanced React

Intermediate / Advanced Redux

Intermediate / Advanced JavaScript

iOS / Android native coding

# Daily Schedule

(times in EST)

| | |
|---|---|
| 9:00 - 9:15 | Check-ins |
| 9:15 - 10:00 | Labs recap |
| 10:00 - 10:15 | Break |
| 10:15 - 11:15 | Lecture |
| 11:15 - 11:30 | Break |
| 11:30 - 12:15 | Lecture / Lab |
| 12:15 - 12:30 | Closing |

# Course structure / takeaways

Repo:

https://github.com/andrewsouthpaw/react-redux-labs

Slides: Under /slides

# Class logistics

What we want

...what we've got

# Class logistics

Eliminate distractions

Enable video

# No really, please enable your video.

# Class logistics


Mute unless speaking

WebEx

# Class logistics

Lecture

Lab

# Class logistics

Be curious

Fail often

# Class logistics

Give feedback

# Class logistics

Ask questions

Get rewards

# Class logistics

Pairing

Screen sharing

# It me!

Contact info:

andrew.southpaw@gmail.com

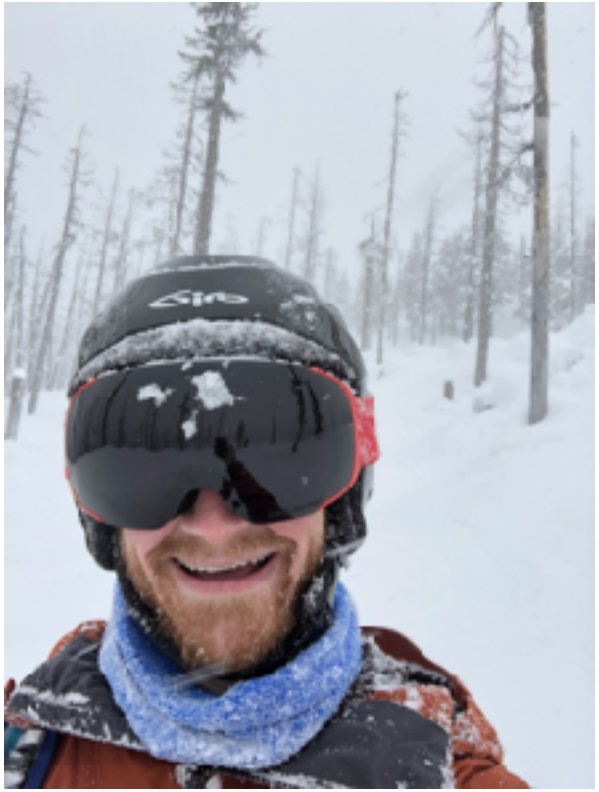
@andrewsouthpaw (Twitter, GitHub, etc.)

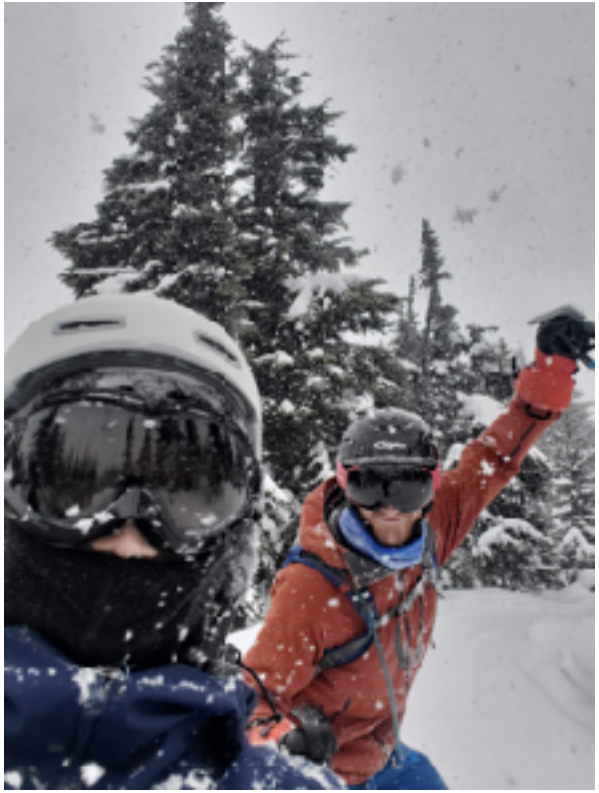
www.andrewsouthpaw.com

# I dance

www.andrewsmithdance.com

# I play in the snow

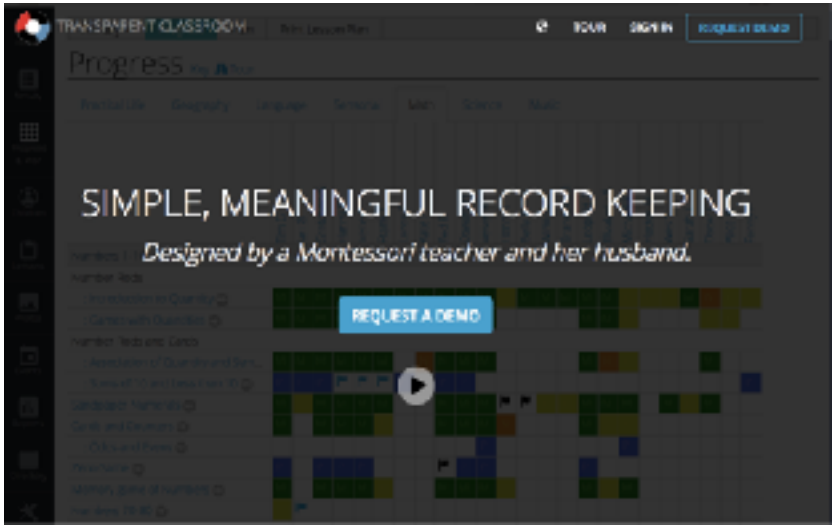


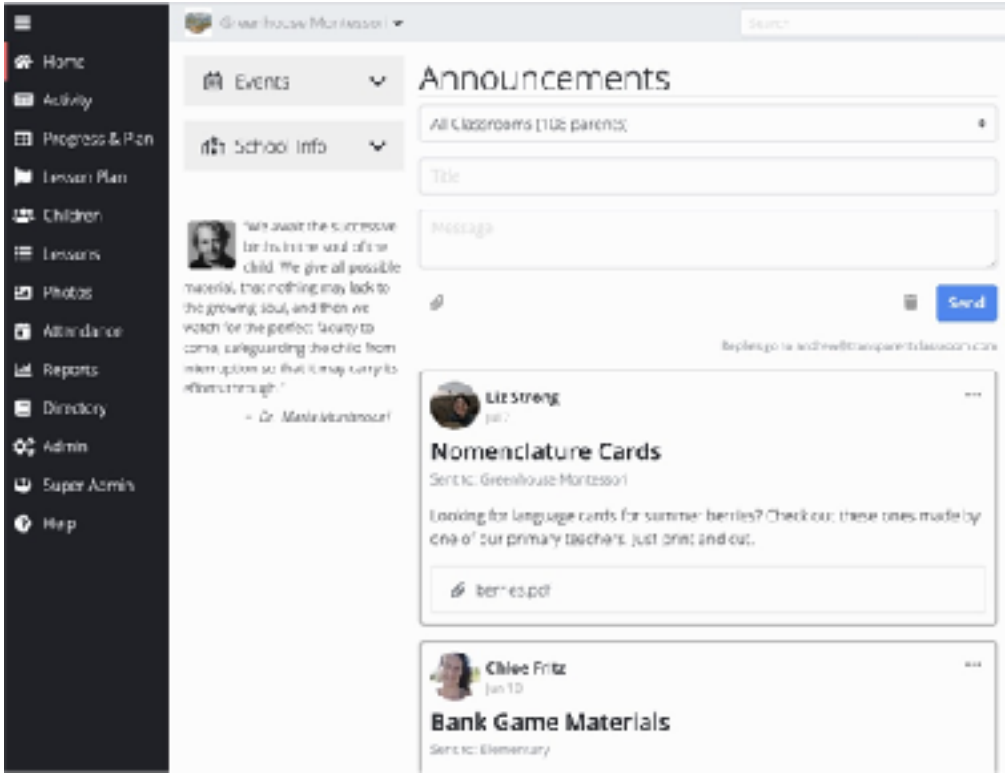


# I bicycle

# I am a Tea ~~Snob~~ Lover 😍

# I lead development



# I lead development

# Get to know you

1. Name
2. Pronouns (she/her, they/them, etc.)
3. Familiarity with React / Redux / React Native
4. Favorite hobby
5. What's something you did in the past 4 months you're proud of?
6. What brings you here today?
7. How are you feeling? (1 - 7)

# What is React Native?

# You are CEO of a small business

Then customers say ...

We love your web app, but we'd have been here earlier if you were in the App Store.

How do I get into the App Store?

Hire an iOS dev who knows Swift

Then customers say ...


Hire an Android dev who knows Java

But you can't afford three developers!

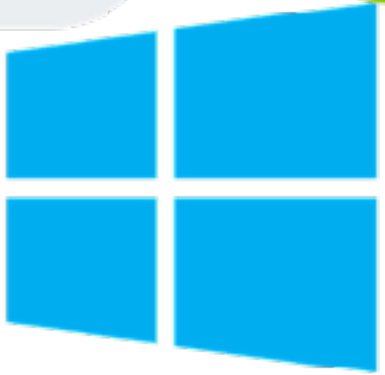
# What is React Native?

"A framework and set of tooling that allows us to write applications with a single set of JavaScript code that will run on devices with different OSs, controlling both the presentation and behavior."

-- Rap Payne

# Create apps for ...





# The app is completely native, indistinguishable from one written in Swift, Objective-C, Java, or Kotlin


- Not in a WebView like Cordova or Ionic
- Not in a host
- Not a hybrid/partially native app like Xamarin.

You don't need ...

Java/Kotlin

Objective-C/Swift

macOS*

Xcode*

You only need the React Native compiler

* But to test iOS, you do need a Mac

# When your app is ready, create an ipa or apk file which can be sent to the Google Play store or the App Store

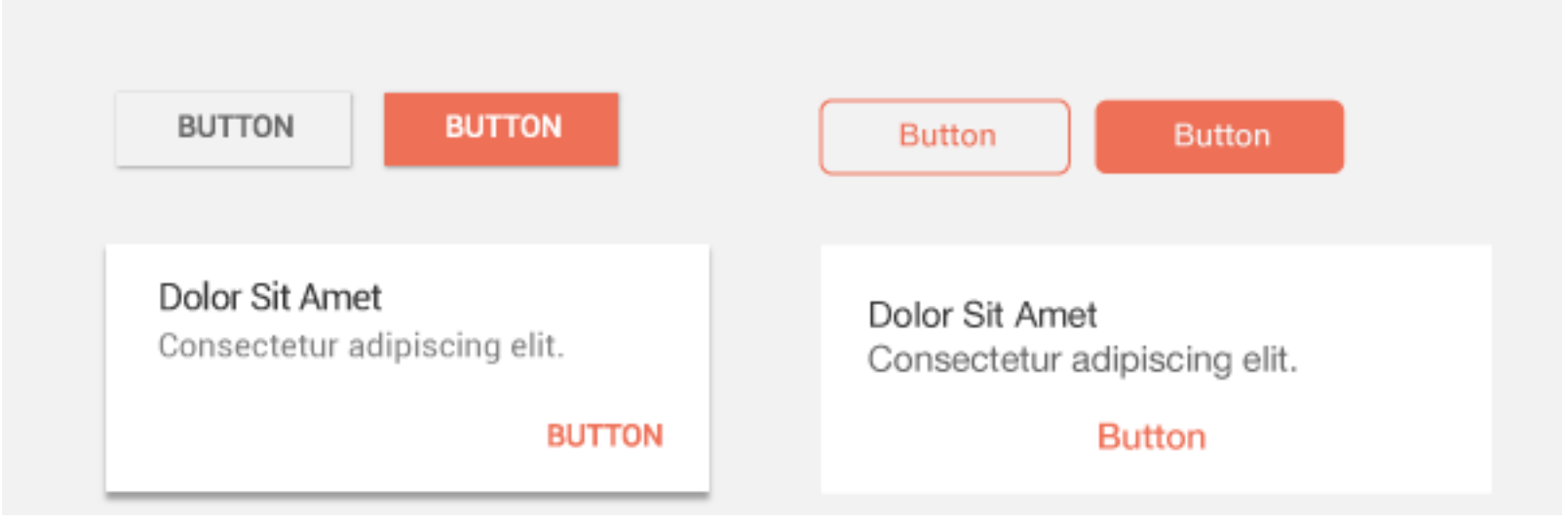


**Of course, you'll still need to be registered to submit**

# Elements render natively

- React Native <Button>s render as iOS buttons on iOS and Android buttons on Android.
- Same with every element where it is possible

(... in some places it is impossible)



# Unfortunately we can't share anything with a web application

No HTML

No CSS

**But you can share pure JavaScript libraries if you're careful.**

# And you can use most of the innumerable libraries in your code

As long as they don't depend on HTML, <canvas>, or SVG

- axios
- date-fns
- frisbee
- immutable
- ramda
- redux
- moment
- etc. etc.


1
E
Electron
2
Rn
React Native
3
Na
Native Apps
4
Cv
Cordova
5
Io
Ionic
6
Ns
NativeScript
8.9%
15.4%
53.2%
19.6%
22.2%
53.2%
18.7%
28%
30
28
10
13.8%
23.4%
39.7%
Never heard of it
Heard of it, not interested
Heard of it, would like to learn
Used it, would not use again
Used it, would use again
Electron
React Native
Nativ

# Should we use React Native?

## Maybe if ...

- You need iOS and Android apps in the stores
- Your team is already strong in functional JavaScript
- Or they're strong in React
- You value speed over quality

## Probably not if ...

- The app is really complex -- like a game
- You already have iOS & Android devs
- The apps need to be different on the different platforms

# But be careful. It isn't perfect

If you want the experience to be different on both platforms, you must create different components that do the same things but look slightly different.

This is a maintenance problem and requires that you have people that know both iOS and Android.

# Even if you want them to be the same ...

- There's a huge variety of Android devices and very few iOS devices. So you may end up doing checks if iOS => do one thing; else => do other things. The more of these you have, the worse RN fits your ideal solution.
- apk size is much bigger with RN (About 10Mb! as of 7/2018)
- RN lags behind iOS features and Android features. ie. When iOS adds a new feature like SafeAreaNavigation, it takes a while before it shows up in RN.
- When a bug occurs, it is really hard to tell if it is in iOS or in Android or in RN.

# ... and it is still new tech

- The framework rules seems inconsistent
- The tooling seems unreliable

- APIs are changing quickly
- Tooling is changing quickly
- Libraries are changing quickly

- You have iOS things that change (API, OS, etc) and Android things that change and you now have RN things that change and probably JavaScript libraries that it depends on that change. That's a lot of breaking changes and things that can be deprecated and require maintenance in one code base!

# What's React Native code look like?

# Categories of RN components

| Category | Some components |
| --- | --- |
| Layout | Modal, View, SafeAreaView, ScrollView, RefreshControl, KeyboardAvoidingView, StatusBar, WebView |
| Single-value | Text, TextInput, Slider, Switch, Image |
| List | Picker, FlatList, SectionList |
| Touchable | Button, TouchableHighlight, TouchableNativeFeedback, TouchableOpacity, TouchableWithoutFeedback |
| Others | ActivityIndicator, Platform-specific components |

# React Native can resemble HTML

| HTML | React Native |
| --- | --- |
|  | <View> |
| <p> | <Text> |
| <img /> | <Image /> |
| <input type='text' /> | <TextInput /> |
| <input type="range" /> | <Slider /> |
| <button> | <Button /> |

**MyComponent.js**

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

class MyComponent extends React.Component {
 render() {
    return (
      <View>
        <Text>Hello World</Text>
      </View>
    );
  }
}

// example usage
<MyComponent />
```

# A class-based component

# A stateless functional component

**MyComponent.js**

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

const MyComponent = () => (
  <View>
    <Text>Hello World </Text>
  </View>
)

// example usage

<MyComponent />
```

**MyComponent.js**

```
class MyComponent extends React.Component {
  render() {
    return (
      <Notify message="Hello World" />
    );
  }
}
```

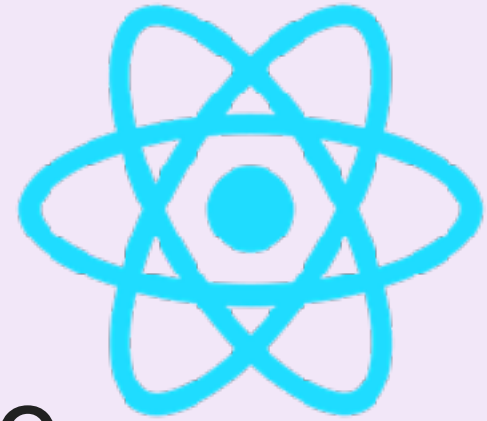
We use props and compose just like in React for the web

**Notify.js**

```
const Notify = props => <Text>{props.message}</Text>
```

# tl;dr

- React Native allows developers to use JavaScript and React to create cross-platform apps
- They can be deployed to the Apple App Store and the Google Play Store because they are 100% native.
- No plugins
- No WebViews
- No tricks



# The React Native Development Process

# tl;dr

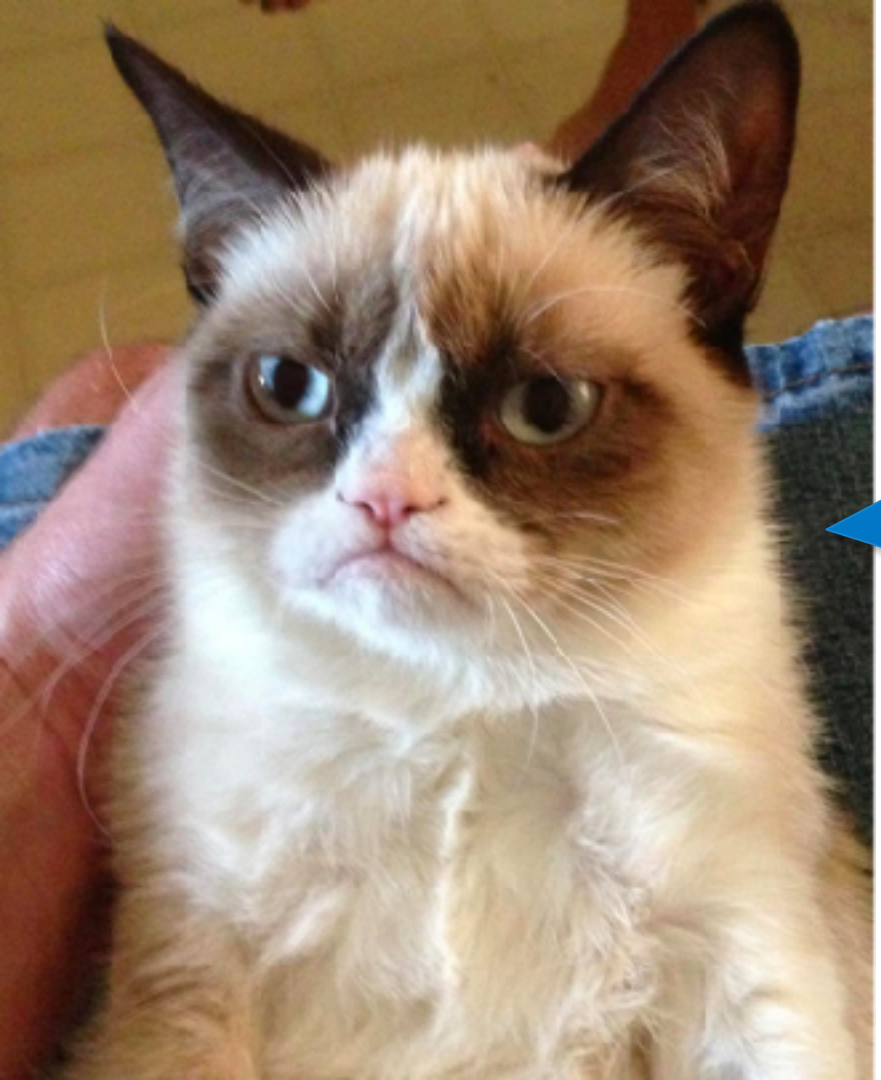
- Your app must run in a device during the development process
- Expo is an emulated environment. You can run it ...

1. on a physical device (with Expo installed) by scanning a QR code
2. in an iOS emulator after installing Xcode and the iOS emulator on your Mac
3. in an Android emulator after installing Android Studio

- ... And there are actually more options

# Three ways to start a RN project

1. Manually
2. react-native init
3. expo init



# Manually

Really?!? We're considering this? Wow.

# react-native-cli

```
npx react-native init cool-app
```

- Creates an initial project
- You then code and deploy to a real device or run in an emulator

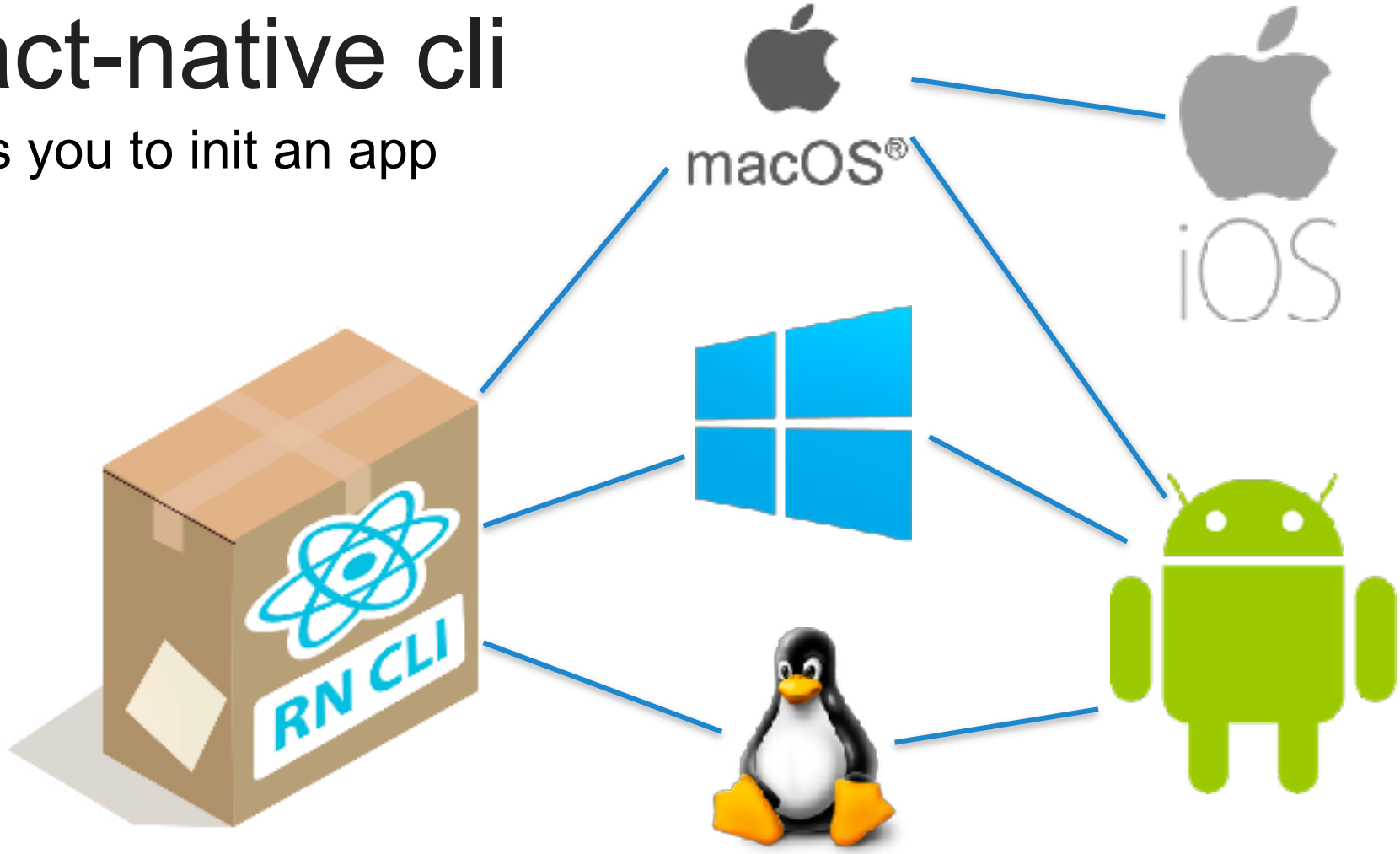
# expo init

- A command-line tool that scaffolds a basic project.

```
$ expo init cool-app
$ cd cool-app
$ expo start
```

# react-native cli

Allows you to init an app



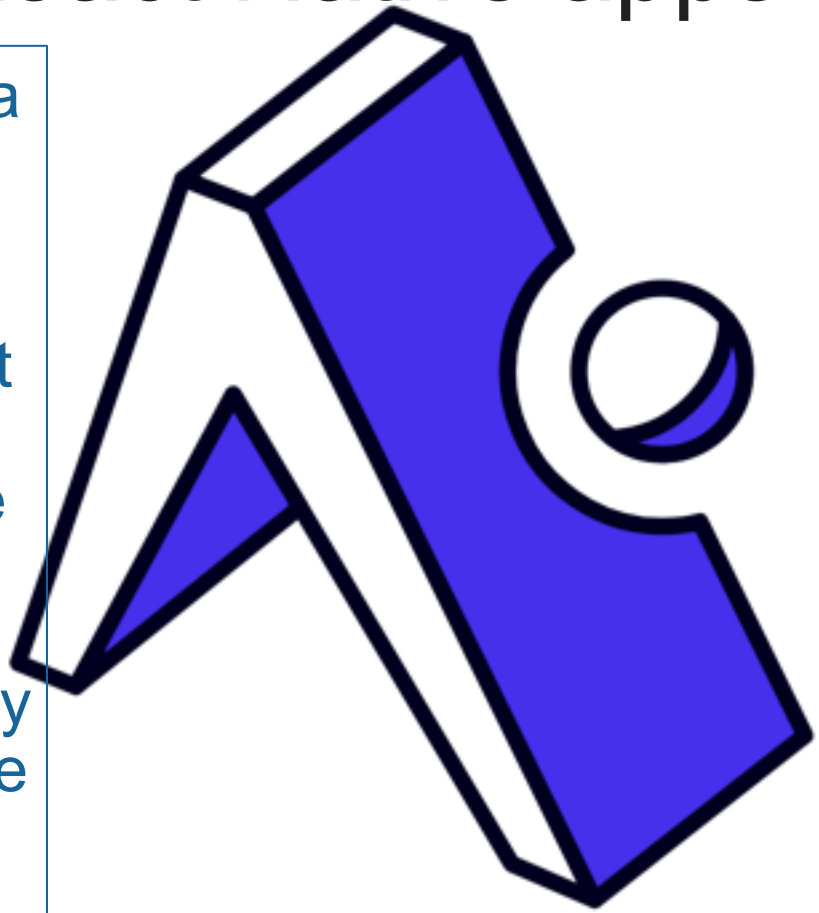
# Which to use?

| react-native init vs. expo | rni | expo |
|---|---|---|
| Can use native modules written in Java/Swift | ☑ | ☒ |
| Can code without Android Studio and XCode | ☒ | ☑ |
| Can develop for iOS on Windows | ☒ | ☑ |
| Can test without being tethered to a device via USB | ☒ | ☑ |
| JavaScript APIs provided/no extra installs needed | ☒ | ☑ |
| Setup and configuration time in minutes vs. hours | ☒ | ☑ |
| Easy to share the app during development | ☒ | ☑ |
| Officially recommended by the React Native team | ☒ | ☑ |

```
$ expo init cool-app
? Choose a template: expo-template-blank
🧶 Using Yarn to install packages. You can pass --npm to use npm instead.
✔ Downloaded and extracted project files.
✔ Installed JavaScript dependencies.
✅ Your project is ready!
To run your project, navigate to the directory and run one of the following yarn commands.
- cd cool-app
- yarn start # you can open iOS, Android, or web from here, or run them directly with the commands below.
- yarn android
- yarn ios
- yarn web
$
```

# Expo apps are React Native apps

... which contain the Expo SDK, a native-and-JS library that provides access to the device's system functionality (camera, contacts, local storage, etc). That means you don't need to use Xcode or Android Studio, or write any native code, and it also makes your pure-JS project very portable because it can run in any native environment containing the Expo SDK



# The five Expo tools

1. XDE: NOT an IDE. Helps to manage your projects and get it compiled on Expo's servers. Nothing you can't do in the Expo CLI.
2. Expo CLI: XDE's capability on a command line.
3. Expo Client: An app for Android and iOS. Runs your React Native project without installing. Allows devs to hot reload on a real device.
4. Expo Snack: https://snack.expo.io, a REPL for React Native apps.
5. Expo SDK: Holds JavaScript APIs that provide Native functionality not found in the base React Native package, accelerometer, camera, notifications, geolocation, etc. Comes baked in with every new project created with Expo.

# How do I run it?

```
$ npm run start
Starting project at /Users/rap/Desktop/cool-app
Expo DevTools is running at http://localhost:19002
Opening DevTools in the browser... (press shift-d to disable)
Starting Metro Bundler.

  exp://192.168.1.16:19000
```

```
  To run the app with live reloading, choose one of:
  • Scan the QR code above with the Expo app (Android) or the Camera app (iOS).
  • Press a for Android emulator, or i for iOS simulator, or w to run
    on web.
  • Press e to send a link to your phone with email.
  • Press s to sign in and enable more options.

 Expo  Press ? to show a list of all available commands.
Logs for your project will appear below. Press Ctrl+C to exit.
```

# To run it, go:

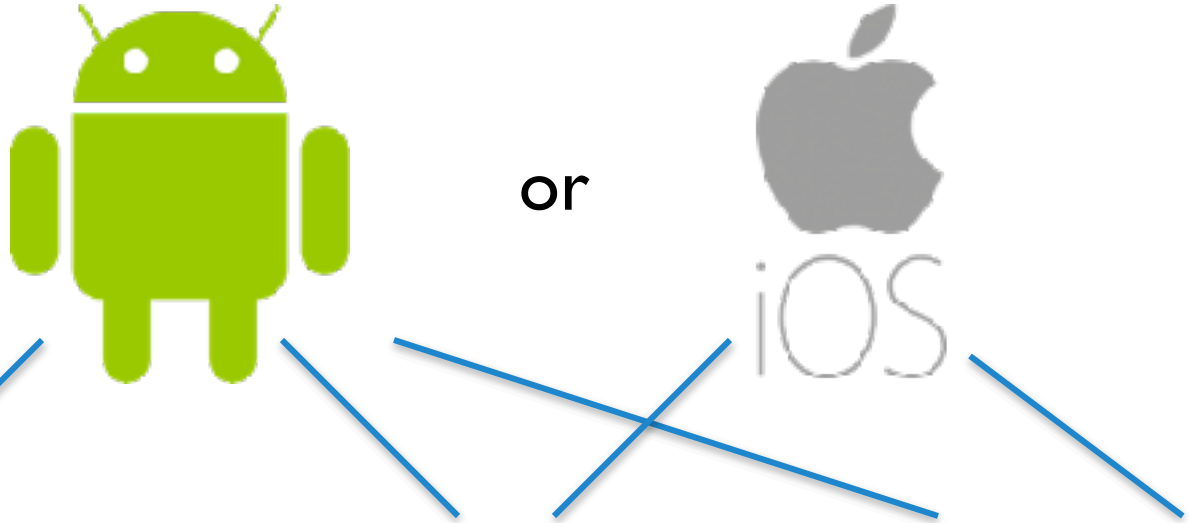
# `npm run start`

# npm run start

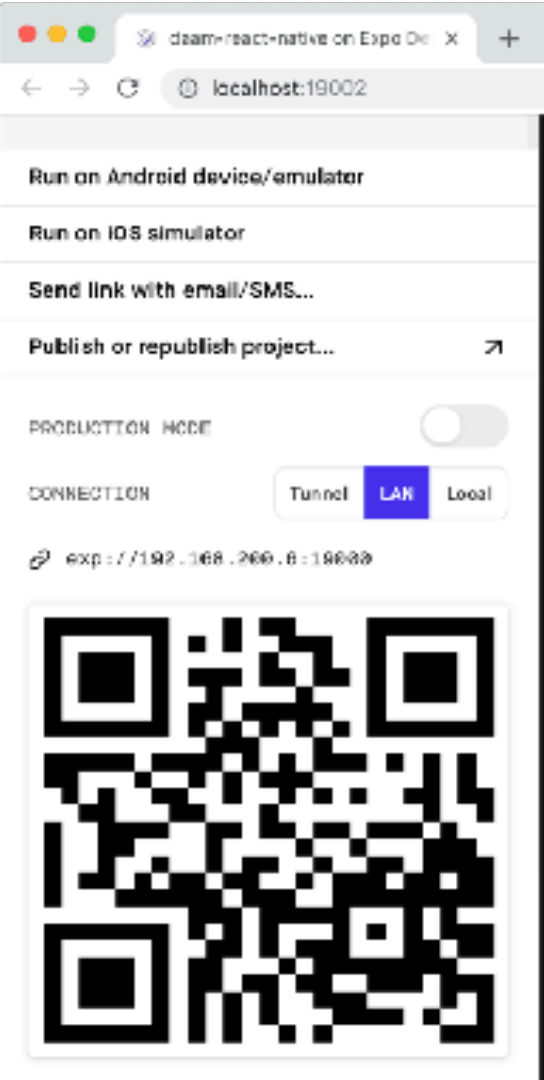
- Builds and bundles your application into memory
- This starts a node server which serves a static json config at a given URL.
- If the expo client browses to that config file, your app will run


```
{
  "sdkVersion": "27.0.0",
  "name": "cool-app",
  "version": "0.1.0",
  "xde": true,
  "developer": {
    "tool": "crna",
    "projectRoot": "/coolApp"
  },
  "packagerOpts": {
    "hostType": "tunnel",
    "lanType": "ip",
    "dev": true,
    "minify": false,
    "urlRandomness": null
  },
  "env": {},
  "bundleUrl":"http://172.16.2.232:19001/...",
  "debuggerHost": "172.16.2.232:19001",
  "mainModuleName": "./node_modules/...",
  "logUrl": "http://172.16.2.232:19000/logs",
  "id": "@anonymous/rn-..."
}
```

# But Expo can't run in a browser. It needs a device like ...




Tethered or Wireless or Emulated

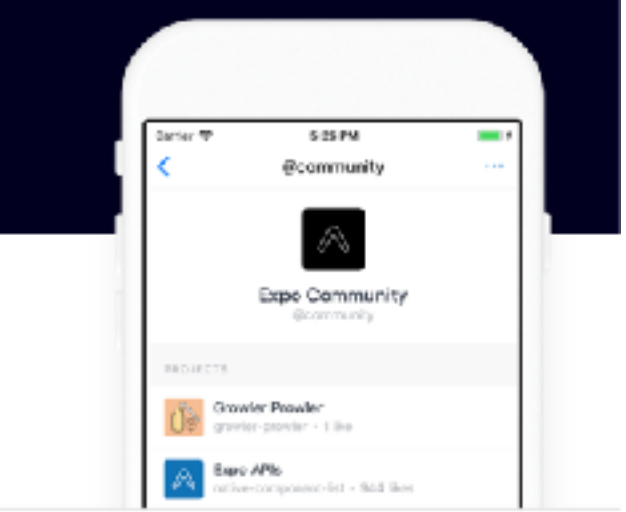



# So we have these options

- q Show a QR code that you scan with your physical device's camera
- a Run in an Android emulator or tethered device
- i Open an iOS emulator and run it in there

# 1. Run it on your non-tethered device

The "q" option




**Expo Client**
for Android & iOS

Run your projects before you deploy. Open projects by scanning QR codes. If you need to, Download IPA 2.8.1 or Download APK 2.8.0.

Star

iOS App  Android App

# Install the Expo client app on the device on which you want to test

- http://expo.io
- Apple App Store
- Google Play Store

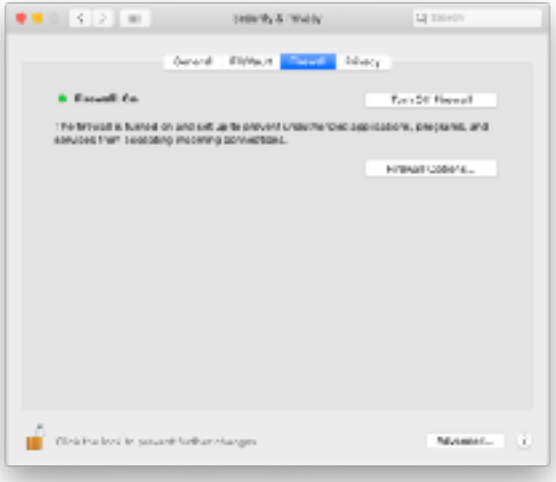
- Displays QR Code that you can scan on your device that has the expo app installed.
- The device must be on the same un-firewalled network as your development machine

# QR code on iOS

- As of 7/2019, the iOS version of the expo app doesn't have QR scanner. (Lame)
- But if you shoot it with your camera, the phone will prompt you to open the link in the expo app. (even cooler!)

# Troubleshooting

- Unable to connect. Timed out.
- B/c we're on a network whose firewall prevents this kind of thing. Weird ports may look like an attack. Try to connect to a more open network.
- Disabling the firewall worked for me.

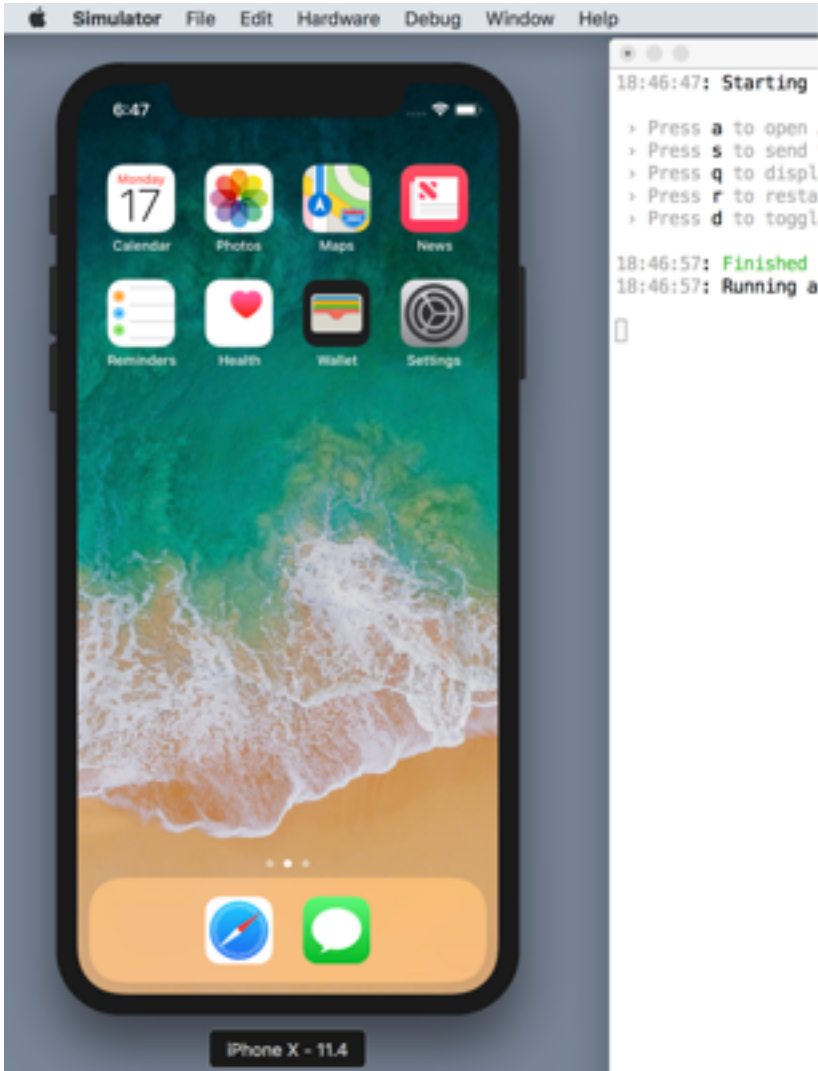


# 2. Run it in an iOS Emulator

The "i" option

# To use the iOS simulator you must ...

- Be on a Mac
- Have Xcode installed and configured
- Have iOS simulator installed and configured

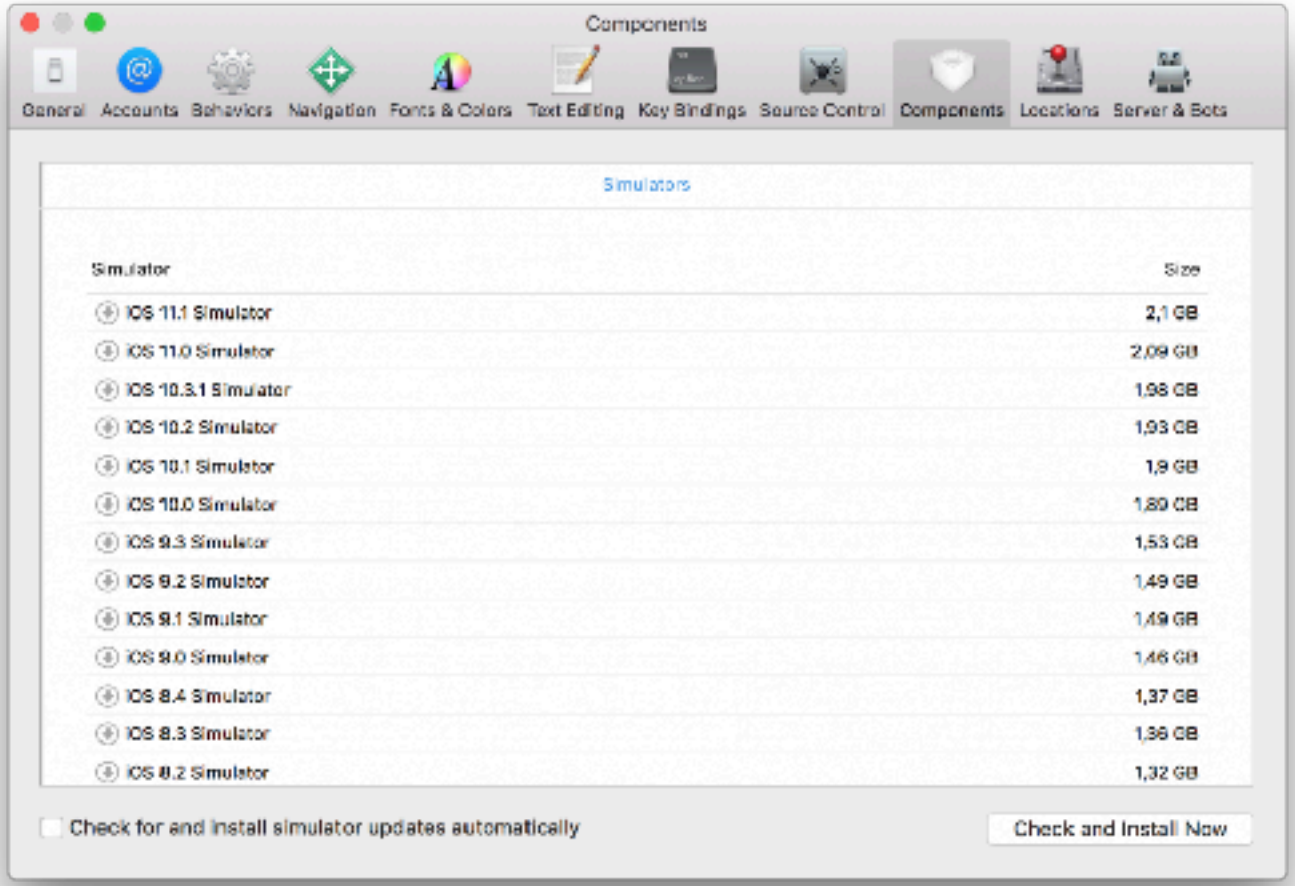


# Troubleshooting

- If the emulator says "This version of the Expo app is out of date. Uninstall the app " or something like that, delete the Expo App from the emulator:
  - Shift-cmd-H
  - Tap and hold the Expo app icon. It'll start shaking and have an "x" in the upper right corner.
  - Hit the x to delete the app.
  - Close the simulator and run your app again.
  - It will "install" Expo fresh. Newest version!

# To install the iOS simulator ...

- Open Xcode
- Go preferences - components - simulators - Pick one.



# 3. Run it in an Android emulator

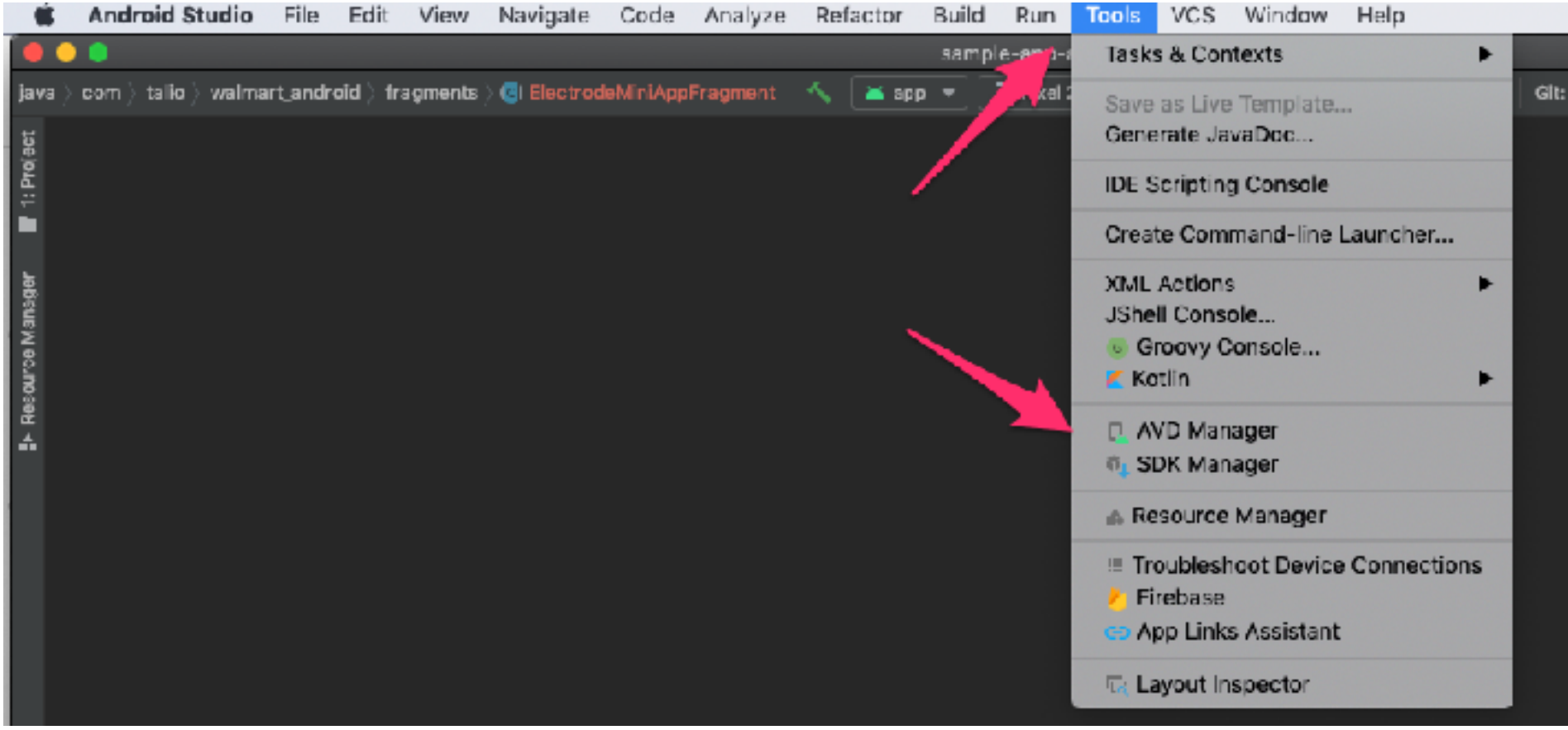
The "a" option

# Android Studio

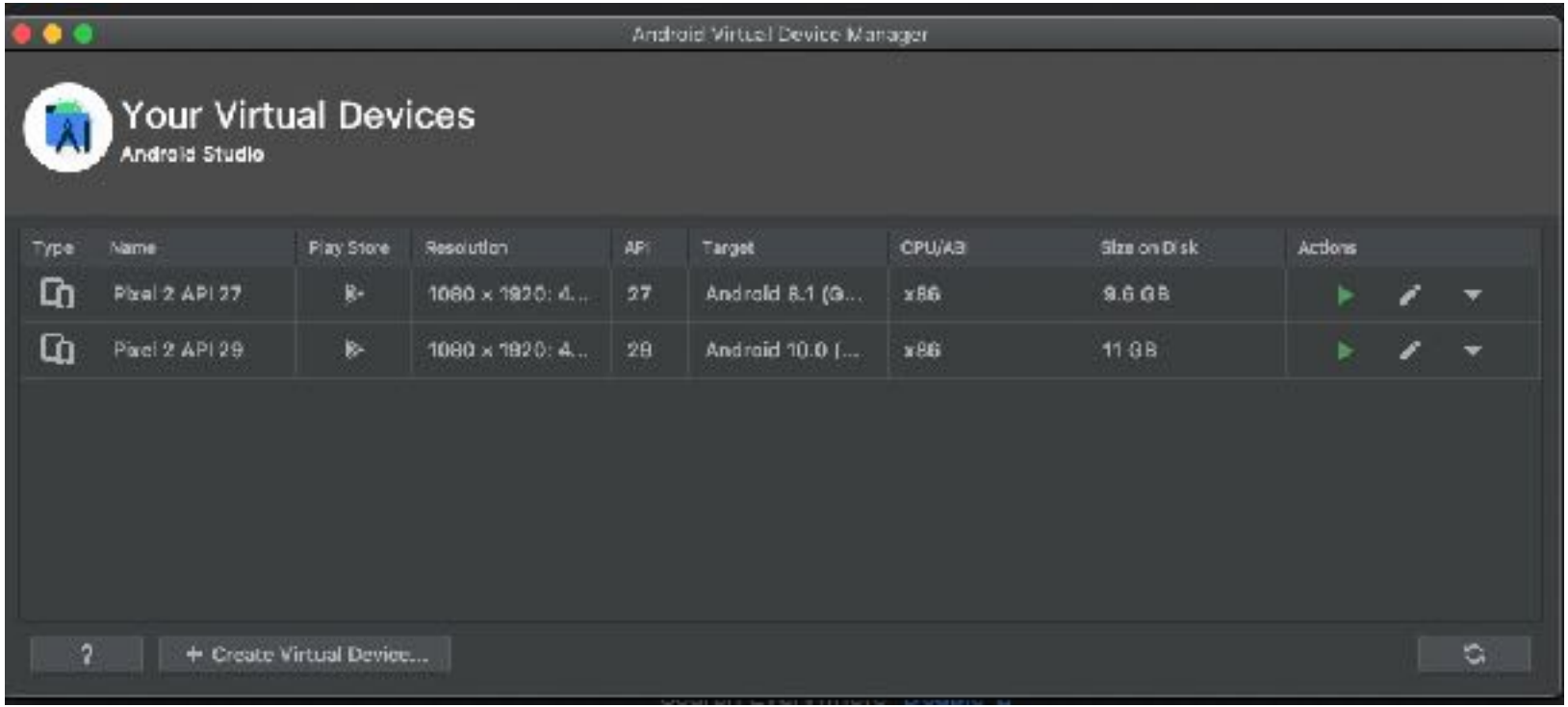

Free Android IDE, plus gives you emulation

https://developer.android.com/studio

# Android Studio

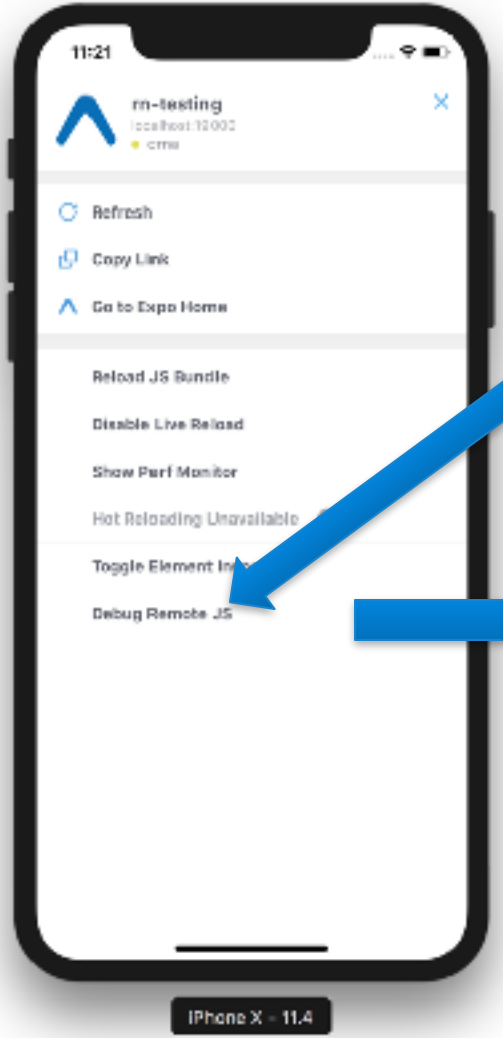


# Android Studio



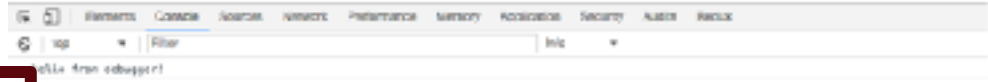
# Debugging

- Normally your app runs JavaScript in the device's engine called JavaScript core.
- But if you enable debugging through an emulator, Expo can connect the emulator to a browser, using the browser's JS engine. Therefore, you can use the typical browser's developer tools!
- console for console.log(), error(), etc
- JavaScript source debugging
- Network for examining Ajax requests and responses



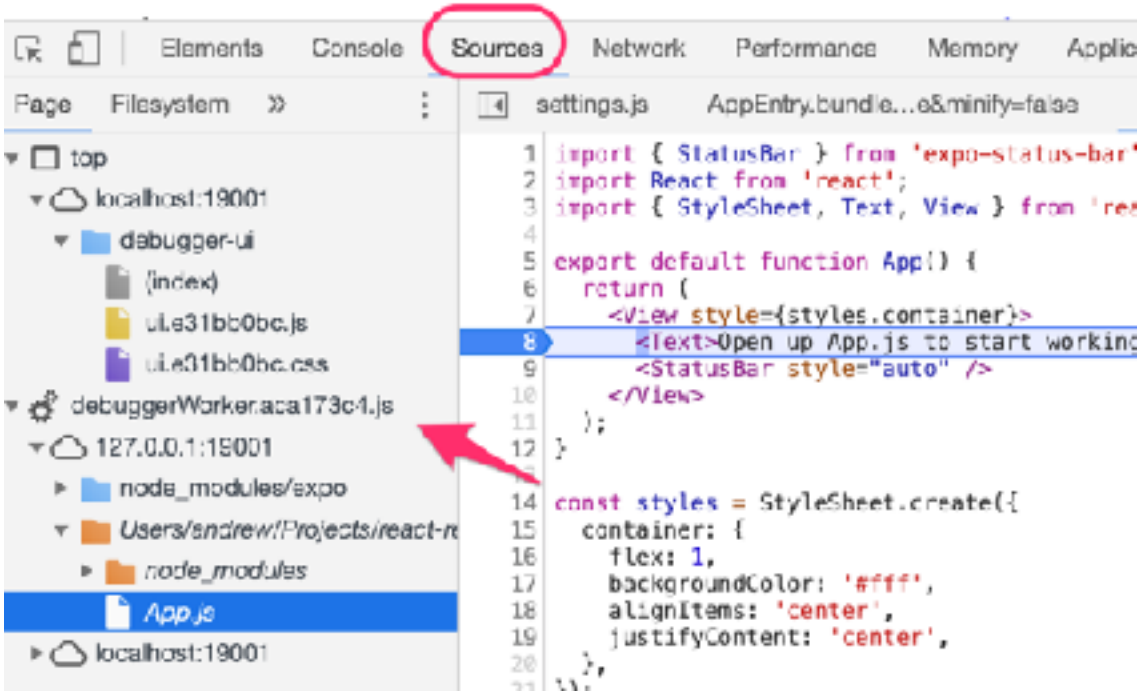

1. Bring up the device menu
   - Use the shake gesture
   - Hit cmd-M (Android) or cmd-D (iOS)
2. Choose "Debug Remote JS"
3. A browser opens with instructions


# You can console.log()

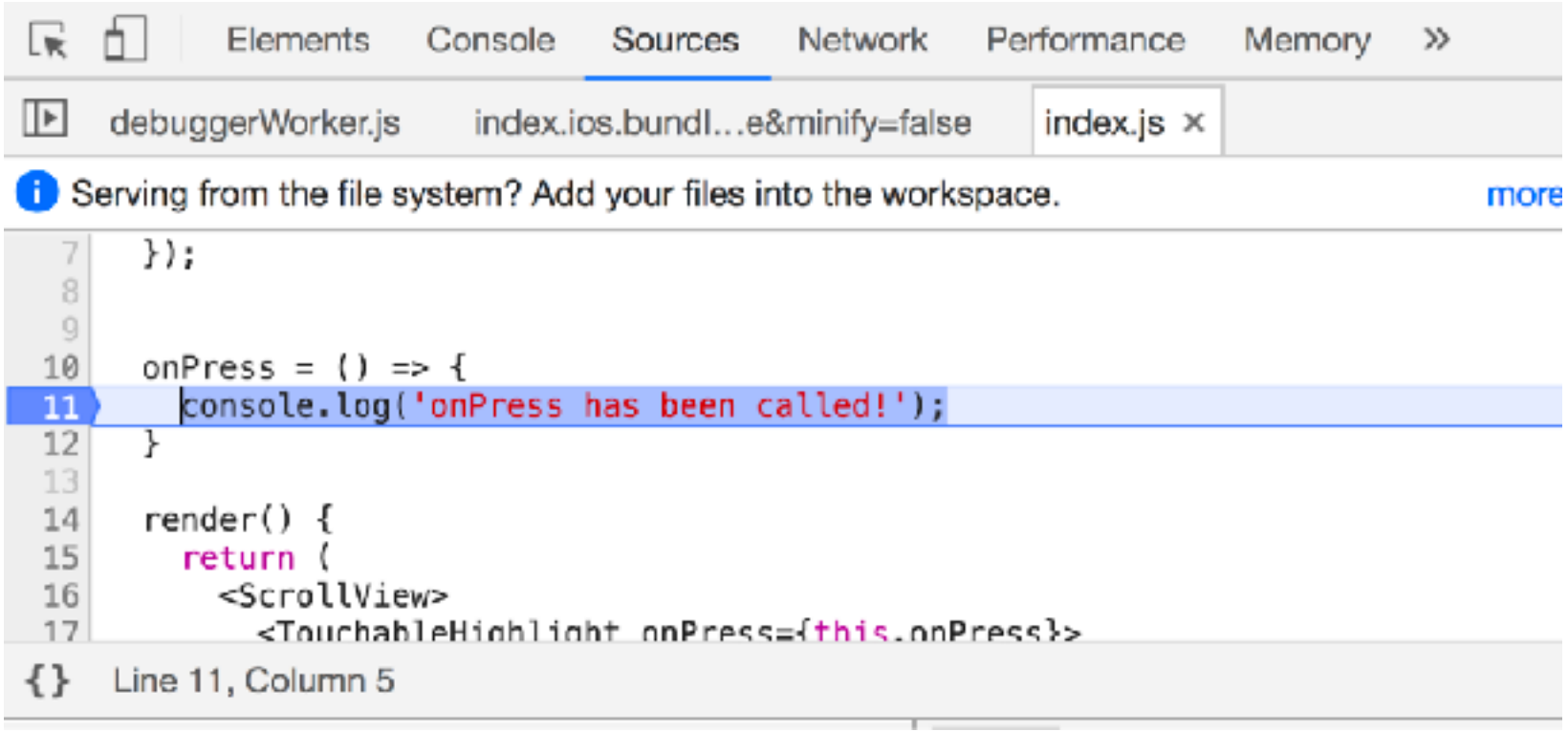



**If it doesn't appear here, look for it in the npm start command window**

You'll find your code under debuggerWorker.js in familiar folders

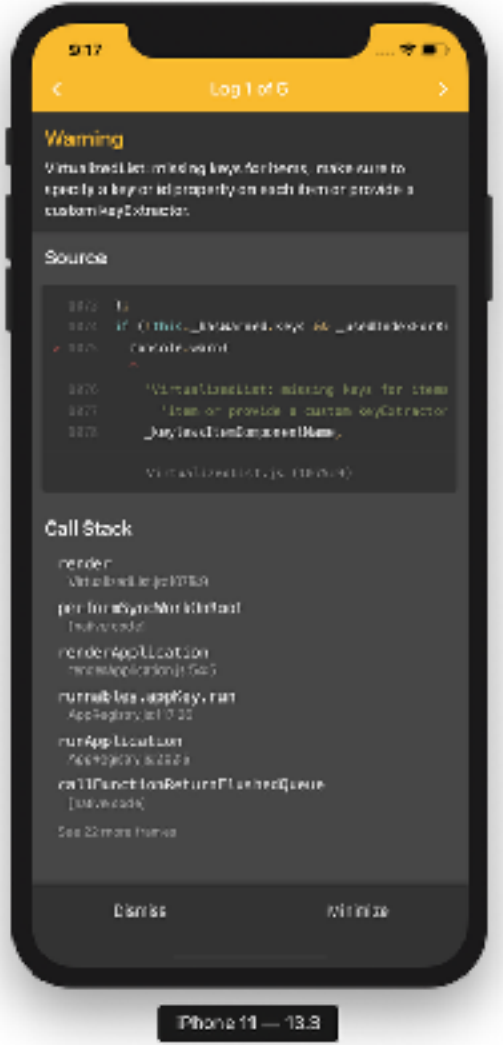


# You can set breakpoints



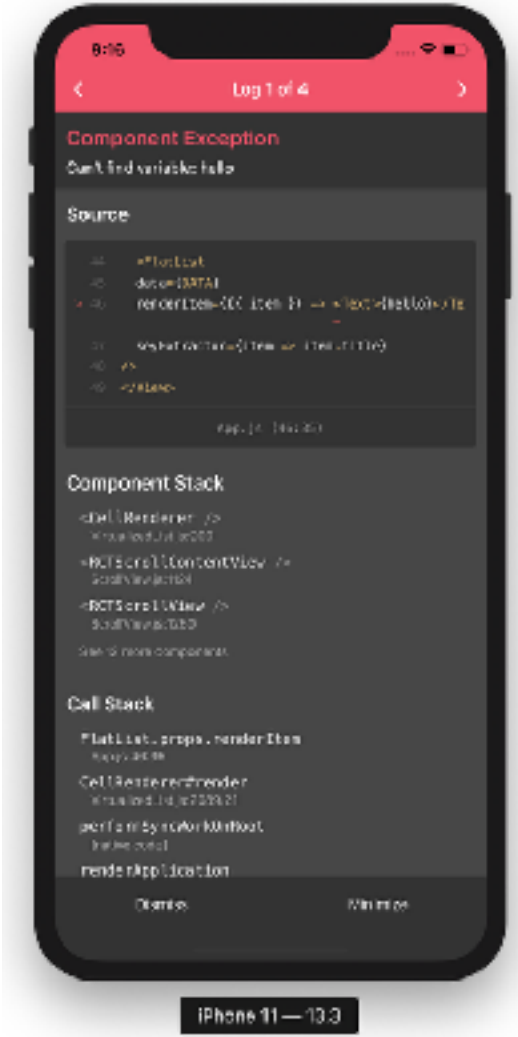
# If there are warnings you see a yellow LogBox

Tap the alert to show details or to dismiss them.



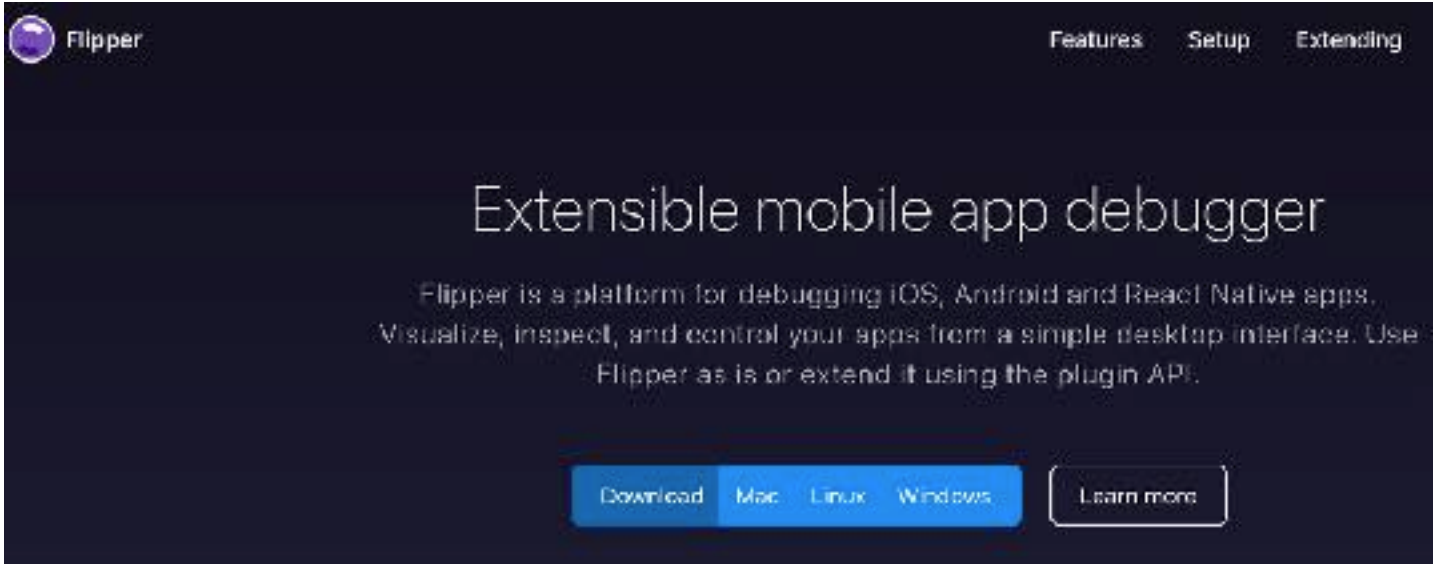
# If there are errors, you see a red LogBox

- This can be dismissed.
- It can be started with console.error();



# Flipper

# Flipper



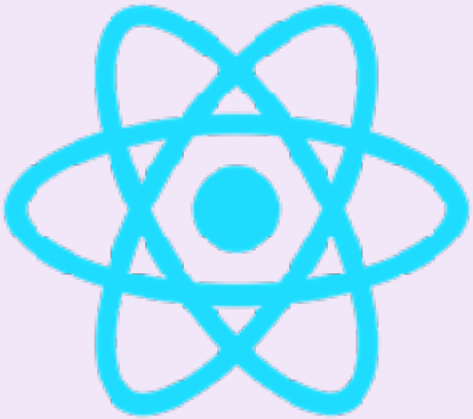

Not for expo 🤦‍♀️

# tl;dr

- Your app must run in a device during the development process
- Expo is an emulated environment. You can run it ...

1. on a physical device (with Expo installed) by scanning a QR code
2. in an iOS emulator after installing xcode and the iOS emulator on your Mac
3. in an Android emulator after installing AVD Manager or Genymotion/Oracle VirtualBox

- ... And there are actually more options



# Single Value Components

# tl;dr

- Components in React Native are not HTML but it might help to relate them to some HTML elements
- Text can't be bare. It must be in a <Text>
- <TextInput> is for editing text. The onChangeText event returns the text in the box
- <Images> are usually bundled with the executable.
- But when they are downloaded live, we must reserve space for them with a width and height or they won't show at all.


# It's important to learn components

So even though React Native looks like HTML and is structured like the HTML-based React, the components are completely different.

# Categories of RN components

| Category | Some components |
| --- | --- |
| Layout | Modal, View, SafeAreaView, ScrollView, RefreshControl, KeyboardAvoidingView, StatusBar, WebView |
| Single-value | Text, TextInput, Slider, Switch, Image |
| List | Picker, FlatList, SectionList |
| Touchable | Button, TouchableHighlight, TouchableNativeFeedback, TouchableOpacity, TouchableWithoutFeedback |
| Others | ActivityIndicator, Platform-specific components |

<Text>

# <Text>

Kind of like a <p>

Only text can appear in it.

**It is permissible to nest another <Text> inside of it for styling purposes**

# Text must be wrapped

From the React Native documentation:

> In React Native, we are more strict about it: you must wrap all the text nodes inside of a <Text> component. You cannot have a text node directly under a <View>.

**RN will throw if there is any text not wrapped in a <Text> element**

# Some Text Props

| Name | Notes |
| --- | --- |
| numberOfLines | Truncate text after this many lines instead of continuing to wrap and grow. |
| ellipsizeMode | When truncating,<br>• head - put the ellipsis at the beginning and cut off the start<br>• tail - put the ellipsis at the end and cut off the end<br>• middle - put the ellipsis in the middle<br>• clip - Don't show any ellipses, just clip the end off |
| selectable | If false, user can't select text, like for copying for example |
| selectionColor | What the text looks like while being selected |
| adjustFontSizeToFit | A bool. If true, shrink the fontSize until all the text fits in the container. (iOS only) |

# Some Text Events

| Name | Notes |
| --- | --- |
| onPress | A tap or click |
| onLongPress | Usually takes the place of a right-click |
| onLayout | Fires when the scene is being laid out |

# <TextInput>

# A component for keyboard entry

Kind of like all these:

```
<input type='text' />
<textarea />
<input type='password' />
<input type="number" />
```

But combined into one control

# Some TextInput Props

| Name | Notes |
| --- | --- |
| autoCorrect | Yep, it'll turn on the sometimes embarrassing feature |
| dataDetectorTypes | Can make the text clickable while the user is typing. If it detects one of 'phoneNumber', 'link', 'address', 'calendarEvent', or 'all' (iOS) |
| keyboardType | number-pad, email-address, phone-pad |
| multiline | bool. If false, it's a single line. Some other properties are ignored if it is single-line. Like borderLeft and borderRight for instance. |
| numberOfLines | Used with multiline=true (Android) |
| placeholder | Just like in the web. Ghost text |
| secureTextEntry | bool. True makes it like a password box |
| value | What is in the textbox |

# Some TextInput Events

| Name | Notes |
| --- | --- |
| onBlur/onFocus | Just like their web counterparts |
| onChange | Sends the event object (See example later) |
| onChangeText | Sends the text (See example later) |
| onKeyPress | note: no onKeyUp, onKeyDown, etc. |

# The event object when needed:

**ModifyPerson.js**

```
export class ModifyPerson extends Component {
 ...
 render() {
  <TextInput value={this.state.first}
    onChange={this.handleChange} />
 }
 handleChange(e) {
this.setState(
   {first: e.nativeEvent.value});
 }
}
```

# onChangeText sends the text itself

**ModifyPerson.js**

```
export class ModifyPerson extends Component {
 ...
 render() {
  <TextInput value={this.state.first}
    onChangeText={this.handleChange} />
 }
 handleChange(text) {
  this.setState({first: text});
 }
}
```


<Image>

# HTML <img> != RN <Image>

| HTML <img> tag | <Image> component |
| --- | --- |
| • For showing an image<br>• Easily scaled<br>• A separate download<br>• Unknown size<br>• Only one virtual machine -- the browser | • For showing an image<br>• Easily scaled<br>• Part of your bundle*<br>• Known size<br>• Many VMs - iPhone X, 7, 6s, etc. Nexus 6, 5, etc. GS4, 5.. |

*often, but not always

# Source

- In the web we ask the server for the image so it must live in a public, static, web server folder.
- In RN, we don't ask for things from the web except for Ajax data requests
- So we place our image files alongside our source

# For local images

- These are compiled into the install package
- For local images:

```
<Image source={require('./assets/foo.jpg')} />
```

# For remote images

- A simple request

```
<Image source={{uri: 'http://foo.com/img.jpg'}} />
```

- A more complex request

```
<Image source={{uri: 'https://foo.com/img.jpg',
 method='POST',
 headers: { 'accepts-type': 'png, jpg, gif' },
 body: { key1: val1, key2: val2 ...}
}} />
```

1. Flow of the page layout happens (kind of) like this...
2. Component is compiled
3. Scene is laid out on the page
4. The layout engine makes room for the image, which at this point is 0 x 0.
5. The image is downloaded to the device and scaled to fit in its allocated space (0x0).

- Result: You see no image.

# Remote images must be sized

To solve the problems, give your image a height and/or width in it style

# Briefly, how to size your Image

```
<Image source={{uri: "http://imgur.com/i.jpg"}}
  style={{
    height:100, width: 100,
    resizeMode: "cover",   // The default
 }} />
```

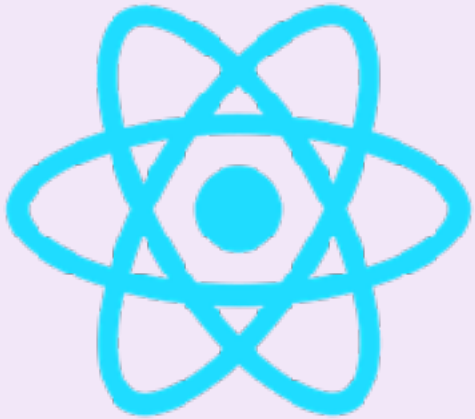
**Much more on styling later in the course**

# Sizing images: resizeMode

- resizeMode: stretch | cover | contain | repeat | center
  - stretch: Lose aspect ratio. Grow both x and y to fill. Nothing is cut off and no blank space top/bottom or left/right
  - cover: Maintain aspect ratio. Grow until x or y covers all blank space. top/bottom or left/right will be cut off. No blank space.
  - contain: Maintain aspect ratio. Grow until x or y fits. Nothing is cut off but there is blank space top/bottom or left/right.
  - repeat: duh
  - center: same as contain.

# tl;dr

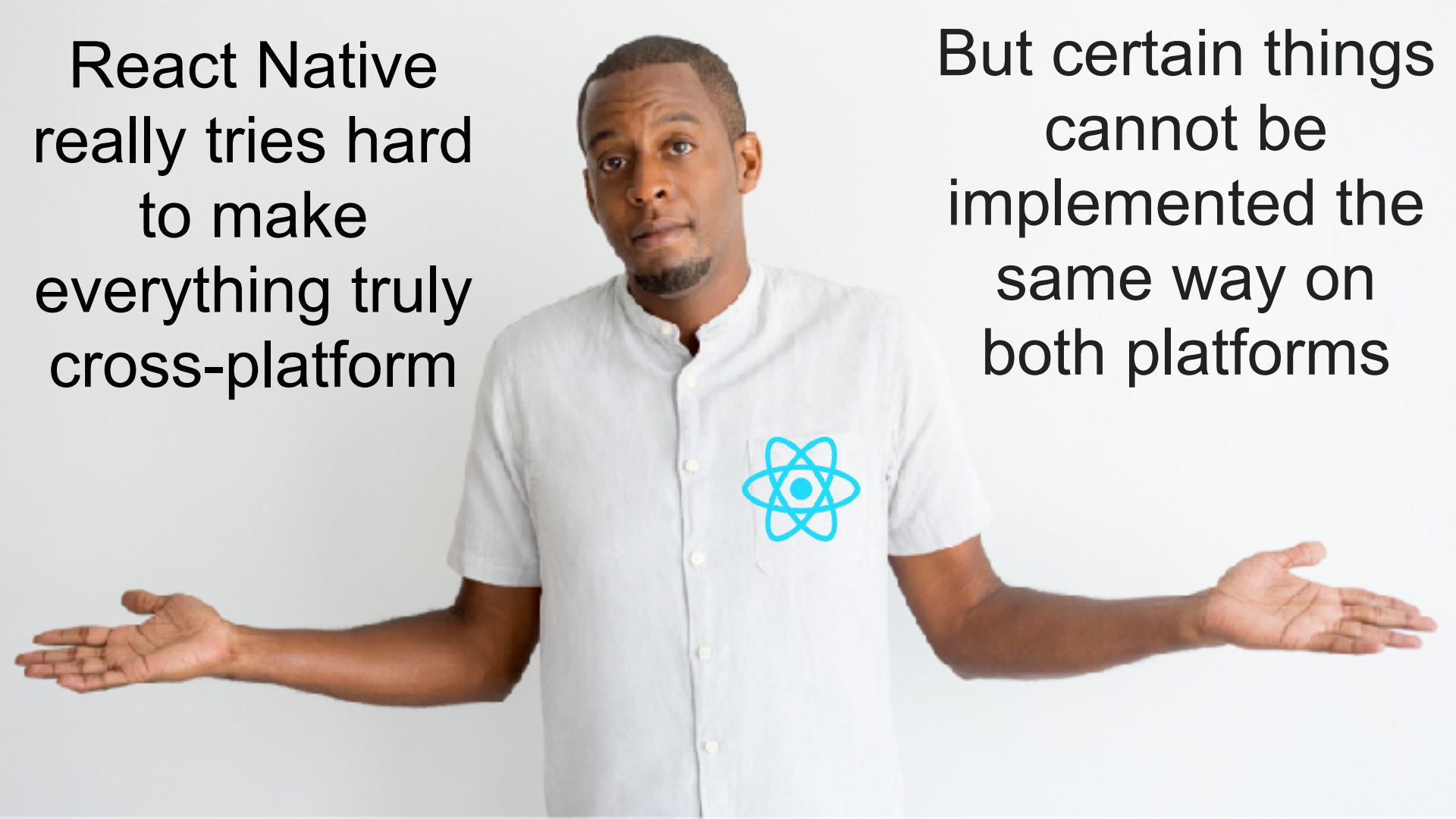
- Components in React Native are not HTML but it might help to relate them to some HTML elements
- Text can't be bare. It must be in a <Text>
- <TextInput> is for editing text. The onChangeText event returns the text in the box
- <Images> are usually bundled with the executable.
- But when they are downloaded live, we must reserve space for them with a width and height or they won't show at all.



# Platform-specific Development

# tl;dr

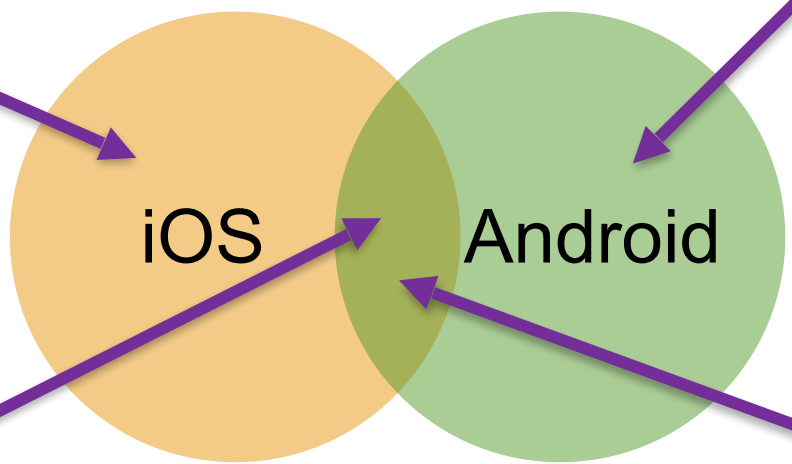
- React Native tries to be 100% cross platform, but that is impossible in some situations (Like DatePicker, for example).
- Besides, we may want there to be differences
- When that happens, we can use
  1. Platform-specific extensions, serving a whole different file
  2. The Platform module which gives us
     - Platform.OS
     - Platform.select()


React Native really tries hard to make everything truly cross-platform
But certain things cannot be implemented the same way on both platforms

# Truly cross-platform development gives us a few choices

1.Focus on iOS. Let Android suffer

2.Focus on Android. Let iOS suffer

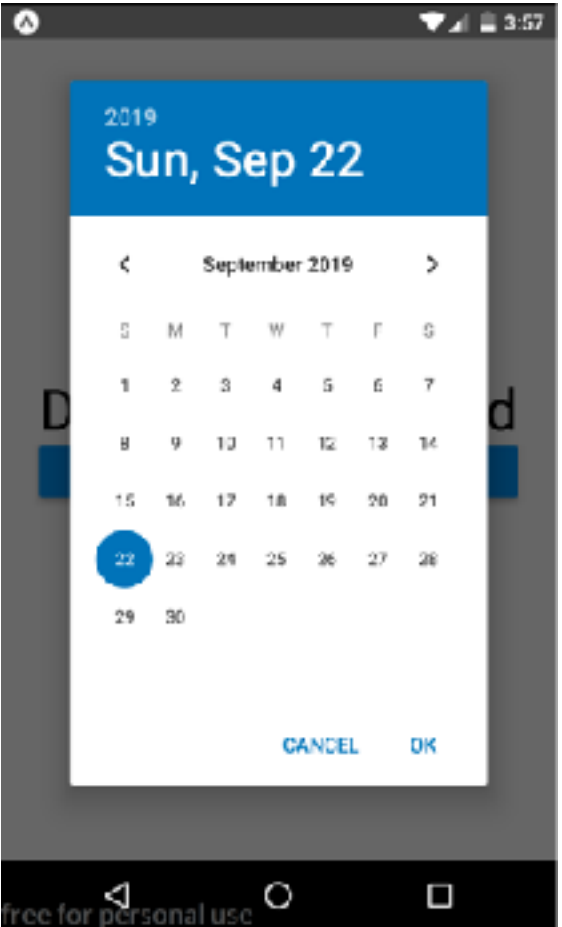



3.Use a limited set of features that are well-supported on both

4. Write a ton of code for each platform to strengthen the features on both

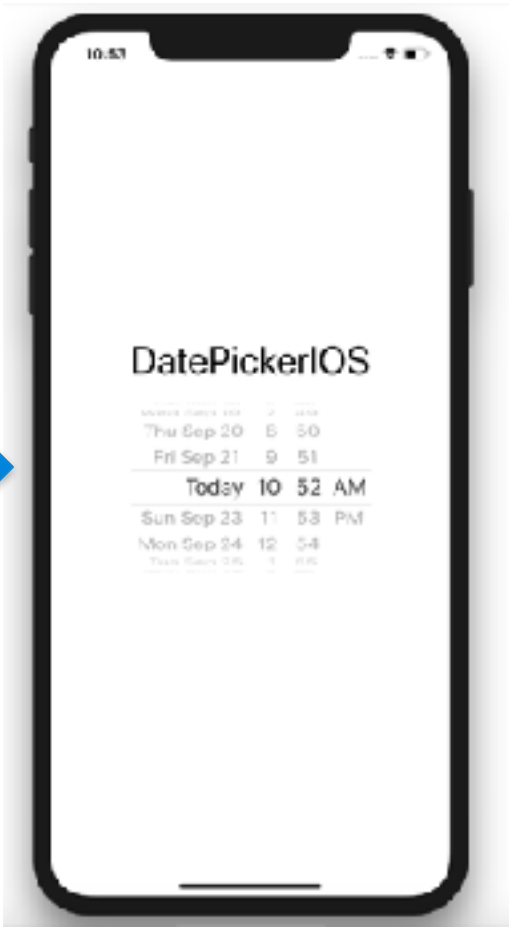
# DatePicker*

# There are two DatePickers




DatePickerAndroid

DatePickerIOS



# To use either on the wrong platform is an error

DatePickerAndroid on iOS


DatePickerIOS is not supported on this platform!

DatePickerIOS on Android

# DatePickerIOS is a JSX component

```
<DatePickerIOS
 date* - The initial date
 onDateChange* - Event when they change a date.
      Returns the date chosen.
 maximumDate, minimumDate
 mode - 'date', 'time', or 'dateTime'
 timeZoneOffsetinMinutes - Force a timezone
     instead of looking at our location
/>
*Required
```

# Simple DatePickerIOS example

**Foo.js**

```
function Foo() {
 return (
  <DatePickerIOS date={new Date()}
                 onDateChange={setDate} />
 )
 function setDate(date) {
  console.log("The date: ", date);
 }
}
```

# DatePickerAndroid is a JS Object

`open(options)` - Opens the standard Android date picker dialog and returns a Promise

```
options = {
  date - The initial date (defaults to today)
  minDate, maxDate
  mode - 'calendar', 'spinner'
}
```

The Promise callback receives an object

- `action` - dateSetAction or dismissedAction
- `year`, `month` (0-11), `day` (1-31)

# Simple DatePickerAndroid example

**Foo.js**

```
function Foo() {
 return (
  <Button onPress={getDate} />
 )
 function getDate() {
  DatePickerAndroid.open({date:new Date()})
   .then( ({month,day,year}) =>
          setDate(new Date(year,month,day)));
 }
 function setDate(date) {
  console.log("The date: ",date);
 }
}
```

# Android-only

- DatePickerAndroid
- DrawerLayoutAndroid
- ProgressBarAndroid
- ToolbarAndroid
- ViewPagerAndroid

# iOS-only

- DatePickerIOS
- MaskedViewIOS
- NavigatorIOS
- PickerIOS
- ProgressViewIOS
- SegmentedControlIOS
- SnapshotViewIOS
- TabBarIOS

# How to handle Platform-specific code

# Two main ways to implement Platform Specific code

1. Platform-specific file extensions
2. The Platform module

# Platform Specific File Extensions

- If you ...

```
import MyButton from './MyButton';
```

- And there is no MyButton.js.
- But there are ...

```
MyButton.ios.js
```

```
MyButton.android.js
```

- Then the compiler will pick the proper one for each platform when compiling

# Platform.OS is set to a string, 'ios' or 'android'

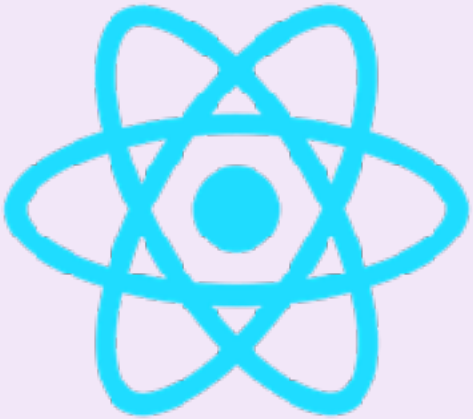
```
function CustomButton() {
 const MyButton = Platform.OS === 'ios' ?
  TouchableHightlight : TouchableWithoutFeedback;
 return (
  <MyButton onPress={console.log}>
   <Text>Press me</Text>
  </MyButton>
 );
}
```

# Platform.select()  returns a different object depending on the platform

```
function CustomButton() {
  const MyButton = Platform.select({
    ios: TouchableHighlight,
    android: TouchableWithoutFeedback,
  });
  return (
    <MyButton onPress={console.log}>
      <Text>Press me</Text>
    </MyButton>
  );
}
```

# tl;dr

- React Native tries to be 100% cross platform, but that is impossible in some situations (Like DatePicker, for example).
- Besides, we may want there to be differences
- When that happens, we can use
  1. Platform-specific extensions, serving a whole different file
  2. The Platform module which gives us
     - Platform.OS
     - Platform.select()



# Layout Components

# tl;dr

- Containers are the main thing we use to control the layout of the screen
- Most were designed to be nested inside one another, each adding their own capabilities
- View is the most fundamental container
- ScrollView allows us to scroll through content that won't fit on a screen
- Modal is a container that can't be ignored
- StatusBar only lets us change the appearance of the status bar

# Categories of RN components

| Category | Some components |
| --- | --- |
| Layout | Modal, View, SafeAreaView, ScrollView, RefreshControl, KeyboardAvoidingView, StatusBar, WebView |
| Single-value | Text, TextInput, Slider, Switch, Image |
| List | Picker, FlatList, SectionList |
| Touchable | Button, TouchableHighlight, TouchableNativeFeedback, TouchableOpacity, TouchableWithoutFeedback |
| Others | ActivityIndicator, Platform-specific components |

# We mentioned containers before but now it's time to dig in to ...

- View
- SafeAreaView
- ScrollView
- KeyboardAvoidingView
- Modal
- StatusBar

# View

# <View>

The most fundamental, most generic of all components.

Like a  in html, this is a container.

# The most used control of them all?

- View is the fundamental unit. Like a  for native devices
- Allows layouts by supporting flexbox
- Can have as many children as you like, including more views.

# SafeAreaView

# SafeAreaView doesn't render in non-viewable areas

- Works cross-platform but doesn't do anything on Android
- Looks great on iPhone, though!

# It wraps your regular view

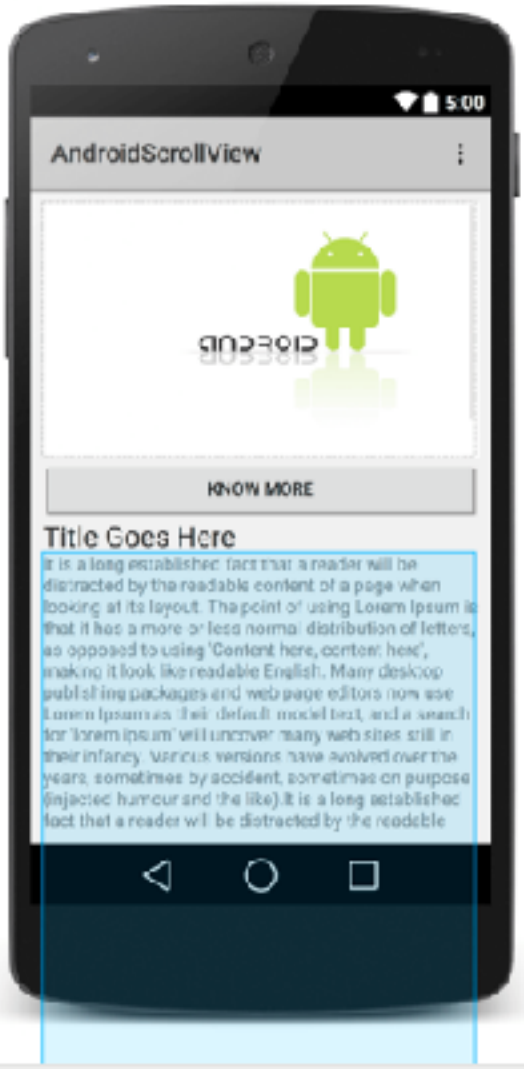
**MyComponent.js**

```
function MyComponent() {
 return (
  <SafeAreaView>
   <View>
    <Text>I'm the regular view</Text>
   </View>
  </SafeAreaView>
}
```

# <ScrollView>

- Without any props, a <ScrollView> makes its contents scrollable immediately.
- Great for when you have too much content in a single view and you want the user to be able to scroll the page to see it all.

```
<ScrollView>
   <TonsMoreContentHere />
</ScrollView>
```



# Makes other views scrollable

- Can set up scrolling horizontally as well as vertically
- Can also set up paging by swiping left/right.
- Nests inside of other Views to make their content scrollable within them.
- All elements inside a ScrollView are rendered even if they're off screen

# To pinch-to-zoom

- The ScrollView has to have a single item.
- Set minimumZoomScale and maximimZoomScale to enable. it.

# KeyboardAvoidingView

- When the user focuses on a <TextInput>, the keyboard slides up.
- This sometimes covers the input they're trying to type into.

KeyboardAvoidingView will move the rest of the view out from behind the keyboard

# Wrap the rest of the view in a <KeyboardAvoidingView>

**webpack.config.js**

```
function MyComponent() {
 return (
  <KeyboardAvoidingView behavior="padding">
   <View>
    <Text>All the other view goes here</Text>
   </View>
  </KeyboardAvoidingView>
)}
```

# Some props

- enabled - boolean. Assumed to be true.
- behavior - height | position | padding
  - height - Force the height of the rest of the view to be the remaining height of the screen
  - position - Move the whole thing up
  - padding - Squeeze everything on the page as much as possible by decreasing the padding around elements

**From the RN docs: "Note: Android and iOS both interact with behavior differently. Android may behave better when given no behavior prop at all, whereas iOS is the opposite.**

<Modal>

# <Modal>

Creates a modal window.

A pop-up that can't be ignored

On top of the other Window

Wraps other views

# Modal props

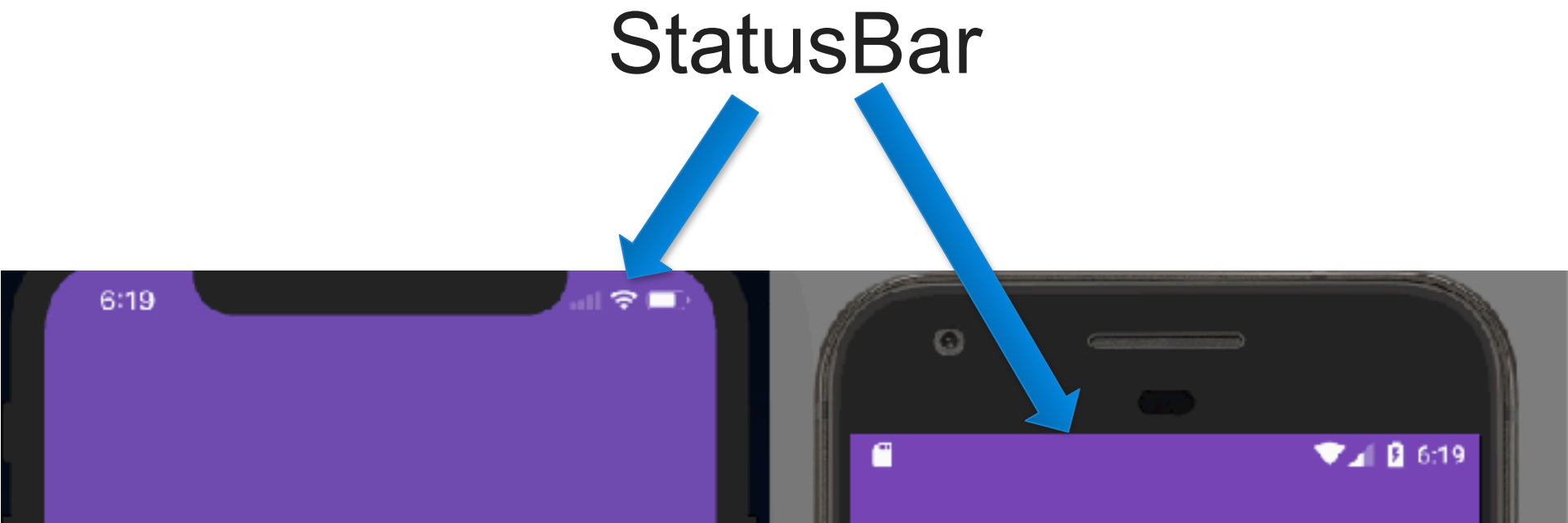
- visible - a bool. Only a true boolean false will hide it. Falsey values cause the Modal to show -- even undefined.
- onShow
- onDismiss
- onOrientationChange

```
const MyComponent = props => {
 const [shown,setShown] = useState(false);
 render() {
  return <View>
   <Modal visible={shown}>
    <View>
     <Text>I'm the modal</Text>
     <Button title="Dismiss"
      onPress={()=>setShown(false)}/>
    </View>
   </Modal>
   <Button title="Show Modal"
    onPress={()=>setShown(true)} />
  </View>
 }
}
```

Toggle its *visible* property to show/hide

# StatusBar

- Your app is not in control of the status bar
- The status bar doesn't honor styles like everything else because it is controlled by the OS and not the app
- All this control can do is change a few styles on the bar

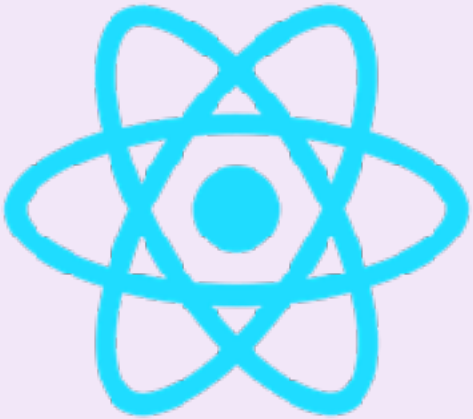


# StatusBar props

- barStyle - An enum - 'dark-content', 'light-content', 'default'
- backgroundColor - Yep, just what the name says
- hidden - bool. When true, the status bar is not drawn at all
- translucent - bool. When true, the app draws behind it. When false, the app is placed below the statusBar

# tl;dr

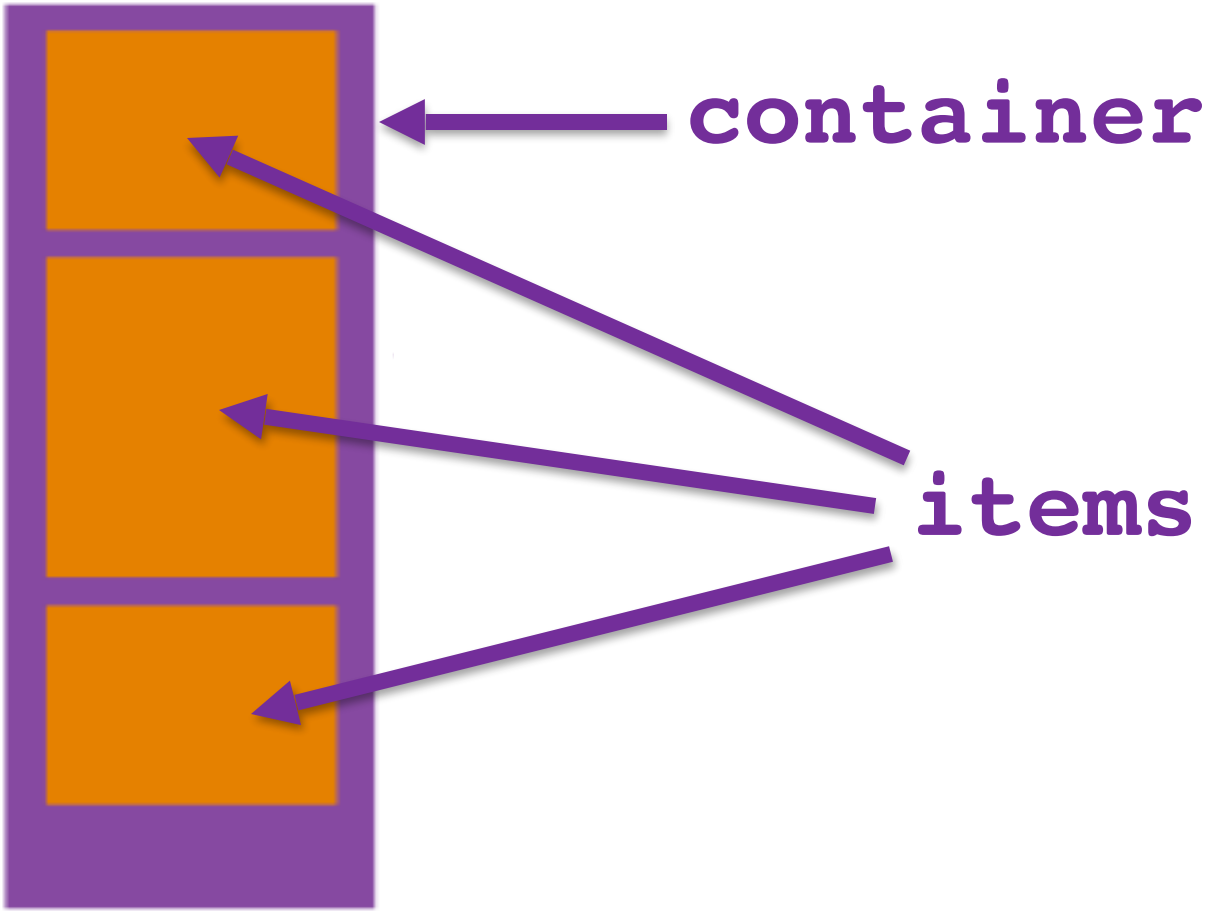
- Containers are the main thing we use to control the layout of the screen
- Most were designed to be nested inside one another, each adding their own capabilities
- View is the most fundamental container
- ScrollView allows us to scroll through content that won't fit on a screen
- Modal is a container that can't be ignored
- StatusBar only lets us change the appearance of the status bar

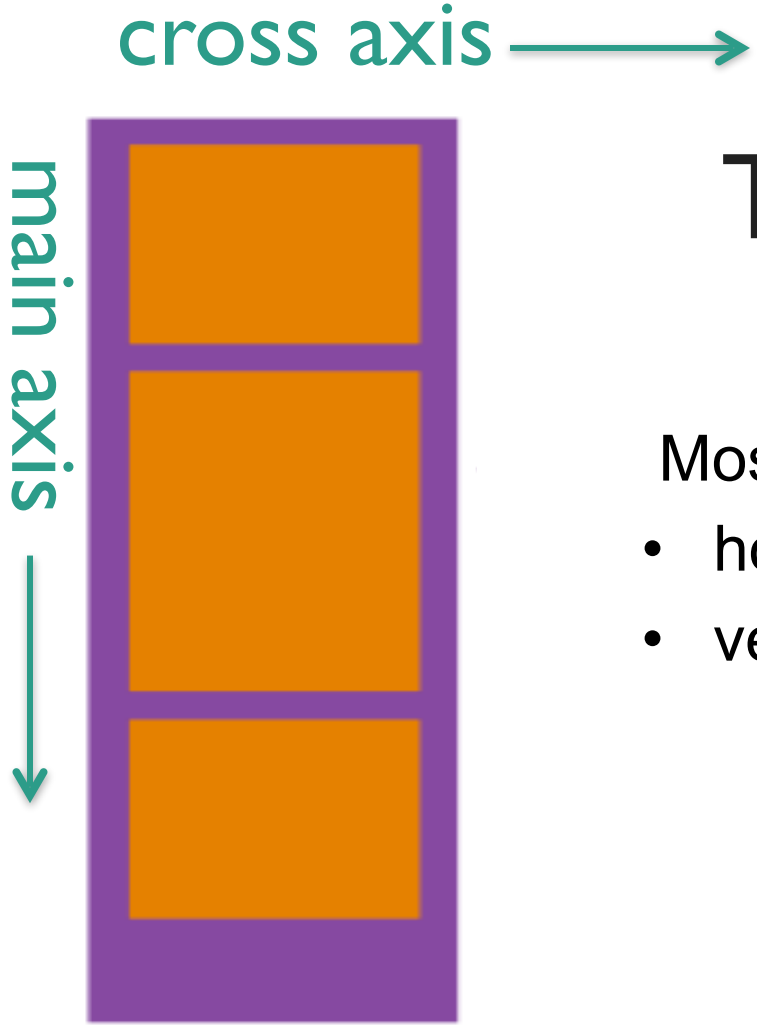


# Layouts with Flexbox


Cary
@caryhartline
.@jensimmons it's difficu
too complicated or if
mindset.
Follow
Curtis
@_CurtisR
why is #flexbox so difficult when it comes to the
easiest la
Follow
AlHasan Husni
@AlHasanps
Flexbox is awes
concepts that ca
interactive... fb.me/7U
Philip Roberts
@philip_roberts
I know flexbox is powerful and awesome; but
boy does it make me feel like a total idiot
everytime I try and use it.
Follow

# flex involves a container and contents





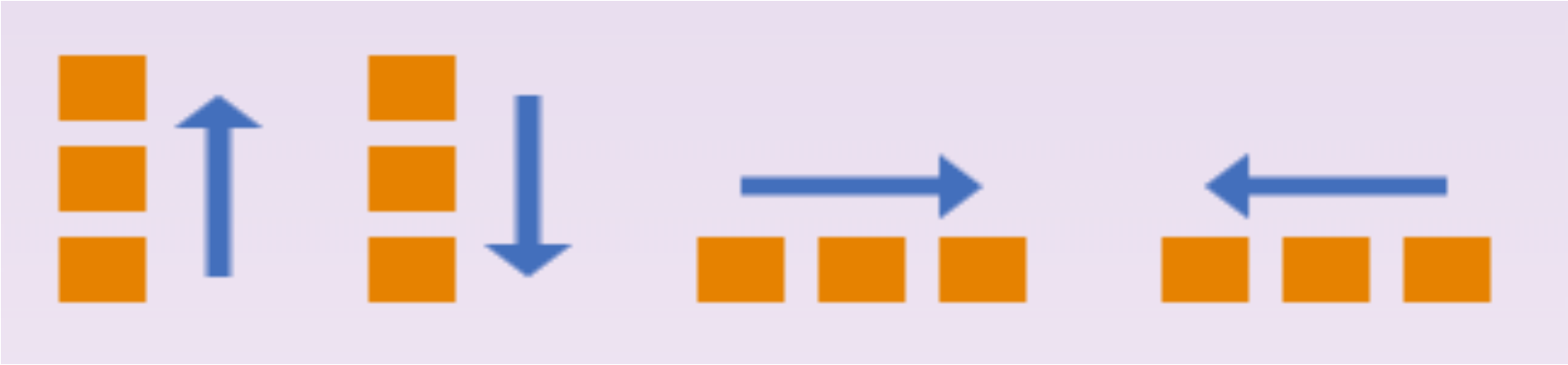

# The container has axes

Most of us would call these

- horizontal
- vertical

# The container lays out the items

```
container: {
  flexDirection: 'column-reverse',
  flexDirection: 'column',
  flexDirection: 'row',
  flexDirection: 'row-reverse',
}
```



# flex items have sizes

They have a "favorite" size

flexBasis

They know how to shrink from that and how to grow from that

flexShrink
flexGrow

# flexBasis is that favorite size

## When flexDirection: 'row'

- flexBasis == width

- width == width
- height == height

## When flexDirection: 'column'

- flexBasis == height

- width == width
- height == height

# Sizes of the items

- flexBasis = the item's "favorite" size.
  - in px
- If the viewport is increased from flex-basis ...
- flexGrow = by how much it grows
  - unitless - a relative number
- if the viewport is decreased from flex-basis ...
- flexShrink= by how much it shrinks
  - unitless - a relative number

# To have fixed widths/heights, set flexBasis on each item

```
.anItem {
  flexBasis: 150;
}
```

**width and height will still work as well**

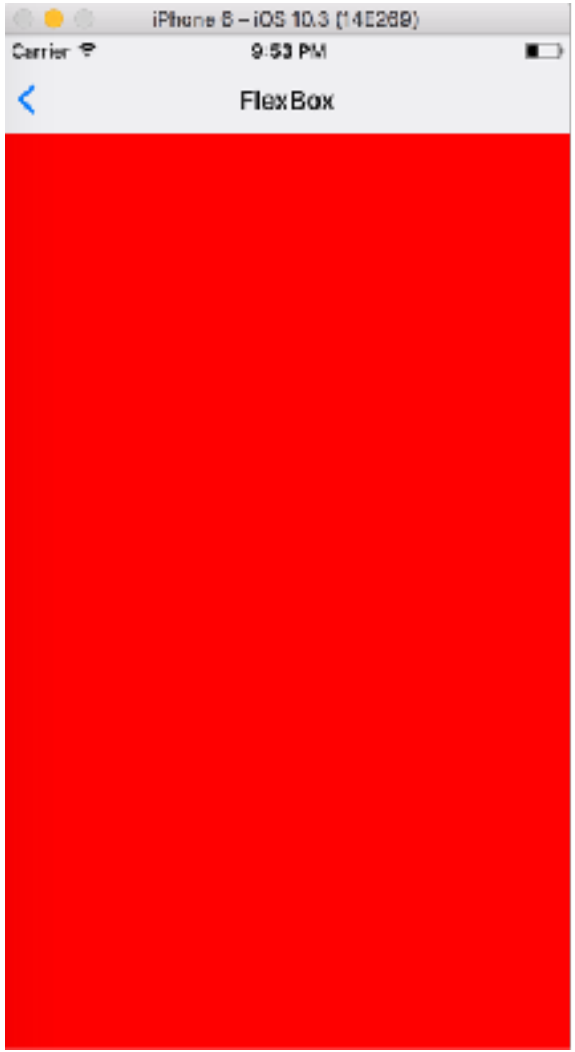
```
item1:{
   flexGrow: 3;
}
item2:{
   flexGrow: 1;
}
item3:{
   flexGrow: 2;
}
```

# To have relative widths/heights set flexGrow/ flexShrink on each item

**These numbers are unitless and relative to one another**

# A single number is a shorthand

If you provide a flex property to elements in the same container, and that has a number, then those numbers are compared and allocated the free space available.

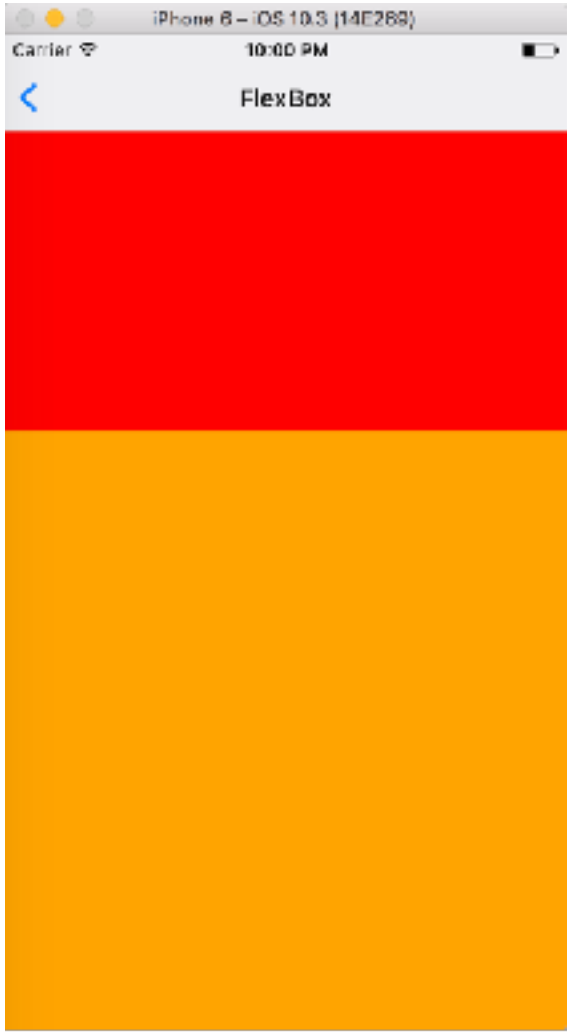



```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},


}
<View style={styles.container}>
 <View style={styles.box1} />

</View>
```

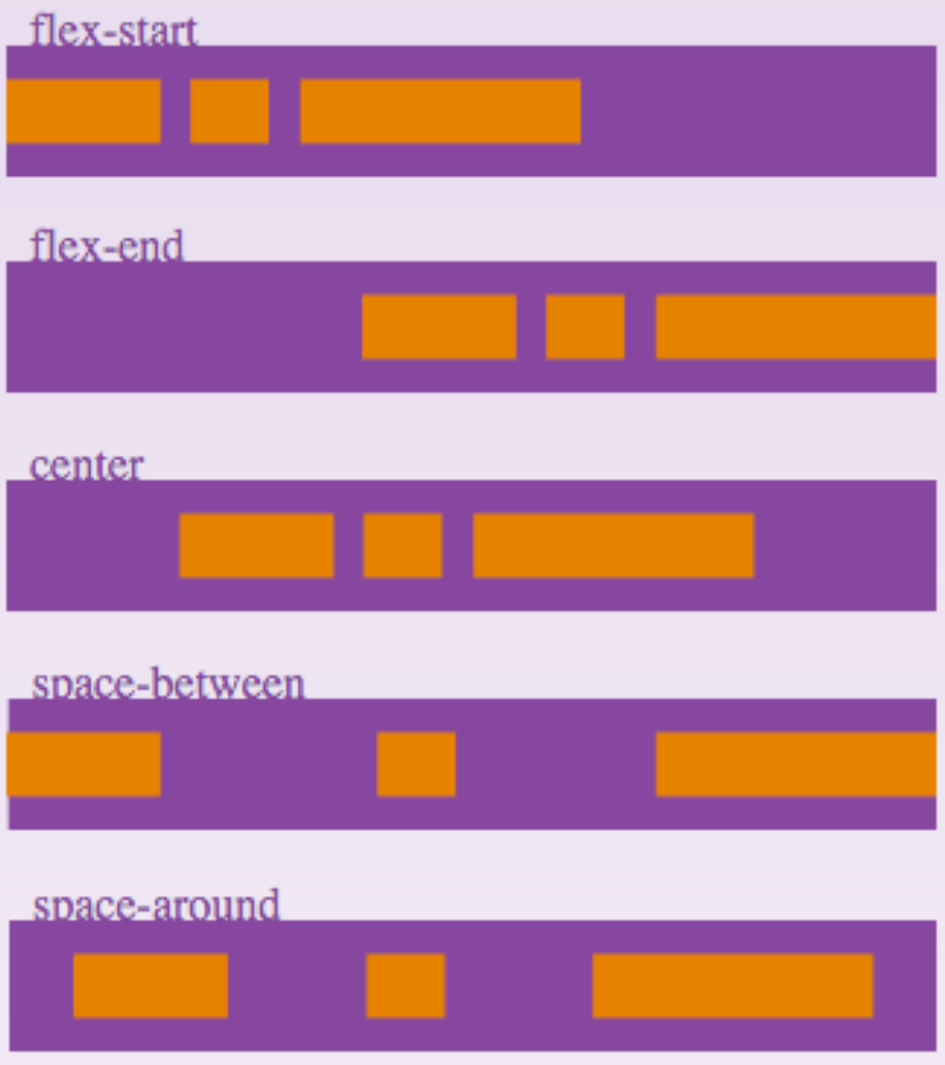

```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},
 box2: {flex: 1,
  backgroundColor: 'orange',},
}
<View style={styles.container}>
 <View style={styles.box1} />
 <View style={styles.box2} />
</View>
```




```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},
 box2: {flex: 2,
  backgroundColor: 'orange',},
}
<View style={styles.container}>
 <View style={styles.box1} />
 <View style={styles.box2} />
</View>
```

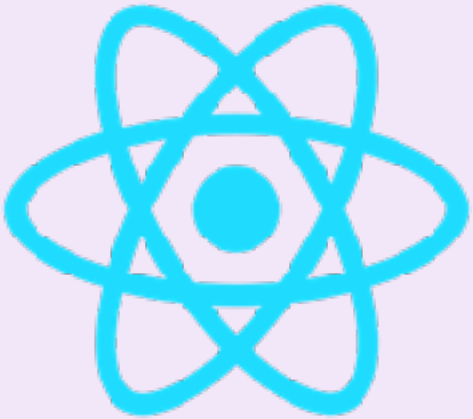
```
container: {
 justifyContent: '_____',
}
```

# justifyContent controls distribution of children along the main axis



# tl;dr

- React Native layouts are handled with a framework that mimics some features of CSS flex layouts
- The learning curve for flexbox is high
- To put things side-by-side, you put them in a container with a flexDirection of 'row'
- Then they will share the space in that row based on
  - flexBasis - their preferred size
  - flexGrow - If there's extra space, how do they share it?
  - flexShrink - Not enough space? How do they donate it?
- "flex: <number>" is a shortcut for flexBasis, flexGrow, and flexShrink



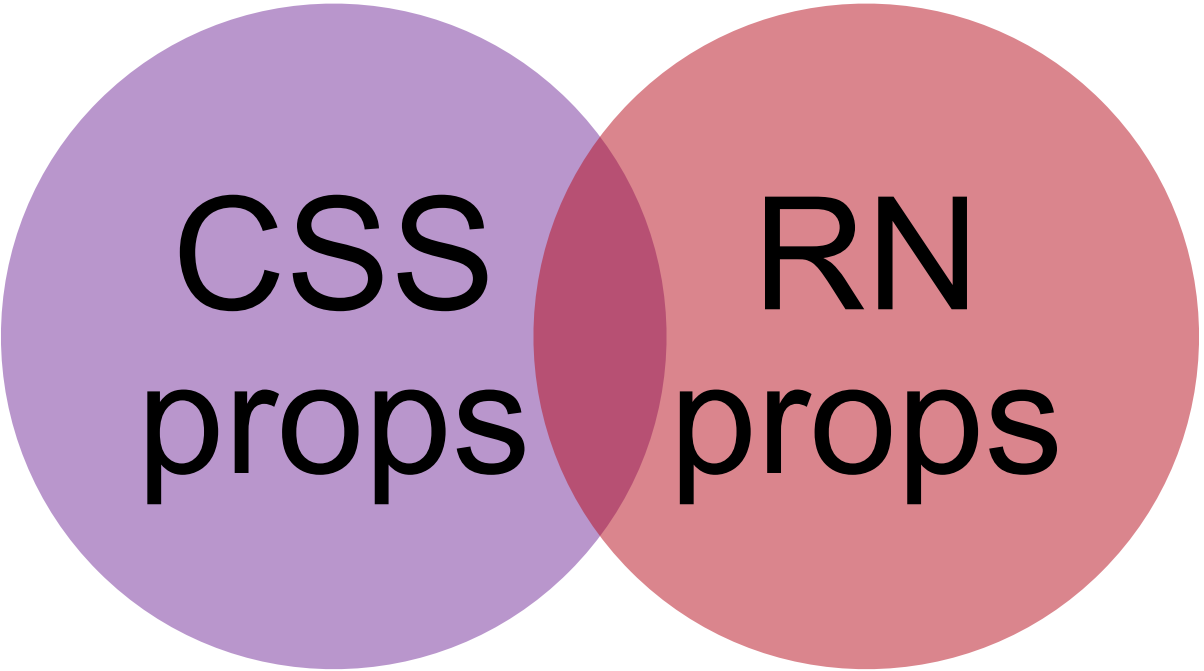
# Styling React Native

# tl;dr

- Native components receive a prop called style which should resolve to a plain JavaScript object which contains keys that look like CSS but are definitely not CSS.
- We can create that style object any way you want, raw, inline, imported, but preferably with StyleSheet.create()
- These can even be applied dynamically using JavaScript logic

# React Native styles are **not** CSS

- They are simple JavaScript objects
- camel-cased, and not kebab-cased




The keys/props may look like CSS, but ...

- Not all CSS properties are supported
- There are some properties that CSS doesn't support

```
<Anything style={someStyleHere} />
```

You don't use a special language for defining styles. You just style your application using JavaScript. All the core components accept a prop named *style*.

# Styles are only applied on the component itself

There are no classes, no ids, no CSS selectors

```
let big={ fontSize: 80 };

<Text style={big}>Lorem</
Text>
```

```
let big={ fontSize: 80 };
let Text={fontFamily:'cochin'};
let #prodLabel={color:'red'};

<Slider class="big" />
<Text class="big" />
<Text class="smaller" />
<Text id="prodLabel" />
```

# You can apply arrays of styles

```
export const MyComponent () => (
 <Text style={[bang, big, shout]}>
  I don't understand the question,
  and I won't respond to it.
 </Text>
);
```

These would "upsert" (Values in the later ones clobber earlier ones if they already exist)

# And combine inline with external

```
export const MyComponent () => (
 <Text style={[big,{color:'red'}]}>
  I don't understand the question,
  and I won't respond to it.
 </Text>
);
```

These would "upsert" (Values in the later ones clobber earlier ones if they already exist)

# How to define styles

# You define styles in 4 ways

1. Inline
2. Inside the component - raw
3. Inside the component - with StyleSheet.create()
4. Separate stylesheet

# 1. Inline

```
<Text style={{ color:'red' }}>
I am going to oblige and answer the nice officer's
questions because I am an honest man with no secrets to
hide.
</Text>
<View style={{ flex: 1, textAlign:'left' }}>
```

**Note the double-curly braces**

# 2. Raw styles

**These are easier to write but slower at runtime**

```
const big = {
 fontSize: 50,
 fontWeight: 'bold'
};

<Text style={big}>
There are so many poorly chosen words in that sentence.
</Text>
```

**MyComponent.js**

```
const styles = StyleSheet.create({
 container: {
  flex: 1,
  backgroundColor: '#ffa1d6',
  alignItems: 'center',
  justifyContent: 'center',
 },
 hdr: {
  fontSize: 25,
  color: '#000000'
  textAlign: 'right',
 },
});
// Your component might go here
```

# 3. StyleSheet.create()

# And here's how to apply them

**MyComponent.js**

```
// Your styles object might be created here.
function MyComponent() {
 return <View style={styles.container}>
  <Text style={styles.hdr}>Cool App</Text>
  <Text>
   There are so many poorly chosen words
   in that sentence.
  </Text>
 </View>
}
```

# 4. Separate stylesheet

**myStyles.js**

```
export const styles =
StyleSheet.create({
 container: {
  flex: 1,
  alignItems: 'left',
 },
 headerText: {
  fontSize: 25,
 },
};
```

... sort of. It's just an object. So export from one file and import it into your component.

**MyComponent.js**

```
import {styles} from './myStyles';
function MyComponent() {
 return (
  <View style={styles.container}>
  ...
  </View>
}
```

# How do these styles work?

What properties exist? Are they the same as in CSS?

Do we have the same fonts available?

Same colors as in CSS?

Do styles inherit like they do in CSS/HTML?

Can you apply logic to the styles?

# Styles are seldom inherited!

- (Developer) React components are designed with strong isolation in mind: You should be able to drop a component anywhere in your application, trusting that as long as the props are the same, it will look and behave the same way. Text properties that could inherit from outside of the props would break this isolation.
- It's easier to list where they *are* and assume that everywhere else is not.
- <Text> embedded inside a <Text> is.

# So how do you inherit?

Pass styles down as props

- or -

Create a styled component and use it at any level

```
const MyText =
 ({children}) => (
  <Text style={{fontSize:20}}>{children}</Text>)
```

# Properties

backgroundColor
color
fontSize
fontWeight
pad
alignContent,
alignItems,
alignSelf,
aspectRatio,
backfaceVisibility,
backgroundColor,
borderBottomColor,
borderBottomLeftRadius,
borderBottomRightRadius,
borderBottomWidth,
borderColor,
borderLeftColor,
borderLeftWidth,
borderRadius,
borderRightColor,
borderRightWidth,
borderStyle,
borderTopColor,
borderTopLeftRadius,
borderTopRightRadius,
borderTopWidth,
borderWidth,
bottom,
color,
decomposedMatrix,
direction,
display,
elevation,
flex,
flexBasis,
flexDirection,
flexGrow,
flexShrink,
flexWrap,
fontFamily,
fontSize,
fontStyle,
fontVariant,
fontWeight,
height,
includeFontPadding,
justifyContent,
left,
letterSpacing,
lineHeight,
margin,
marginBottom,
marginHorizontal,
marginLeft,
marginRight,
marginTop,
marginVertical,
maxHeight,
maxWidth,
minHeight,
minWidth,
opacity,
overflow,
overlayColor,
padding,
paddingBottom,
paddingHorizontal,
paddingLeft,
paddingRight,
paddingTop,
paddingVertical,
position,
resizeMode,
right,
rotation,
scaleX,
scaleY,
shadowColor,
shadowOffset,
shadowOpacity,
shadowRadius,
textAlign,
textAlignVertical,
textDecorationColor,
textDecorationLine,
textDecorationStyle,
textShadowColor,
textShadowOffset,
textShadowRadius,
tintColor,
top,
transform,
transformMatrix,
translateX,
translateY,
width,
writingDirection,
zIndex

**There's a great cheatsheet of React Native style properties here: http://bit.ly/RNStyles**

# Fonts on Android

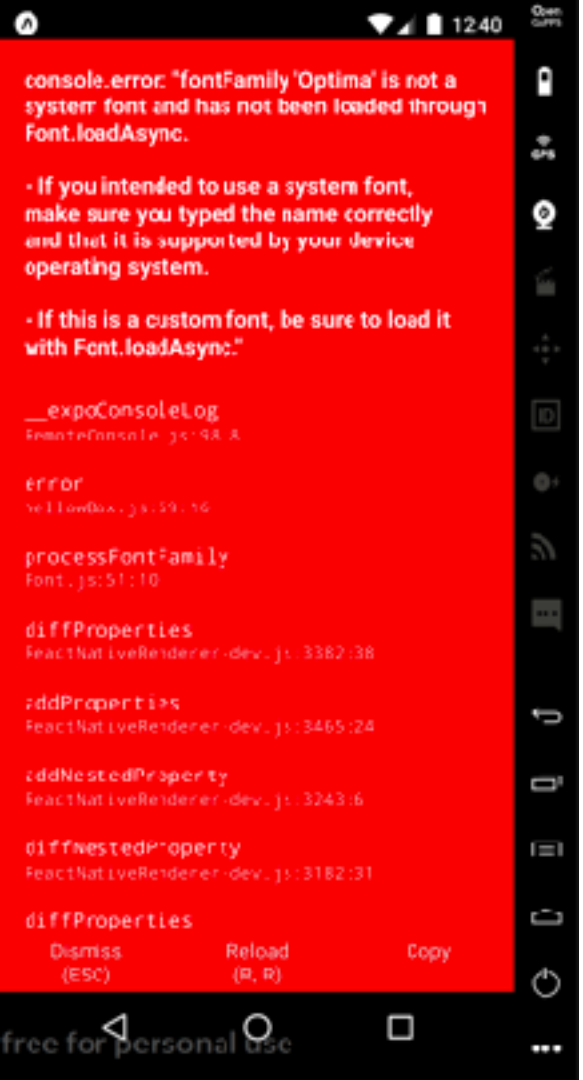
- normal
- notoserif
- sans-serif
- sans-serif-light
- sans-serif-thin
- sans-serif-condensed
- sans-serif-medium
- serif
- Roboto
- monospace

# Fonts on iOS

- Academy Engraved
- Al Nile
- American Typewriter
- Apple Color Emoji
- AppleSDGothicNeo
- Arial
- Arial Hebrew
- Avenir
- Avenir-Roman
- Bangla Sangam MN
- Baskerville
- Bodoni 72
- Bodoni Ornaments
- Bradley Hand
- Chalkboard SE
- Chalkduster
- Cochin
- Copperplate
- Courier
- Damascus
- Devanagari Sangam
- Didot
- DiwanMishafi
- Euphemia UCAS
- Farah
- Futura
- Geeza Pro
- Georgia
- Gill Sans
- Gujarati Sangam MN
- Gurmukhi MN
- Heiti SC
- Helvetica
- Hiragino Mincho ProN
- Hiragino Sans
- Hoefler Text
- Iowan Old Style
- Kailasa
- Kannada Sangam MN
- Khmer Sangam MN
- Kohinoor Bangla
- Lao Sangam MN
- Malayalam Sangam
- Marker Felt
- Menlo
- Mishafi
- Noteworthy
- Optima
- Oriya Sangam MN
- Palatino
- Papyrus
- Party LET
- PingFang HK
- Savoye LET
- Sinhala Sangam MN
- Snell Roundhand
- Symbol
- Tamil Sangam MN
- Telugu Sangam MN
- Thonburi
- Times New Roman
- Trebuchet MS
- Verdana
- Zapf Dingbats
- Zapfino.

# Fonts on both Android and iOS

- ...




console.error: "fontFamily 'Optima' is not a system font and has not been loaded through Font.loadAsync.

- If you intended to use a system font, make sure you typed the name correctly

# To use an unsupported font is an error.

## So how do you use fonts?

```
label: {
 color: 'blue',
 fontFamily: Platform.OS === 'ios' ? 'Optima' : 'serif',
 fontSize: 20,
 fontWeight: 'bold',
},
alert: {
 color: 'red',
 fontFamily: Platform.select({
  ios:'San Francisco',
  android:'sans-serif'}),
 fontSize: 15,
},
```

You use Platform to pick Platform-specific fonts

# What if I want to have the same font on both platforms?

- You can't.  Well ... not natively.

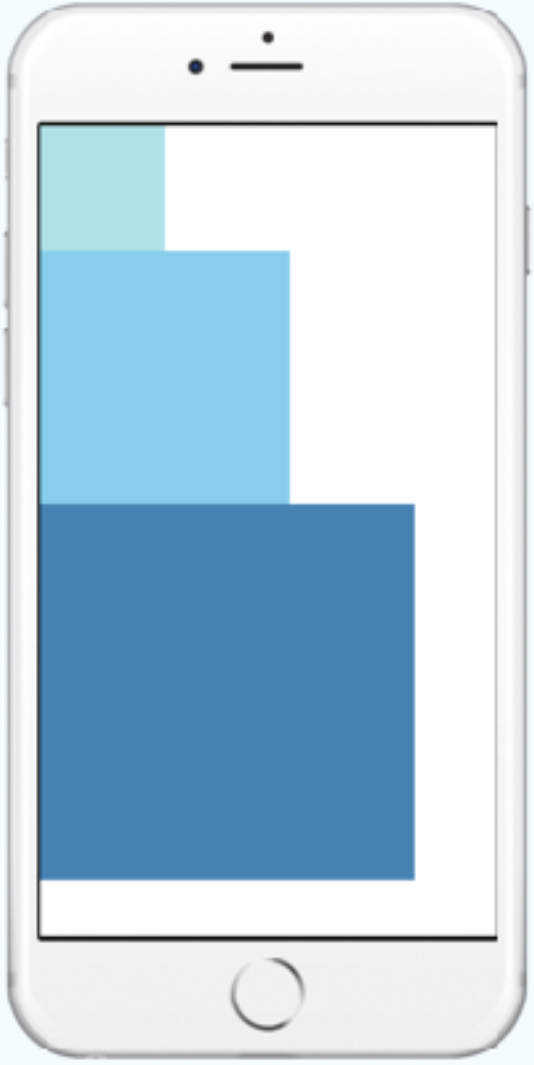
1. Obtain a custom font
2. Install the same font on both platforms using font.loadAsync
3. Use that font

**How to use cross-platform custom fonts:**
**http://bit.ly/RNCustomFonts**

# Sizes are unitless and independent of pixel density

They represent pixels, just not *actual* pixels.

```
<View style={{width: 50, height: 50,
  backgroundColor: 'powderblue'}} />
 <View style={{width:100, height:100,
  backgroundColor: 'skyblue'}} />
 <View style={{width:150, height:150,
  backgroundColor: 'steelblue'}} />
</View>
```



# What values can color take on?

| | |
|---|---|
| 'red' | All the named colors from the W3C spec - about 100 of them |
| '#fff' | #rgb |
| '#fa75ed' | #rrggbb |
| 'rgb(255, 100, 0)' | Red/Green/Blue |
| 'rgba(255, 100, 0, 0.8)' | RGB with alpha channel |
| 'hsl(360, 100%, 50%)' | Hue, saturation, lightness |
| 'hsla(360, 100%, 50%, 0.75)' | HSL with alpha channel |

# You can conditionally style

```
state = {
 danger: true,
}
render() {
 const color = this.state.danger ? 'red' : 'black';
 return (
  <View>
   <Text style={{ color: color }}>
    I like being with you. Really? Did nothing cancel?
   </Text>
  </View>
 );
}
```

**It's just JavaScript, so apply logic.**

# You can conditionally style

```
const Product = ({product}) => {
 const style = [
  product.defaultStyle,
  product.isInStock && {backgroundColor:'green'},
  product.isOnSale && {borderColor:'red', borderWidth:2}
 ];
 return (
  <View style={style}>
    <Text>{product.description}</Text>
  <View>
 )
}
```

**It's just JavaScript, so apply logic.**

# tl;dr

- Native components receive a prop called style which should resolve to a plain JavaScript object which contains keys that look like CSS but are definitely not CSS.
- We can create that style object any way you want, raw, inline, imported, but preferably with StyleSheet.create()
- These can even be applied dynamically using JavaScript logic

# React Native Navigation

... with react-navigation

# tl;dr

- Navigation is giving the user access to other scenes in our app
- It is from a 3rd party but has been adopted by the community
- We declaratively describe the route organization
- You'll navigate using
  - Stack navigators
  - Tab navigators
  - Drawer navigators
- All of them are created imperatively (They're not JSX) but they create components that will be placed in other components

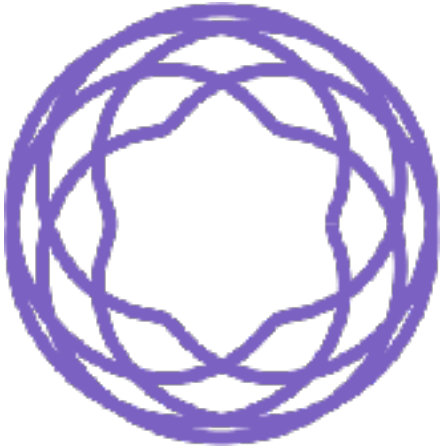
# What is navigation?

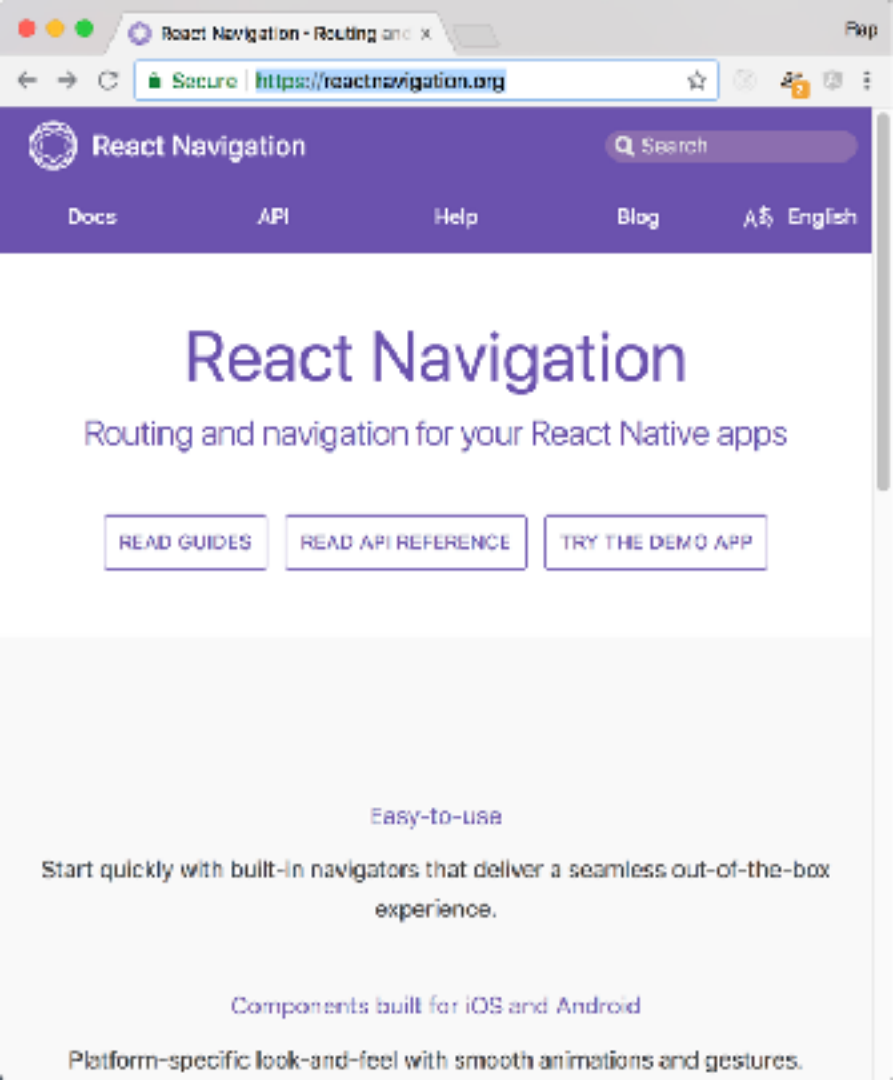
- It is the swapping out of one component for another.
- It makes the user feel like they are moving around in our app.

**There is much less navigation on a device -- you don't swap views out that much. It's just the nature of devices.**

# Navigation

- The community solution to navigation is a standalone library that allows developers to set up the screens of an app with just a few lines of code.






# It comes from ReactNavigation.org

# We must install react-navigation

```
npm install react-navigation
```

# All of the navigators are created imperatively

For example:

```
const dnav = createDrawerNavigator(foo, bar);
```

- They're not in JSX
- They're in JavaScript

# You must also create a container

**App.js**

```
// More stuff up here
const stackNav = createStackNavigator(routes, routeConfig);
const NavContainer = createAppContainer(stackNav);


export default function App() {
return (
   <NavContainer />
 );
}
```

- It's a needed step

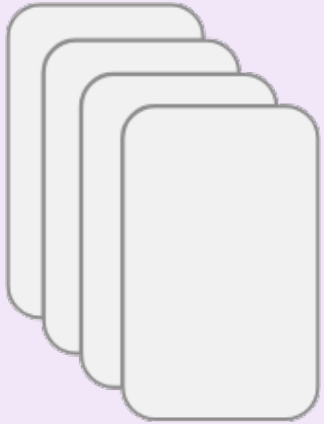
# Three types of navigators

1. StackNavigator

2. TabNavigator

3. DrawerNavigator

# 1. Stack Navigator



A pile of scenes that you push onto or pop off.

# A Stack Navigator is the most fundamental navigation component

- You don't see anything different on the screen. No tabs. No menus. Whatever navigation you want done will be done manually.
- Navigation is merely ...
  1. Forward - Push a new screen on top of the stack (aka. the other screens).
  2. Back - Pop the top screen off the stack, revealing the previous one.

# Make one with createStackNavigator()

This is a React Native component that you host inside another component

```
Nav = createStackNavigator(
routingObject,
configObject)
```

Defines all the components and how we will navigate to them

Holds configuration - modifications to how navigation should work here

# The Routing Object maps routes to components

The simplest routing:

```
{
 Home: Main,
 Contact: Contact,
 Buy: BuyComponent
}
```

More complete routing:

```
{
 Home: {screen:Main},
 Contact: {
  screen: Contact,
 },
 Buy: {
  screen: BuyComponent,
 }
}
```

# The Config Object

The ones you need to know:

```
{
 initialRouteName: Where we should start,
 initialRouteParams: Object for initial draw,
 ... and tons more about gestures, header
configs, and more
}
```

**See the React Navigator docs at http://bit.ly/2OTSrJH for lots more options**

You can create a static property called navigationOptions which will override the universal settings only when you're on that scene

# What if I want different config settings per page?

```
class Foo extends Component {
  static navigationOptions = {
    header: null,
  };
  ... Rest of your component here
}
```

... or ...

**Only navigationOptions can be overridden.**

```
Foo.navigationOptions = {
  title: "We are awesome",
}
```

**App.js**

```
import { Main } from './Main';
import { Contact } from './Contact';
import { Buy } from './Buy';

const Nav = createStackNavigator({
 Home: Main,
 Contact: Contact,
 Buy: Buy
});

export function App() {
 return <View>
  <StatusBar />
  <Nav />
 </View>
}
```

This ...

... can go here

# The component is placed like any other

# Then it's activated with props.navigation.navigate(route)

**Navigable components get a ".navigation" object in their props.**

| | |
|---|---|
| **.navigate(routeName)** | will push that component onto the stack and the user sees it |
| **.goBack()** | Pop off the nav stack |
| **.push()** | Navigate to a route you're already on but with different props |

# So navigation might look like this

```
function Contact(props) {
 const nav = props.navigation;
 return <View>
  <Button title="Purchase"
   onPress={()=> nav.navigate('Buy')} />
  <Button title="Back to home"
   onPress={()=> nav.goBack()} />
 </View>
}
```

# Passing params when navigating

# Remember, When you nest components, you pass values from one to another in props:

Nice and simple

```
function Inner(props) {
  console.log(props);
  return <SomeJSX />
}
function Host() {
 return <Inner foo=1 bar=2 />
}
```

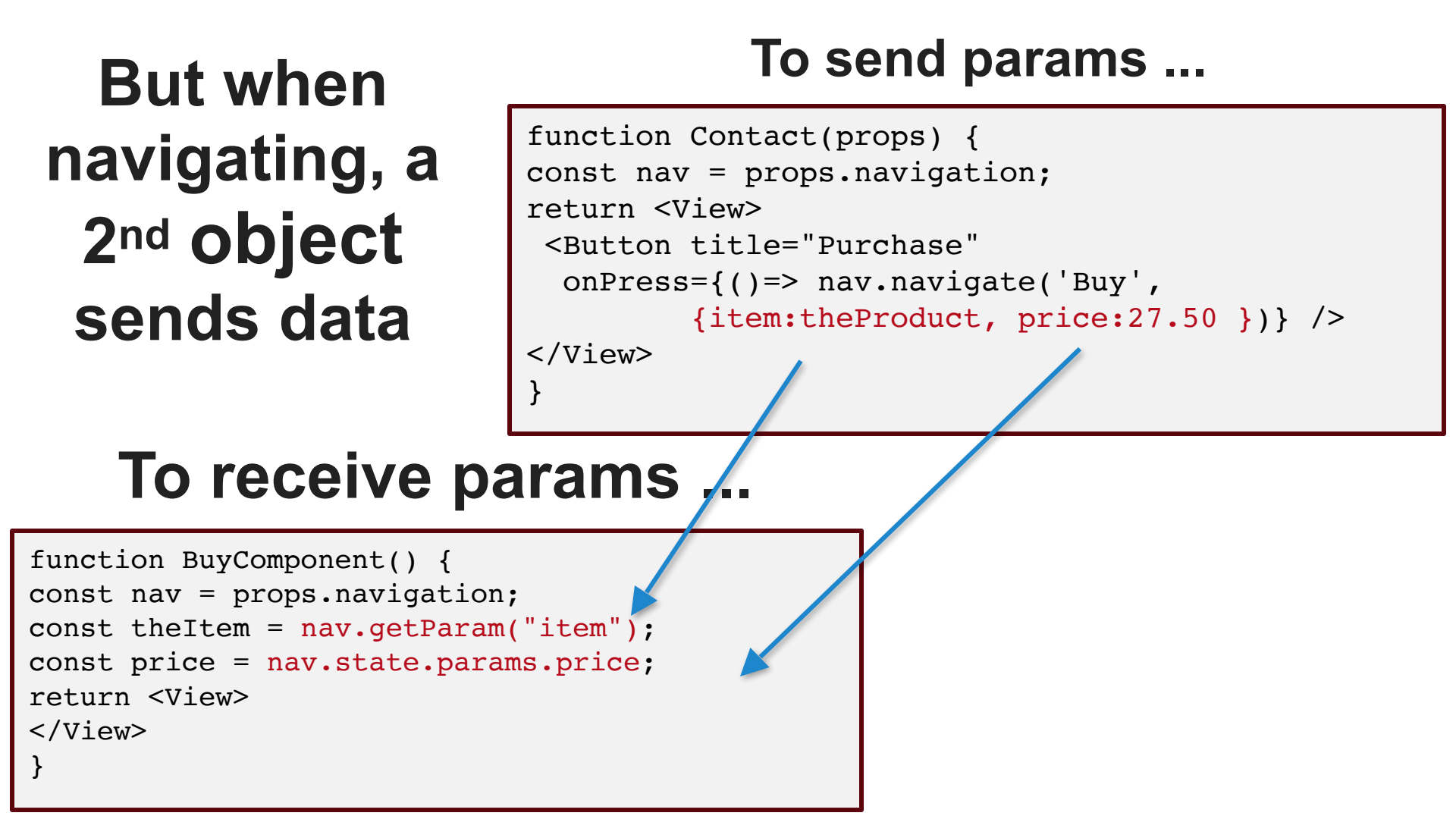
# But when navigating, a 2nd object sends data

## To send params ...

```
function Contact(props) {
const nav = props.navigation;
return <View>
 <Button title="Purchase"
  onPress={()=> nav.navigate('Buy',
         {item:theProduct, price:27.50 })} />
</View>
}
```

## To receive params ...

```
function BuyComponent() {
const nav = props.navigation;
const theItem = nav.getParam("item");
const price = nav.state.params.price;
return <View>
</View>
}
```



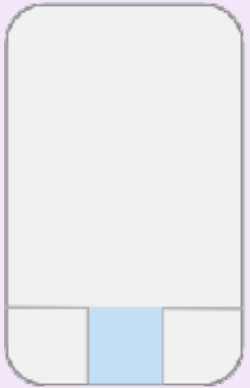
# Since the two techniques are so different ...

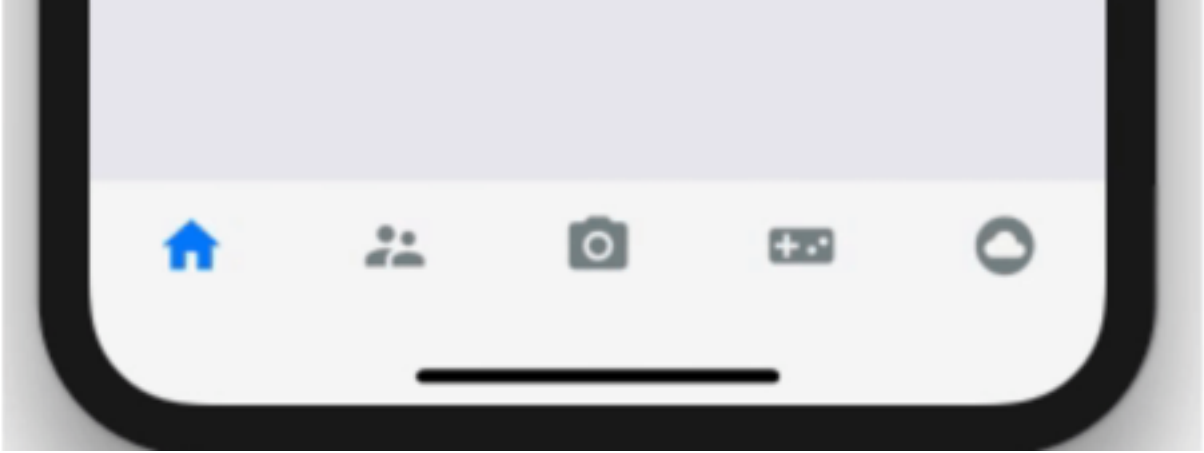
You generally must decide beforehand if a component will be nested or navigated to

But if you **must** do both you can...

```
const {item,price} = props.navigation ?
  props.navigation.state.params : props;
```

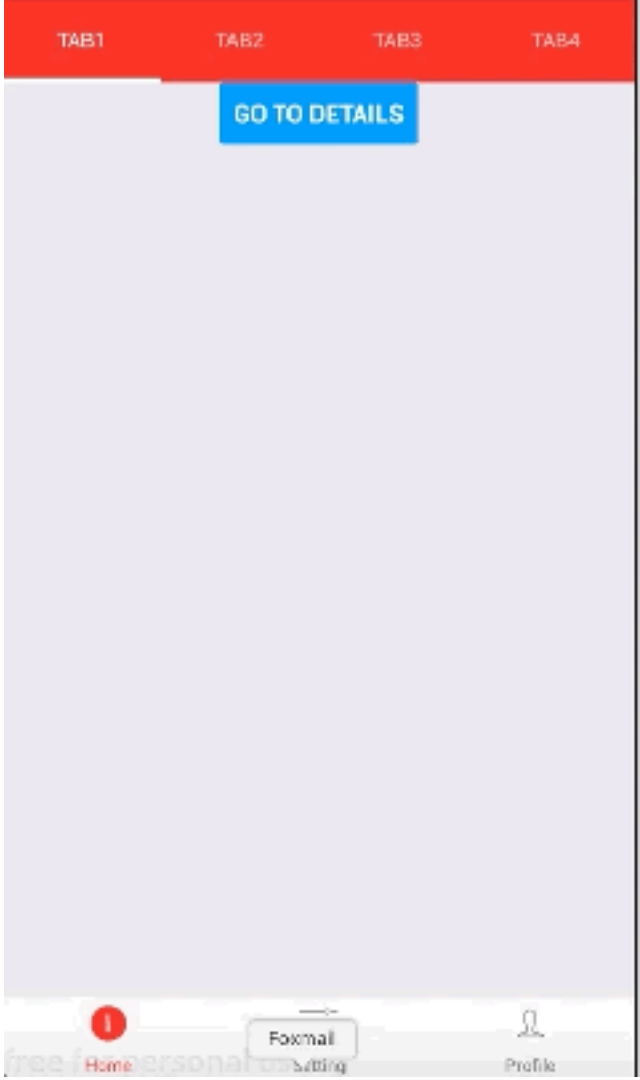
# 2. Tab Navigators



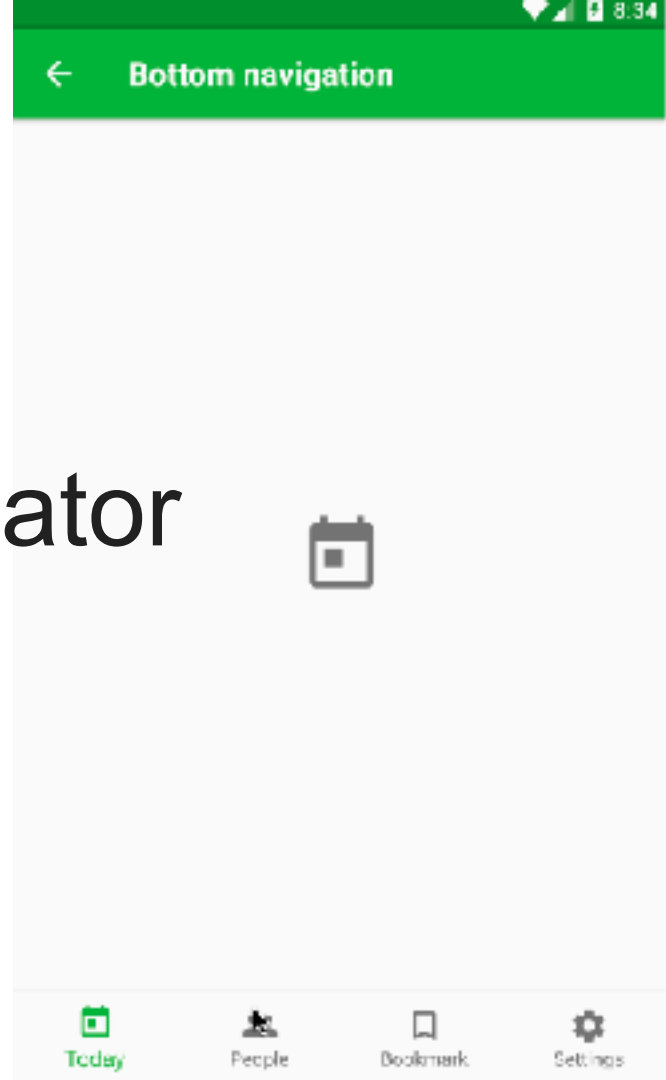


# BottomTabNavigator

# MaterialTopTabNavigator



# MaterialBottomTabNavigator



createBottomTabNavigator(RouteConfigs, NavConfig)

createMaterialBottomTabNavigator(RouteConfigs, NavConfig)

createMaterialTopTabNavigator(RouteConfigs, NavConfig)

```
const App = createBottomTabNavigator({
  Home: { screen: Home },
  Info: { screen: Info }
},{
  tabBarOptions:{
    activeTintColor: '#ff9900',
    // showIcon is only necessary for android
    showIcon: true,
  },
});
```

# Adding Icons

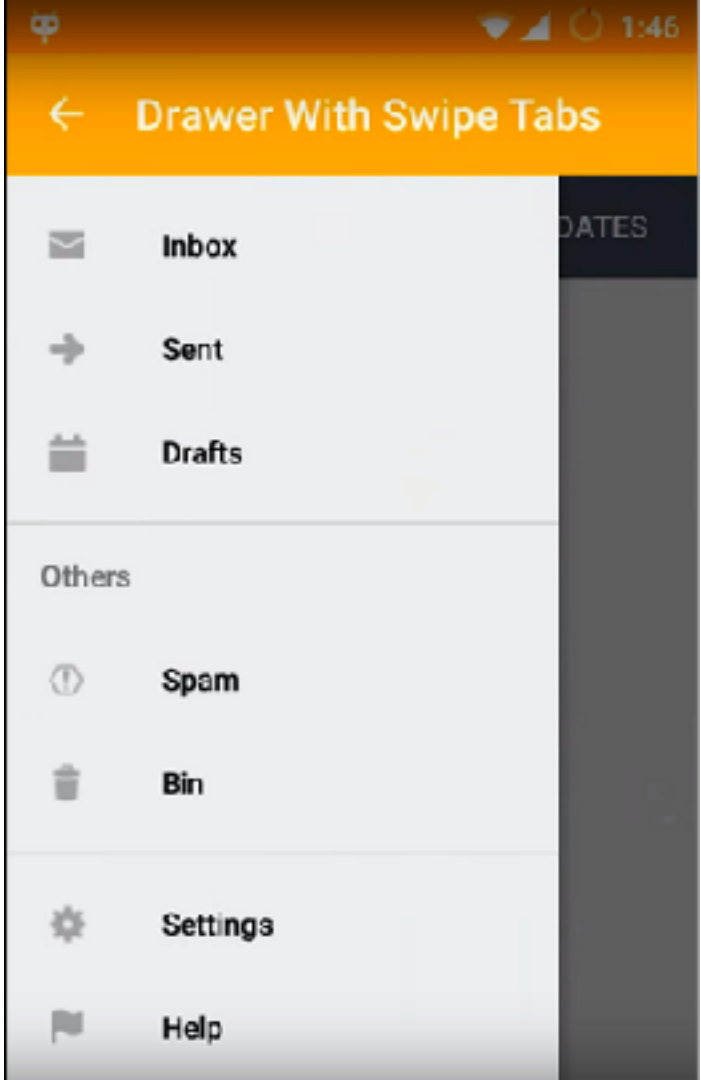
```
const Home = () => (
  <Text>Hello from Home</Text>
)

Home.navigationOptions = {
  tabBarIcon: ({ tintColor }) => (
    <Image
      source={require('./homeicon.png')}
      style={{ width: 24, height: 24, tintColor }}
    />
  ),
}
```

# 3. Drawer Navigator

# DrawerNavigator

- The navigator is hidden, but you can swipe right to reveal it



# Drawer Navigator

Step 1: import createDrawerNavigator into your component

```
import {
  createDrawerNavigator,
} from 'react-navigation';
```

# createDrawerNavigator

createDrawerNavigator(RouteConfigs, DrawerNavigatorConfig)

```
const NavComponent= createDrawerNavigator({
  Home: { screen: Home },
  Info: { screen: Info }
},{
  contentOptions:{
    activeTintColor: 'pink'
  },
});
```

# A simple example

```
const Home = ({ navigation }) => (
  <Text onPress={() => navigation.openDrawer()}>
    Open Drawer</Text>
)

const Info = ({ navigation }) => (
   <Text onPress={() => navigation.closeDrawer()}>
     Close Drawer</Text>
)

const App = createDrawerNavigator({
  Home: { screen: Home },
  Info: { screen: Info }
});
```

# Adding Icons

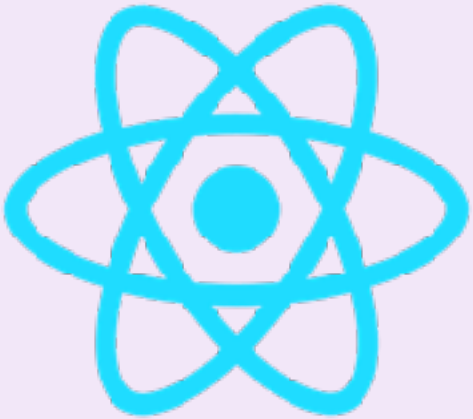
```
const Home = ({ navigation }) => (
  <Text onPress={() => navigation.navigate('DrawerOpen')}>Open Drawer</Text>
)
Home.navigationOptions = {
  drawerIcon: ({ tintColor }) => (
    <Image
      source={require('./homeicon.png')}
      style={{ width: 24, height: 24, tintColor }}
    />
  ),
}
```

# tl;dr

- Navigation is giving the user access to other scenes in our app
- It is from a 3rd party but has been adopted by the community
- We declaratively describe the route organization
- You'll navigate using
  - Stack navigators
  - Tab navigators
  - Drawer navigators
- All of them are created imperatively (They're not JSX) but they create components that will be placed in other components

# Further Study

- React Navigation Docs
  - https://reactnavigation.org/docs/getting-started.html
- Daniel Merril's Medium series
  - http://bit.ly/react-navigation-up-and-running
  - (From where I got the navigator icons)



# Buttons and Touchables

# tl;dr

- A <Button> is easy to understand; you press it and run a method in response.
- But a button only contains text with limited ability to style. If you want more, use Touchables.
- TouchableWithoutFeedback - Generally avoid
- TouchableNativeFeedback - Android only
- TouchableOpacity - Becomes transparent
- TouchableHighlight - Changes color

# Categories of RN components

| Category | Some components |
| --- | --- |
| Layout | Modal, View, SafeAreaView, ScrollView, RefreshControl, KeyboardAvoidingView, StatusBar, WebView |
| Single-value | Text, TextInput, Slider, Switch, Image |
| List | Picker, FlatList, SectionList |
| Touchable | Button, TouchableHighlight, TouchableNativeFeedback, TouchableOpacity, TouchableWithoutFeedback |
| Others | ActivityIndicator, Platform-specific components |

# Buttons

# Some Button Events

| Name | Notes |
| --- | --- |
| onPress | A tap or click (required) |
| onPressIn | Before the press as the user is touching. Like mouseDown |
| onPressOut | After the press. Like mouseUp |
| onLongPress | Usually takes the place of a right-click |
| onLayout | Fires when the scene is being laid out |

# A note about events

- In React, when an event points to a method that doesn't exist, it throws.
- In React Native, it sometimes just silently fails.
- No notification and no traceability. Tough to debug.

# Some Button Props

| Name | Notes |
| --- | --- |
| title | The text in the button |
| disabled | A bool. When true, prevent all interactions. You can't press it. |
| selectable | If false, user can't select text, like for copying for example |
| selectionColor | What the text looks like while being selected |
| adjustFontSizeToFit | A bool. If true, shrink the fontSize until all the text fits in the container. (iOS only) |

# What if I want my button to have a picture in it?

- A button can only hold a title.
- It can't hold an image or structure or anything.
- You want to press it? Wrap it in a touchable.
- Anything can be wrapped in a touchable and begin to behave like a button

This is why we need touchables

# Touchables

# Four types of touchables

1. TouchableWithoutFeedback
2. TouchableNativeFeedback
3. TouchableHighlight
4. TouchableOpacity

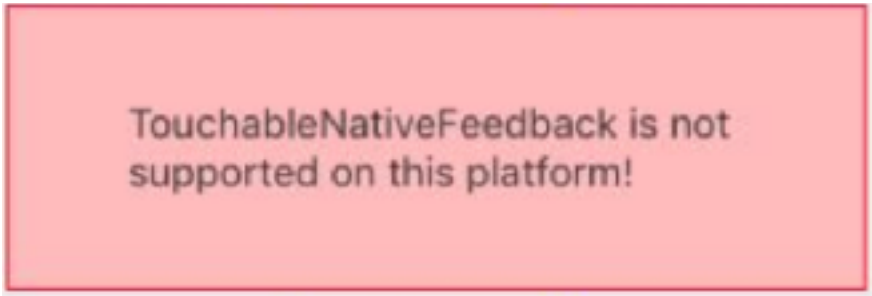
# TouchableWithoutFeedback

- Gives no feedback whatsoever


From the React Native documentation

# TouchableNativeFeedback

- "Ink surface ripples on touch"
- Android only
- on iOS



# TouchableHighlight

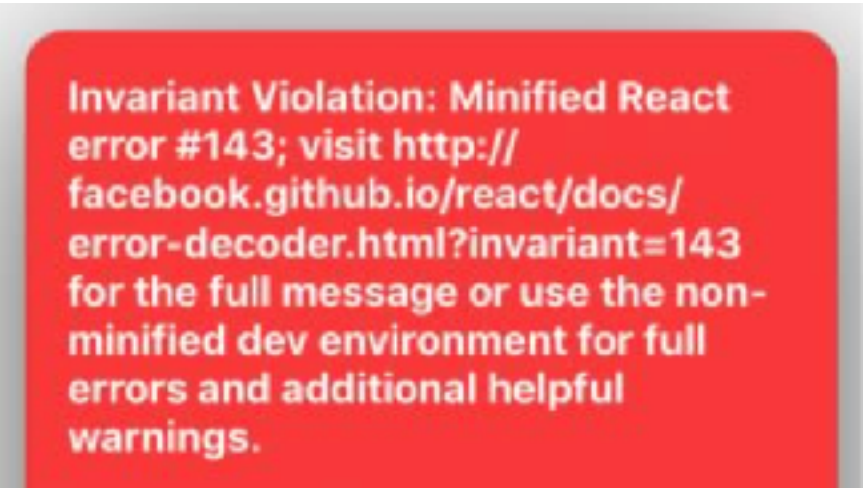
- Background darkens when tapped
- Can only have one child.

# TouchableOpacity

- User can see stuff behind the button while it's being touched

```
<TouchableOpacity
 onPress={handlePress}
 activeOpacity={0.75}>
  <Text>Press me!</Text>
</TouchableOpacity>
```

# There are some surprises




- Two don't honor styles (WithoutFeedback, NativeFeedback)
- WithoutFeedback and Highlight can only hold one inner
- TouchableHighlight doesn't highlight without an onPress event (Even if it doesn't do anything)
- TouchableNativeFeedback breaks on iOS

# Disabling a touchable

# It's not as simple as setting a property

```
<TouchableOpacity disabled={true}>
```

# To manually 'disable' a touchable

```
const { disabled } = this.state;
<TouchableOpacity
 style={[styles.button,
         disabled && disabledStyle]}
 onPress={disabled || handlePress}
 activeOpacity={disabled ? 1.0 : 0.5 }
 >
  <Text>Press me!</Text>
</TouchableOpacity>
```

# tl;dr

- A <Button> is easy to understand; you press it and run a method in response.
- But a button only contains text with limited ability to style. If you want more, use Touchables.
- TouchableWithoutFeedback - Generally avoid
- TouchableNativeFeedback - Android only
- TouchableOpacity - Becomes transparent
- TouchableHighlight - Changes color