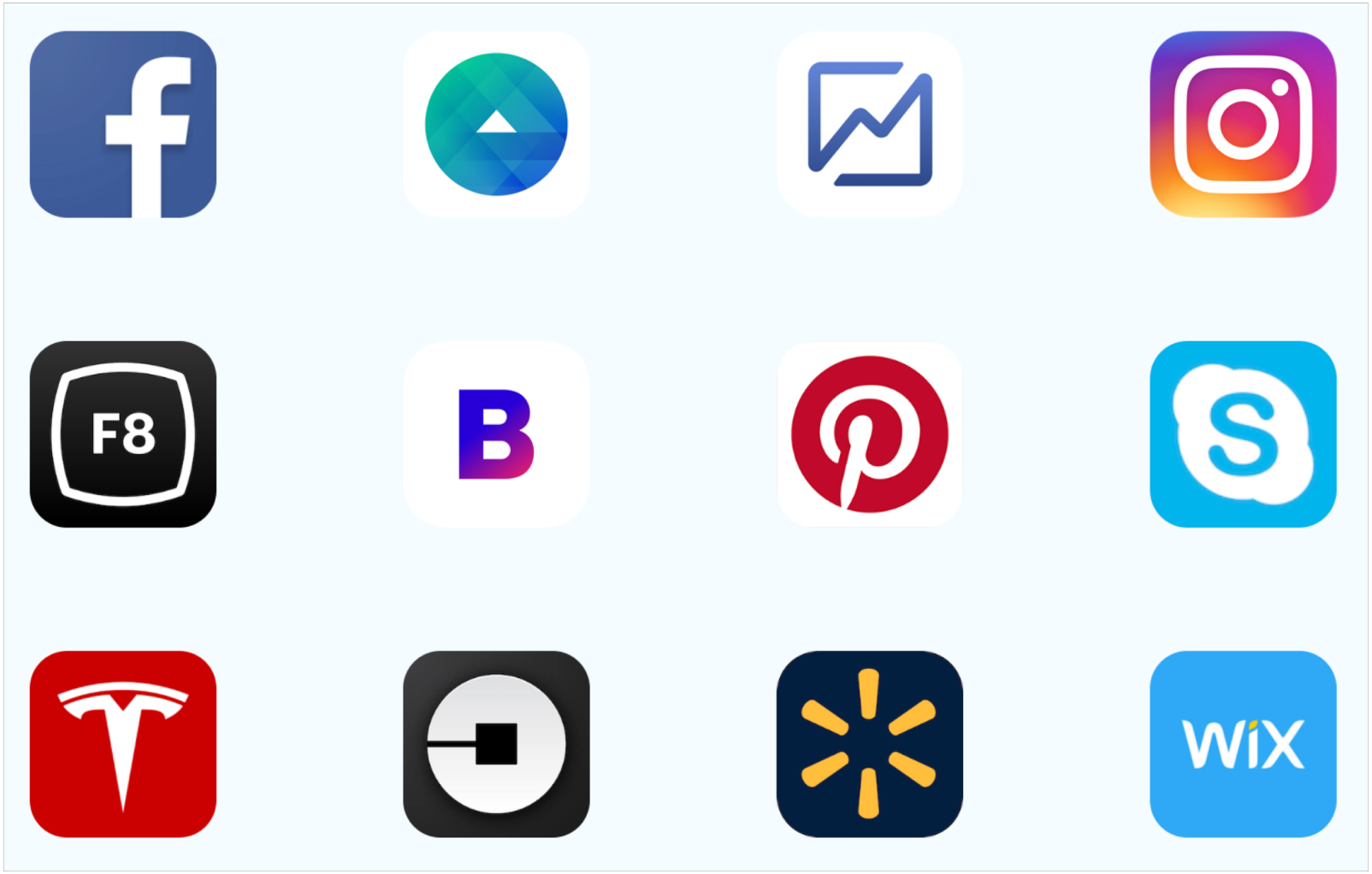
React Native

"An open source, cross-platform framework for building native mobile apps with JavaScript and React using declarative components."

https://facebook.github.io/react-native/

Who is using React Native?



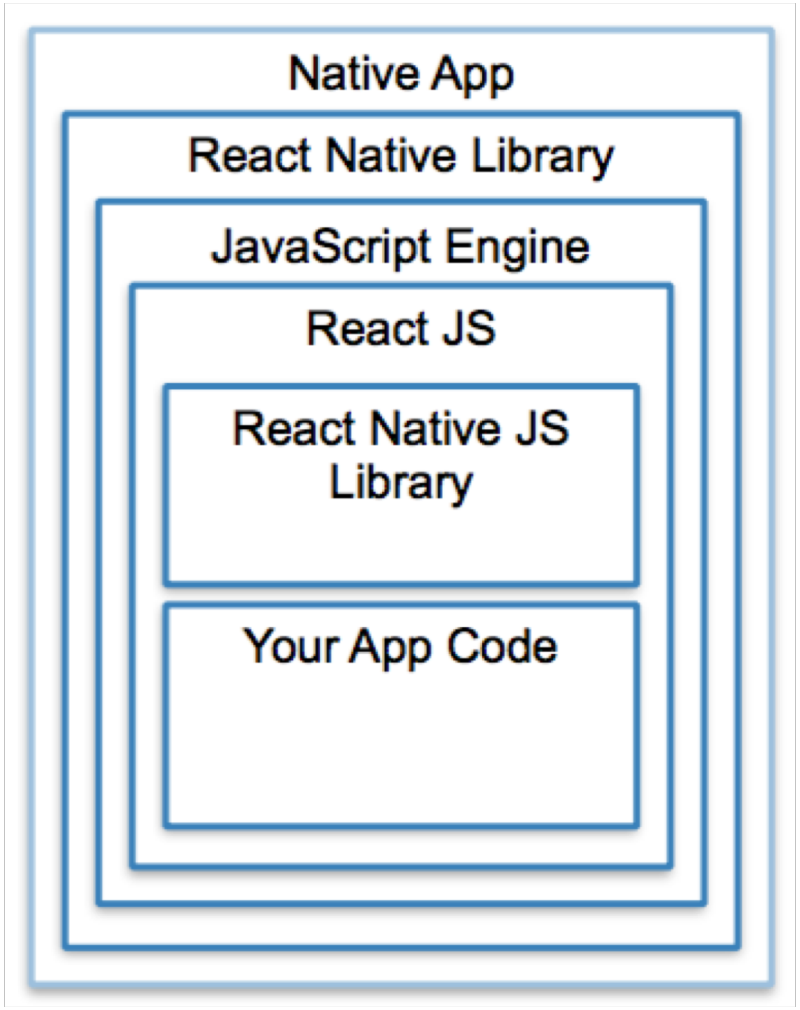
React Native

- React Native (RN) is a framework for building native apps using Javascript
  - Cordova-based apps run inside a webview
  - RN apps render using native views
    - RN apps have direct access to all the native APIs and views offered by the underlying mobile OS
    - RN apps have the same feel and performance as that of a native application
- RN can be considered as a set of React components
  - The developer will be writing the code just like for any other React web app but the output will be a native application

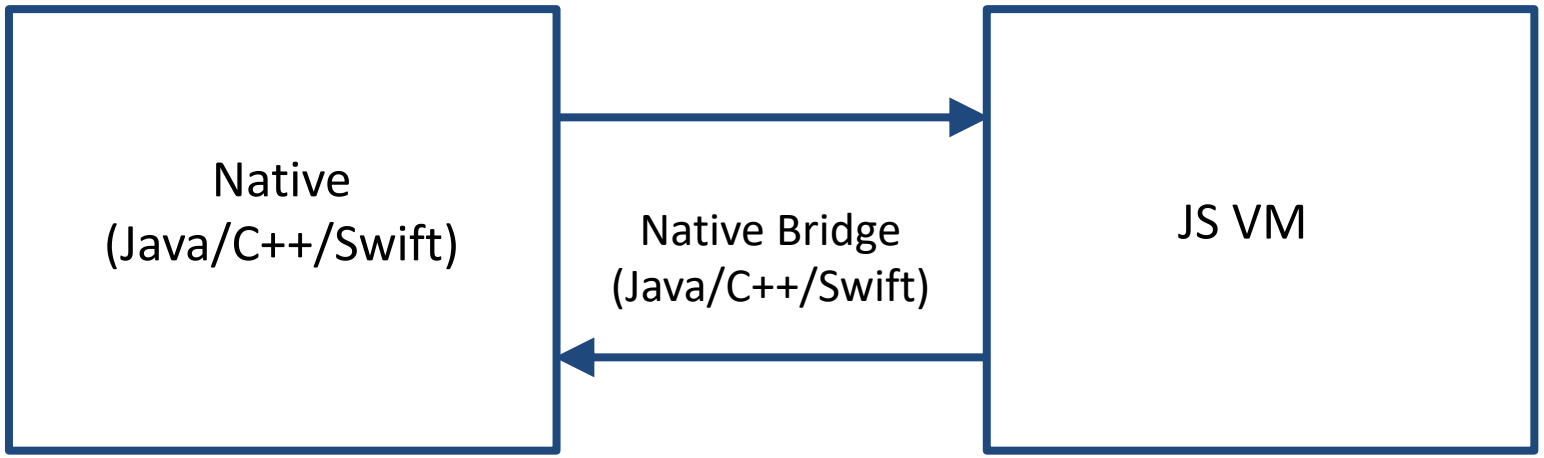
React Native

- Comes with a bunch of atomic components
  - They do not cover all the desidered functionality (e.g, a component for a MapView is not available)
  - To solve this problem leverage the community!
- Most of the atomic components are platform agnostic, but some are not
- We can implement our own atomic components by leveraging RN helpers for Java and Swift

React Native



RN architecture



RN Architecture

- **Native Code/Modules**: Swift and Java code
- **Javascript VM**: RN uses JavaScriptCore (Safari)
  - In case of Android, RN bundles the JavaScriptCore along with the application
  - In case of Chrome debugging mode, RN uses the V8 engine and communicates with native code via WebSocket
- **React Native Bridge**: a C++/Java bridge responsible for communication between the native and Javascript threads
  - A custom protocol is used for message passing

Threading model

- **Main thread** is spawned as soon as the application launches
  - Loads the app and starts the JS thread to execute the JS code
  - Listens to UI events and passes them to the JS thread via the RN Bridge
- **Javascript thread** runs bundled JS/RN code
- **Additional threads** can be spawned on custom native modules to speed up the performance
  - For example, animations are handled in React Native by a separate native thread to offload the work from the JS thread

RN basics

- We need to understand some of the basic React concepts, like JSX, components, state, and props
  - If you already know React, you still need to learn some React-Native-specific stuff, like native components
- RN ships with ES2015 (ES6)
  - For example: import, from, class, and extends
- JSX is a syntax for embedding XML within JavaScript
  - Many frameworks use a special templating language to embed code inside markup language
  - JSX lets you write your markup language inside code
  - You can put any JavaScript expression inside braces in JSX

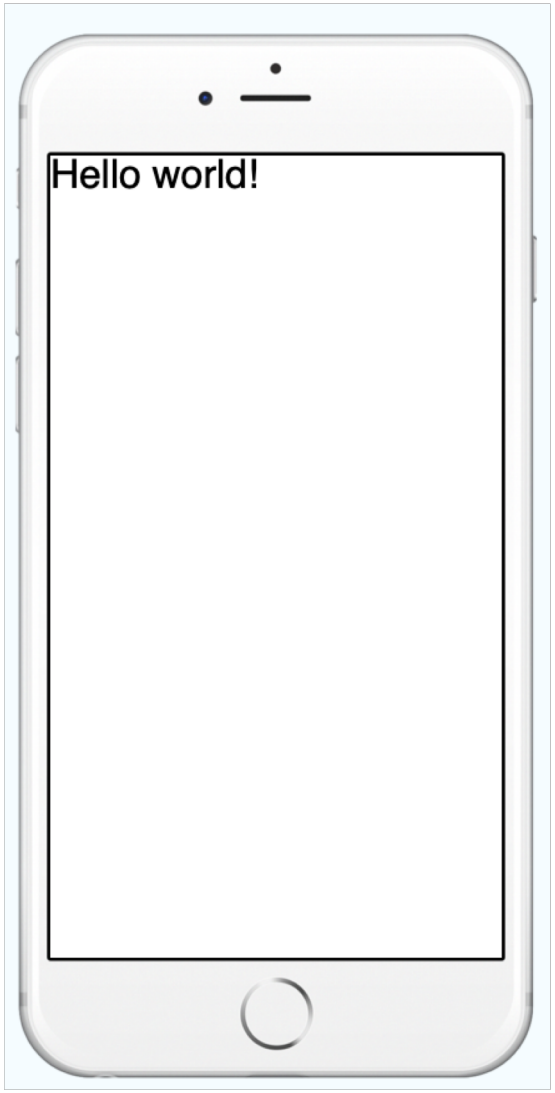
Components as JS classes

- We can define React components by using an ES6 class that extends React.Component

```
import React from 'react';

class MyComponent extends React.Component {
  constructor() {
    super();
    // This constructor defines the state of the component
  }

  render() {
    return(// This defines the reactive UI associated with the component)
  }

  my_method() {
    //This is a custom method
  }
}
```

First example

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

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View>
        <Text>Hello world!</Text>
      </View>
    );
  }
}
```

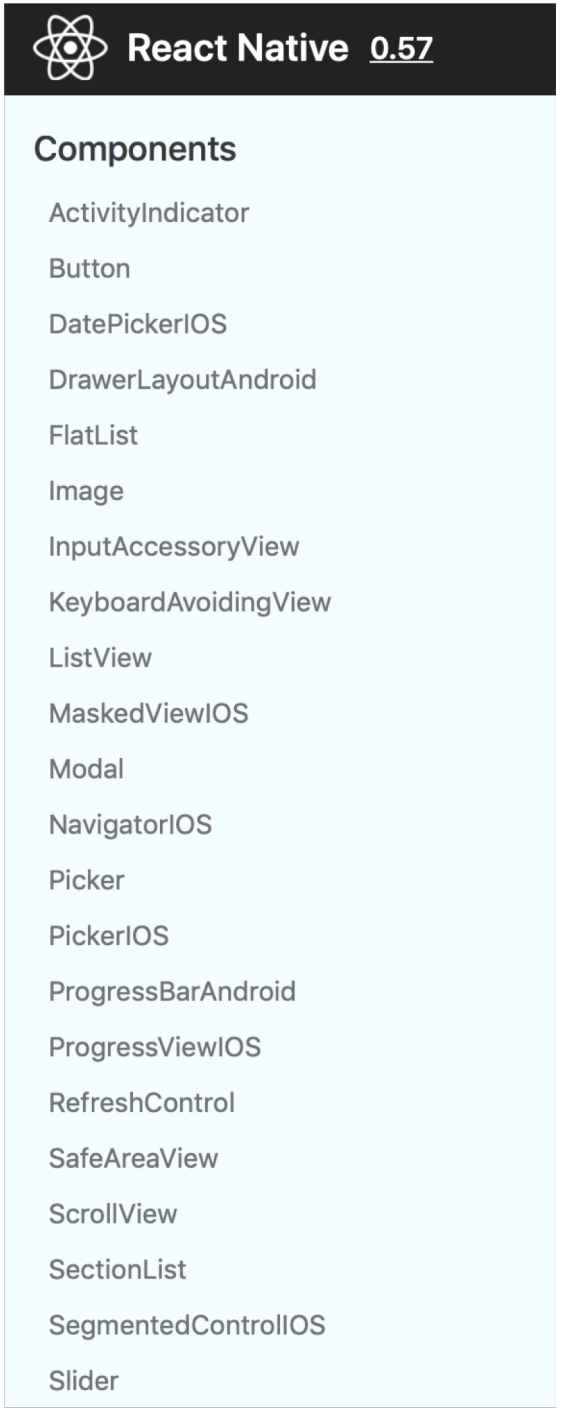


Components

- HelloWorldApp is a new component
- RN apps use many components
  - Anything you see on the screen is some sort of component
- A component can be pretty simple
  - The only thing that is required is a render function to get some JSX to render
- Most components can be customized when they are created
  - These creation parameters are called props

React Native Components

- They have the same characteristics of React ones, but they map to Native UI elements
  - <View> is a generic container of other UI elements
  - <Text> is a piece of text
  - <Image> is an image
  - <Button> is a button
  - <FlatList> is a list that renders only visible elements
  - <Switch> is a switch

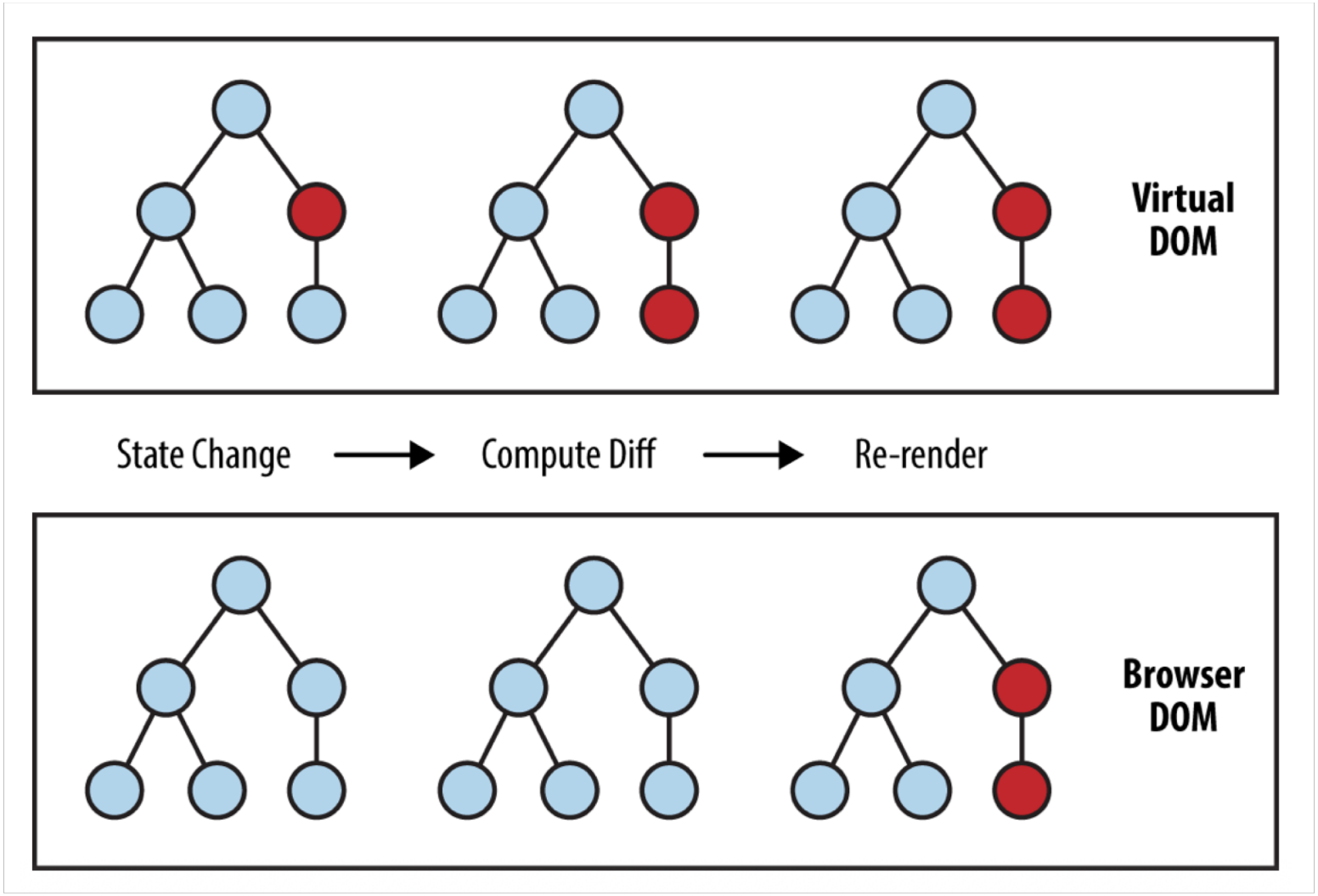


Reactive UI

- Whenever the state of a component or its properties change, the UI associated with the component should be re-rendered
- React maintains a virtual representation of the UI (Virtual DOM) of all the components and uses it to understand what pieces of UI need to be re-rendered
- These pieces of UI are those defined using the state or the props of the application

```
<div>
    <h1>{this.state.title}</h1>
    <p>{this.props.paragraph}</p>
</div>
```

Virtual DOM



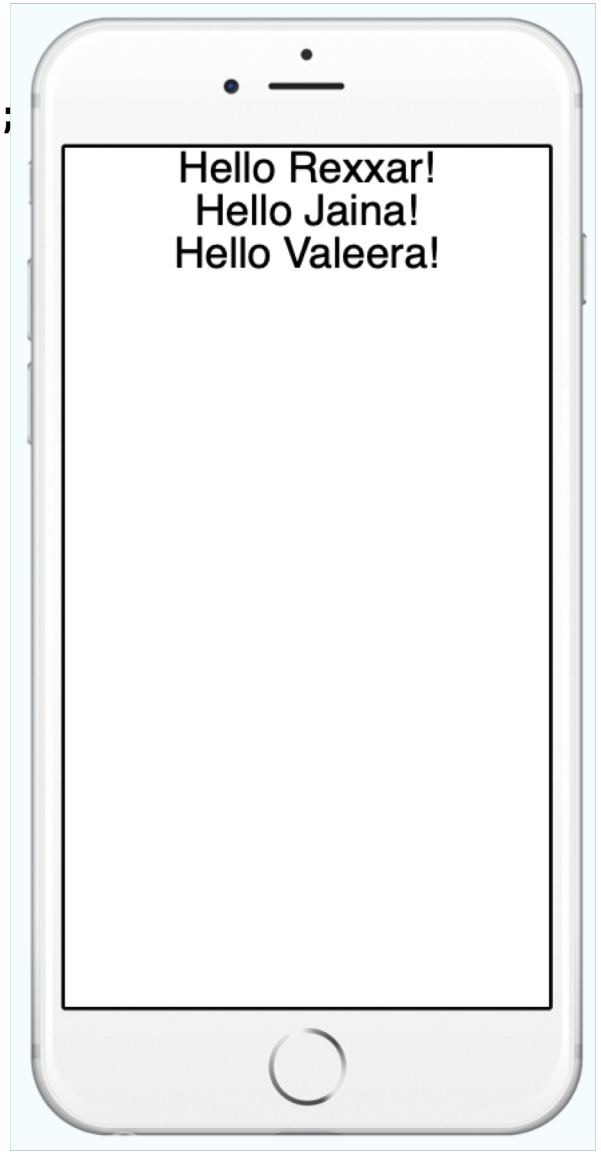
https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html

Example

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

export default class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://www.yoursite.org/bananas.jpg'
    };
    return (
      <Image source={pic} style={{width: 193, height:
    );
  }
}
```

Special purpose components

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

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

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



State

- Props are fixed throughout the lifetime of a component
- State is used for data that is going to change
- We should initialize state in the constructor and then call setState when you want to change it
  - For example, if we want to make text that blinks all the time
    - The text itself is a prop and is set once when the blinking component is created
    - Its blinking capabilities should be kept in state
- You can also use a state container like Redux to control data flow

State

- this.setState merges the current state with the given one
  - It intersects the states and it changes properly the current state
- this.setState is not synchronous
  - React may batch multiple state changes to guarantee better performance

```
constructor() {
      super();
      this.state = {notes : []}
}
```

```
my_method() {
      const myNotes = /*Here you construct the notes you want to set*/
      this.setState({notes : myNotes});
}
```

```
this.state = { notes : [note1, note2],
                    filter :  'NONE'}
this.setState({notes : [note3]})
```

→

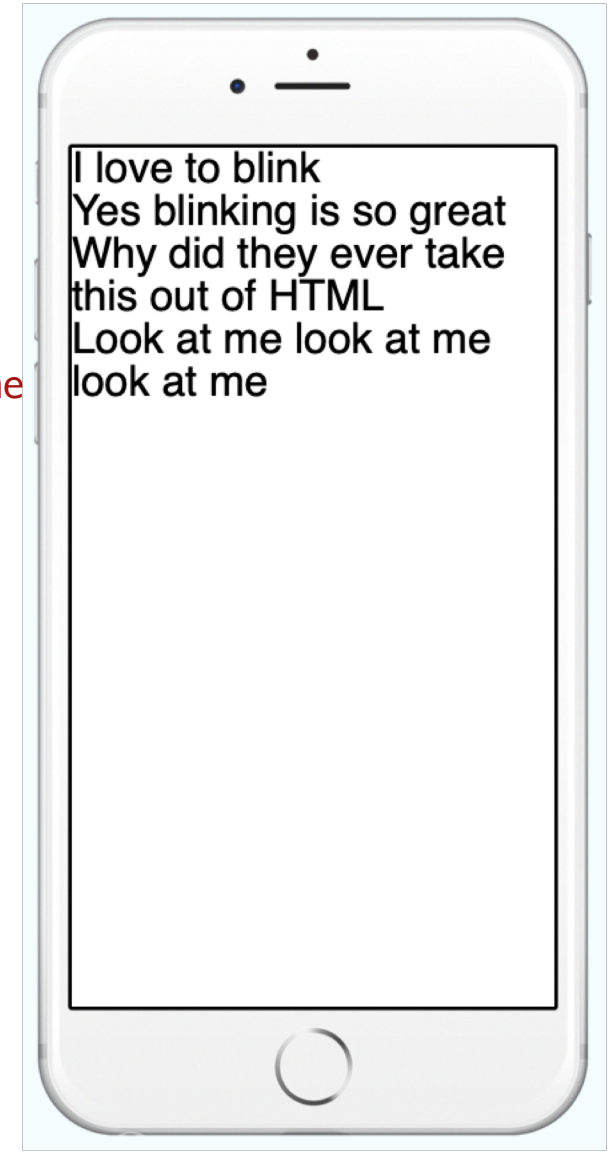
```
{
      notes : [note3],
      filer: 'NONE'
}
```

Example (I)

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

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

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

  render() {
    let display = this.state.isShowingText ? this.props.text : ' ‘;
    return (
      <Text>{display}</Text>
    );
  }
}
```

Example (II)

```
export default class BlinkApp extends Component {
  render() {
    return (
      <View>
        <Blink text='I love to blink' />
        <Blink text='Yes blinking is so great' />
        <Blink text='Why did they ever take this out
        <Blink text='Look at me look at me look at me
      </View>
    );
  }
}
```

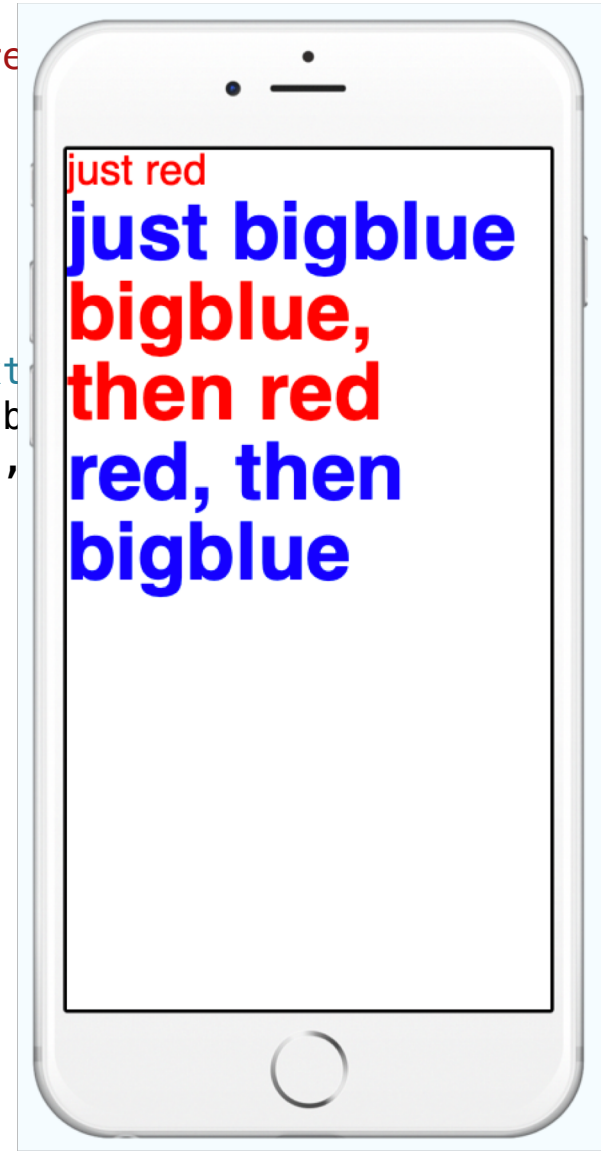


Style

- All core components accept a prop named style
  - Style names and values usually match how CSS works, but names are written using camel casing (e.g backgroundColor rather than background-color)
- Prop style can be a plain old JavaScript object
- It is often cleaner to use StyleSheet.create to define several styles in one place
- Styles can "cascade" the way they do in CSS

Example

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View } from 're

export default class LotsOfStyles extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>just red</Text>
        <Text style={styles.bigblue}>just bigblue</Text
        <Text style={[styles.bigblue, styles.red]}>bigb
        <Text style={[styles.red, styles.bigblue]}>red,
      </View>
    );
  }
}

const styles = StyleSheet.create({
  bigblue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});
```



The Basics

Getting Started
Learn the Basics
Props
State
Style
Height and Width
Layout with Flexbox
Handling Text Input
Handling Touches
Using a ScrollView
Using List Views
Networking
More Resources

Guides

Components and APIs
Platform Specific Code
Navigating Between Screens
Images
Animations
Accessibility
Improving User Experience

Text

EDIT

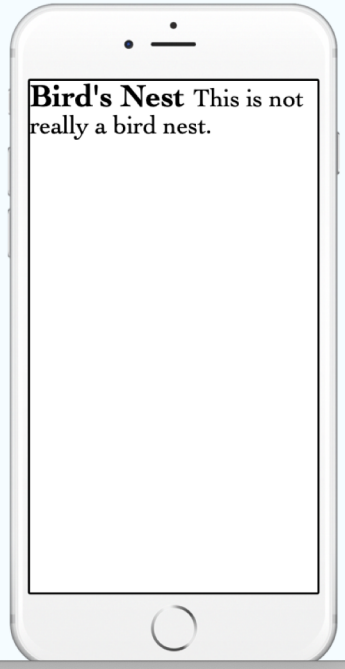
A React component for displaying text.

`Text` supports nesting, styling, and touch handling.

In the following example, the nested title and body text will inherit the `fontFamily` from `styles.baseText`, but the title provides its own additional styles. The title and body will stack on top of each other on account of the literal newlines:

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

export default class TextInANest extends Component {
  constructor(props) {
    super(props);
    this.state = {
      titleText: "Bird's Nest",
      bodyText: 'This is not really a bird nest.'
    };
  }

  render() {
    return (
      <Text style={styles.baseText}>
        <Text style={styles.titleText} onPress={this.onPressTitl
          {this.state.titleText}{'\n'}{'\n'}
        </Text>
        <Text numberOfLines={5}>
```

No Errors Show Details

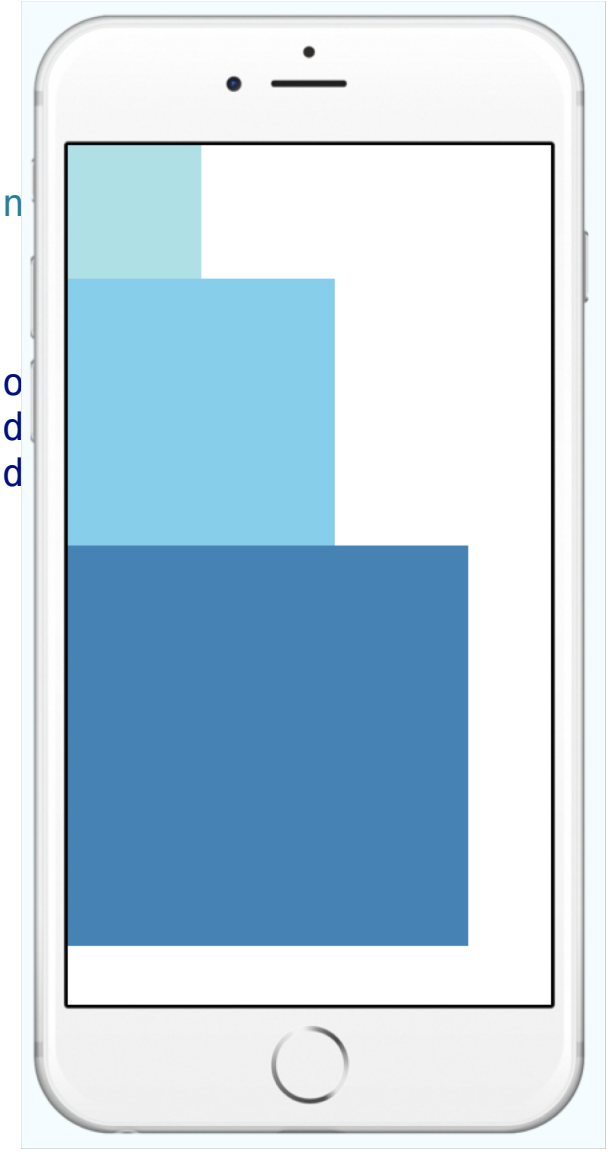


Component size

- Fixed dimensions are set through fixed width and height to style
  - All dimensions in React Native are unitless, and represent density-independent pixels
- flex allows a component to expand and shrink dynamically
  - flex: 1 tells a component to fill all available space
  - The higher flex is, the higher the ratio of space a component will take compared to its siblings is
  - A component can expand if its parent has dimensions greater than 0
    - If a parent does not have either a fixed width and height or flex, the parent will have dimensions of 0 and the flex children will not be visible

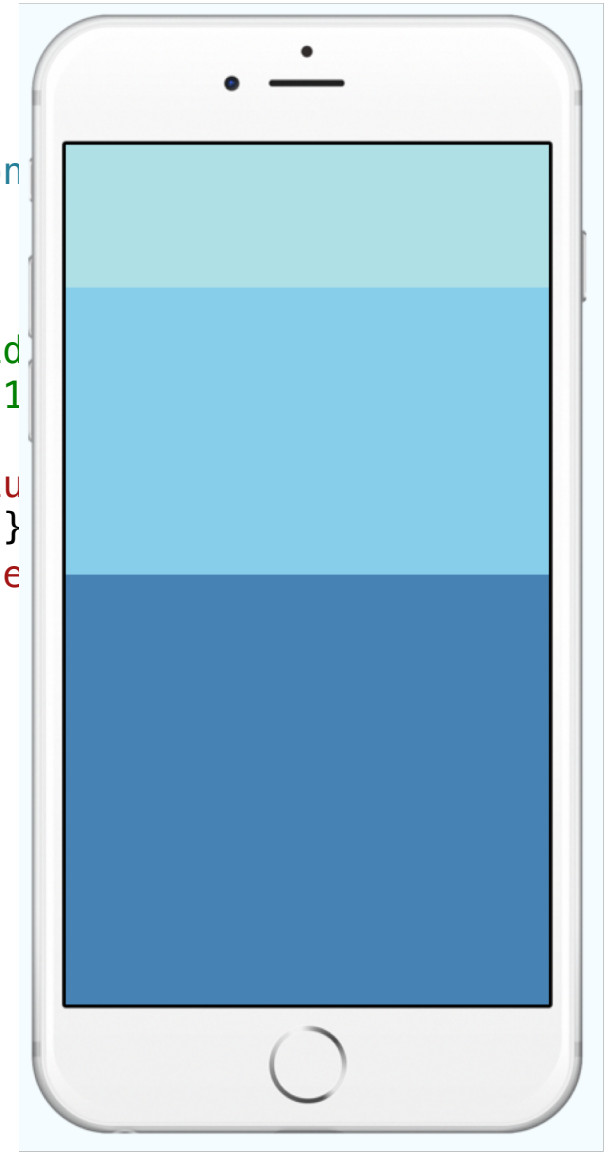
Fixed dimensions

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

export default class FixedDimensionsBasics extends Compon
  render() {
    return (
      <View>
        <View style={{width: 50, height: 50, backgroundCo
        <View style={{width: 100, height: 100, background
        <View style={{width: 150, height: 150, background
      </View>
    );
  }
}
```



Flex dimensions

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

export default class FlexDimensionsBasics extends Compon
  render() {
    return (
    // Try removing the `flex: 1` on the parent View.
    // The parent will not have dimensions, so the child
    // What if you add `height: 300` instead of `flex: 1
    <View style={{flex: 1}}>
      <View style={{flex: 1, backgroundColor: 'powderblu
      <View style={{flex: 2, backgroundColor: 'skyblue'}
      <View style={{flex: 3, backgroundColor: 'steelblue
    </View>
    );
  }
}
```

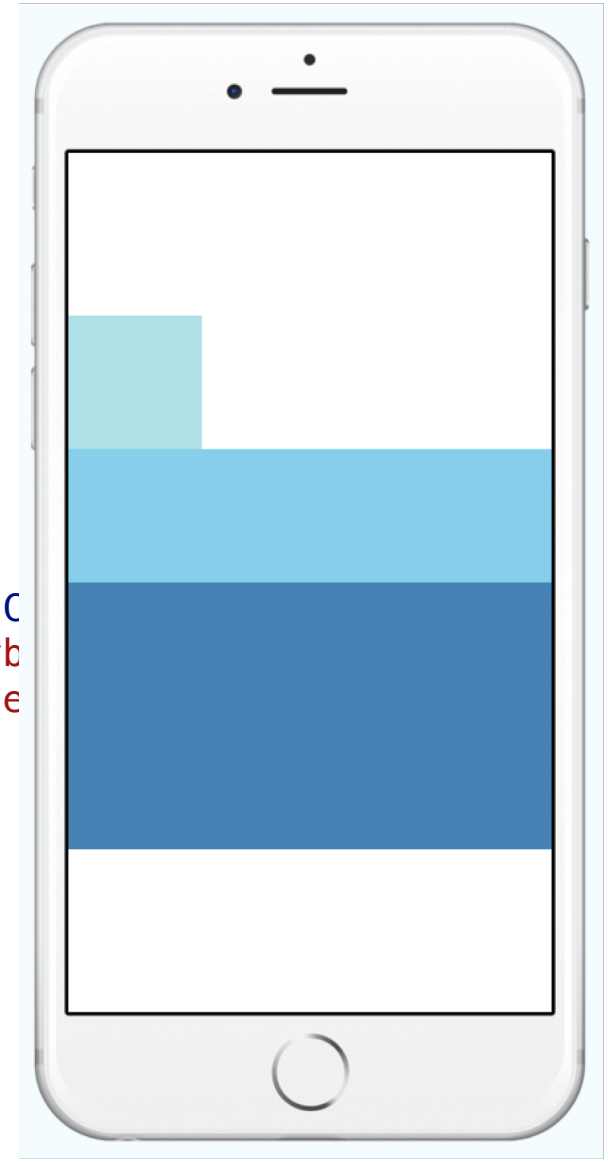


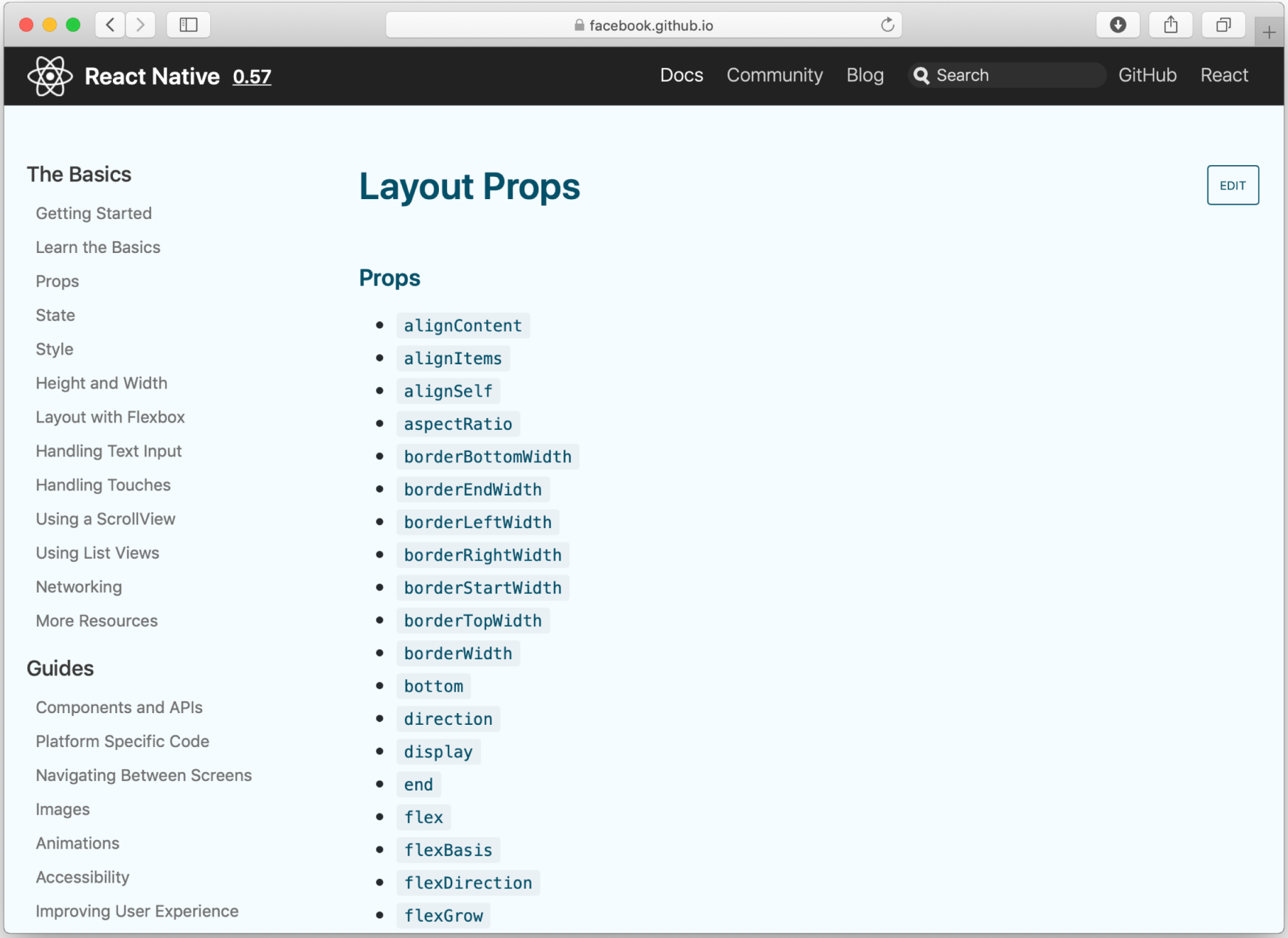
Layout with Flexbox

- Flexbox is designed to provide a consistent layout on different screen sizes
- flexDirection determines the primary axis of its layout
  - row or column, the default is column
- justifyContent determines the distribution of children along the primary axis
  - Available options are flex-start, center, flex-end, space-around, space-between and space-evenly
- alignItems determines the alignment of children along the secondary axis
  - Available options are flex-start, center, flex-end, and stretch

Example

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

export default class AlignItemsBasics extends Component
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'stretch',
      }}>
        <View style={{width: 50, height: 50, backgroundC
        <View style={{height: 50, backgroundColor: 'skyb
        <View style={{height: 100, backgroundColor: 'ste
      </View>
    );
  }
};
```

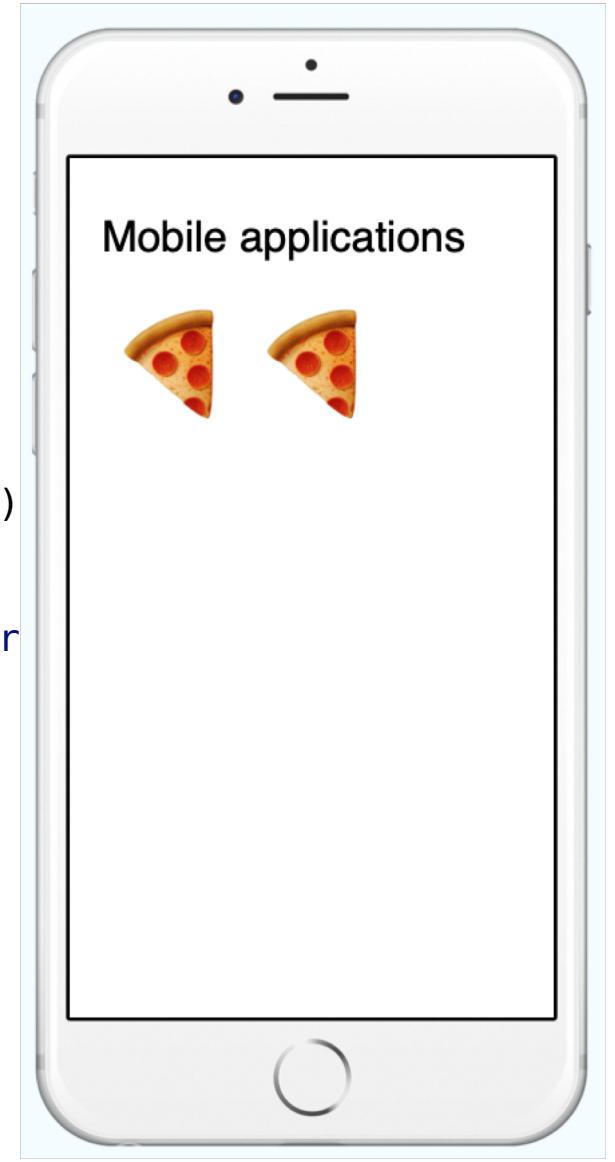



facebook.github.io
React Native 0.57
Docs
Community
Blog
Search
GitHub
React
The Basics
Getting Started
Learn the Basics
Props
State
Style
Height and Width
Layout with Flexbox
Handling Text Input
Handling Touches
Using a ScrollView
Using List Views
Networking
More Resources
Guides
Components and APIs
Platform Specific Code
Navigating Between Screens
Images
Animations
Accessibility
Improving User Experience
Layout Props
EDIT
Props
alignContent
alignItems
alignSelf
aspectRatio
borderBottomWidth
borderEndWidth
borderLeftWidth
borderRightWidth
borderStartWidth
borderTopWidth
borderWidth
bottom
direction
display
end
flex
flexBasis
flexDirection
flexGrow

Handling Text Input

- TextInput is a basic component that allows the user to enter text
  - Prop onChangeText takes a function called every time the text is changed
  - Prop onSubmitEditing takes a function called when the text is submitted
- More on controlled components

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

export default class PizzaTranslator extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ''};
  }

  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Type here to translate!"
          onChangeText={(text) => this.setState({text})
        />
        <Text style={{padding: 10, fontSize: 42}}>
          {this.state.text.split(' ').map((word) => wor
        </Text>
      </View>
    );
  }
}
```



Handling Touches

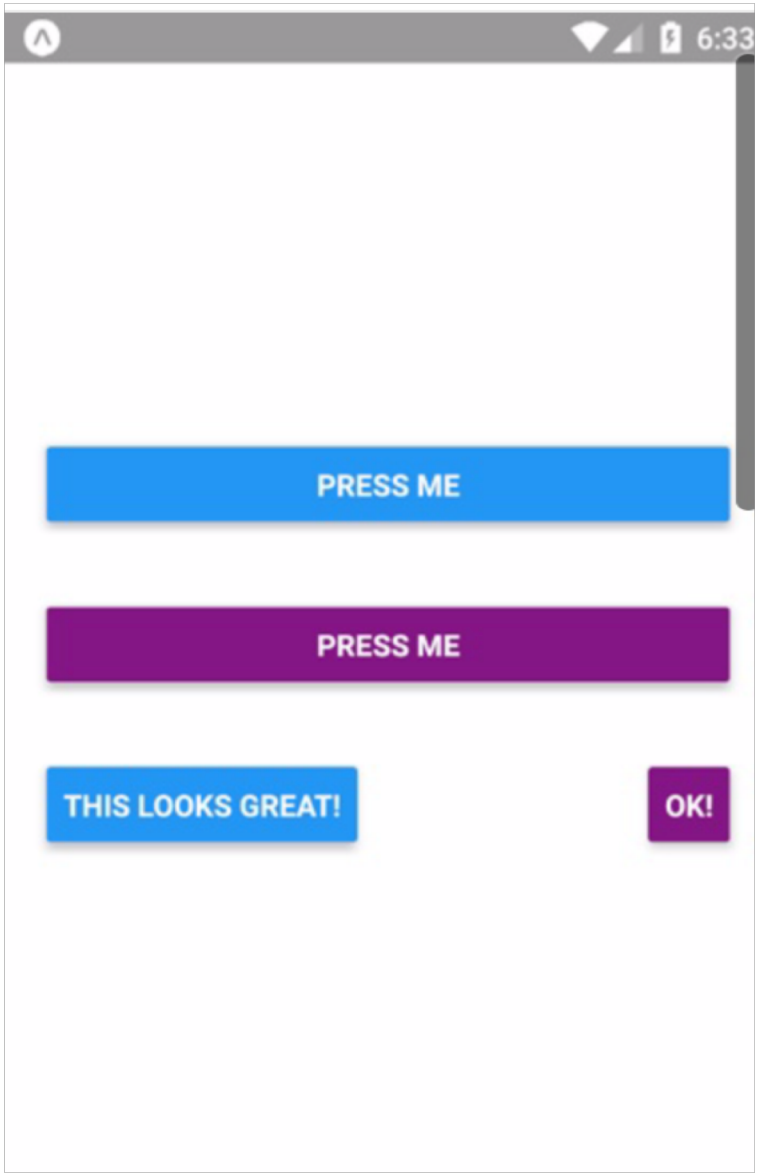
- React Native provides components to handle all sorts of common gestures
- Button provides a basic button component that is rendered nicely on all platforms

```
<Button
  onPress={() => {
    Alert.alert('You tapped the button!');
  }}
  title="Press Me"
/>
```

- This renders a blue label on iOS, and a blue rounded rectangle with white text on Android
  - We can specify a prop color to change its color

```
import React, { Component } from 'react';
import { Alert, AppRegistry, Button, StyleSheet, View } from 'react-native';

export default class ButtonBasics extends Component {
  _onPressButton() {
    Alert.alert('You tapped the button!')
  }

  render() {
    return (
      <View style={styles.container}>
        <View style={styles.buttonContainer}>
          <Button
            onPress={this._onPressButton}
            title="Press Me"
          />
        </View>
        <View style={styles.buttonContainer}>
          <Button
            onPress={this._onPressButton}
            title="Press Me"
            color="#841584"
          />
        </View>
```

```
        </View>
          <View style={styles.alternativeLayoutButtonContainer}>
            <Button
              onPress={this._onPressButton}
              title="This looks great!"
            />
            <Button
              onPress={this._onPressButton}
              title="OK!"
              color="#841584"
            />
          </View>
        </View>
      );
    }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  buttonContainer: {
    margin: 20
  },
  alternativeLayoutButtonContainer: {
    margin: 20,
    flexDirection: 'row',
    justifyContent: 'space-between'
  }
});
```



Touchables

- We can build our own button using any of the "Touchable" components provided by RN
  - They provide the capability to capture tapping gestures, and can display feedback when a gesture is recognized
  - They do not provide any default styling
- Long presses can be handled by passing a function to prop onLongPress

We can use

- **TouchableHighlight** anywhere you would use a button or web link
- **TouchableNativeFeedback** on Android to display ink surface reaction ripples that respond to the user's touch
- **TouchableOpacity** to provide feedback by reducing the opacity of the button
- **TouchableWithoutFeedback** to handle a tap gesture but without displaying any feedback

ScrollView

- ScrollView is a generic scrolling container that can host multiple components and views
  - Scrollable items need not be homogeneous
  - We can scroll both vertically and horizontally (by setting the horizontal property)
- ScrollViews can be configured to allow paging through views using swiping gestures by using prop pagingEnabled
- A ScrollView with a single item can be used to allow the user to zoom content
  - Props maximumZoomScale and minimumZoomScale allow the user to use pinch and expand gestures to zoom in and out

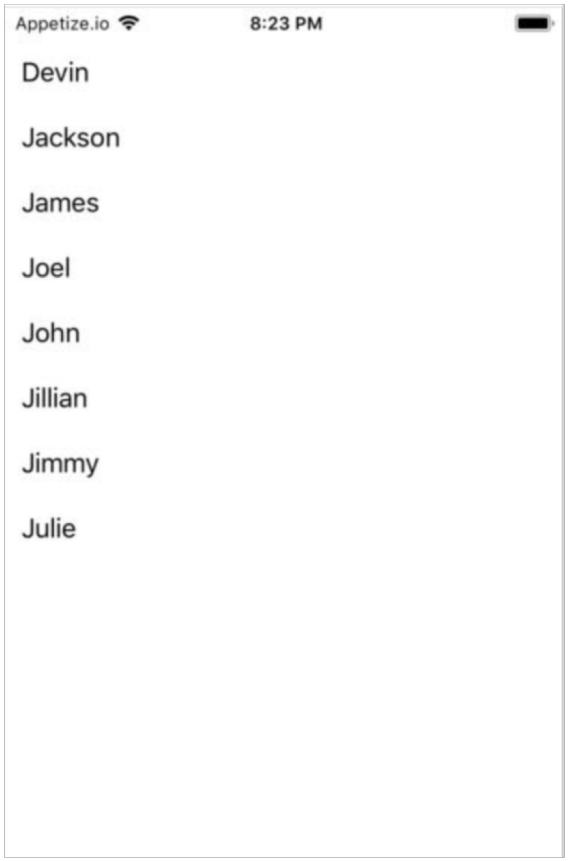
Example

```
render() {
  return (
    <ScrollView>
      <Text style={{fontSize:96}}>Scroll me plz</Text>
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Text style={{fontSize:96}}>If you like</Text>
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Image source={{uri: "https://.../image.png", width: 64, height: 64}} />
      <Text style={{fontSize:96}}>Scrolling down</Text>
    </ScrollView>
  );
}
```

List Views

- FlatList displays a scrolling list of changing, but similarly structured, data
  - It works well for long lists of data, where the number of items might change over time
  - It only renders elements that are currently showing on the screen, not all the elements at once
- It requires two props
  - data is the source of information for the list
  - renderItem takes one item from the source and returns a formatted component to render
- SectionList renders a set of data broken into logical sections
  - Maybe with section headers

```
export default class FlatListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <FlatList
          data={[{key: 'Devin'}, {key: 'Jackson'}, {key: 'James'}, {key: 'Joel'},
                 {key: 'John'}, {key: 'Jillian'}, {key: 'Jimmy'}, {key: 'Julie'},
                ]}
          renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
        />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 22
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
})
```



Networking

- RN provides the Fetch API for networking
  - Similar to XMLHttpRequest or other networking APIs
- Networking is an inherently asynchronous operation
  - Fetch methods return a Promise that makes it straightforward to write code that works asynchronously

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

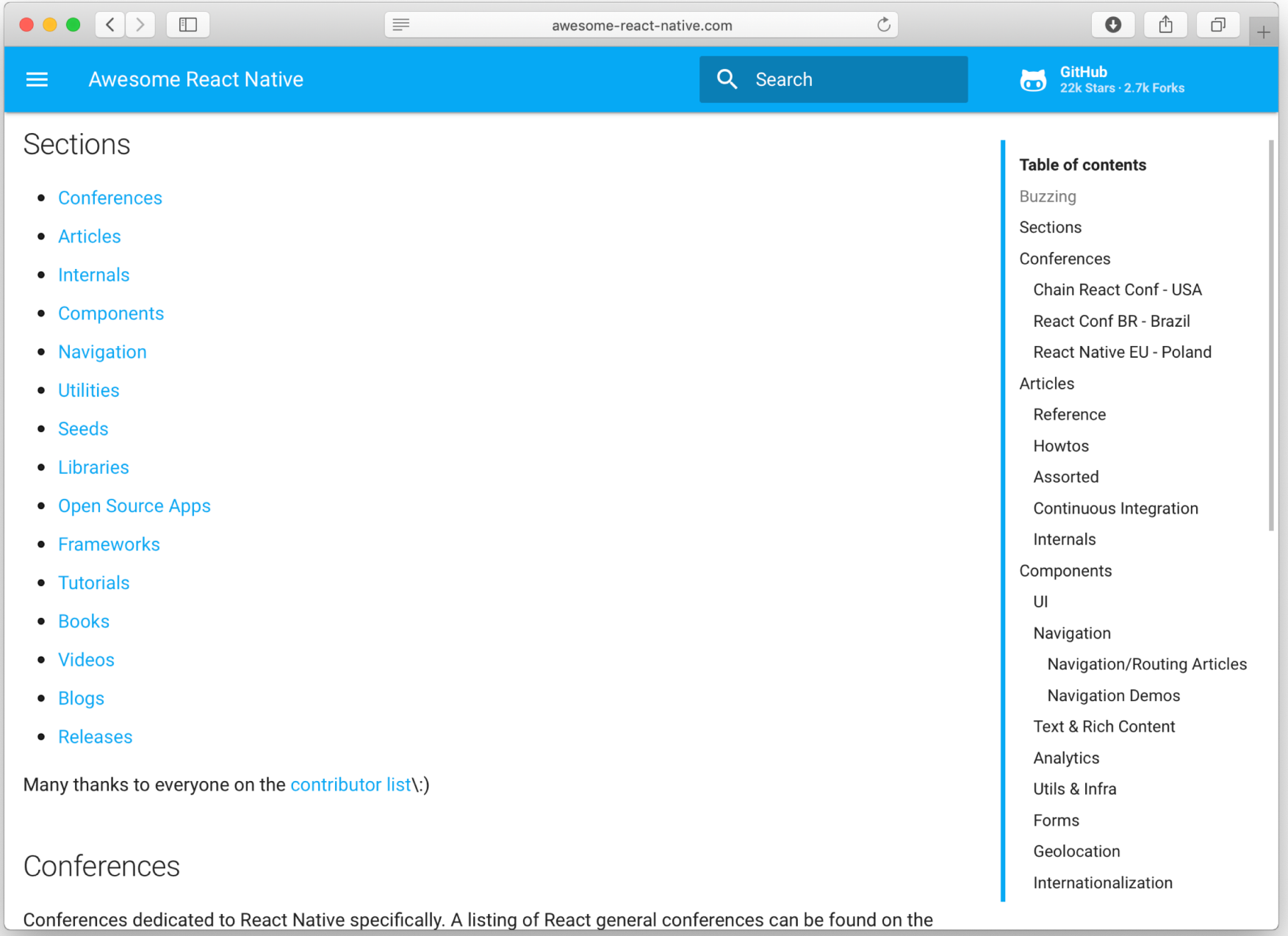
- We can also use the proposed ES2017 async/await syntax in RN

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

export default class FetchExample extends React.Component {

  constructor(props){
    super(props);
    this.state ={ isLoading: true}
  }

  componentDidMount(){
    return fetch('https://facebook.github.io/react-native/movies.json')
      .then((response) => response.json())
      .then((responseJson) => {
        this.setState({
          isLoading: false,
          dataSource: responseJson.movies,
        }, function(){
       });
      })
      .catch((error) =>{
        console.error(error);
      });
  }
```

```
render(){
  if(this.state.isLoading){
    return(
      <View style={{flex: 1, padding: 20}}>
        <ActivityIndicator/>
      </View>
    )
  }

  return(
    <View style={{flex: 1, paddingTop:20}}>
      <FlatList
        data={this.state.dataSource}
        renderItem={({item}) => <Text>{item.title}, {item.releaseYear}</Text>}
        keyExtractor={({id}, index) => id}
      />
      </View>
  );
}
```

Other options

- The XMLHttpRequest API is built in
  - We can use third party libraries, such as frisbee or axios, that depend on it, or we can use the API directly
- RN also supports WebSockets, a protocol that provides full-duplex communication channels over a single TCP connection



Sections

- Conferences
- Articles
- Internals
- Components
- Navigation
- Utilities
- Seeds
- Libraries
- Open Source Apps
- Frameworks
- Tutorials
- Books
- Videos
- Blogs
- Releases

Many thanks to everyone on the contributor list\:)

Conferences

Conferences dedicated to React Native specifically. A listing of React general conferences can be found on the

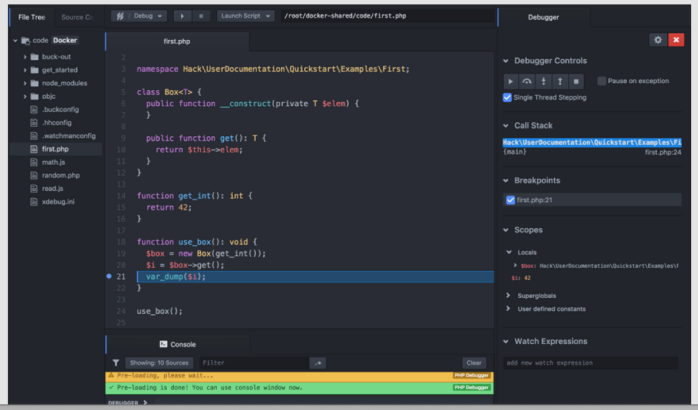
Table of contents

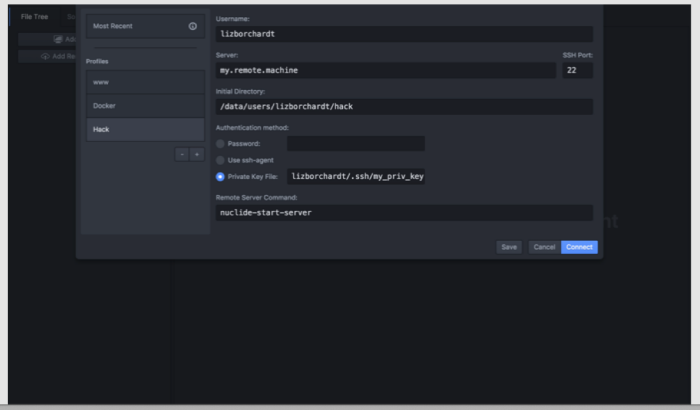
Buzzing
Sections
Conferences
Chain React Conf - USA
React Conf BR - Brazil
React Native EU - Poland
Articles
Reference
Howtos
Assorted
Continuous Integration
Internals
Components
UI
Navigation
Navigation/Routing Articles
Navigation Demos
Text & Rich Content
Analytics
Utils & Infra
Forms
Geolocation
Internationalization

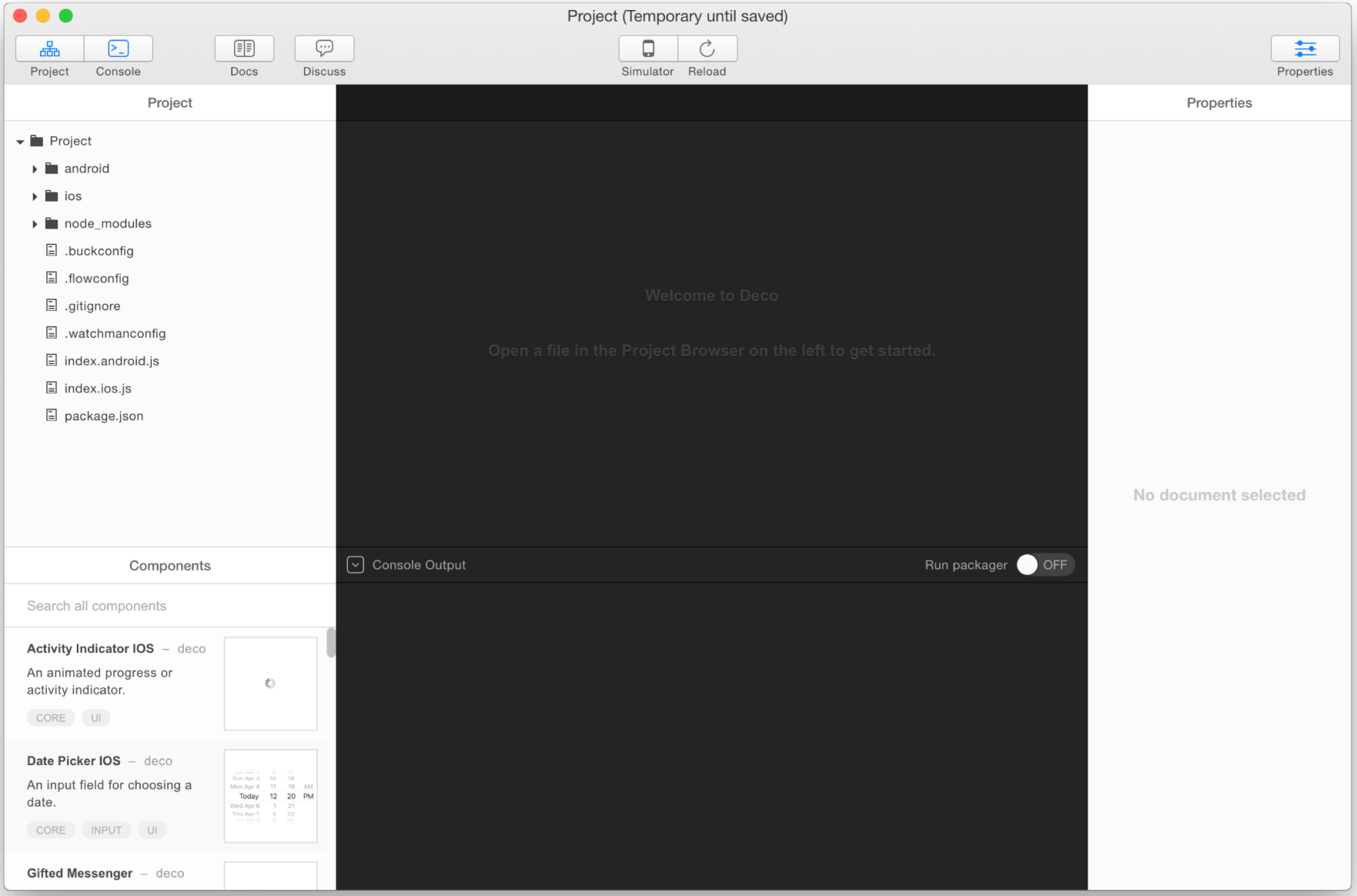
A unified developer experience for software development

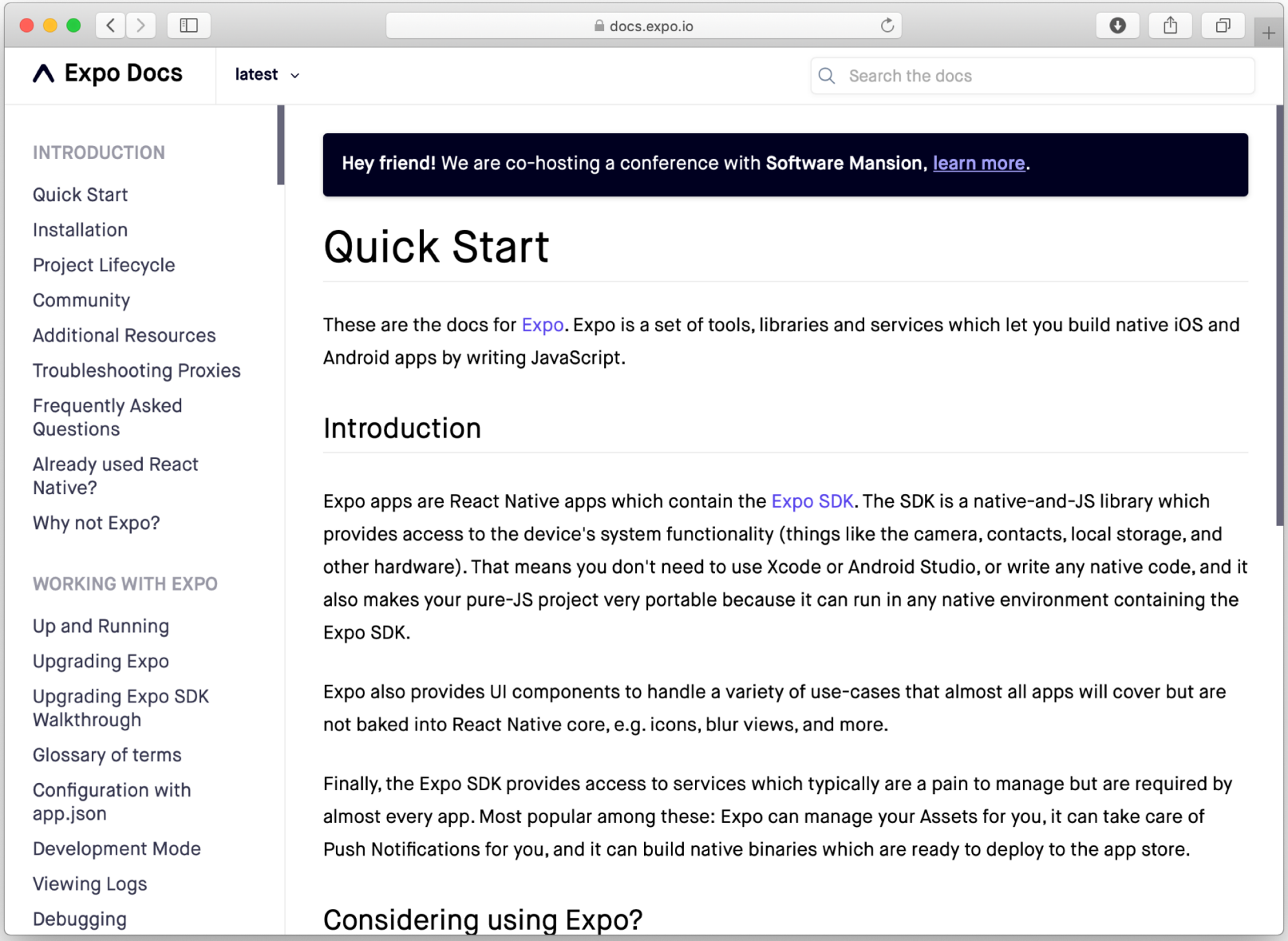
Nuclide is built as a single package on top of Atom to provide hackability and the support of an active community. It provides a first-class development environment for React Native, Hack and Flow projects.

Get Started or read more about using Nuclide for React Native, iOS, or Web development.






Project (Temporary until saved)
Project
Console
Docs
Discuss
Simulator
Reload
Properties
Project
Project
android
ios
node_modules
.buckconfig
.flowconfig
.gitignore
.watchmanconfig
index.android.js
index.ios.js
package.json
Welcome to Deco
Open a file in the Project Browser on the left to get started.
Properties
No document selected
Components
Search all components
Activity Indicator IOS – deco
An animated progress or activity indicator.
CORE
UI
Date Picker IOS – deco
An input field for choosing a date.
CORE
INPUT
UI
Gifted Messenger – deco
Console Output
Run packager
OFF


docs.expo.io
Expo Docs
latest
Search the docs
INTRODUCTION
Quick Start
Installation
Project Lifecycle
Community
Additional Resources
Troubleshooting Proxies
Frequently Asked Questions
Already used React Native?
Why not Expo?
WORKING WITH EXPO
Up and Running
Upgrading Expo
Upgrading Expo SDK Walkthrough
Glossary of terms
Configuration with app.json
Development Mode
Viewing Logs
Debugging
Hey friend! We are co-hosting a conference with Software Mansion, learn more.
Quick Start
These are the docs for Expo. Expo is a set of tools, libraries and services which let you build native iOS and Android apps by writing JavaScript.
Introduction
Expo apps are React Native apps which contain the Expo SDK. The SDK is a native-and-JS library which provides access to the device's system functionality (things like the camera, contacts, local storage, and other hardware). 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.
Expo also provides UI components to handle a variety of use-cases that almost all apps will cover but are not baked into React Native core, e.g. icons, blur views, and more.
Finally, the Expo SDK provides access to services which typically are a pain to manage but are required by almost every app. Most popular among these: Expo can manage your Assets for you, it can take care of Push Notifications for you, and it can build native binaries which are ready to deploy to the app store.
Considering using Expo?