React Native Components & API

Bala Krishna Ragala

The Plan

- What are React Native Components
- View
- Text
- Image
- ScrollView
- Touchables
- FlatList
- SectionList
- Modal
- WebView
- TabBar
- TextInput
- Slider
- Switch
- Picker

Native Components?

- Wrapper on platform native UI components available as components in JS code
- Available in **“react-native”** module
- Classified as following
  - Basic Components
  - User Interface
  - List Views
  - iOS-specific
  - Android-specific
  - Others

Basic Components

- Most apps will end up using one of these basic components

View	Text	Image
The most fundamental component for building a UI.	A component for displaying text.	A component for displaying images.

TextInput	ScrollView	StyleSheet
A component for inputting text into the app via a keyboard.	Provides a scrolling container that can host multiple components and views.	Provides an abstraction layer similar to CSS stylesheets.

User Interface

- Render common user interface controls on any platform using the following components

Button

A basic button component for handling touches that should render nicely on any platform.

Picker

Renders the native picker component on iOS and Android.

Slider

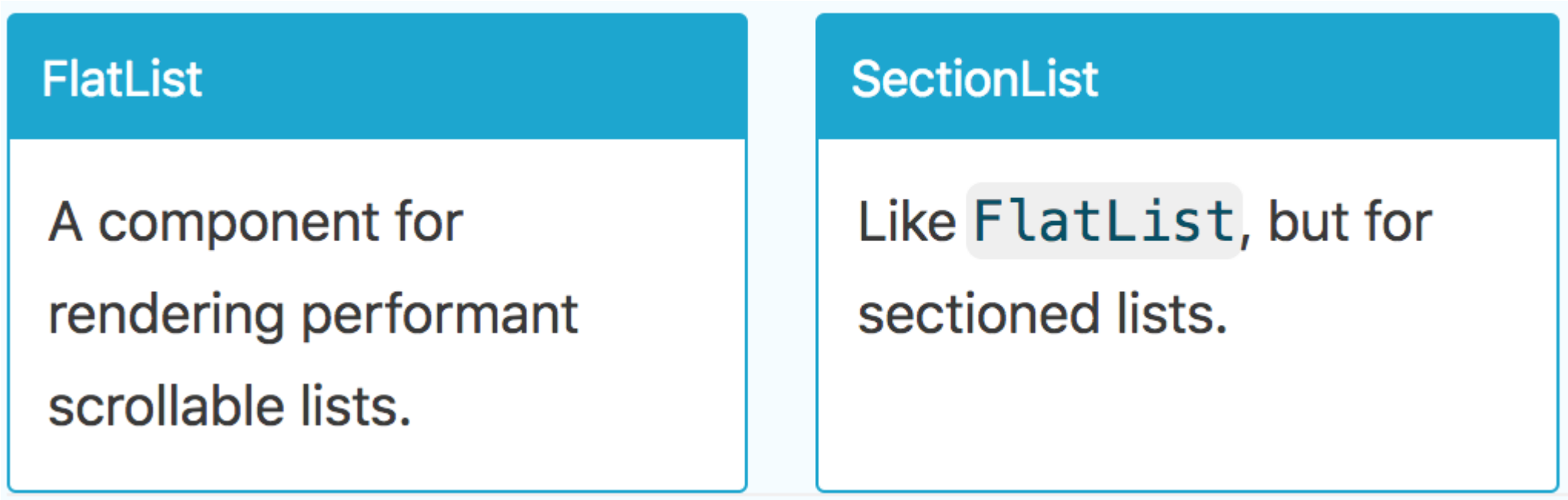
A component used to select a single value from a range of values.

Switch

Renders a boolean input.

ListViews

- Unlike the more generic ScrollView, the following list view components only render elements that are currently showing on the screen. This makes them a great choice for displaying long lists of data.



iOS Components and APIs

- Many of the following components provide wrappers for commonly used UIKit classes

ActionSheetIOS

API to display an iOS action sheet or share sheet.

AlertIOS

Create an iOS alert dialog with a message or create a prompt for user input.

DatePickerIOS

Renders a date/time picker (selector) on iOS.

ImagePickerIOS

Renders a image picker on iOS.

NavigatorIOS

A wrapper around `UINavigationController`, enabling you to implement a navigation stack.

ProgressViewIOS

Renders a `UIProgressView` on iOS.

PushNotificationIOS

Handle push notifications for your app, including permission handling and icon badge number.

SegmentedControlIOS

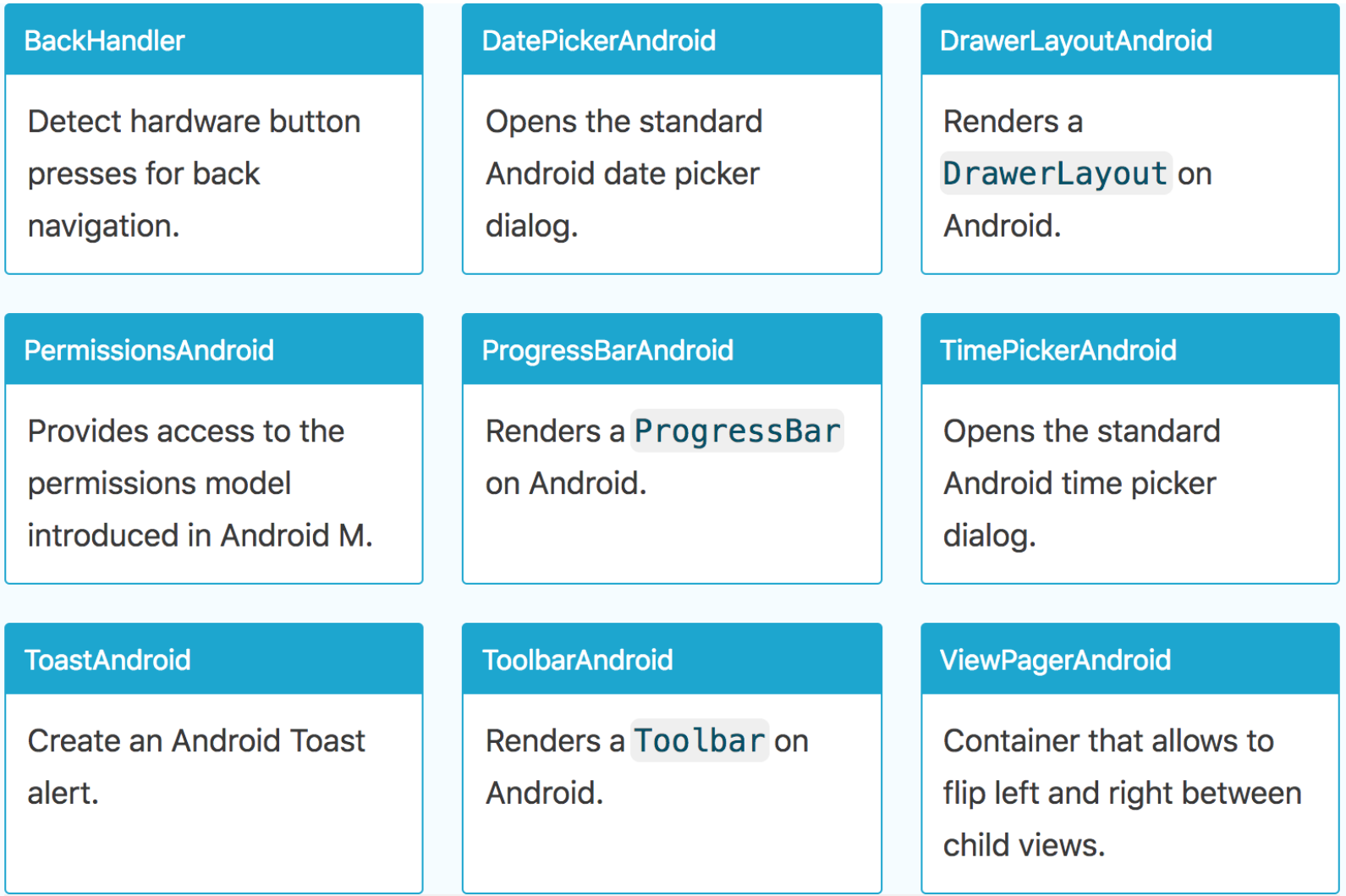
Renders a `UISegmentedControl` on iOS.

TabBarIOS

Renders a `UITabViewController` on iOS. Use with TabBarIOS.Item.

Android Components and APIs

- Many of the following components provide wrappers for commonly used Android classes



Others

- These components may come in handy for certain applications
- ActivityIndicator - Displays a circular loading indicator.
- Alert - Launches an alert dialog with the specified title and message.
- Animated - A library for creating fluid, powerful animations that are easy to build and maintain.
- CameraRoll - Provides access to the local camera roll / gallery.
- Clipboard - Provides an interface for setting and getting content from the clipboard on both iOS and Android.

Others (contd…)

- Dimensions - Provides an interface for getting device dimensions.
- KeyboardAvoidingView - Provides a view that moves out of the way of the virtual keyboard automatically.
- Linking - Provides a general interface to interact with both incoming and outgoing app links.
- Modal - Provides a simple way to present content above an enclosing view.
- PixelRatio - Provides access to the device pixel density.

Others (contd…)

- RefreshControl - This component is used inside a ScrollView to add pull to refresh functionality.
- StatusBar - Component to control the app status bar.
- WebView - A component that renders web content in a native view.

View

- The most fundamental component for building a UI, View is a container that supports layout with **flexbox, style, some touch handling, and accessibility** controls. View maps directly to the native view equivalent on whatever platform React Native is running on, whether that is a UIView, , android.view, etc.
- View is designed to be **nested** inside other views and can have **0 to many children of any type**

View - Example

```
class ViewColoredBoxesWithText extends Component {
  render() {
    return (
      <View
        style={{
          flexDirection: 'row',
          height: 100,
          padding: 20,
        }}>
        <View style={{backgroundColor: 'blue', flex: 0.3}} />
        <View style={{backgroundColor: 'red', flex: 0.5}} />
        <Text>Hello World!</Text>
      </View>
    );
  }
}
```

ScrollView

- Component that wraps platform ScrollView while providing integration with touch locking "responder" system
- ScrollViews must have a bounded height in order to work, since they contain unbounded-height children into a bounded container (via a scroll interaction).
- In order to bound the height of a ScrollView, either set the height of the view directly (discouraged) or make sure all parent views have bounded height.(setting flex:1 for full parent hierarchy)

ScrollView (contd..)

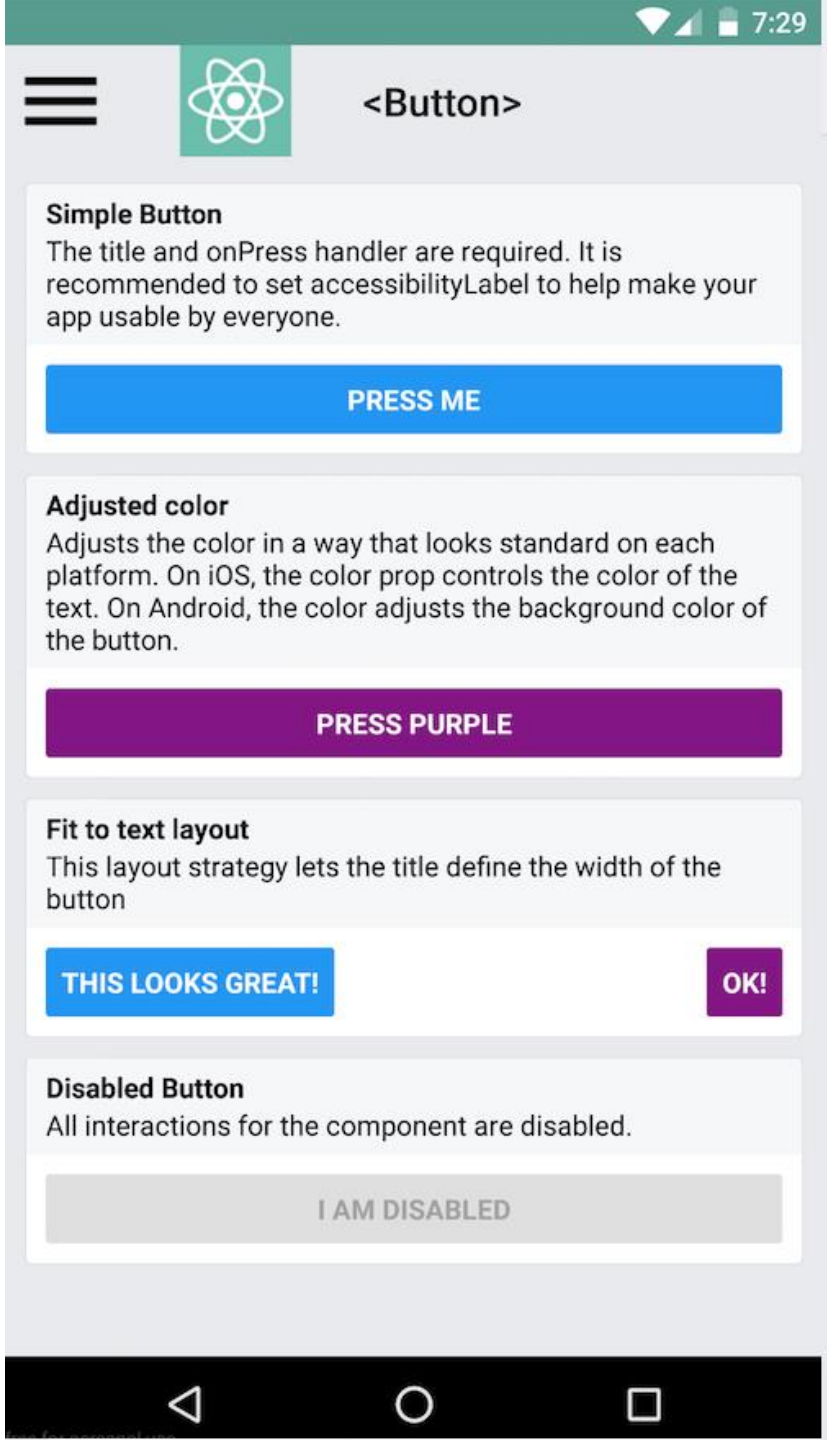
- **ScrollView simply renders all its react child components at once**
- A **ScrollView** with the single item can be used to allow a user to zoom content.
  - Set up a **maximumZoomScale** and **minimumZoomScale** props and the user will be able to use pinch and expand gestures to zoom in and zoom out.
- Scrollable items need not be **homogeneous**, and you can scroll both vertically and horizontally
- By default scrolls vertically Set **horizontal** property to **true** for horizontal scrolling

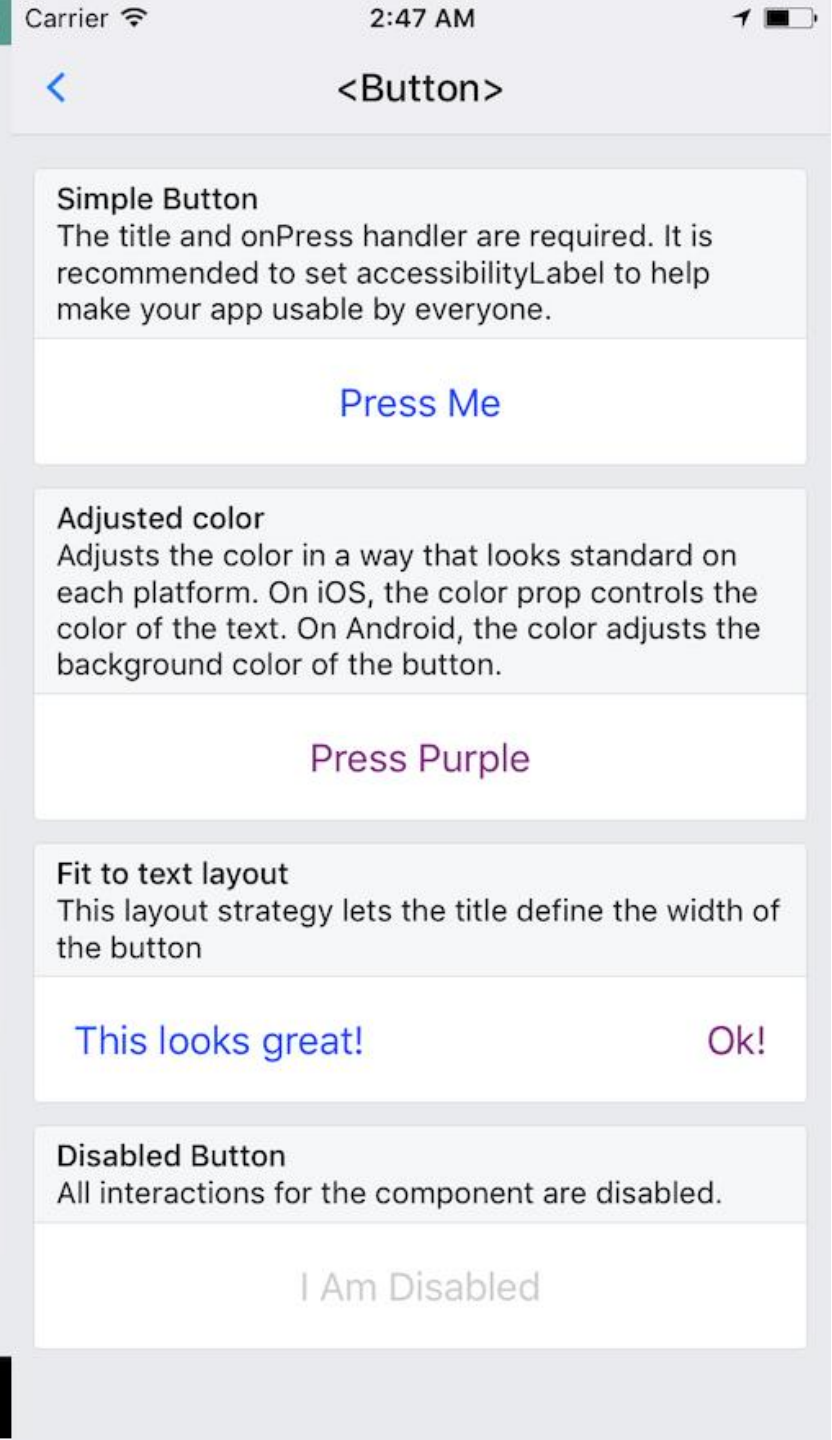
Gesture Responder System

- The gesture responder system manages the lifecycle of gestures in your app.
- A touch can go through several phases as the app determines what the user's intention is. For example, the app needs to determine if the touch is scrolling, sliding on a widget, or tapping. This can even change during the duration of a touch. There can also be multiple simultaneous touches.
- The touch responder system is needed to allow components to negotiate these touch interactions without any additional knowledge about their parent or child components.

Button

- A basic button component that should render nicely on any platform. Supports a minimal level of customization.
- A wrapper on Touchable





Button Usage & Props

- title
- onPress
- color
- disabled

```
import { Button } from 'react-native';
...

<Button
  onPress={onPressLearnMore}
  title="Learn More"
  color="#841584"
  accessibilityLabel="Learn more about this purple button"
/>
```

TouchableHighlight

- A wrapper for making views respond properly to touches.
- On press down, the opacity of the wrapped view is decreased, which allows the underlay color to show through, darkening or tinting the view.
- The underlay comes from wrapping the child in a new View, which can affect layout, and sometimes cause unwanted visual artifacts if not used correctly, for example if the backgroundColor of the wrapped view isn't explicitly set to an opaque color.
- **TouchableHighlight must have one child (not zero or more than one).** If you wish to have several child components, wrap them in a View.

TouchableHighlight Usage & Props

- activeOpacity
- underlayColor
- onPress
- onPressIn
- onPressOut
- onLongPress
- delayLongPress
- delayPressIn
- delayPressOut
- disabled

```
<TouchableHighlight onPress={this._onPressButton}>
  <Image
    style={styles.button}
    source={require('./myButton.png')}
  />
</TouchableHighlight>
```

TouchableOpacity

- A wrapper for making views respond properly to touches. On press down, the opacity of the wrapped view is decreased, dimming it.
- Opacity is controlled by wrapping the children in an Animated.View, which is added to the view hierarchy

TouchableOpacity Usage & Props

- activeOpacity
- onPress
- onPressIn
- onPressOut
- onLongPress
- delayLongPress
- delayPressIn
- delayPressOut
- disabled

```
<TouchableOpacity onPress={this._onPressButton}>
  <Image
    style={styles.button}
    source={require('./myButton.png')}
  />
</TouchableOpacity>
```

RefreshControl

- This component is used inside a ScrollView or FlatList or SectionList to add pull to refresh functionality.
- When the ScrollView is at **scrollY: 0**, swiping down triggers an **onRefresh** event.

```
<RefreshControl
  refreshing={this.state.refreshing}
  onRefresh={this._onRefresh.bind(this)}
/>
```

RefreshControl Props

- refreshing
- onRefresh
- colors
- enabled
- progressBackgroundColor
- size
- tintColor (ios)
- Title (ios)
- titleColor

FlatList

- A performant interface for rendering simple, flat lists, supporting the most handy features:
  - Fully cross-platform.
  - Optional horizontal mode.
  - Configurable viewability callbacks.
  - Header support.
  - Footer support.
  - Separator support.
  - Pull to Refresh.
  - Scroll loading.
  - ScrollToIndex support.

FlatList Usage & Props

```
<FlatList
  data={this.props.data}
  extraData={this.state}
  keyExtractor={this._keyExtractor}
  renderItem={this._renderItem}
/>
```

- onScroll
- onScrollBeginDrag
- onScrollEndDrag
- pagingEnabled
- refreshControl
- scrollEnabled
- showsHorizontalScrollIndicator
- showsVerticalScrollIndicator
- renderItem
- data
- ItemSeparatorComponent
- ListEmptyComponent
- ListFooterComponent
- ListHeaderComponent
- extraData
- horizontal

Picker

- Renders the native picker component on iOS and Android

```
<Picker
  selectedValue={this.state.language}
  style={{ height: 50, width: 100 }}
  onValueChange={(itemValue, itemIndex) => this.setState({language: itemValue})}>
  <Picker.Item label="Java" value="java" />
  <Picker.Item label="JavaScript" value="js" />
</Picker>
```

TextInput

- A foundational component for inputting text into the app via a keyboard. Props provide configurability for several features, such as auto-correction, auto-capitalization, placeholder text, and different keyboard types, such as a numeric keypad.

```
<TextInput
  style={{height: 40, borderColor: 'gray', borderWidth: 1}}
  onChangeText={(text) => this.setState({text})}
  value={this.state.text}
/>
```

TextInput Props

- onChange
- multiline
- Style
- Palceholder
- placeholderTextColor
- value
- numberOfLines
- maxLength
- keyboardType

https://facebook.github.io/react-native/docs/textinput.html

ActivityIndicator

- Displays a circular loading indicator.

```
<View style={[styles.container, styles.horizontal]}>
  <ActivityIndicator size="large" color="#0000ff" />
  <ActivityIndicator size="small" color="#00ff00" />
  <ActivityIndicator size="large" color="#0000ff" />
  <ActivityIndicator size="small" color="#00ff00" />
</View>
```

ActivityIndicator Props

- animating
- color
- Size - 'small’ / 'large'

WebView

- renders web content in a native view.

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

class MyWeb extends Component {
  render() {
    return (
      <WebView
        source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
      />
    );
  }
}
```

React Native APIs

What are Native APIs?

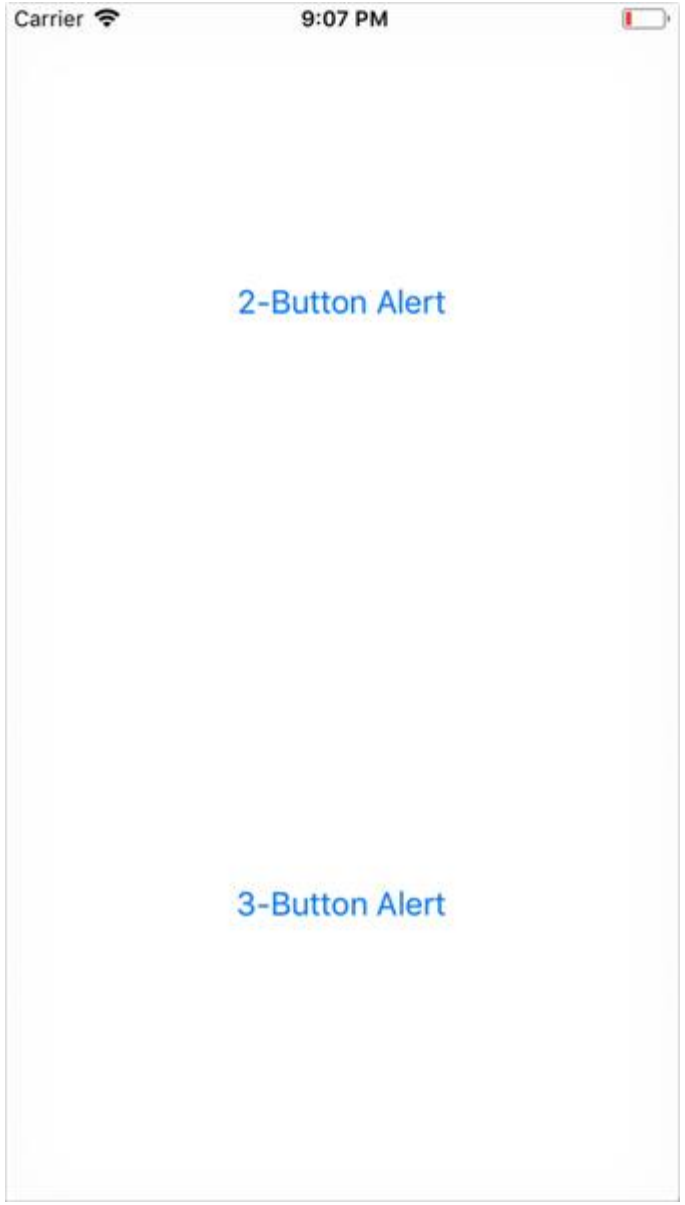
- JavaScript wrappers on native services / apis

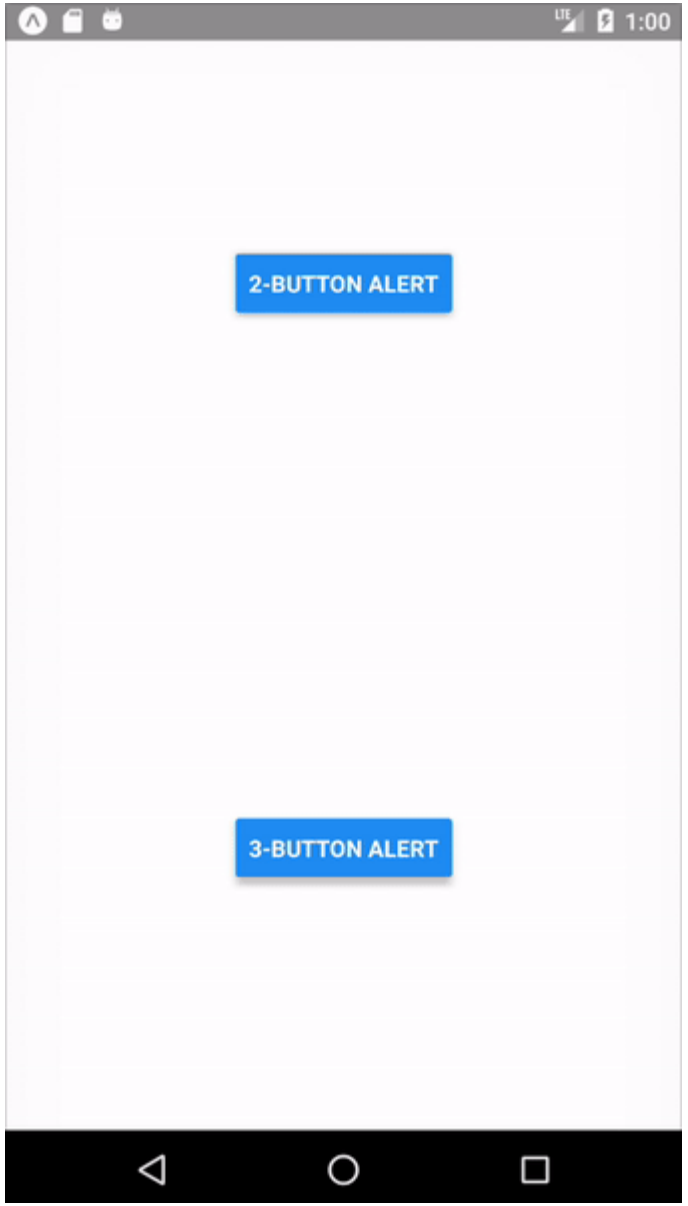
Alert

- Launches an alert dialog with the specified title and message.
- Optionally provide a list of buttons. Tapping any button will fire the respective **onPress** callback and dismiss the alert.
- **By default, the only button will be an 'OK' button.**

```
// Works on both iOS and Android
Alert.alert(
  'Alert Title',
  'My Alert Msg',
  [
    {text: 'Ask me later', onPress: () => console.log('Ask me later pressed')},
    {text: 'Cancel', onPress: () => console.log('Cancel Pressed'), style: 'cancel'},
    {text: 'OK', onPress: () => console.log('OK Pressed')},
  ],
  { cancelable: false }
)
```

Alert - Examples





ToastAndroid

- a very simple API for Android. It simply shows toast message.
- ONLY android

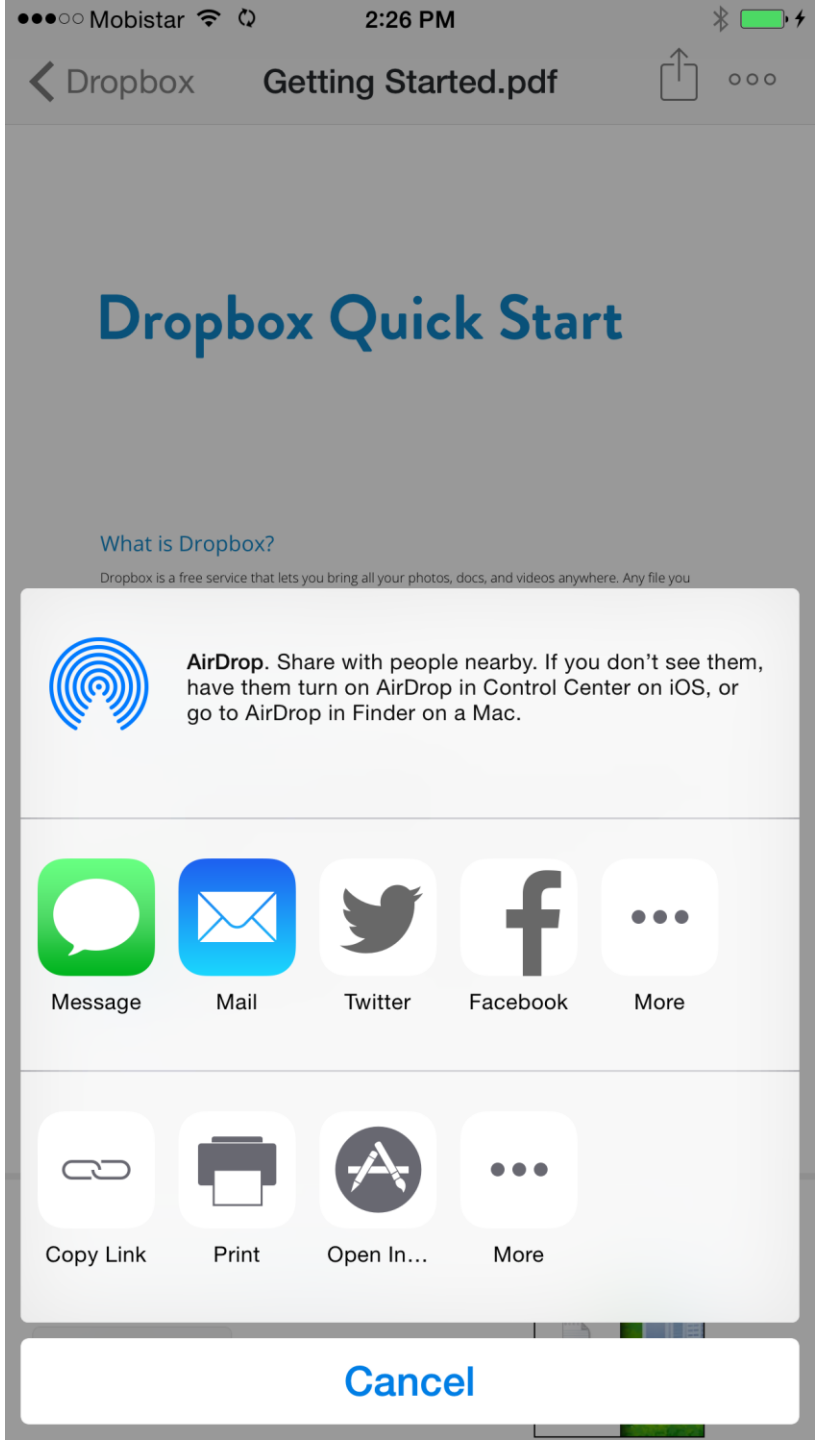
```
ToastAndroid.show('Toast message for android', ToastAndroid.LONG)
```

```
ToastAndroid.showWithGravity('Toast message for android’,
                             ToastAndroid.LONG,
                             ToastAndroid.TOP)
```

ActionSheetIOS

- IOS action sheets are a commonly used option in IOS apps to provide multiple selection options to the user.
- **ActionSheetIOS** is used to create IOS action sheets
- 2 Methods
  - showActionSheetWithOptions()
  - showShareActionSheetWithOptions()


Carrier
9:15 PM
Show case Action sheet
This is a simple action sheet show case
one
two
three
four
exit
cancel


Mobistar
2:26 PM
Dropbox
Getting Started.pdf
Dropbox Quick Start
What is Dropbox?
Dropbox is a free service that lets you bring all your photos, docs, and videos anywhere. Any file you
AirDrop. Share with people nearby. If you don't see them, have them turn on AirDrop in Control Center on iOS, or go to AirDrop in Finder on a Mac.
Message
Mail
Twitter
Facebook
More
Copy Link
Print
Open In...
More
Cancel

Vibration

- Vibration API is exposed at **Vibration.vibrate()**

```
const DURATION = 10000
const PATTERN = [1000, 2000, 3000]

Vibration.vibrate(DURATION)
// Android: vibrate for 10s
// iOS: duration is not configurable, vibrate for fixed time (about 500ms)

Vibration.vibrate(PATTERN)
// Android: wait 1s -> vibrate 2s -> wait 3s
// iOS: wait 1s -> vibrate -> wait 2s -> vibrate -> wait 3s -> vibrate

Vibration.vibrate(PATTERN, true)
// Android: wait 1s -> vibrate 2s -> wait 3s -> wait 1s -> vibrate 2s -> wait 3s -> ...
// iOS: wait 1s -> vibrate -> wait 2s -> vibrate -> wait 3s -> vibrate -> wait 1s -> vibrate -> wait 2s -> vib

Vibration.cancel()
// Android: vibration stopped
// iOS: vibration stopped
```

AppState

- can tell you if the app is in the foreground or background, and notify you when the state changes.
- AppState is frequently used to determine the intent and proper behavior when handling push notifications.
- States
  - active
  - background
  - inactive -This is a state that occurs when transitioning between foreground & background, and during periods of inactivity such as entering the Multitasking view or in the event of an incoming call

AppState - Usage

```
componentDidMount() {
        AppState.addEventListener('change', this._handleAppStateChange);
}

componentWillUnmount() {
        AppState.removeEventListener('change', this._handleAppStateChange);
}

_handleAppStateChange = (nextAppState) => {
    if (this.state.appState.match(/inactive|background/) && nextAppState === 'active') {
        console.log('App has come to the foreground!')
    }

}
```

NetInfo

- exposes info about online/offline status

```
NetInfo.getConnectionInfo().then((connectionInfo) => {
  console.log('Initial, type: ' + connectionInfo.type + ', effectiveType: ' + connectionInfo.effectiveType);
});
function handleFirstConnectivityChange(connectionInfo) {
  console.log('First change, type: ' + connectionInfo.type + ', effectiveType: ' + connectionInfo.effectiveType)
  NetInfo.removeEventListener(
    'connectionChange',
    handleFirstConnectivityChange
  );
}
NetInfo.addEventListener(
  'connectionChange',
  handleFirstConnectivityChange
);
```

https://facebook.github.io/react-native/docs/netinfo.html

AsyncStorage

- Simple, unencrypted, asynchronous, persistent, key-value storage system that is global to the app.
- It should be used instead of LocalStorage
- On **iOS**, AsyncStorage is backed by native code that stores small values in a serialized dictionary and larger values in separate files
- On **Android**, AsyncStorage will use either **RocksDB** or **SQLite** based on what is available.

AsyncStorage - Write

```
try {
  await AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.');
} catch (error) {
  // Error saving data
}
```

AsyncStorage - Read

```
try {
  const value = await AsyncStorage.getItem('@MySuperStore:key');
  if (value !== null){
    // We have data!!
    console.log(value);
  }
} catch (error) {
  // Error retrieving data
}
```

AsyncStorage - API

- getItem
- setItem
- removeItem
- mergeItem
- clear
- getAllKeys
- multiGet
- multiSet
- multiRemove
- multiMerge

https://facebook.github.io/react-native/docs/asyncstorage.html

Dimensions

- Used to retrieve screen width and height for layout-related calculation
- usually used for styling or animation calculations

```
const { width, height } = Dimensions.get('window');
```

Platform

- Used to retrive platfrom information
- Used for platform specific actions

PermissionsAndroid

- Project with Native Code Required
- This API only works in projects made with **react-native init** or in those made with **Create React Native App** which have since **ejected**
- provides access to Android M's new permissions model.
- Some permissions are granted by default when the application is installed so long as they appear in **AndroidManifest.xml**
- "dangerous" permissions require a dialog prompt. You should use this module for those permissions.

Permission Android - Usage

```
import { PermissionsAndroid } from 'react-native';

async function requestCameraPermission() {
  try {
    const granted = await PermissionsAndroid.request(
      PermissionsAndroid.PERMISSIONS.CAMERA,
      {
        'title': 'Cool Photo App Camera Permission',
        'message': 'Cool Photo App needs access to your camera ' +
                   'so you can take awesome pictures.'
      }
    )
    if (granted === PermissionsAndroid.RESULTS.GRANTED) {
      console.log("You can use the camera")
    } else {
      console.log("Camera permission denied")
    }
  } catch (err) {
    console.warn(err)
  }
}
```

https://facebook.github.io/react-native/docs/permissionsandroid.html

Permission Android - API

- **check(permission)** - Returns a promise resolving to a boolean value as to whether the specified permissions has been granted.
- **request(permission, [rationale])** - Prompts the user to enable a permission and returns a promise resolving to a string value (see result strings above) indicating whether the user allowed or denied the request or does not want to be asked again.

Geolocation

- Geolocation API extends the Geolocation web spec.
- vailable through the **navigator.geolocation** global - you do not need to **import** it.
- navigator.geolocation.getCurrentPosition
- navigator.geolocation.watchPosition
- navigator.geolocation.clearWatch

Geolocation - Usage

```
componentDidMount() {
  navigator.geolocation.getCurrentPosition(
    (position) => {
      this.setState({
        latitude: position.coords.latitude,
        longitude: position.coords.longitude,
        error: null,
      });
    },
    (error) => this.setState({ error: error.message }),
    { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 },
  );
}
```

Question Time

- Name components that provide scroll behavior?
- Which scrolling component supports lazy fetch?
- What is the purpose of refresh control?
- Which component gives you choice to select from list of values?
- What is the default width and height of View?
- Can you nest Text in Text?
- How many children are need to make TouchableHighlight legible?
- Is possible to embed webpage in a View?
- Which component shows you the loader?