Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2018 - Week 11

Dr Nick Hayward

React Native - Platform Structure

cross-platform

- React Native gives us a default directory and script structure
  - *part of the structure for a newly initialised app*
- modify stucture as app grows in complexity and scope
- React Native provides app initialisation files
  - *index.js & App.js*
- create a custom directory for app, e.g.
  - *`src` or `app` &c.*
  - *add directories for UI components, assets, scripts for APIs...*
- import `App.js` from `src` &c. directory

```
import App from './src/App';
```

React Native - Platform Structure

Android & iOS

- then start to add platform specific requirements
  - *including components, styles, images...*
- customisation is being encouraged with the `Platform` component. e.g.

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

- add checks to the logic of our app to add platform specific customisations,

```
const titles = Platform.select({
  ios: 'iOS custom title...',
  android: 'Android custom title...',
});
```

- to use this in our app's code
  - *do not need to specify iOS or Android*
  - *simply add the required output for `titles`. e.g.*

```
...
<View>
  <Text>{titles}</Text>
</View>
...
```

React Native - component usage

StatusBar

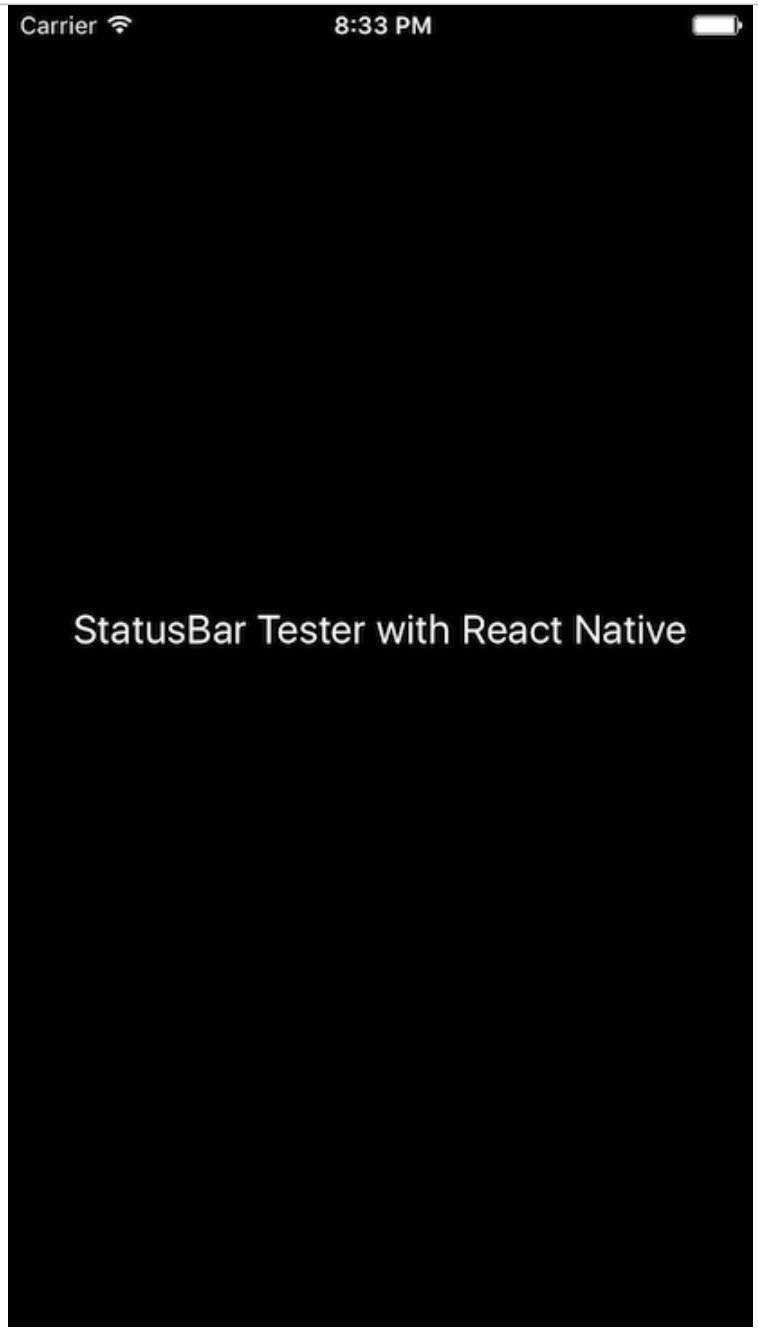
- add customisation to our app's *Status Bar*
  - *top bar with network icon, data, battery status, notification icons &c*
- various customisation options for each platform
  - *animate this bar*
  - *modify its colour*
  - *add custom style to match the current mode or status within our app*
- simple modification is to update the background colour
  - *from light to dark, and vice versa...*
  - *e.g. inform user of status change by animating the colour change and update*
- need to import the *StatusBar* component
  - *add an `animated` prop for the component*
  - *and specify a star for the bar itself*
- e.g. set the background colour of the bar to white

```
<StatusBar animated barStyle="light-content" />
```

- we might also set the `barStyle` to dark using the value `dark-content`
  - *sets colour of status bar text*
- we can only use the `barStyle` prop with iOS
- for Android, we can set props for `backgroundColor` and `translucent`
- additional options for working with the *StatusBar*, including static functions
  - *StatusBar*

Image - React Native - Component Usage

StatusBar



React Native - StatusBar

React Native - component usage

images

- use `Image` component to add images
  - *and various static resources as well*
- `Image` component works with local and remote sources
  - *able to fetch remote images from a specified URL or server address*

```
...
<Image
  style={styles.image}
  resizeMode="contain"
  source={{
    uri: 'http://www.test.com/images/image.png'
  }}
/>
...
```

or

```
<Image
  style={styles.image}
  resizeMode="contain"
  source={require('./images/camel-icon.png')}
/>
```

- `resizeMode` prop may accept various values to help with layout and design
  - *cover, contain, stretch, repeat (only iOS), center*
- also check and use additional lifecycle props with images, including
  - *`onLoad`*
  - *`onLoadEnd`*
  - *`onLoadStart`*
- also get the size of a specifed image before rendering it to the View

```
Image.getSize
```

Image - React Native - Component Usage

Image component

React Native - Image component

React Native - component usage

activity indicator

- `ActivityIndicator` component gives us a default spinning loader for an app
  - *a small default component*
  - *useful for async loading, animations...*
- in addition to standard `View` props - also accepts the following
  - *animating - boolean value to determine whether to spin or not*
  - *color - specify the foreground colour of the spinner*
  - *size - pass `small` or `large` string for iOS, and a size value for Android*

React Native - component usage

activity indicator - example

- might want to use the `ActivityIndicator` to delay showing an image
- add a property to *state* - use as a simple boolean check for loading of the image
- initial *state* set as follows,

```
state = {
  showImage: false,
  loading: false
}
```

- image is not shown by default
  - *and the `ActivityIndicator` is not visible or active either*
- create a function to allow us to update the state
  - *will show the activity indicator and image*
- we're using ES6 classes for these examples
  - *need to start binding our functions as we pass them as props*
  - *e.g.*

```
// instantiate object
constructor(props) {
  super(props);
  // bind function
  this.showImage = this.showImage.bind(this);
}
```

- `showImage` function can now be added

```
showImage() {
  this.setState({
    loading: true
  });
  setTimeout(() => {
    this.setState({
      showImage: true,
      loading: false
    })
  }, 2500)
}
```

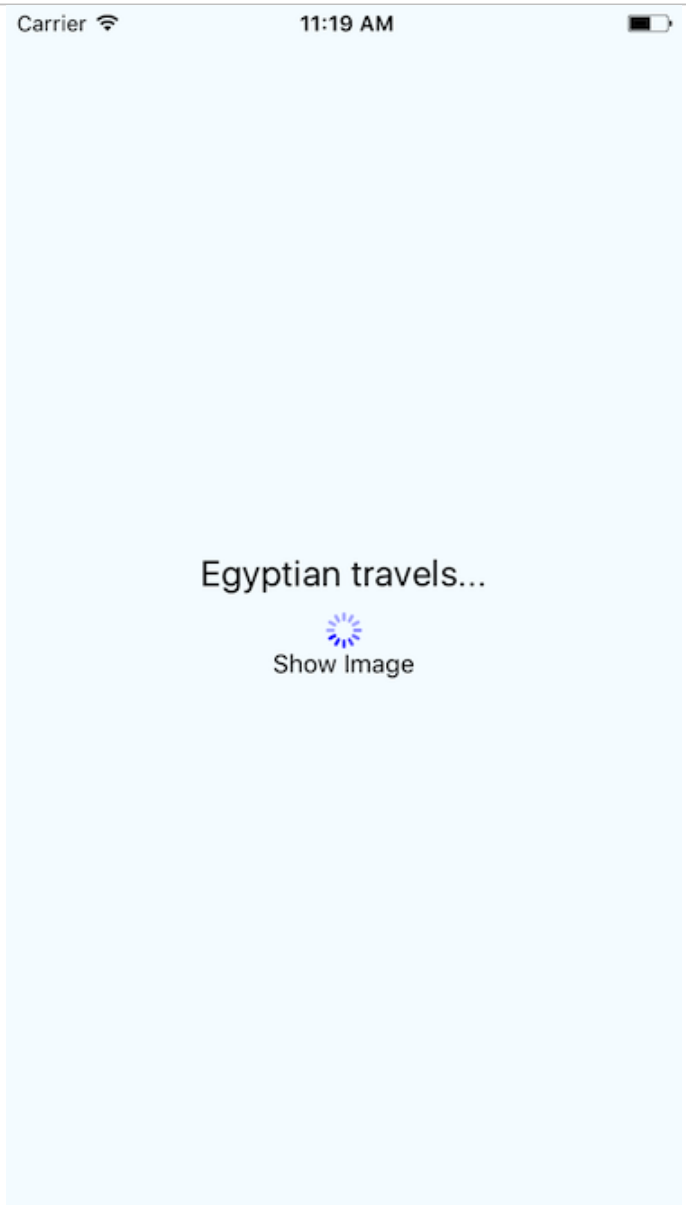
Image - React Native - Component Usage

ActivityIndicator component - part 1

React Native - ActivityIndicator component

Image - React Native - Component Usage

ActivityIndicator component - part 2



React Native - ActivityIndicator component

Image - React Native - Component Usage

ActivityIndicator component - part 3

React Native - ActivityIndicator component

React Native - component usage

custom modal

- React Native also supports a `Modal` component by default
- use it for success messages, feedback or prompts to a user, &c.
- also nest various child components to create the necessary output
- `Modal` component will accept the following props
  - *animationType*
  - *Transparent*
  - *Visible*
  - *onShow*
- also some custom props for each mobile platform
  - *e.g.* `presentationStyle` *for iOS*

React Native - component usage

custom modal - example

```
...
state = {
  modalVisible: true,
}

setModalVisible(visible) {
  this.setState({modalVisible: visible});
}

<Modal
  animationType="slide"
  transparent={false}
  visible={this.state.modalVisible}
  >
  <View style={styles.modal}>
    <TouchableHighlight onPress={() => {
    this.setModalVisible(!this.state.modalVisible)
    }}>
      <Text style={styles.modalClose}>close</Text>
    </TouchableHighlight>
    <Text style={styles.modalText}>Greetings from Egypt</Text>
  </View>
</Modal>
```

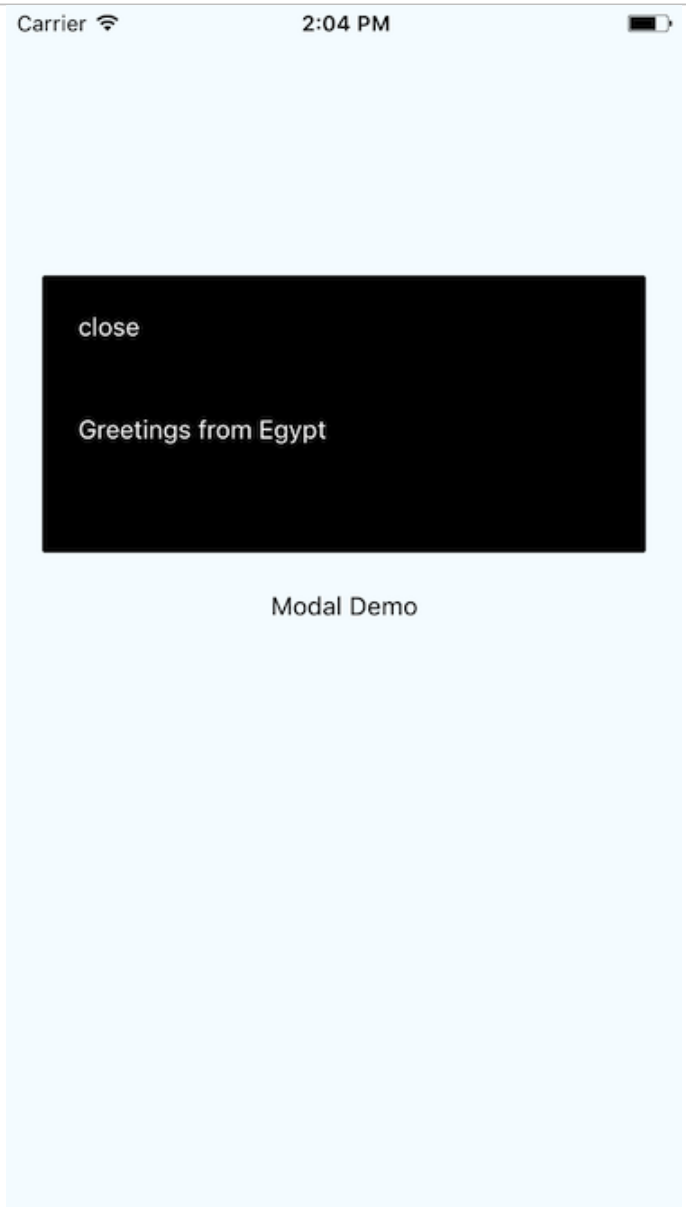
Image - React Native - Component Usage

custom modal component - part 1

React Native - modal component - show

Image - React Native - Component Usage

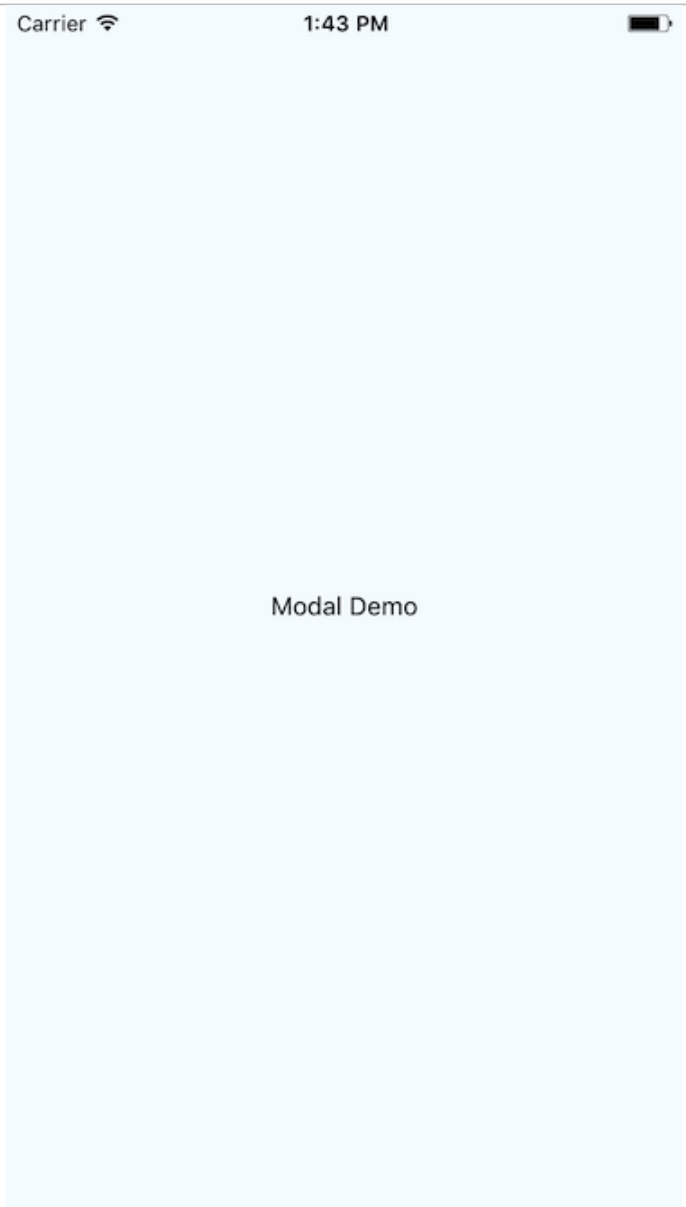
custom modal component - part 2



React Native - modal component - show with transparency

Image - React Native - Component Usage

custom modal component - part 3



React Native - modal component - closed

Mobile Design & Development - UI Components & Usage

Fun Exercise

Four groups, two apps

- Fashion - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/fashion/
- Travel Notes - http://linode4.cs.luc.edu/teaching/cs/demos/422/videos/travelnotes/

For each app, consider the following

- define UI components for the app?
- which components may be reused to create different effects?
- which components could be abstracted to extend a parent component?
- how is the UI influenced by the use of such components?

~ 10 minutes

React Native - component usage

lists - FlatList

- React Native provides suggested view components for lists
  - *two primary examples include `FlatList` and `SectionList`*
- `FlatList` is meant to be used for long lists of data
  - *in particular where data items may change during the lifecycle of an app*
- `FlatList` will only render elements currently shown on screen
  - *not all of the available elements at the same time*

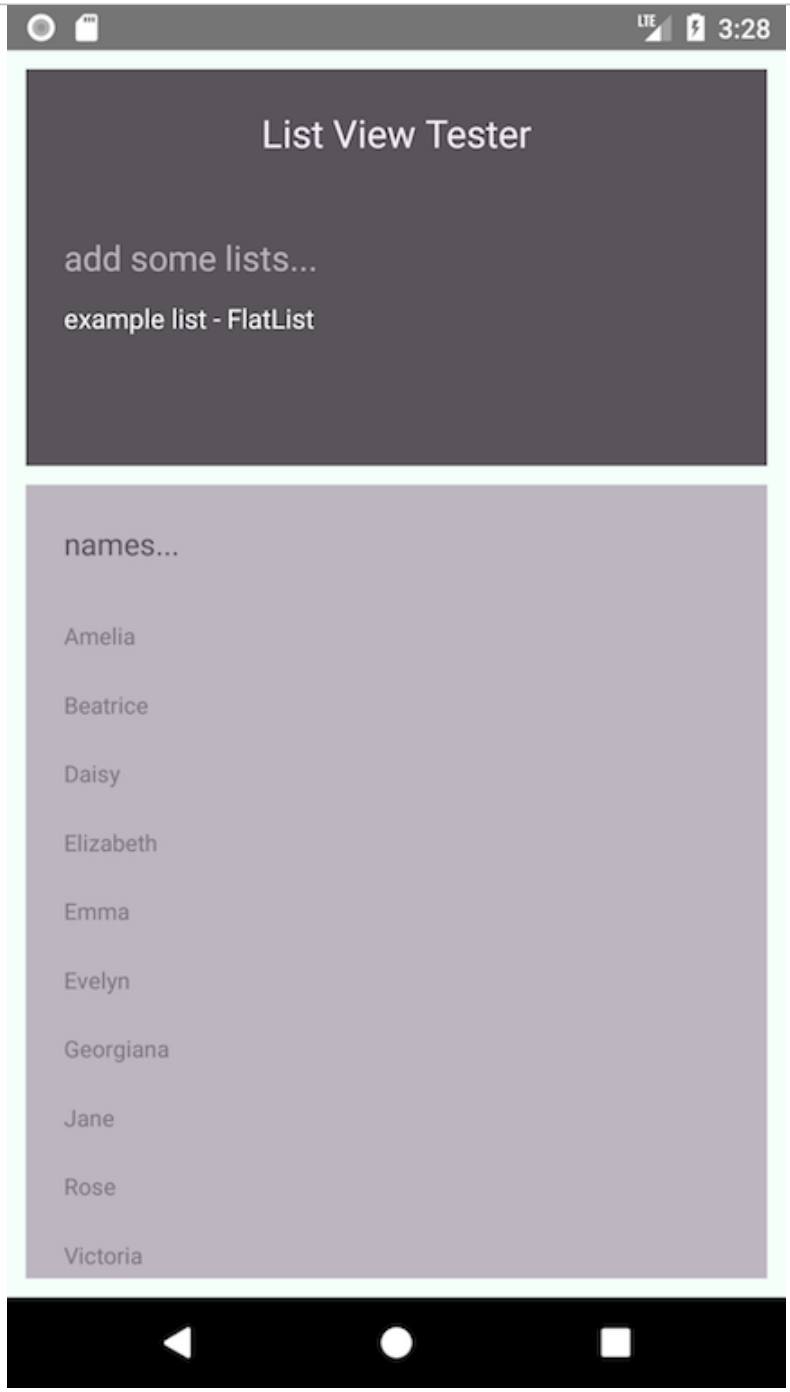
```
<FlatList
    data={[
        {key: 'Amelia'},
        {key: 'Beatrice'},
        {key: 'Daisy'},
    ]
    }
    renderItem={({item}) => <Text style={styles.listItem}>{item.key}</Text>}
/>
```

- component expects two *props*
  - *`data` for the list itself*
  - *`renderItem` to define the output structure for each list item*

```
renderItem={() => <Text></Text>}
```

Image - React Native - Component Usage

lists - FlatList



React Native - FlatList

React Native - component usage

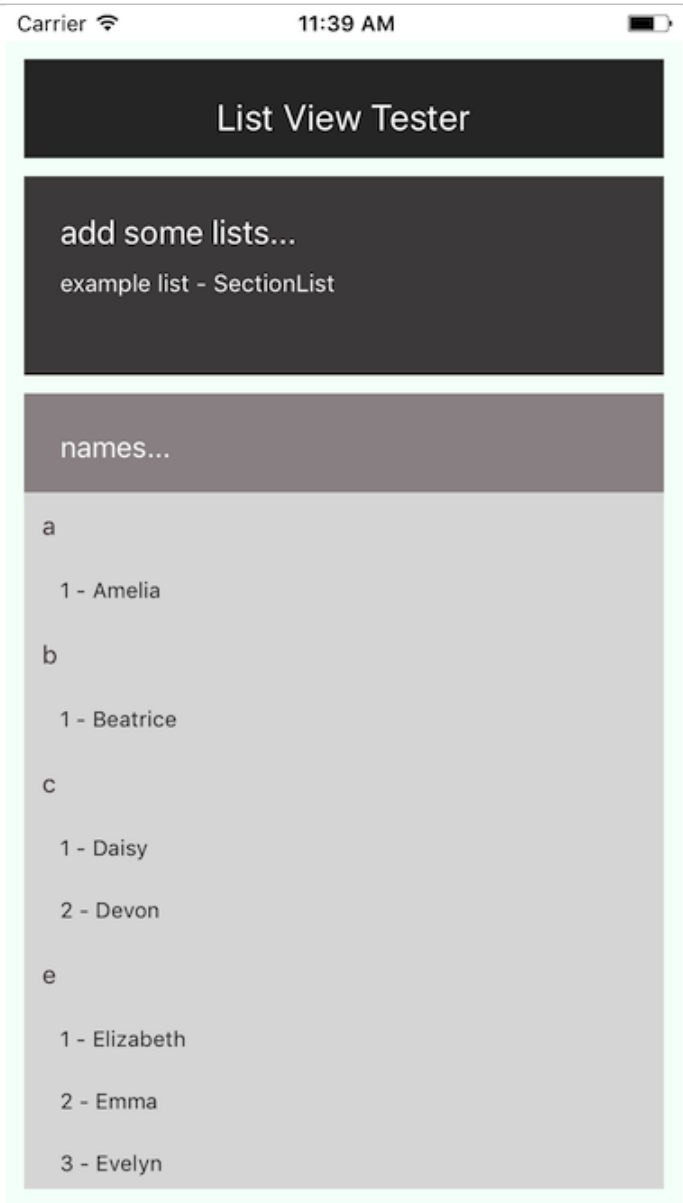
lists - SectionList

- may also create section breaks in a list of data. e.g.

```
<SectionList
  sections={[
        {title: 'a', data:[{key: 1, name: 'Amelia'}]},
        {title: 'b', data:[{key: 1, name: 'Beatrice'}]},
    {title: 'c', data: [{key: 1, name: 'Daisy'}, {key: 2, name: 'Devon'}]},
        {title: 'e', data: [{key: 1, name: 'Elizabeth'}, {key: 2, name: 'Emma'}, {key: 3, n
        {title: 'g', data:[{key: 1, name: 'Georgiana'}]},
        {title: 'j', data:[{key: 1, name: 'Jane'}]},
        {title: 'r', data:[{key: 1, name: 'Rose'}]},
        {title: 'v', data: [{key: 1, name: 'Victoria'}, {key: 2, name: 'Violet'}]},
        {title: 'y', data:[{key: 1, name: 'Yvaine'}]},
    ]}
    //keyExtractor={item => item}
  renderItem={({item}) => <Text style={styles.listItem}>{item.key} - {item.name}</Text>}
  renderSectionHeader={({section}) => <Text style={styles.heading4}>{section.title}</Text>}
/>
```

Image - React Native - Component Usage

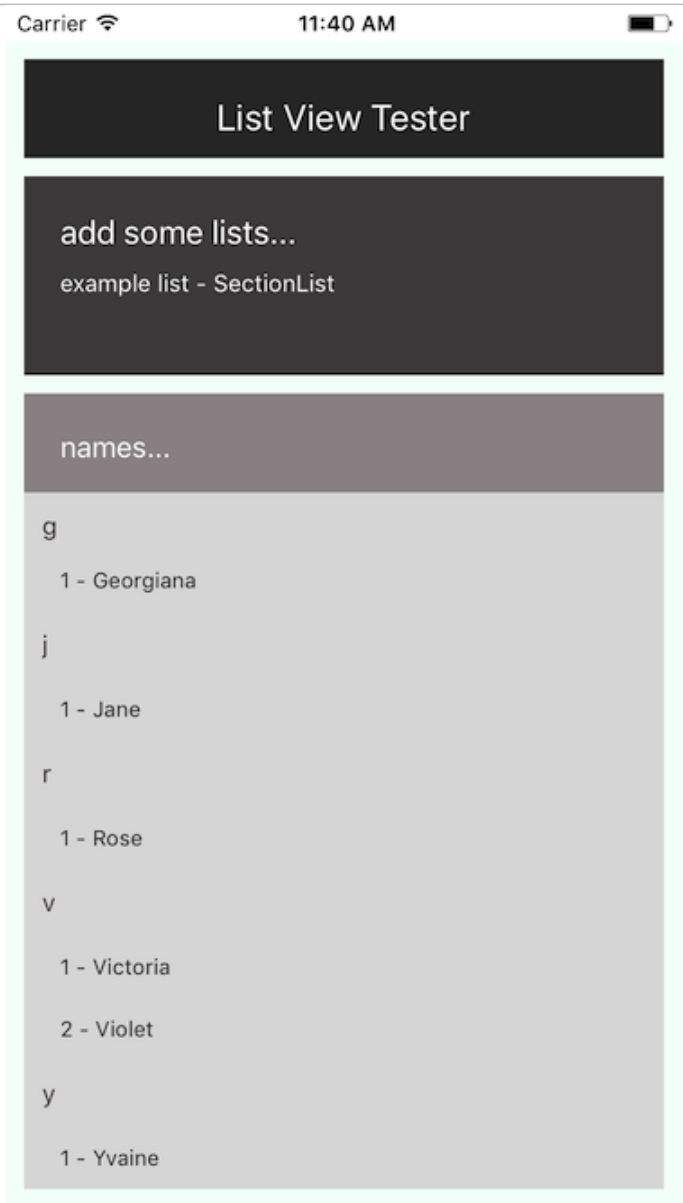
lists - SectiontList - top



React Native - SectionList - top

Image - React Native - Component Usage

lists - SectiontList - bottom



React Native - SectionList - bottom

React Native - component usage

ScrollView

- scrolling in React Native apps is achieved with a generic scrolling container
  - `ScrollView`
- specific view container can itself accept multiple child components and views
- scrollview container option to specify direction
  - *either horizontal or vertical*
- general usage
  - *add a `ScrollView` using the same general pattern as a standard `View` component*
  - *return a `ScrollView` as eiher the primary container for a component*
  - *or a child of a standard `View`*
  - *an app's screen may either scroll top to bottom*
  - *or simply present a component with scroll features*

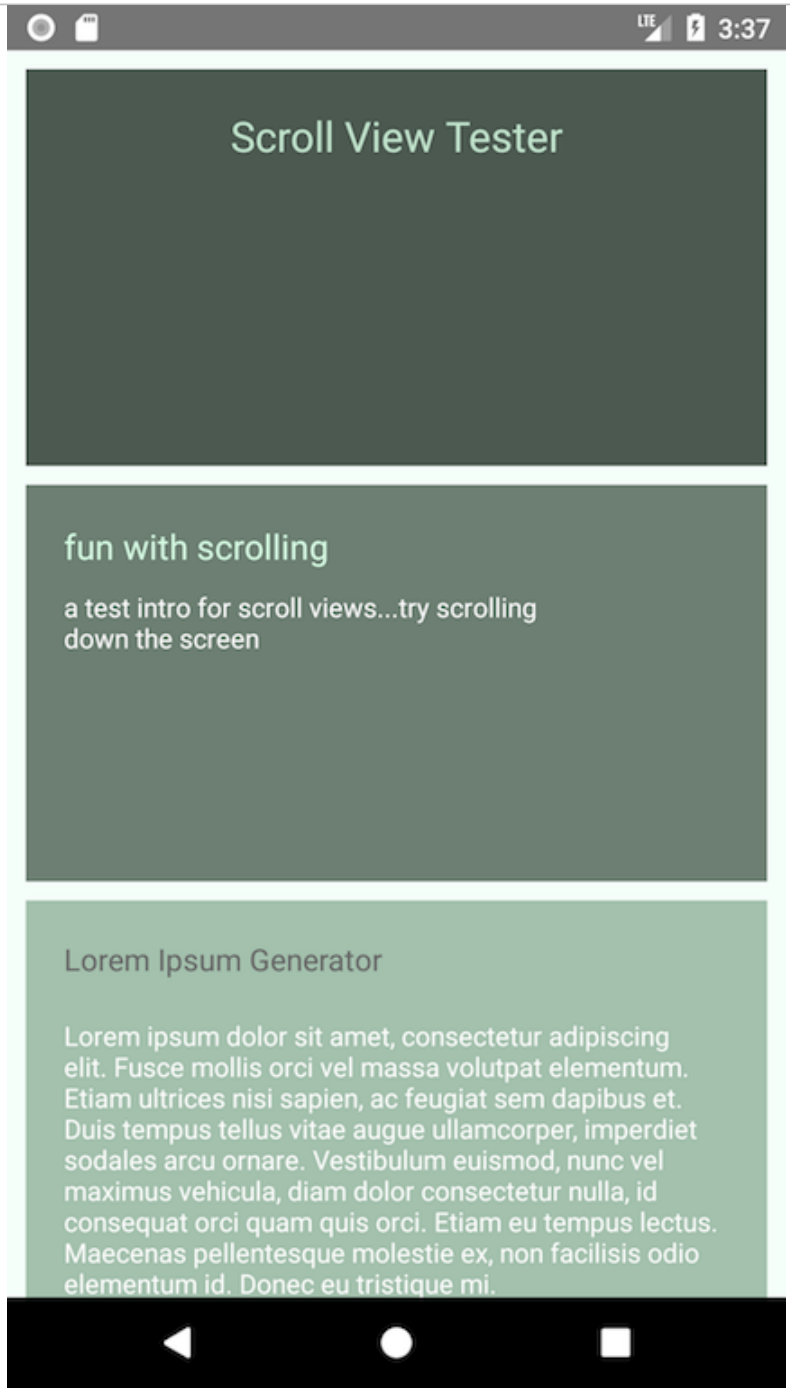
React Native - component usage

ScrollView - example

```
export default class ScrollTester extends Component {
  render() {
    return (
                <View style={styles.container}>
                    <View style={styles.headingBox}>
            <Text style={styles.heading1}>
                Scroll View Tester
            </Text>
            </View>
            <View style={styles.subHeadingBox}>
            <Text style={styles.heading2}>
                {intro.heading}
            </Text>
            <Text style={styles.content}>
                {intro.description}
            </Text>
            </View>
                    <ScrollView>
                    <View style={styles.contentBox}>
            <Text style={styles.heading3}>
                Lorem Ipsum Generator
            </Text>
            <Text style={styles.content}>
                            ...
                        </Text>
            </View>
                    </ScrollView>
        </View>
    );
  }
}
```

Image - React Native - Component Usage

lists - ScrollView



React Native - ScrollView

React Native - Component usage

text input

- a default component to handle user text input
- component `TextInput` is similar to a standard input field
  - *allowing a user to simply enter any required text content*
- to use `TextInput` with an app
  - *need to add the default module from React Native*
  - *add as part of the standard `import` statement*
- `TextInput` component includes a useful *prop*, `onChangeText`
  - *accepts callback function for each time text is changed in input field*
- also includes a complementary *prop*, `onSubmitEditing`
  - *handles text as it is submitted*
  - *again using a defined callback function*

React Native - Component usage

text input - props usage

- might accept user text input for a given value
  - *such as a name, place, &c.*
- then dynamically update the *view*
- e.g.

```
<TextInput
  style={styles.textInput}
  placeholder={this.state.quoteInput}
  onChangeText={(quoteText) => this.setState({quoteText})}
/>
```

React Native - Component usage

text input - props and state

- example relies upon calling and setting state for the app
  - *relative to `TextInput` and various `Text` components*
- simple constructor for this app
  - *pass required `props` and define intial values for `state`*

```
export default class TextUpdater extends Component {
  constructor(props) {
    super(props);
    this.state = {
      quoteInput: 'enter a favourite quotation...',
      quoteText: 'the unexamined life is not worth living...'
    };
  }
}
```

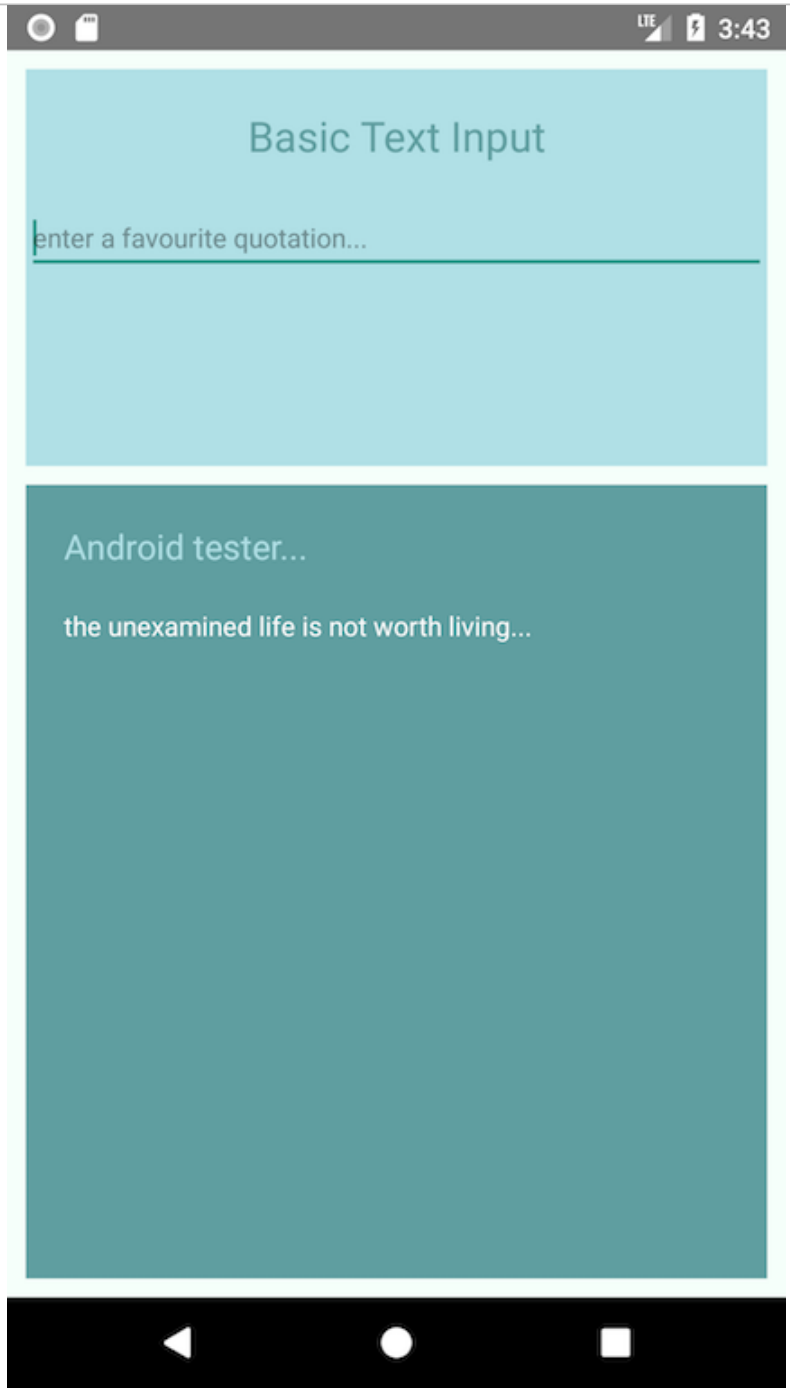
- then use the properties on `state`
  - *to set initial values for the text input field and the text output,*

```
<TextInput
  style={styles.textInput}
  placeholder={this.state.quoteInput}
  onChangeText={(quoteText) => this.setState({quoteText})}
/>
```

```
<Text style={styles.content}>
    {this.state.quoteText}
</Text>
```

Image - React Native - Component Usage

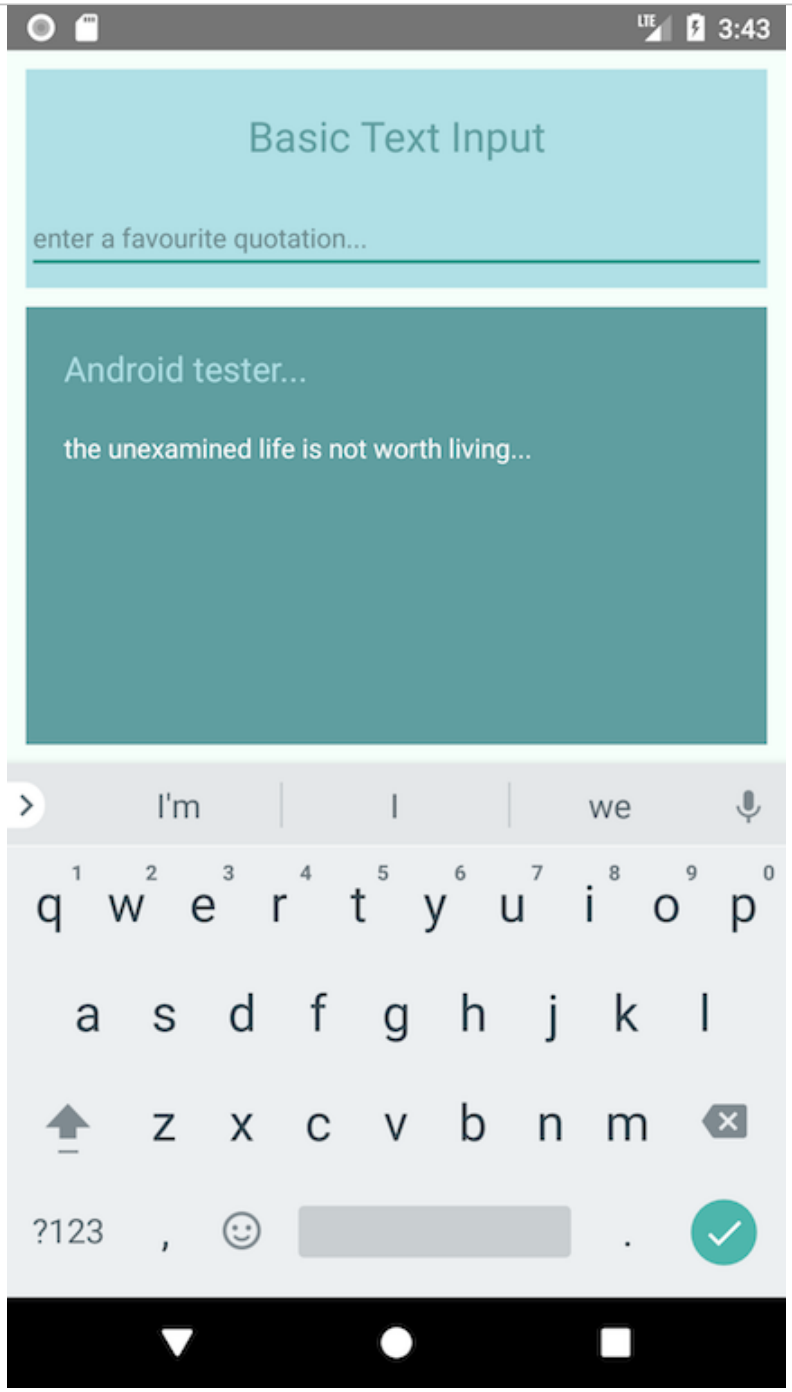
text input



React Native - text input

Image - React Native - Component Usage

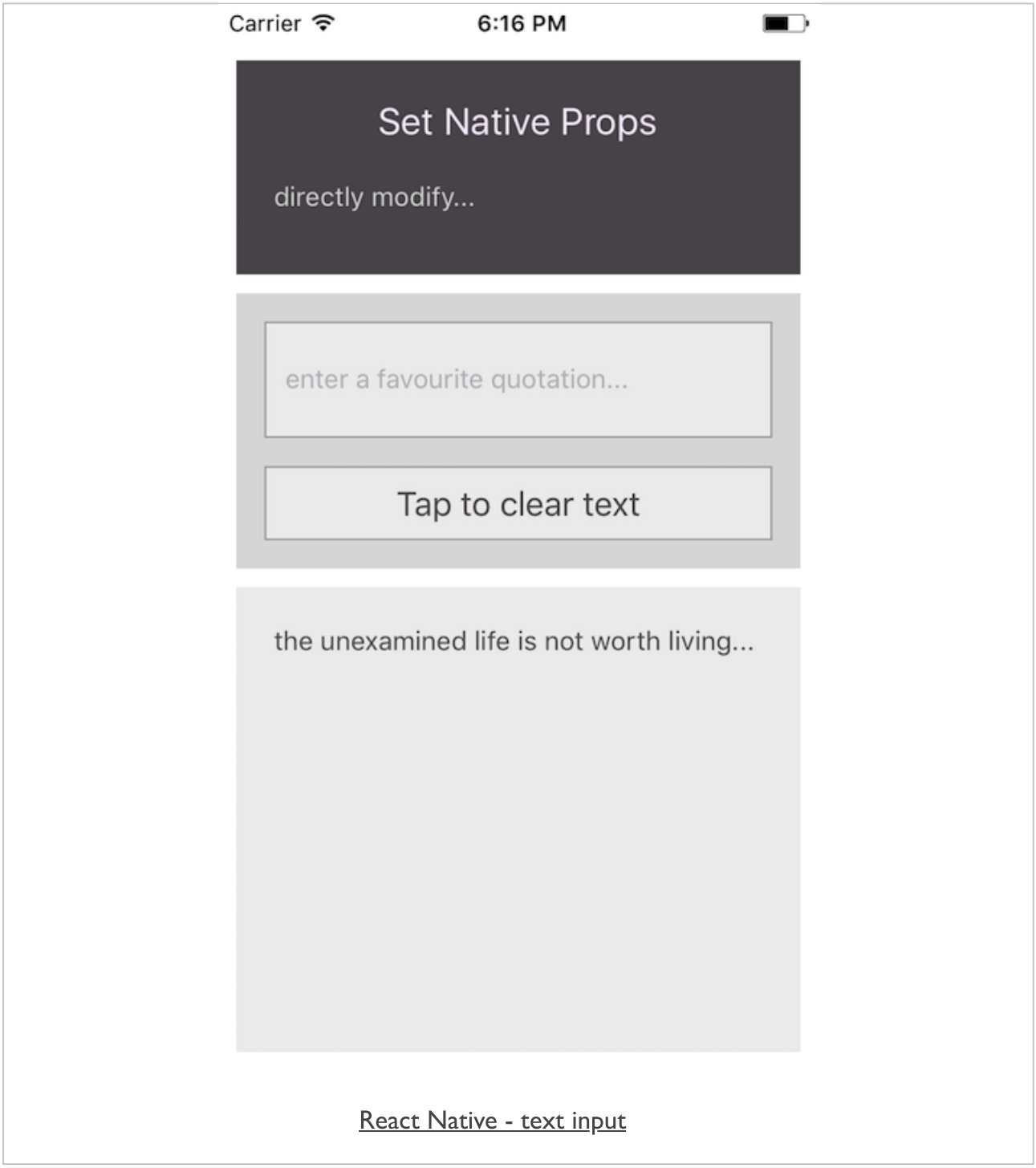
text input



React Native - text input

Image - React Native - Component Usage

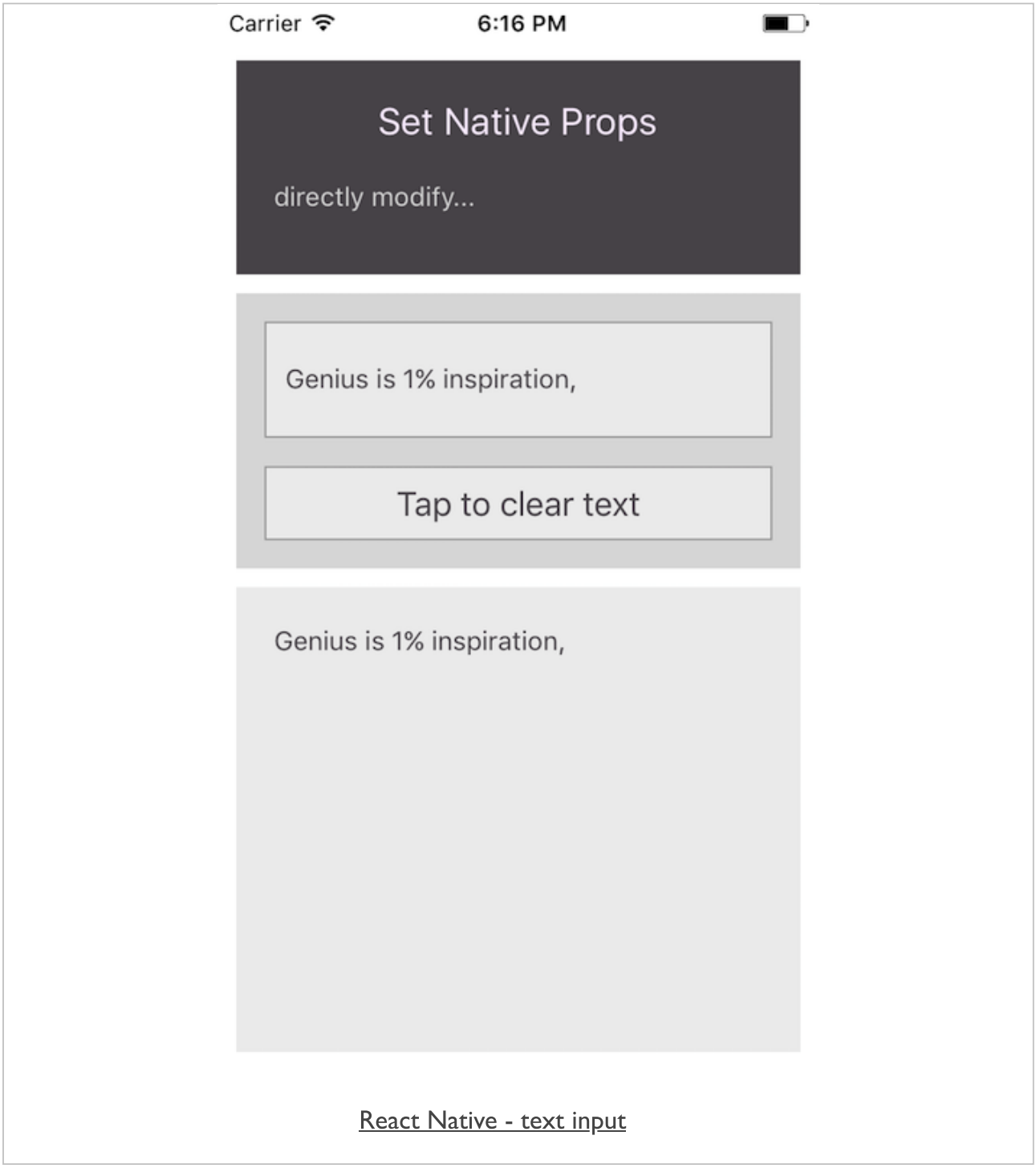
text input - use setNativeProps



React Native - text input

Image - React Native - Component Usage

text input - use setNativeProps



React Native - text input

React Native - component usage

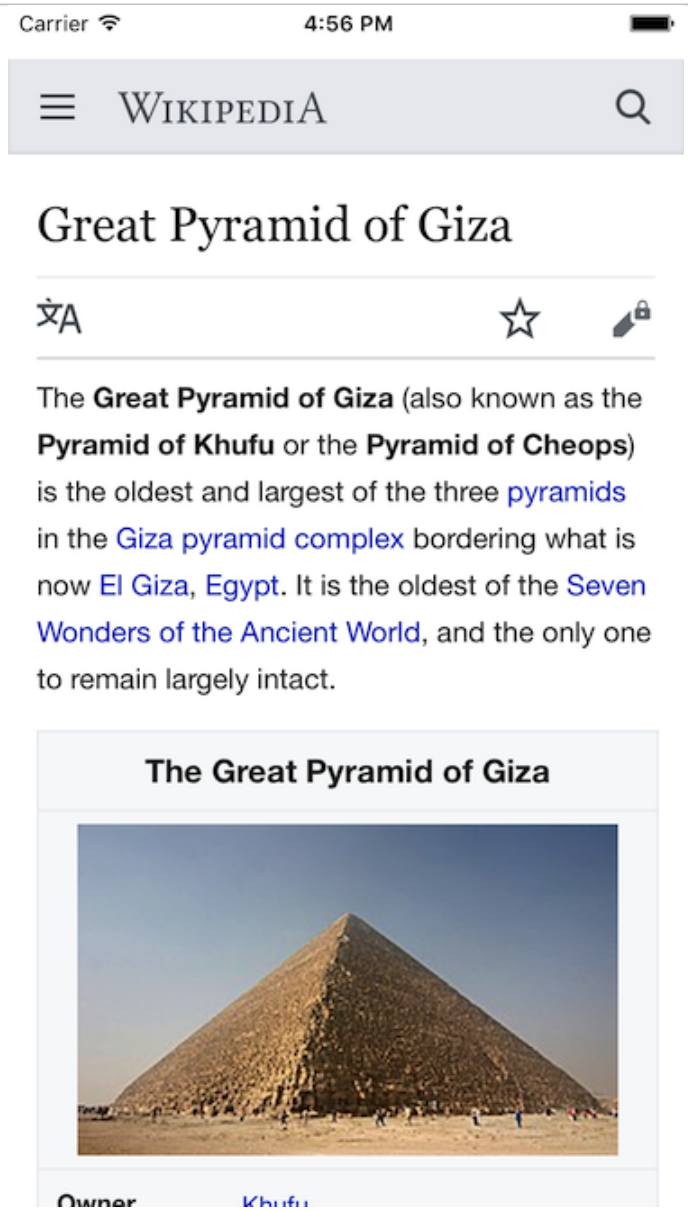
embed web content

- React Native offers a component solution for embedding web content
  - *embedded directly in a WebView*
  - *as a child to an existing view &c.*
- similar functionality to native WebView modules
- `WebView` component provides developers with a variety of props
  - *to help manipulate and structure a rendered web page*
- also use various available callbacks
  - *provide an option to register to specific events*
  - *e.g. error handling, message responses, navigation state change...*

```
<WebView style={styles.web}
  scalesPageToFit
  automaticallyAdjustContentInsets
  source={{
  uri: 'https://en.wikipedia.org/wiki/Great_Pyramid_of_Giza'
  }} />
```

Image - React Native - Component Usage

WebView - load external page &c.



React Native - WebView

React Native - component usage

iOS - SegmentedControlIOS

- some components in React Native may be specific to a given mobile OS
  - *e.g. Segmented Control component is specific to iOS*
- offers a simple split option to switch between two groupings of content
- e.g. we might use this component as follows

```
<SegmentedControlIOS
  values={['Giza', 'Luxor']}
  selectedIndex={this.state.selectedIndex}
  onChange={(event) => {
    this.setState({selectedIndex: event.nativeEvent.selectedSegmentIndex});
  }}
/>
```

- instead of passing expected `onValueChange` props
  - *we can pass a callback prop for `onChange`*
- prop will receive an event argument
  - *e.g. from `nativeEvent` as shown in this example*
- also abstract this usage to pass in required values for each segment

Mobile Design & Development - More UI Components & Usage

Fun Exercise

Four groups, two apps

- Music - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/music/
- Travel Booking - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/travelbooking/

For each app, consider the following

- define UI components for the app?
- how is the app using lists for various views?
- how is the app combining multiple components to create the required UI layout?
  - *e.g. various list views, scrolling, text input &c.*
- how are the UI components defining UX for the app?

~ 10 minutes

References

- React DevTools
- React Native - Layout Props
- React Native - StatusBar