React Native 2

CS571: Building User Interfaces

Cole Nelson

Announcements

- Thursday's class will be recorded & uploaded to Kaltura. CypherCon 2023. **No in-person class.**
- Modality Feedback:
  - **In-Person:** Asking questions, meeting people, consistent schedule, food, end-of-class q's
  - **Online:** Flexibility, pausing/speed control, no Bascom Hill

What will we learn today?

- A review of React Native
- Other core React Native concepts
- Using animations
- Using modals

What is "True Native" Development?

Building specifically for the device (e.g. Android or iOS) that you want to support.

iOS: Objective-C or Swift w/ Cocoapods
Android: Java or Kotlin w/ Maven or Gradle

Pros and Cons of True Native

Pros

- Organic User Experience
- Optimized Apps
- Fine-Grained Control

Cons

- Expensive
- Little Code Reuse
- Less Sense of Abstraction

Alternatives to True Native

No mobile app! Do we really need an app? Could a responsive webpage be just as effective?

WebView! Can we take our existing code and just slap it into a WebView? e.g. Apache Cordova

Cross-Platform! Can we use a library or framework that will make our code work natively on Android *and* iOS? e.g. React Native

What is React Native?

A JS framework for building native, cross-platform mobile applications using React, developed by Facebook in 2015.

Unlike ReactJS, which was a library, React Native is a framework that includes everything* that we will need to build mobile applications.

React Native supports iOS and Android development.

What stays the same?

- Using NPM for our library management
- Using complex APIs
- Core React features
  - React Hooks (useEffect, useState, etc.)
  - Passing props and state management
  - Controlled vs Uncontrolled Inputs
  - Memoization

What changes?

- This isn't a browser!
  - No more DOM!
  - No more CSS!
    - No more Bootstrap!
  - No more sessionStorage, localStorage, or cookies.
- Wider variety of inputs
  - Sensors
  - Gestures
- React Navigation vs React Router

Conversions to Know

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Image Source

Hello World!

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

function MyApp() {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
      <Text>
        Try editing me! 🎉
      </Text>
    </View>
  );
}

export default MyApp;
```

Expo Snack

Styling

Because React Native does not use a "browser", we can't use CSS styles. Instead, we create JavaScript stylesheets.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 40,
  },
  ...
});
```

Styling

Style definitions can be done inline or via stylesheets. You can also combine both methods.

```
<View>
 <Text style={styles.label}>First label</Text>
 <Text style={{fontSize: 28, color:'tomato'}}>Second label</Text>
 <Text style={[styles.label, {fontSize: 20, color:'gray'}]}>Third label</Text>
</View>
```

Snack Solution

Images

`Image` not `img` (must be imported!)

Must specify a width and height: the default is 0!

`source` not `src` which takes an object (not a string)

```
<Image
  style={{
    width: 100,
    height: 100
  }}
  source={{
    uri: "https://example.com/me.png"
  }}
/>
```

Cross-Platform Development

React Native provides a number of components that utilize platform capabilities that may not be available in other platforms, thus for cross-platform development, we need to utilize multiple platform-specific components.

e.g. `TouchableNativeFeedback` only works on Android; a *similar* effect can be achieved using `TouchableHighlight` on iOS.

Differentiating by Platform

```
if (Platform.OS === 'android') {
  return (
    <TouchableNativeFeedback> ... </TouchableNativeFeedback>
  );
} else {
  return (
    <TouchableHighlight> ... </TouchableHighlight>
  );
}
```

Optionally, create two components e.g. `MyButton.ios.js` and `MyButton.android.js`.

Snack Solution

Cross-Platform: Dimensions

Mobile devices vary significantly in screen size, and we o"en need to obtain screen dimensions of the device using the `Dimensions` class in `react-native`.

```
getScreenSize = () => {
 const screenWidth = Math.round(Dimensions.get('window').width);
 const screenHeight = Math.round(Dimensions.get('window').height);
 return { screenWidth: screenWidth, screenHeight: screenHeight };
}
```

ScrollView

Like a `View`, but scrollable! Make sure it is flex-ible.

```
<View style={{flex: 1}}>
  <ScrollView>
    { /* A bunch of content here! */ }
  </ScrollView>
</View>
```

Alternatively, `FlatList` functions the same, but does lazy loading. Furthermore, `SectionList` provides more configuration options.

FindMyBadgers Demo

As a React Native app.

Snack Solution

React Native

Things that differentiate mobile from the web...

- Animations
- Navigation
- Gestures
- Sensing

Animations using `Animated`

Provides *visual aesthetics*, *entertainment* and *feedback*.

```
import { Animated } from 'react-native'
```

May also consider using a third-party-library like react-native-reanimated or lottie.

Animated Docs

Animated

`Animated` provides animations for...

- `View`
- `Text`
- `Image`
- `ScrollView`
- `FlatList`
- `SectionList`

... e.g. `<Animated.View>{/* ... */}</Animated.View>`

Animated

These are animated using...

- `Animated.timing`,
- `Animated.spring`
- `Animated.decay`

... which manipulate an `Animated.Value`, e.g.

```
Animated.timing(opVal, {
  toValue: 1,
  duration: 10000, // in ms
  useNativeDriver: true // must include
})
```

Animated

`Animated.Value` is used in combination with useRef.

```
const opVal = useRef(new Animated.Value(0)).current
```

To run an animation on page load...

```
useEffect(() => {
  Animated.timing(opVal, {
    toValue: 1,
    duration: 10000,
    useNativeDriver: true
  }).start() // don't forget this!
}, [])
```

```
export default function FadeInView(props) {
  const opVal = useRef(new Animated.Value(0)).current;
  useEffect(() => {
    Animated.timing(opVal, {
      toValue: 1,
      duration: 5000,
      useNativeDriver: true,
    }).start();
  }, []);
  return (
    <View>
      <Animated.View
        style={{
          height: 100, width: 100, opacity: opVal
          backgroundColor: "cyan",
        }}>
      </Animated.View>
    </View>
  );
}
```

Expo Snack

Animated

Can control many animations using...

- `Animated.parallel`
- `Animated.sequence`
- `Animated.loop`

`start()` and `stop()` apply to the set of animations

In parallel...

```
useEffect(() => {
  Animated.parallel([
    Animated.timing(height, {
      toValue: 800,
      duration: 10000,
      useNativeDriver: false, // cannot use native driver for height/width!
    }),
    Animated.timing(width, {
      toValue: 500,
      duration: 10000,
      useNativeDriver: false,
    })
  ]).start()
}, []);
```

In sequence...

```
useEffect(() => {
  Animated.sequence([
    Animated.timing(sizeVal, {
      toValue: 500,
      duration: 10000,
      useNativeDriver: false,
    }),
    Animated.timing(sizeVal, {
      toValue: 0,
      duration: 10000,
      useNativeDriver: false,
    })
  ]).start()
}, []);
```

In loop...

```
useEffect(() => {
  Animated.loop( // not an array!
    Animated.sequence([
      Animated.timing(sizeVal, {
        toValue: 500,
        duration: 10000,
        useNativeDriver: false,
      }),
      Animated.timing(sizeVal, {
        toValue: 0,
        duration: 10000,
        useNativeDriver: false,
      })
    ])
  ).start()
}, []);
```

Animated Demo

In class, let's build this together.

Animated

You cannot *directly* add/subtract/multiply/divide `Animated.value`. Instead, you must use...

- `Animated.add(v1, v2)`
- `Animated.subtract(v1, v2)`
- `Animated.multiply(v1, v2)`
- `Animated.divide(v1, v2)`

Animated

e.g. start at 50 and grow from there.

```
<Animated.View
  style={{
    backgroundColor: "blue",
    height: Animated.add(height, 50),
    width: Animated.add(width, 50)
  }}>
</Animated.View>
```

Your turn!

Fork FindMyBadger (React Native). Can you add animations so that...

- The picture fades in?
- The picture gradually becomes bigger?
- The screen automatically scrolls down to the bottom?

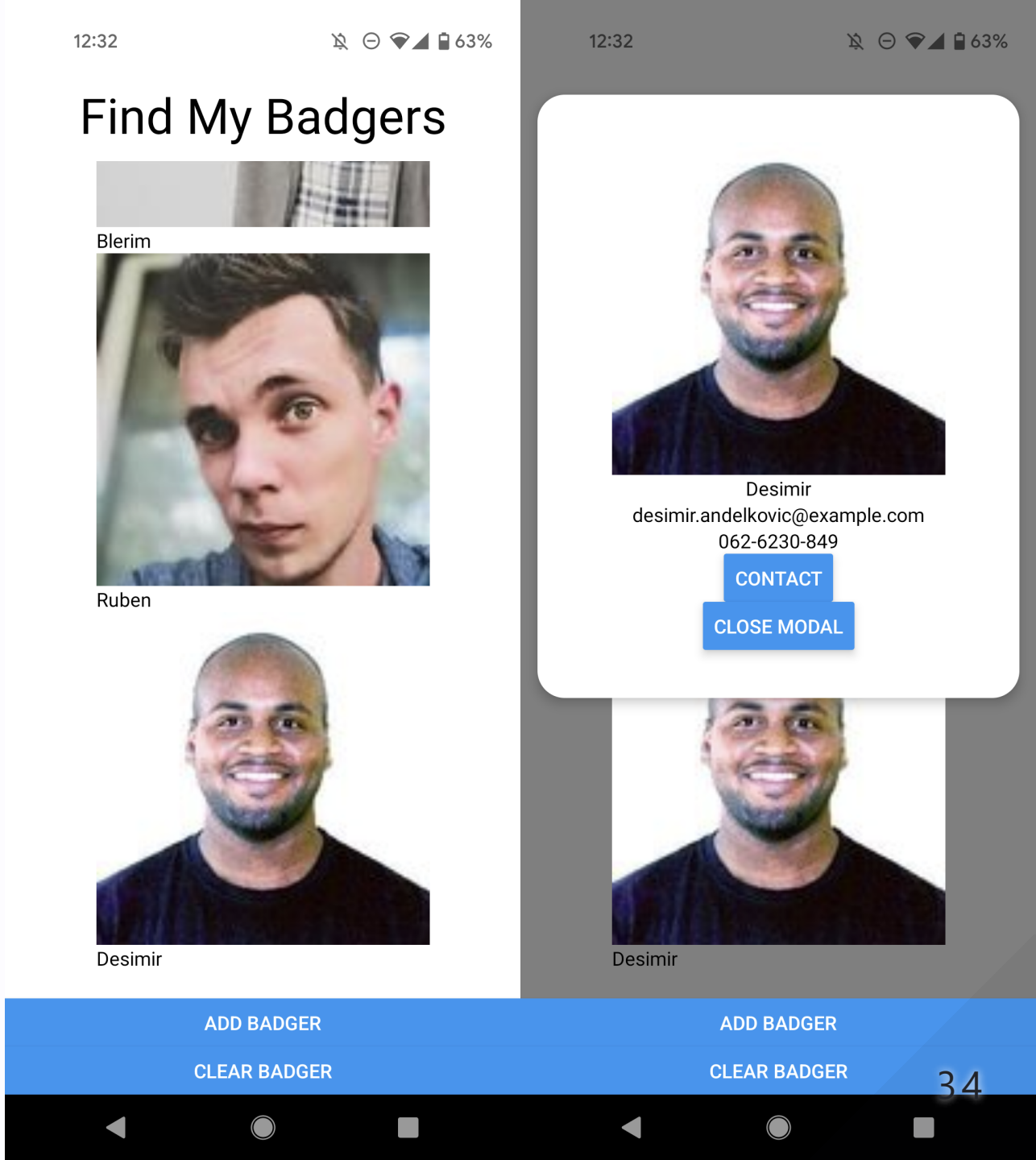
Expo Snack Solution

Modal

A secondary window.

```
import { Modal } from 'react-native'
```

Alternative: react-native-modal



Modal

What you need...

1. Something to open the modal
2. Some content inside the modal
3. Something to close the modal

We often manage whether the modal is open or closed using a state variable, e.g.

```
const [modalVisible, setModalVisible] = useState(false);
```

Modal

A modal is nothing more than a secondary overlay! You will need to style it.

```
const styles = StyleSheet.create({
  modalView: {
    margin: 20,
    alignItems: 'center',
    shadowColor: '#000',
    shadowOpacity: 0.25,
    shadowRadius: 4,
    elevation: 5,
    // ...
```

Modal Properties

- `animationType` : 'slide', 'fade', 'none'
- `onShow` : callback func
- `onRequestClose` : callback func
- `transparent` : true/false
- `visible` : true/false often handled by a state variable

Modal Docs

Modal Snack

FindMyBadgers Demo

As a React Native app with animations and modals.

Snack Solution

What did we learn today?

- A review of React Native
- Other core React Native concepts
- Using animations
- Using modals

Questions?