Mobile Dev 2

CS571: Building User Interfaces

Cole Nelson

Today's Warmup

- Set back and relax! :)
- We'll try our in-class exercise through "Expo Snacks" today instead of downloading starter code.

Learning Objectives

1. Be able to create a scrollable interface.
2. Be able to create generic custom components.
3. Be able to perform animations.
4. Be able to provide for navigation.

How can I display more?

Introducing the `ScrollView`.

ScrollView

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

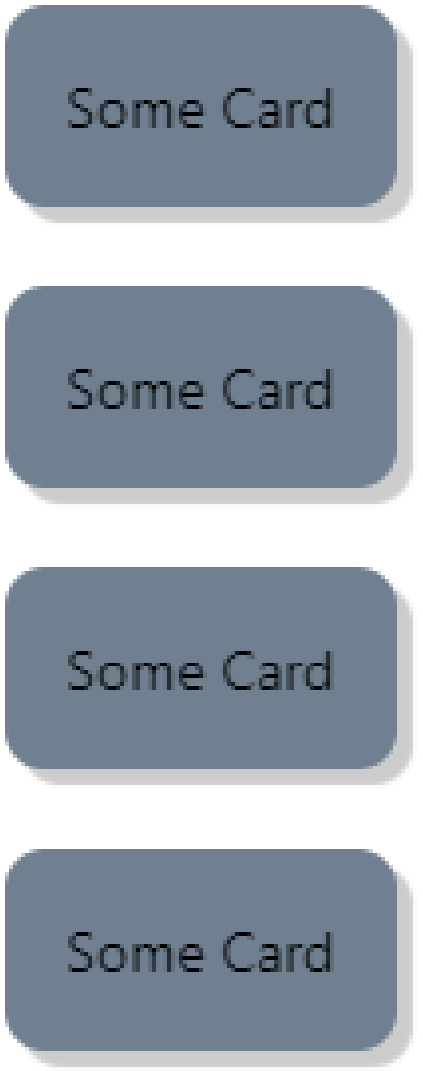
```
<View style={{flex: 1}}>
  <ScrollView>
    { /* A bunch of content here! */ }
  </ScrollView>
</View>
```

Snack Solution

Card

React Native does not have the concept of a "card"...

1. Use a third-party library like `react-native-paper`.
2. Create our own component!



Card

```
export default function BadgerCard(props) {
    return <Pressable onPress={props.onPress}>
        <View style={[styles.card, props.style]}>
            {props.children}
        </View>
    </Pressable>
}
```

Pressable | React Native Paper Card

Card

Adding in the `styles.card` ...

```
const styles = StyleSheet.create({
    card: {
        padding: 16,
        elevation: 5,
        borderRadius: 10,
        backgroundColor: 'slategray',
    }
})
```

Card

Using the BadgerCard...

```
function App() {
  return <View style={styles.main}>
    <BadgerCard
      onPress={() => Alert.alert("Hello", "World!")}
      style={{backgroundColor: "red"}}
    >
      <Text>Testing Custom Card</Text>
    </BadgerCard>
  </View>
};
```

Adding Gestures

```
export default function BadgerCard(props) {
    return <Pressable
      onPress={props.onPress}
      onLongPress={props.onLongPress}
    >
        <View style={[styles.card, props.style]}>
            {props.children}
        </View>
    </Pressable>
}
```

Snack Solution

Animations

Providing *feedback* in a *visually aesthetic* way.

Animations using `Animated`

We must specify *what* we want to animate, and *how* we want to animate it.

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

Animated Docs

`Animated` : Choosing *What*

`Animated` provides animations for…

- `View`
- `Text`
- `Image`
- `ScrollView`

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

`Animated` : Choosing *What*

```
<Animated.View style={{ opacity: op }}>
  <Image
    style={styles.pic}
    source={{
      uri: props.img,
    }}></Image>
  <Text>{props.name}</Text>
</Animated.View>
```

Could apply the animation to the `Image` and `Text` individually, or wrap both in a `View`.

`Animated` : Choosing *How*

A much more complicated question!

- Do I want the component to fade in?
- Do I want the component to grow in size?
- Do I want the component to bounce around?
- Do I want to do a combination of these things?
  - Do I want to do it in sequence?
  - Do I want to do it in parallel?

`Animated` : Choosing *How*

Also consider...

- When should it be triggered?
  - On component load?
  - On button press?
  - On gesture?
- How long should it last?
  - Can it be started or stopped?

Expo Snack

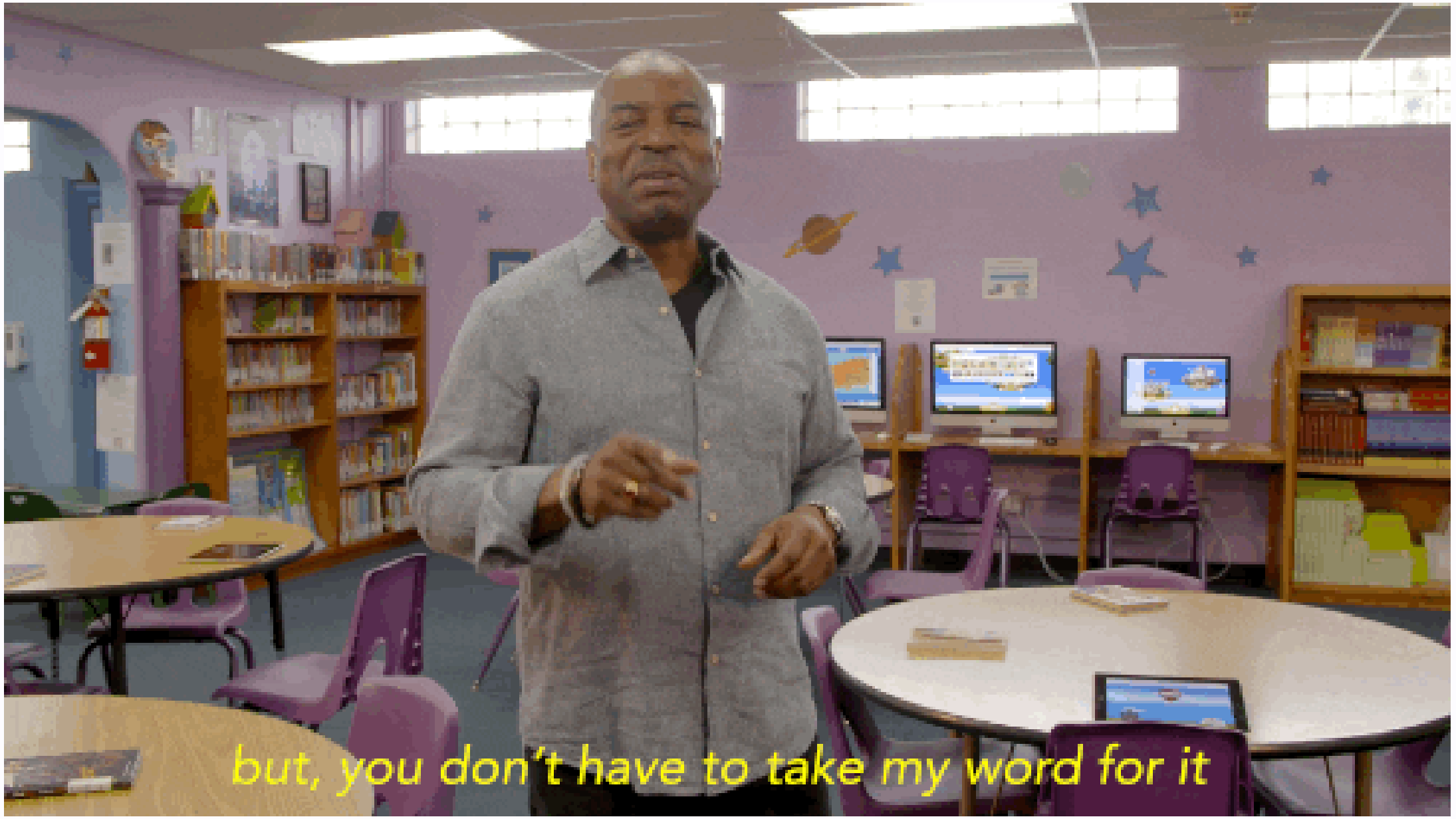
Animated

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

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

To run this animation...

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


but, you don't have to take my word for it

What does this do?

```
Animated.timing(opVal.current, {
  toValue: 1,
  duration: 10000, // in ms

})
```

Increases the `Animated.Value` stored within `opVal` from 0 to 1 over 10 seconds.

What does this do?

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

Uses the animation libraries *native* to Android/iOS as opposed to JavaScript calculated values.

Cannot be done for animations like increasing size!

What does this do?

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

Starts the animation.

What does this do?

```
Animated.timing(opVal.current, {
  toValue: 1,
  duration: 10000, // in ms
  useNativeDriver: true
}).start((finished) => {
  // TODO I'm done!
})
```

Optional: Run a function on animation completion.

What does this do?

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

We know what `useEffect` does!

Runs the animation on component load.

What does this do?

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

Creates a *reference* to `Animated.Value`. Can be referred to with `opVal.current`.

Notice: This is *not* a state variable! A change to `opVal`'s referenced value will not trigger a re-render. In fact, animations occur outside of React.

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

In sequence...

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

In loop...

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

Animated Demo

Expo Snack

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.current, 50),
    width: Animated.add(width.current, 50)
  }}>
</Animated.View>
```

Your turn!

Take this snack and make it so that the Badgers fade in as they are added.

Hint: Only change the `Badger` component.

Expo Solution

Navigation in React Native

A more mobile-centric library.

React Navigation Alternatives

React Native is a framework* but still lacks support for things like navigation.

- React Router previously!
- React Navigation new!
- `return isHome ? <HomeScreen> : <SettingsScreen>`

React Navigation

We will use...

- Tab Navigation: `@react-navigation/bottom-tabs`
- Drawer Navigation: `@react-navigation/drawer`
- Stack Navigation: `@react-navigation/native-stack`

...others exist!

Navigation Basics

- Must be nested inside of a `NavigationContainer`
- Create navigators via a function `createNAVIGATOR()`
  e.g. `createBottomTabNavigator()`
- Navigators consist of a *navigator* and a set of *screens*

```
<NavigationContainer>
 <SomeNav.Navigator>
   <SomeNav.Screen name="Bookstore" component={BookstoreScreen}/>
   <SomeNav.Screen name="Book" component={BookScreen}/>
 </SomeNav.Navigator>
</NavigationContainer>
```

Navigation Basics

- `useNavigation` is a custom React hook that can be used to help us navigate
  - Supports `navigate`, `reset`, `goBack` among others
- Information can be passed from screen to screen via *route params* (see Native Stack Navigator example)
- Navigators can be styled
- Navigators can be nested

Tab Navigation

```
const SocialTabs = createBottomTabNavigator();

<NavigationContainer>
  <SocialTabs.Navigator>
    <SocialTabs.Screen name="NewsFeed" component={NewsFeedScreen}/>
    <SocialTabs.Screen name="Notifications" component={NotificationScreen}/>
    <SocialTabs.Screen name="AboutMe" component={AboutMeScreen} />
  </SocialTabs.Navigator>
</NavigationContainer>
```

Expo Snack Solution

Drawer Navigation

```
const SocialDrawer = createDrawerNavigator();

<NavigationContainer>
  <SocialDrawer.Navigator>
    <SocialDrawer.Screen name="NewsFeed" component={NewsFeedScreen}/>
    <SocialDrawer.Screen name="Notifications" component={NotificationScreen}/>
    <SocialDrawer.Screen name="AboutMe" component={AboutMeScreen} />
  </SocialDrawer.Navigator>
</NavigationContainer>
```

Expo Snack Solution

Stack Navigation

```
const BadgerStack = createNativeStackNavigator();

<NavigationContainer>
  <BadgerStack.Navigator>
    <BadgerStack.Screen name="People" component={AllBadgersScreen}/>
    <BadgerStack.Screen name="Person" component={SpecificBadgerScreen}/>
  </BadgerStack.Navigator>
</NavigationContainer>
```

Expo Snack Solution

Stack Navigation

Can push a screen onto the history stack via `navigation.push(screenName, params)`

- `screenName` is the name of the screen to navigate to, e.g. `Book`
- `params` is an optional object of parameters *sent* to the receiving screen.
- `params` is *received* as `props.route.params`

Nested Navigation

- Navigators can be nested.
  - Stack in Tabs
  - Stack in Drawer
  - Stack in Tabs in Drawer (e.g. Example Below)
  - Stack in Stack in Tabs
  - Stack in Stack in Stack in Stack in Stack
- Make use of the `headerShown` option!

Expo Snack Solution

Questions?