Week 3 Workshop

React Native Course

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Crankwheel	10 minutes
Check-In	10 minutes
Weekly Recap, Quiz Review	60 minutes
Tasks 1 & 2	40 minutes
BREAK	15 minutes
Tasks 2 & 3	90 minutes
Check-Out	15 minutes

Check-In

- How was this week? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- You must complete all Exercises before beginning the Workshop Assignment. We will review some of the Exercises together next, along with this week's concepts.

Week 3 Recap - Overview

Most of the New Concepts You Learned This Week

• Swipe • Alert • Animated	• Animatable • Gestures / PanResponder • Persist Redux Store

Next slides will review these concepts,
along with most of the Exercises you did this week.

Deleting Favorites

You saw the use of this code in the Swipe exercise to delete a favorite from the state in Redux (in favorites.js):

```
case ActionTypes.DELETE_FAVORITE:
    return state.filter(favorite => favorite !== action.payload);
```

- **Discuss together:** Let's make sure that everyone understands how and why this code works to remove a favorite. Is there any student volunteer who will explain it verbally for the class?

Alert API

With Android, there are at most three buttons that can be specified in an Alert dialog.

As an exercise in consulting the documentation: What is the number of buttons that can be specified for iOS?

Alert API

Answer: Unlimited / any number

Animated API

- In **Exercise: Animations**, you learned to use the built-in Animated API for React Native.
- In order to animate a component for this exercise, you had to:
  - Create and initialize an Animated.Value
  - Use an Animated.timing() method with a toValue and duration, and .start() it by calling it from somewhere in your code (in this case, the componentDidMount() lifecycle method)
  - ...what was the last step? **Discuss together.**

Animated API (cont)

- The last step is to configure a style prop, such as scale or color, by setting the Animated.Value as its value, as we did in the HomeComponent:

```
<Animated.ScrollView style={{transform: [{scale: this.state.scaleValue}]}}>
```

Animatable

Animatable is a third party library built using the Animated API and provides easier ways to use a wide range of basic animations

It provides both a *declarative* and *imperative* usage.

You saw its declarative usage in the *Animatable* exercise, where you used Animatable.View and passed it animation, duration, and delay props to quickly make a component animate when mounted:

```
<Animatable.View animation='fadeInDown' duration={2000} delay={1000}>
```

You saw its imperative usage in the *Gestures Part 2* exercise (reviewed in later slide)

Handling gestures with PanResponder

- The built-in PanResponder API is a sophisticated way of handling many different types of gestures, from multi-finger gestures, to short taps, to swipes
- Support for entire lifecycle of the gesture, from the beginning of the gesture to the end, using various panHandlers
- Each panHandler receives by default the native event object (typically named as e, evt, or event) and an object containing information about the gesture, typically named as gestureState

Handling gestures with PanResponder (cont)

- Each panHandler receives by default the native event object (typically named as e, evt, or event) and an object containing information about the gesture, typically named as gestureState
- According to the RN documentation, you have access to these properties of the event and gestureState objects:

`nativeEvent`

- `changedTouches` - Array of all touch events that have changed since the last event
- `identifier` - The ID of the touch
- `locationX` - The X position of the touch, relative to the element
- `locationY` - The Y position of the touch, relative to the element
- `pageX` - The X position of the touch, relative to the root element
- `pageY` - The Y position of the touch, relative to the root element
- `target` - The node id of the element receiving the touch event
- `timestamp` - A time identifier for the touch, useful for velocity calculation
- `touches` - Array of all current touches on the screen

A `gestureState` object has the following:

- `stateID` - ID of the gestureState- persisted as long as there at least one touch on screen
- `moveX` - the latest screen coordinates of the recently-moved touch
- `moveY` - the latest screen coordinates of the recently-moved touch
- `x0` - the screen coordinates of the responder grant
- `y0` - the screen coordinates of the responder grant
- `dx` - accumulated distance of the gesture since the touch started
- `dy` - accumulated distance of the gesture since the touch started
- `vx` - current velocity of the gesture
- `vy` - current velocity of the gesture
- `numberActiveTouches` - Number of touches currently on screen

Handling gestures with PanResponder (cont)

- We can use these properties in combination with the panHandler to gain information about a gesture and respond to it however we wish
- In **Gestures Part 1**, for example, we used onPanResponderEnd to wait until a gesture ended, then checked the dx value from its gestureState to figure out if it was over 200 pixels from right to left
- Then we responded to it by popping up an Alert – but our response could have been anything, not limited to Alerts

Animatable & PanResponder

In **Gestures Part 2**, you learned to combine gesture support and animations in the CampsiteInfo component by using the Animatable library in a different (imperative) way, along with a React ref:

- We created a new ref in the RenderCampsite component using the React.createRef method:

```
const view = React.createRef();
```

- We then used it as the value for the ref prop for the Animatable.View component:

```
ref={view}
```

- Then, the onPanResponderGrant panHandler fires when a panResponder becomes activated, and we set it to call the rubberBand method from the Animated library on the currently mounted instance of the Animatable.View component that the view ref refers to:

```
onPanResponderGrant: () => {
    view.current.rubberBand(1000)
```

Redux Persist Store

- The `redux-persist` library allows you to access permanent storage in your device and write information to it, then 'rehydrate' your information when an application is reopened
- Different ways: local storage, session storage, AsyncStorage, secure store, etc.
- By default, `redux-persist` (versions 5 and below) uses local storage. (This has changed in versions 6 and up)

Exercise: Persist Redux Store

- In this exercise, you:
  - Installed the redux-persist library
  - In configureStore.js, added persistence support to reducers by creating a config object to set key, storage, and debug, then replacing combineReducers with persistCombineReducers and passing it the config object
  - Also used persistStore function to obtain a persistor object from the store:
  - Returned both the store and the persistor from configureStore.js:

```
const persistor = persistStore(store);
```

  - Imported both into App.js, then wrapped the root component in App.js with a <PersistGate> component, passing the persistor to it.

Quiz & Code Challenges

- The Workshop Assignment this week will be shorter than usual, so less time has been allotted for it in the Agenda
- There should be time to go over the Quiz questions together before the assignment
- If everyone finishes the assignment early, you can use the extra time to go over the Code Challenges

Workshop Assignment

- It's time to start the workshop assignment!
- Sit near your workshop partner.
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner, you should consult each other throughout.
- Follow the workshop instructions very closely
  - both the video and written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Submit your 2 assignment files in the learning portal:
  - *ReservationComponent.js*
  - *CampsiteInfoComponent.js*
- Wrap up – Retrospective
  - What went well
  - What could improve
  - Action items
- Start Week 4 or work on your Portfolio Project