Platform-Specific Development Lab

We really try to make all of our code work on all platforms. It would be great if we could do that 100% of the time! But we've seen that sometimes it's necessary to detect the type of OS our app is running on at the time and change how it behaves. We'll work on that in this lab by adding a DatePicker that is intelligent enough to change how it displays based on the OS.

There is a library that does that already. In fact, your IDE will tell you that DatePickerIOS and DatePickerAndroid are deprecated in favor of that library. But we want some practice in doing this ourselves rather than relying on a library. So ignore the warnings and press on!

Implementing DatePickerIOS

1. Edit Landing.js. In the JSX, add a DatePickerIOS just above the list of films, setting the date to today. Write the onDateChange event to just console.log() the date picked..
2. If you're able to debug this on an iPhone device or iOS simulator, run and test. Make sure you can see the date chosen in the console.

This is great, but it'll only work on iOS. Let's see if we can support it on Android.

Implementing DatePickerAndroid

3. Add a Button. Since we haven't really covered <Button> yet, here's how it might look:

```
<Button title="Pick a showing date" onPress={pickDate} />
```

4. The pickDate function should call DatePickerAndroid.open. You can either use async/await or handle the Promise, but you want to get the date chosen in a local variable.
5. Console.log() that local variable.
6. Run and test on an Android device or emulator until you can see the chosen date in the console.

Using the Platform module

At this point we have an app that works on both platforms but also errors on both platforms. Let's get rid of those errors by detecting the OS. We can do that with the Platform module.

7. import Platform at the top of Landing.js.
8. In the JSX, find where you're including the <DatePickerIOS> and use an expression to make this conditional based on us being on an iOS device. Something like this:

```
{ Platform.OS === "ios" && <DatePickerIOS .... >}
```

9. Then, where you're drawing the button, make it conditional on the Platform.OS being "android"
10. Run and test. Adjust until it is working.

Extracting the DatePicker

This is great and all but we're going to want to have this functionality on another component. Let's extract it so it can be reused.

11. Create a new component called DatePicker.
12. It should receive in selectedDate via props.
13. It should get a dispatch using the react-redux hook to communicate with the store
14. Make the JSX return a Button:

```
<Button onPress={() => showModal()}
  title={`Showing times for ${selectedDate.toDateString()}`} />
```

15. Write the showModal function at the top of the DatePicker function. If we're on Android, it should DatePickerAndoid.open(). (Hint: Take out the logic from Landing.js and put it here to set a date on Android).
16. In that success callback, go ahead and ...

```
store.dispatch(
 {type: "SET_SELECTED_DATE", date:new Date(date.year, date.month, date.day)})
```

17. And of course you'll need to handle that action in reducers.js. Put in another case sort of like this:

```
case "SET_SELECTED_DATE":
  return { ...state, selectedDate: action.date };
```

18. Pause. We have selectedDate managed in Redux state. How do we access that? We could access in App and “prop drill” through Landing to pass it along, or you could access it directly in Landing. Think about why you like it one way or another.

(Hint: Don't forget to add imports at the top of DatePicker.js and remove the unused ones from Landing.js. We're assuming that as experienced React developers you know this already).

19. Run and test. On iOS nothing works yet. But on Android, you should be able to press the button and then pick a date which is saved in our application state and displayed on the button.

Now for iOS

20. At the top of the DatePicker component, add a state hook:
21. We're going to want to toggle show/hiding the DatePickerIOS, so add a state hook to maintain it:

```
const [showIosPicker, setShowIosPicker] = useState(false);
```

22. In showModal if we're on iOS, set a variable called showIOSPicker in state:

```
setShowIosPicker(!showIOSPicker );
```

    Hint: Don't forget all the rules of JavaScript scoping. You will want to move the showModal function into the right scope if you have troubles with it recognizing showIOSPicker and setShowIOSPicker. Or you can pass references into showModal.
23. Back in the JSX, conditionally show the DatePickerIOS control. If showIOSPicker is truthy, show it. If not, don't.
24. Set your DatePickerIOS's date property to the prop selectedDate.
25. Make its onDateChange event call setShowIOSPicker(false) and dispatch the "SET_SELECTED_DATE" action to the store. (Hint: it will be kind of like the Android dispatch, but not exactly like the Android dispatch. Watch out!)
26. Run and test on iOS. You should be able to hit the button and see an iOS date picker show. When the user chooses a date, the selector should disappear and the date displayed in the button should change.

At this point you should have the ability to select a date on iOS and on Android despite the hurdle that they're both very different in how they handle date pickers. That's pretty cool. Our app may not look pretty yet but don't worry, we'll handle that in future labs.

Bonus!! If you don't like how choosing a date closes the iOS date picker immediately, you can add a button to your DatePicker component and only close it when the user presses the button.