Navigation Lab

Our app is getting close! We have some really cool scenes. But the whole app is disjoint because we can't get to all the scenes naturally. In this lab, we're going to join them together. From the Landing or Film Details scenes, if the user picks a showing, we'll navigate them to Pick Seats. Then, when they hit the checkout button, they should go to the Checkout scene. And finally, when they pay with a credit card, we'll send them back to Landing so they can choose more movies if they want.

They're not tapping on tabs to get around so a tabNavigator would be inappropriate. They are just going from scene to scene through taps. That is the perfect situation for a stackNavigator.

1. Before we can navigate we have to install it
2. npm install react-navigation

Setting up navigation

3. Open App.js in your IDE.
4. import createStackNavigator at the top.
5. Then (outside of your class, call createStackNavigation and pass a simple routing object to it. Make it look like this:
6.

```
const routes = {
  Landing: {
    screen: Landing,
  },
};
const MyStackNavigator = createStackNavigator(routes);
```

Remember, this function creates a React Native component that you must render in JSX to use.

7. Render this component instead of <Landing>
8. Run and test. It should fail in the Landing component. Probably something about you trying to .map() through an undefined. Regardless of the error, the first problem is that in Landing, you're reading props but we can't pass props to something we're navigating to.

There are few ways to solve this problem, but the best way may be to move state out of App.js and into Landing.js.

9. Convert Landing to a class-based component if it isn't already one.
10. In the constructor go ...

```
this.state = {...store.getState()};
store.subscribe(() => this.setState(store.getState()));
```

And now that we have this.state, we should be using it instead of props throughout.

11. You could do a global search for "props." and replace it with "this.state." but here's another thought: In the render method do something like this:

```
const { showings, films, selected_film, showFilmDetails } = {...this.state};
```

... and then remove all the "props." (If you didn't follow that, don't worry. Just do a global search and replace).

12. Run and test again. You should see your list of films again. You should also be able to tap any one of them and see the details of that film.

Navigation Config Options
The layout and look of our page has changed some. There's now a new area at the top of our page. That's the navigation header which is added by navigation to hold a place for the back/forward buttons. That area is customizable.

13. Edit App.js and add this:
```
const stackNavConfig = {
  initialRouteName: 'Landing',
  navigationOptions: {
    headerStyle: {
      backgroundColor: 'rgb(11, 134, 232)',  // Or whatever color you like
    },
    headerTintColor: 'rgb(17, 190, 255)',  // For the back/forward buttons
    headerTitleStyle: {
      fontWeight: 'bold',
    },
  },
}
```
14. Run and test. Adjust the colors until you're happy with it.

Overriding nav settings
These settings will be for all scenes that we navigate to. And it may be appropriate to have a back button on some pages and a forward button on others, but don't you think it looks bizarre on the Landing scene where this bar does nothing for them? Let's remove it only for the Landing scene.

15. Edit Landing.js. Add a static property called navigationOptions that has one property, *header* which is set to *null*.
16. Run and test. Adjust until the header is gone.

Navigating to PickSeats
The list of showings in ShowingTimes should be tappable. When the user taps one, we should navigate them to PickSeats. Unfortunately the *navigation* function is only on this.props.navigation in a component that we navigated to. And ShowingTimes was not navigated to. It was nested. So...

17. Open Landing.js in your IDE and add this method:
```
chooseTime = showing => {
  this.navigation.navigate('PickSeats');
}
```
18. Then pass it down to FilmDetails as a prop (hint: chooseTime={this.chooseTime}.
19. Edit FilmDetails.js and pass it down again to ShowingTimes as a prop.
20. Open ShowingTimes.js in your IDE and find where we're creating the <Text>s with the times. Add an onPress event to each.
```
onPress={() => props.chooseTime(showing)}
```
21. Run and test. You should be navigating to PickSeats when the user taps on a time in ShowingTimes but you'll probably encounter an error or two due to props not being populated.

Do you know why? It is because we can't pass props to PickSeats when we're navigating to it. And when navigating, input values aren't sent as props, they're sent as navigation params. We have to send them as params and then read them as params

22. In Landing.js, change the chooseTime method to send params:
```
this.navigation.navigate('PickSeats', { showing });
```
23. Edit PickSeats and look for everywhere you're reading a value from props. Those values should be taken from props.navigation.state.params instead. Maybe do some destructuring like we did above?

```
const {selected_film,selected_date,showing} = props.navigation.state.params;
```

24. Run and test. Now when you tap a time, the Landing scene should navigate you to the PickSeats scene.

25. Bonus! You may have noticed that we can't really navigate until the Modal hosting FilmDetails is dismissed. If you have extra time, go ahead and fix that. Two ways to do it: 1) Pull it out of the modal and convert it to a navigable scene, passing params and adding it to the StackNavigator's config object. Or 2) just going "store.dispatch({ type: 'HIDE_FILM_DETAILS' }" in chooseTime() in Landing.js.

Navigating to CheckOut

Now let's make the check out button at the bottom of PickSeats navigate us to the CheckOut scene.

26. Go ahead and edit PickSeats.js. Find the check out button and make it navigate to Checkout.
27. Run and test. Are you getting to Checkout? Move on when you are.

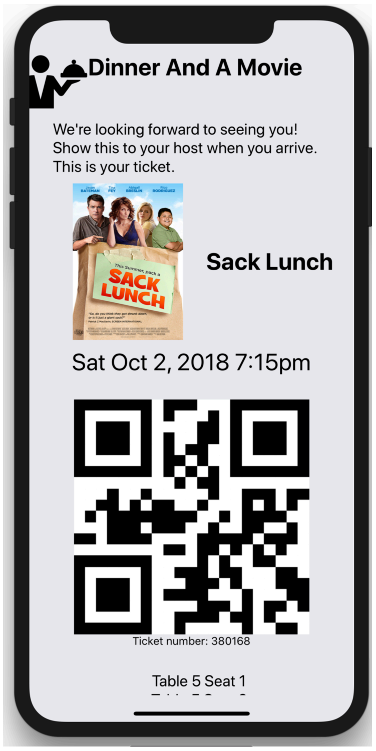
One final Scene!

In Checkout, when the user hits the purchase button, nothing happens. Let's send them to a Ticket scene with a confirmation number so when they arrive at our theater we can link this person to the seats they just purchased.

28. Create a new component called Ticket.js.
29. In it, simulate creating a ticket_number. Randomly-generate a number between 50,000 and a million.
30. The component should say "We're looking forward to seeing you soon. Please show this to the host when you arrive. This is your ticket."
31. Then say "Ticket number: {ticket_number}"
32. Go into Landing.js. Add this scene to your route configuration.
33. Edit Checkout.js. Make the button navigate to Receipts.
34. Run and test. You should now be able to get to any screen you like.

35. Bonus! In the Ticket component, tell the user what movie they're seeing, the date and time of the showing and the table and seats they reserved.

36. Extra bonus! Make the ticket component print a barcode or QR code that our hosts can scan when the customers arrive. (Hint: you'll have to install an npm library to make this happen. react-native-barcode-builder and react-native-qrcode might help you out.)



We're looking forward to seeing you!
Show this to your host when you arrive.
This is your ticket.

Sack Lunch

Sat Oct 2, 2018 7:15pm

Ticket number: 380168

Table 5 Seat 1