Navigation Lab

Our app is getting close! We have some really cool scenes. But the whole app is disjointed because we can't get to all the scenes naturally. In this lab, we're going to join them together. From the Film Details component, 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 show them an electronic ticket and send them back to Landing so they can choose more movies.

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.

Remember, each type of navigation (stack, tab, drawer) share a common setup. Let's do that first.

Setting up common navigation

1. Before we can navigate we have to install it

```
npm install @react-navigation/native
```

2. After that, you'll need to install some peer dependencies. You can use expo install to pick the right version that's compatible with your expo SDK:

```
expo install react-native-screens react-native-safe-area-context
```

3. Open App.js in your IDE. Wrap everything in a NavigationContainer. (Hint: Don't forget to import).

Creating the basic stack navigator

Now that the common setup is finished, we could implement any of the three types of navigation schemes. All three have their own installs. Since we're going to use the stack navigator, let's set it up next.

4. Do the install

```
npm install @react-navigation/native-stack
```

5. import createNativeStackNavigator at the top of App.js.
6. Create your stack navigator. Call it Nav.

```
const Nav = createNativeStackNavigator();
```

Your scenes (pages, screens, whatever you prefer to call them) will appear on the screen wherever your "navigator" is. If you want to put in a header View that never changes, that would be fine. If you desire, go ahead and put that in a View with a logo and big text or whatever. But it also makes sense to allow each scene to take up the entire screen. These instructions will be written that way.

7. Inside the outermost enclosing view, go ahead and put a <Nav.Navigator>.

Let's give it one route to begin with, the landing route.

8. Add a <Nav.Screen> for the landing page. I might make the screen name "Landing" or "Home" but you can name it whatever you choose.
9. Make sure you have an expression as a child of the <Nav.Screen>. It must be a function and that function must return the <Landing> component in JSX.
10. Run and test. You shouldn't see any changes at this point but you shouldn't see any errors either.

Adding the other routes

11. Add a <Nav.Screen> for the PickSeats scene and another for the Checkout scene.
12. If you change the order of these Screens and then re-run, you'll be able to see one and only one scene at a time. Give that a try.

13. Now do the same thing but don't change the order, change the <Nav.Navigator>'s initialRouteName prop. Make sure you can see each of your other scenes as you change them. (Note!! A change to initialRouteName is one of those very few things that a refresh won't catch automatically. You'll need to refresh manually after each change to initialRouteName).
14. Set Landing as the initial Route. That's just common sense, right?

Okay, fine we have set up the navigation and it is not erroring but it also isn't allowing us to navigate. Are you ready to make the navigation work? Let's go!

ShowingTimes to PickSeats

Remember our goal: when the user presses a showing in ShowingTimes we want the user to be sent to PickSeats for the correct showing.

15. Open ShowingTimes and add an onPress event to each <Text>:

```
onPress={() => pickShowingTime(showing)}
```

16. Then write the pickShowingTime method. It should do two things:

```
dispatch({type: "HIDE_FILM_DETAILS"});  // Hide the modal
navigation.push("PickSeats");           // Navigate to PickSeats
```

17. If you run at this point, it errors because ShowingTimes doesn't know what *navigation* is.
18. At the top of ShowingTimes, import the useNavigation hook from @react-navigation/native.
19. Call useNavigation() and store it in a const called navigation.
20. Now that navigation is defined, run your app. You should be able to tap/press a showing and navigate to PickSeats.

Reading values

Notice that when you navigate to PickSeats, the seat map never changes. You know why? Because we have hardcoded our list. But the showing id should change based upon the showing time chosen.

21. Prove that to yourself by console.log()ging the showing.id in PickSeats:

```
console.log(showing.id);
```

Ha! Did I trick you? There's no way that can work because showing has not been defined.

22. Discuss with your partner ... how the heck do we get data from one component to another when we have no JSX to put props in?
23. Here's the trick, pass an object from ShowingTimes:

```
navigation.push("PickSeats", { showing: showing });
```

24. Then read it in PickSeats:

```
import { useRoute } from 'react-navigation/native'; // <--Put this at the top
const route = useRoute();           // <--Put this inside the component
const showing = route.params.showing;
```

25. Now when you navigate to PickSeats, it see the object passed from ShowingTimes. Make sure you get the value you expected. We'll use that showing ID later to pull the actual showing from the server via a network request.

Navigating to CheckOut

Now let's make the check out button at the bottom of PickSeats navigate us to the CheckOut scene. Since you're now a pro at stack navigation, we'll give you fewer details.

26. 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. Add a button to the bottom titled "Look for another movie". It should navigate the user to the Landing page.
33. Go into App.js. Add this Ticket scene to your route configuration.
34. Edit Checkout.js. Make the button navigate to Ticket.
35. Run and test. You should now be able to get to any screen you like.

36. Bonus! In the Ticket component, tell the user what movie they're seeing, and the date and time of the showing. (Hint: the showing and selected_film can either be put into state or passed as route parameters. Your choice).

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