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:

```
npm install --save react-navigation react-navigation-stack
```

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-gesture-handler react-native-reanimated
react-native-screens react-native-safe-area-context
@react-native-community/masked-view
```

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.

```
import { createStackNavigator } from 'react-navigation-stack'
import { createAppContainer } from 'react-navigation'

const AppNavigator = createStackNavigator({
  Landing: { screen: Landing }
})

const AppContainer = createAppContainer(AppNavigator)
```

The `AppContainer` is a JSX you can render.

7. Render **only** this component, instead of the <View><Landing /></View> that you were doing before.
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.
9. 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. Remember how you accessed Redux state inside your App? It's time to move some of that logic to Landing.
10. 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.

11. The **second argument** to the `createStackNavigator` call is an options object, you can find docs here: https://reactnavigation.org/docs/4.x/stack-navigator/#stacknavigatorconfig

12. Try setting an initial route name, change the background color of the **header style**, and change the **header tint color** for the back/forward buttons. If you're wanting a simple color palette, here are some colors you can work with:

```
Colors
Primary light: rgb(232, 229, 153) Pale yellow
Secondary light: rgb(230, 255, 13) Bright yellow
Primary dark: rgb(11, 134, 232)  Darker blue
Secondary dark: rgb(17, 190, 255) Pale blue
Highlight: rgb(255, 0, 0) Red
```

13. 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.

You can provide component-level navigation options for screens inside the stack navigator. Here's the API:
https://reactnavigation.org/docs/4.x/stack-navigator/#navigationoptions-for-screens-inside-of-the-navigator

14. Edit Landing.js. Add a static property called navigationOptions, which is an object. Remember that "static properties" for function components are simply a matter of setting a new property on the exported function.

```
export const Landing = () => { /* ... */ }

Landing.navigationOptions = {/* ... */}
```

15. Add a navigation option to not show the header. (Hint: search for "headerShown" in the docs)
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. When the user choose a time, two things need to happen:

a. Navigate to the PickSeats screen
b. Somehow tell the PickSeats which showing was chosen.

Normally, we'd reach for our trusty tool Redux to store it as global state that can be accessed by PickSeats. In this case, we'll practice sending **navigation state params** instead.

Components that aren't "screens" (directly rendered in the Navigator) don't magically receive the navigation prop. So, you have a choice with how to allow a child component to navigate:

a. Wrap the component in `withNavigation` (https://reactnavigation.org/docs/4.x/with-navigation/), which injects the `navigation` prop into your component, or
b. Pass down a press handler

What you do is a matter of taste and style. I tend to dislike prop drilling, so I favor connecting with the HOC, but the downside is that it can make the components harder to test. The choice is yours!

17. Decide how you're going to access the navigation prop in ShowingTimes: either through withNavigation or pass it a onChooseTime prop that you drill down from Landing. You can give an `onPress` prop to a Text element. Once the time is pressed, you should tell it to navigate:

```
const handleChooseTime = () => {
  console.log('navigating...')
  navigation.navigate('PickSeats')
}
return (
  ...
  <Text onPress={handleChooseTime}>{time}</Text>
  ...
)
```

18. Run and test. When you do this, it won't go anywhere, because you haven't told react-navigation about this screen in the stack navigator.
19. The instructions are being deliberately less prescriptive to get you to think critically about how to achieve the high-level goals in your app. Be patient and keep trying!
20. When you tap a time, you should be navigated 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. (Docs: https://reactnavigation.org/docs/4.x/params) You pass params like this:

```
navigation.navigate('RouteName', { /* params go here */ })
```

and then read navigation params like this:

```
navigation.getParam(paramName, defaultValue)
```

21. Send along the selectedFilm, selectedDate, and showing with the navigation.
22. Edit PickSeats and get those props from the navigation state params instead of directly reading them as React props
23. Run and test. Now when you tap a time, the Landing scene should navigate you to the PickSeats scene.

24. Bonus! You may have noticed that we can't see that we've navigated 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 (just like you've done with PickSeats), passing params and adding it to the StackNavigator. Or 2) just dispatch { type: 'HIDE_FILM_DETAILS' } when the time is pressed.

Navigating to CheckOut

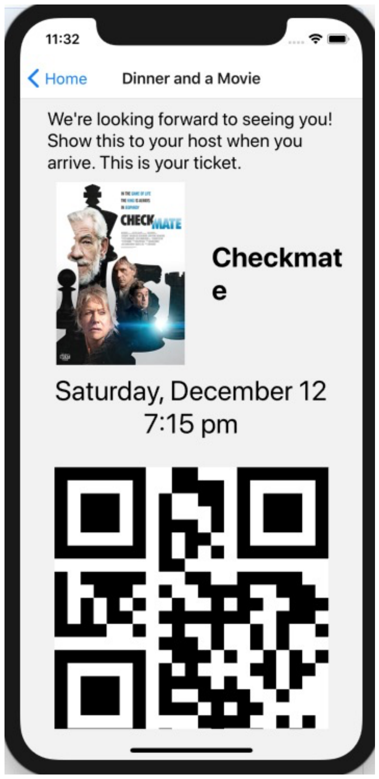
Now let's make the check out button at the bottom of PickSeats navigate us to the CheckOut scene.

25. Go ahead and edit PickSeats.js. Find the check out button and make it navigate to Checkout.
26. 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.

27. Create a new component called Ticket.js.
28. In it, simulate creating a ticketNumber. Randomly-generate a number between 50,000 and a million.
29. 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."
30. Then say "Ticket number: {ticketNumber}"
31. Go into Landing.js. Add this scene to your route configuration.
32. Edit Checkout.js. Make the button navigate to Ticket. (Note: the seats is currently hardcoded in Checkout as reading from `carts.json`. For now, just pass along the seats as a state variable.)
33. Run and test. You should now be able to get to any screen you like.
34. Now add a way for them to return to the Landing screen. Hint: use the StackActions popToTop functionality: https://reactnavigation.org/docs/4.x/stack-actions#poptotop. As a bonus challenge, put it as a header button that overrides the default "back" behavior of a stack navigator to jump all the way back to the Landing back. https://reactnavigation.org/docs/4.x/header-buttons#overriding-the-back-button Here's a hint: you'll use navigationOptions, and you can render the button like this:



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

Checkmat
e

Saturday, December 12
7:15 pm

```
<HeaderBackButton
  label="Home"
  onPress={() => {
    navigation.dispatch(StackActions.popToTop())
  }}
/>
```

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.)