Touchable Components Lab

We've been hearing a complaint from the users of our app. They say that they can reserve seats just fine but that they can't envision where the seats are in the theater. Some peole like to sit close to the screen. Others want to back way off. Some like to sit on the sides or in the back where they don't feel so cramped. Still others want to sit right in the center.

So they're asking for a seat map when they reserve their seats. Let's see if we can do that.

1. Open a browser or some other tool where you can make GET requests and make one to http://localhost:5000/api/theaters/:theaterId/tables/:table_id/seats with a valid theaterId and table_id. You'll notice that there is an x and a y location for each table. This number is what we'll use to place the seats.
2. Create a new component called SeatMap.js. Feel free to copy its contents directly from PickSeats.js since it is going to have the same basic functionality -- the user will be able to add seats to their cart.

We will need to change the JSX obviously.

3. Leave the headers at the top and the checkout button at the bottom. But replace all the stuff between them with a <View> that takes up 100% of the remaining space (hint: flex: 1).

Pinch-to-Zoom

Remember that <ScrollView> will allow us to pinch-zoom if we set it up just right, but it only works on iOS. So let's install a component that will work cross-platform.

4. Do this:

```
npm install react-native-pinch-zoom-view
```

Feel free to read up on this thing to make sure you understand how it works. But it will expose a new React Native component called <PinchZoomView>.

5. Go ahead and place one of those inside the <View> you just added.

Making it absolutely positioned

6. Put a <View> inside your <PinchZoomView>. Make this view have a style of position: 'absolute'.
7. Put some placeholder text and/or images inside of it to make sure you like how it works to start.
8. Move your placeholder text and images into different locations by setting the left and top properties of their container view.
9. Run and test several times, adjusting top and left until you get a feel for how to position things in the <View>.

Adding the tables back in

10. In the JSX of SeatMap, remove your placeholder text/images and do this instead:

```
tables.map(table => <Table />)
```

Note: in one of the prior labs there was a bonus step to extract <Table> and <Seat> components. If you didn't do that before, go ahead and do it now. They can just render a <Text> whose title is the table or seat number.

11. In addition to the props you were passing in before, make sure that the table's x and y position are being passed in as well. Remember, you'll need these tables to be positioned absolutely. (Hint: They may already be being passed in if you just sent the whole table object down somehow).
12. Use these x and y values to position your <Table>s. The x values are from the top and y values are from the left. (It's backwards from what you'd normally think, don't stress about it too much: top = table.x, left = table.y) You may need to do some math to spread them out across your device. Don't worry about getting them perfect just yet. We'll adjust later.
13. Run and test to make sure the <Table>s appear with some room between them.

Placing the seats

Remember that each table has one to four seats. Let's place those just like you did with the <Table>.

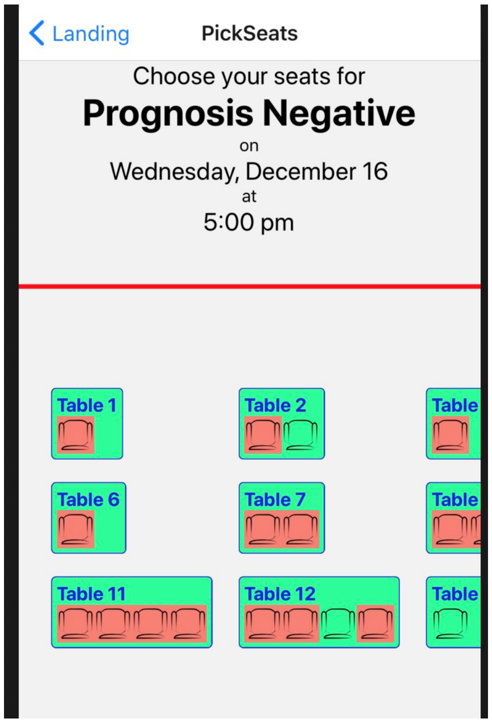
14. As before, if you don't already have a <Seat> component, create that now. It can render a <View> that has the seat number in it.
15. And again, using the x and y positions of the table, do a little math to position the seats next to the table. Don't worry about getting them perfectly placed just yet.
16. Run and test. Once you can see the right amount of seats next to each table, you can move on.

Making them look right

Let's make the seats look like seats.

17. Take a look in the project's starters. You'll find a file called seat.png. Copy this to your project's folder so it'll be compiled into the app.
18. Add it to the <Seat> as an <Image>. Resize it and place it as needed.
19. Now is where you can adjust the tables and the chairs. Change their sizes and locations to fit within your <PinchZoomView> control.
20. Right now you only pan when you're touching one of the tables in the <PinchZoomView>. That's because PinchZoomView is giving gesture handlers, but only to gesture events that happen on elements inside it. Let's take that first <View> that sits inside the <PinchZoomView> and give it a much bigger size, so it can catch all the gestures the user is making. (Hint: add a borderColor and borderWidth to the View to make it easier for debugging.)

As a rough sketch, they might lay out something like this:



21. Right now as you're panning around, the seats and end up on top of other things like the label. Let's go ahead and fix that. By default, all elements have a z-index of 0, and an element's later siblings will occlude it. That's why the Seats end up hovering over the label for the movie. Let's fix that!
22. Place the label top portion into a View with a z-index higher than 0. Then you'll need to give it a background color, otherwise it's transparent and it won't really cover up the map.
23. Do something similar for the Checkout button at the bottom.

Making it Touchable

We want the user to be able to press and choose multiple seats to reserve them.

24. Provide a way to track which seats the user has chosen. If they press a seat they already selected, it should deselect it. This time, try managing it using React state.
25. Wrap each seat in a TouchableHighlight. Put an onPress event on it to help manage your state.
26. Tapping on reserved seats should do nothing.
27. You should distinguish "reserved" seats from "chosen" seats
28. Once the user is finished and they press "Checkout", it should send along the seat choices to the Checkout screen.
29. Run and test in the debugger. Just to make sure that your touchable is indeed dispatching the right action and has a seat_number and showing object.

Pull out toggling logic

30. Let's pull out the business logic of toggling seat selections into a React custom hook. That will allow our business logic to remain separated from our view layer. (Also it's easier to test hooks!)
31. Create a hook like `useSeatManager` which encapsulates the logic, and returns a few pieces of data that the consumer needs to know about: `onPress` and `seatSelections`. Or something like that, you can be creative. The key part is that `useSeatManager` handles the press events and the logic to select/deselect a seat. How you communicate that knowledge into the React component tree is up to you.