Layouts with Flexbox Lab

We have a few cool components. And these components may work okay but they can use some help in styling and layout. Why don't we work on the layout for a while?

As you can see below, you've been given some simple wireframes of how our client's UX experts would like the views to lay out. Your job will be to use what you've learned about flexbox to make all of that happen. Once you've got them laid out in the code, try to look at them on a number of devices and don't forget to rotate your device to make sure they look good portrait and landscape.

Note: You're not responsible to make the font sizes or centering work just yet. We'll have a chance to do that later. Just concern yourself with layout at this point.

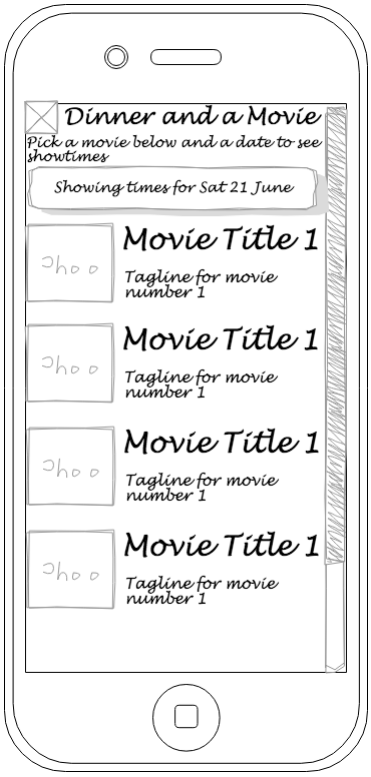
For starters, read through this guide on flexbox:

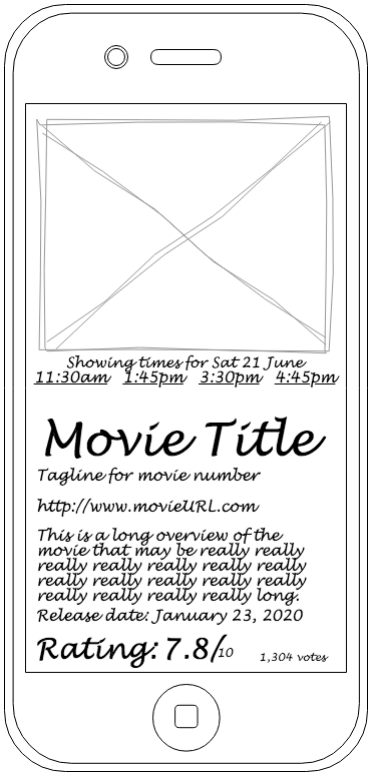
https://reactnative.dev/docs/flexbox?redirected

Once you're feeling ready to try your hand at flex, let's get started.

The Landing Component

1. Add a company logo to the upper-left. Use one of your own or use daam.png in the starters folder. If you need to size it, set the style's height and/or width property. But the size of images should be the only styling you'll do in this lab.
2. Place the logo and our company name side-by-side
3. Make the movie poster lay out to the left of the movie title and the tagline.





The Film Details Component

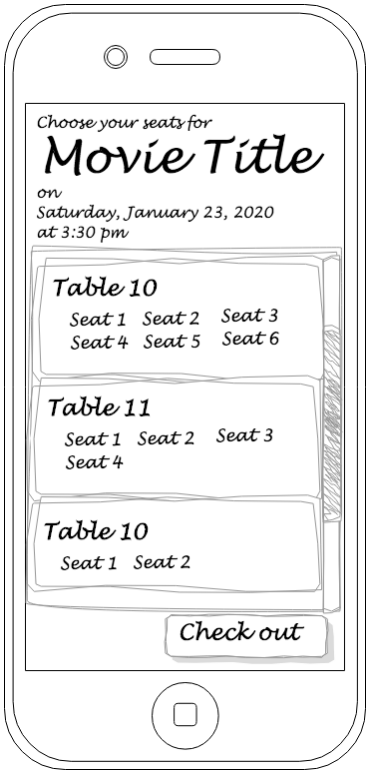
4. Make the movie poster centered and bigger
5. List your showing times side-by-side
6. Put the words "Release date" before the release_date.
7. Make a few additions:
   - Add a hardcoded "Rating" <Text> just before your vote_average
   - Add a hardcoded "/" after vote_average
   - Put a hardcoded "10" immediately after that
   - Then put the vote_count
   - Add a hardcoded "votes" after that
   - Then put all of those things on the same line

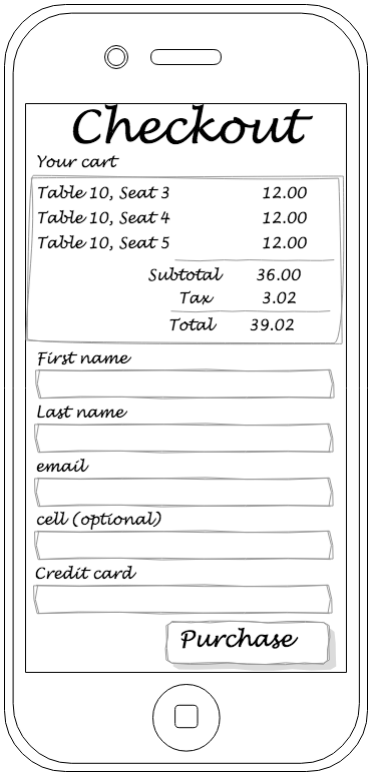
The Pick Seats Component

8. This is a new component. Go ahead and create it in PickSeats.js. It will not receive any props. Instead, it'll eventually get its values from the Redux store.
9. For testing in the meantime, copy starters/tables.json to the assets folder and go:
   `import tables from './assets/tables.json';`
   at the top of PickSeats.js. That JSON file has a list of tables and seats that you can .map() through.
10. Edit App.js. Render <PickSeats> instead of <Landing> for testing the layout here.

Pick Seats will eventually be shown when a user picks a movie, a date, and a showing. Then they will be able to reserve seats at a table. This scene lets them choose where they want to sit.

11. Make it look like the sketch to the right. ---->
12. Some notes:
    - Don't worry about making Movie Title big or about padding/margins/alignment for anything. We'll fix that later.
    - Each of the Tables will be in a View.
    - Each table has a table_number.
    - Each table has an array of seats. The tables will have a varying number of seats.
    - The table list will be in a <ScrollView>





The Checkout Component

13. For testing your layout, go ahead and make <Checkout> the startup component in App.js. Don't forget to make <Landing> the startup component when this lab is finished.
14. Open Checkout.js in your IDE.
15. First, add a <View> that will serve as your cart.
16. If you want some data that you can .map() through, look in the the starters/json folder for a file called cart.json. This is some hardcoded data that will work for now.
17. Note that you are going to want to calculate a subtotal, tax, and a grand total.
18. You're going to want to replicate the layout as best as you can but once again, don't worry about the styling, padding, margins, or alignment.

Got four laid-out components? Cool! You're done!

Bonus!! Getting data from the server

If you have time and know Redux, go ahead and implement the middleware and reducers to populate the PickSeats component. You should ...

19. Add these reducer cases:

```
case "SET_SHOWINGS":
  return { ...state, showings: action.showings };
case "SET_TABLES":
  return { ...state, tables: action.tables };
case "SET_RESERVATIONS":
  return { ...state, reservations: action.reservations };
```

20. Add a middleware function called fetchTablesAndSeatsMiddleware to get the data from the /api/theaters/<theaterId>/tables/ API. (Hint: The action should have the theaterId as its payload and the callback should dispatch({type: SET_TABLES, tables: tables});)
21. Add a middleware function called fetchShowingsForDateMiddleware to get the showings from the /api/showings/${filmId}/${selectedDate} API. (HInt: The action should hold the filmId and the selectedDate. The callback should dispatch({ type: "SET_SHOWINGS", showings }).
22. Lastly add a middleware function called fetchReservationsMiddleware to get the reservations from the /api/showings/${action.showingId}/reservations/ endpoint. (Hint: The action should hold the showingId. The callback should dispatch({ type: "SET_RESERVATIONS", reservations });)

At this point, you are ready to get your data live from the RESTful API server instead of a hardcode. We just need to let PickSeats know which showing the user wants. That'll happen later in the navigation chapter.