Styling Components Lab

Now that we know the better and worse ways of styling React Native components, let apply some of those to (finally) getting our app to look good. This lab will have fewer explicit instructions because we want you to express yourself in the look and feel of your own app.

And since it is a simpler topic, this would be a great time to erase any technical debt you might have built up. As you look through your components, take any poor designs or antipatterns and refactor them so that your code is stronger, more robust, more encapsulated, more abstract, or simply easier to understand.

Starting a theme

In this section we're going to create a theme object in its own file. This is a living file that will be included in nearly every other file to set the styles. As such, it will be constantly changing throughout the life of your project. Bottom line here: Don't fret about getting it perfect immediately. It is normal to put in simple placeholder values and tune them over time.

1. Create a new file called theme.js. In it, put an object that will become your color palette. Give these colors a name that describes the color itself, not how it'll be used. For example, maybe you want a white, a black, a red, a green, a blue.
2. Add to it a theme object. Make this the thing that is being exported from the file.
3. The theme object should have colors and text.
4. colors should be an object with semantically named colors like mainLight, mainDark, altLight, altDark and so forth. Use your imagination. These colors should be set to values defined in the palette above. Use colors like palette.white, palette.black and so forth.
5. text should be an object with different text styles like *normal* and *headline*. These styles are of course sub-objects with sizes, weights, and colors.
6. import the theme object at the top of the Landing scene.
7. Once you've got these started we can try out some individual styles and fold the ones you like into your new theme.

Working with fonts

Experiment with a few different fonts. We generally like to stick to the standard fonts throughout, but you may have an eye for design and want a different look.

8. Choose a <Text> element somewhere. Apply a style to it. Then edit that style and add this:

```
fontFamily: 'Papyrus',
```

9. Run and test on both platforms. It should work great on iOS but throw on Android.
10. Now change fontFamily to 'serif'. It should work great on Android but throw on iOS.
11. Make these fonts work on the same label, choosing the supported font for each platform.
12. Once you do, feel free to experiment with a few fonts until you find one you like.
13. Edit your theme.js file. Find your headline text style. Make it pick a font for iOS and a different font for Android. (Hint: You'll need to import Platform from react-native).
14. Apply that style to the "Headline", the <Text> that says "Dinner and a Movie" at the top.

Making a styled element

15. Notice that we have the movie title in several places in the app. We want that to be consistent, so ...
16. Create a new component called Title.js
17. It should do something like ...

```
export const Title = props =>
  <Text style={theme.text.movieTitle}>{props.children}</Text>
```

18. Add a theme.text.movieTitle style to your theme.js file.

This <Title> is a React Native component that can be used for a very consistent look. These things are called "styled components" because they carry their style along with them.

19. Now, go through each screen and change the movie titles to use <Title> instead of <Text>. Don't forget to import! (Hint: FilmBrief, FilmDetails, PickSeats)
20. Run and test. Adjust as needed.

Nesting <Texts>

21. Look at the ratings of the film at the bottom of the FilmDetails component. It says "X.Y/10". Let's say that our UX designers wanted the "X.Y/" portion to be large and the "10" portion to be small. If you haven't already done so, use Text nesting. Write it something like this:

```
<Text style={theme.text.rating}>
 X.Y/
 <Text style={theme.text.ratingSmallText}>10</Text>
</Text>
```

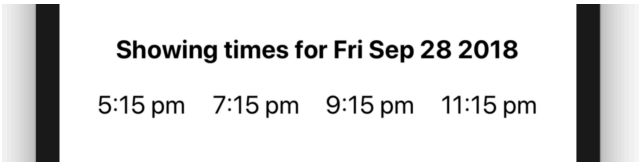
22. Then, of course, improve the styles in themes.js to satisfy our UX designers.

Scaling images

23. Make your movie posters resize nicely. Right now, you probably have them scaled, but parts of them may be cut off. Change the resizeMode property until you like how it looks better.
24. Adjust the sizes until you like their look

Showing Times

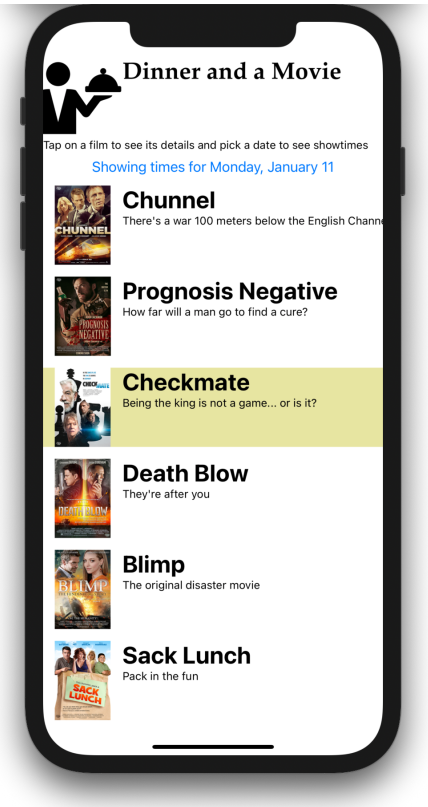
25. In ShowingTimes.js, make the headline bigger and bold. Do this as part of the theme.
26. Also make it centered. But layout like centering shouldn't be part of a theme. So put the centering as a style directly in ShowingTimes.js. (Hint: You'll have to combine them somehow. Think object spread).
27. Make each of the times large enough to be tapped easily. Something like this:



Perfecting the styles

Below you'll notice some screen shots. These can serve as inspiration to you. Your mission for each of them is to make your design as good or better. ('Better' should not be too much of a challenge). You'll also see that there are some specific instructions on a few of the scenes to improve the operation.

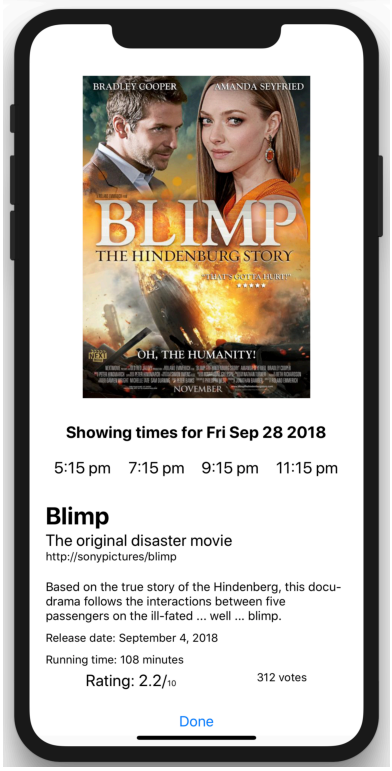
28. For each one, adjust styles as needed to make the scenes look more impressive!



Landing Scene

29. When the user chooses a film, change that film's style slightly to let the user know they've chosen it. You can change background color, increase the margins or put a border around it. Whatever looks good to you.

Film Details

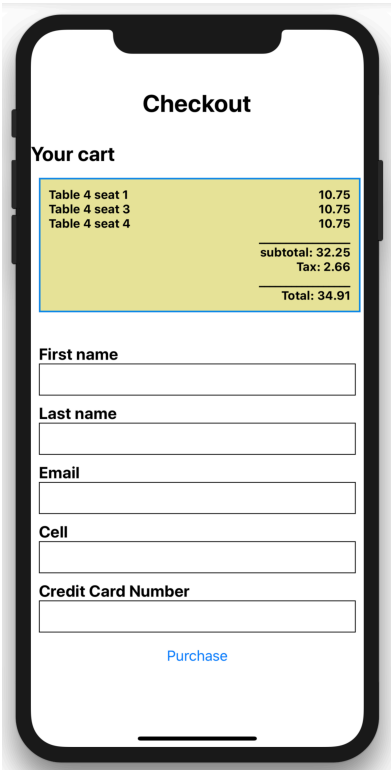


Pick Seats

We're going to use styling to let the user know the status of a seat.

30. Look at the data in your tables.json file Notice that some seats have a status of 'seatIsTaken', others are 'seatIsSelected', and other have no status at all.
31. Make the styling of a seat conditional. If the seatIsTaken, make the text light with a red background. If the seatIsSelected, make it light with an orange background. If the status is falsey, it is available. Make it bold and dark. Do all of this using your theme.

Checkout



32. Bonus!! If you get finished early, note that PickSeats is getting pretty heavy. Let's extract Table into its own component and Seat into its own.