

Layout Components Lab

By now you've surely seen that not all of your movies show on the landing scene. We're going to fix that first by using a `ScrollView`.

1. Open `Landing.js` and wrap your `<View>` in a `<ScrollView>`
2. Run and test. You can scroll now! Well, that was easy, wasn't it?

Now you might notice that some of the content is way too high on the screen, especially on an iPhone.

3. Add a `<SafeAreaView>` that wraps your `<ScrollView>`.
4. Run and test. If you're on Android, it works but just doesn't do much for us. But if you're on iOS, this should look great.

Notice that the status bar is covering the top of our app. Let's see what it looks like if we hide the status bar.

5. Open `App.js`. Add the `<StatusBar>` control. First change its `barStyle` property to `'dark-content'` and then `'light-content'`. See which you like better.
6. Then set the `hidden` property to `true`.
7. Run and test. The status bar should now be hidden. Note that you can still get to the status bar if you pull down from the top.

Selecting a film

When we have real customers on this app eventually, we'll show them the list of film briefs so they can select one by tapping it. Unfortunately, `<View>`s are not clickable. So let's wrap it in something that is.

8. In `FilmBrief.js`, surround your main `<View>` with a `<TouchableHighlight>`.
9. Add an `onPress` event to the `<TouchableHighlight>` and have it call `selectThisFilm`. In `selectThisFilm`, dispatch an action to the store. Something like this may work:
`store.dispatch({type: "SET_SELECTED_FILM", film: props.film});`
10. Of course do all the necessary work in Redux to make `state.selected_film` the one just tapped.
11. App's JSX should pass `selected_film={this.state.selected_film}` into `<Landing>` as a prop and `Landing.js` should pass `isSelected` (a boolean) down into `<FilmBrief>`.

Working with a Modal

`FilmBrief` was just a little bit of info about the movie. But when the user wants more details, let's show them those details in a Modal.

12. Inside `Landing.js`, add a `<Modal>` view. It should be just inside the top-level view (You probably wrote that as a `<SafeAreaView>` above).
13. Put a little `<Text>` in that Modal. Just something to see. And a `<Button>` titled "Done".
14. Add an `onPress` event to the Button. It should
`store.dispatch({type: "HIDE_FILM_DETAILS"});`
15. Open `FilmBrief` ... briefly. :) In the `onPress` event for the Touchable, also
`store.dispatch({type: "SHOW_FILM_DETAILS"}).`
16. Go back to `Landing.js`. Add a visible prop to the Modal. Set it equal to `props.showFilmDetails`.
17. In `App.js`, make sure you're passing `state.showFilmDetails` as a prop to `Landing.js`
18. Run and test. You should be seeing your modal and there is no way to dismiss it yet.
19. Edit `reducer.js`. Add a case for `HIDE_FILM_DETAILS` and another for `SHOW_FILM_DETAILS`. All they need to do is set `showFilmDetails` to `false` and to `true` respectively.
20. Run and test again. Now you can see the Modal to begin with but you can dismiss it with a click of the "Done" button.

21. Change the store's initial state to set showFilmDetails to false and test it one more time. Now it is hidden initially but shows/hides on button clicks.

Showing the film details

22. Create a new component called FilmDetails.js. It should receive a single film object as a prop.
23. Make this component show all of the details of the film. Put them in <Text>s inside a root <View>. Again, don't worry about layout or styling until later.
24. Let's tell the user when they can see the movie:

```
<View>
  <Text>Showing times for {props.selected_date.toDateString()}</Text>
  {props.showings.map(showing => <Text key={showing.id}>
    {showing.showing_time}
  </Text> )}
</View>
```

25. Finally nest this new <FilmDetails /> in the Modal instead of the placeholder text you added earlier.
26. Run and test by hardcoding some values. Or if you'd prefer, there's a file of showings in starters called showings.json you can use.
27. Bonus!! If you have extra time, put the details inside a ScrollView to make it layout just a bit better.
28. One last thing. It would be cleaner to have those showing times in their own component. Besides, we may want to reuse that component in multiple places. Go ahead and extract it into its own component called ShowingTimes.js which should receive showings and selected_date as props.

Use a keyboardHidingView

After the user has selected their movie and date and has selected their seats we'd like for them to actually pay us. That'd be nice, wouldn't it? So let's create a Checkout component.

29. Write a new class-based component called Checkout.js. Why class-based? Because we'll need state.
30. Add a <View> for the scene. Put that inside of a <ScrollView>.
31. Put a <Text> at the top to tell the user we're checking out.
32. Add <TextInput>s for first and last name, credit card, email, phone. Go ahead and give them <Text>s above each so the user knows what each is for.
33. Make the credit card <TextInput> show a number-pad keyboard.
34. Make the cell <TextInput> show a phone-pad keyboard.
35. Make the email <TextInput> show an email-address keyboard.
36. Add a <Button> titled "Purchase". It'll force you to add an onPress event. Make that run a function called this.purchase().
37. We'll learn how to properly navigate to Checkout later. But for now, just comment out <Landing> and add <Checkout> to App.js.
38. Run and test. Try to enter some text in each <TextInput>. Depending on the emulator you're using, the soft keyboard may slide up. If not, force it to come up through the emulator settings.

You may notice a problem. When the keyboard slides in the view it covers (or occults) some content behind it. This could be a problem someday. The solution is a KeyboardAvoidingView.

39. Edit Checkout.js. Add a <KeyboardAvoidingView> around the root <View> of the component.
40. Run and test again. Do you like this better? (Note: You may need to put a bunch of Lorem Ipsum text above your form to see the behavior. Try that if you don't see a difference).
41. Add a behavior property to the KeyboardAvoidingView. Switch between position, padding, and height to see the differences. Set it to the one you like best.
42. Bonus!! Add a <ScrollView> just inside the <KeyboardAvoidingView> so whether the keyboard is up or out, we can scroll through the content.
43. Before finishing, don't forget to uncomment <Landing> and comment out <Checkout> in App.js. Also delete all your Lorem Ipsum test. Then, you can be finished.