State and Subscriptions Lab

Right now we're reading our state object in App.js directly from a file. App.js **should** be getting its state from the SSOT - The Single Source of Truth - at all times. This is the store. Let's fix that first.

Initializing state

1. Remove the reading of the file from App.js
2. Change App.js to pass the person into the Register component:

```
<Register person={this.state.person} />
```

This doesn't work yet, but it will eventually because we are about to put a person object into state in the store. Ready? Here we go ...

3. Edit store.js. Add an import statement to read the user file into a const called user. This is just like the line you removed from App.js.
4. Change the initialState object to have a property called user. Set it to the user imported from the JSON file.
5. Change the data in the email and cell fields at the bottom. Do they change? (They should not). But are you seeing something in the console? (You should be).

Responding to data changes

Note that whenever any value in any field is changed, the changePerson method is called.
In it, we have JavaScript code capable of handling our defined fields. Your mission will be to notify the store that the email field and cell fields have changed. (You can ignore the other fields for now).

6. Find the method where the email field change is handled. Add a dispatch like so:

```
store.dispatch({type: "SET_EMAIL", email:e.target.value});
```

   As you know, this will tell redux we want to run the reducer.
7. Edit store.js and find your reducer function. Add a new condition for the action type "SET_EMAIL". If the action is "SET_EMAIL", return back a new object that looks just like the old one, but is using the action's email property.
8. Run and test. The UI should now change but only for the email field. and you should still have no errors but if you stop it in the debugger or console.log() the state that is coming out of that reducer function, you should see an object with the new email address. This will prove that the data is changing.
9. Do the same for the *cell* field.

Subscribing

Now that the data is changing, let's refresh the page based upon it.

10. Edit App.js. In the constructor, subscribe to a method called this.refresh.
11. Add a method called refresh:

```
refresh = () => {
  // You have to fill in this line here.
  this.setState(newState);
}
```

12. Run and test. You will know it is working when you can change the email or cell number values and watch the page change live.

13. Bonus! If you have time, implement a reducer case for the first name property.

14. Extra bonus! Feel free to remove the fake action we put at the top of changePerson whenever you are ready.