Composing Reducers Lab

If you think about it, our reducer could get fairly complicated if we just keep adding cases. Remember that we have objects within objects within objects. That complicates it. We can simplify this by creating sub-reducers.

Since we have some data (like cell and email for instance) immediately inside the main object, but other data in sub-objects and sub-sub-objects, the data is not "square". So we can't use Redux's built-in combine reducer. We'll have to combine reducers manually.

Let's do a little preparation for this lab.

1. Create an actionType for SET_NAME_TITLE in actionTypes.js.
2. Create an action creator for setNametitle(title) in actions.js.
3. Edit Register.js and find the case where we should be dispatching an action to set the title. In there, go ahead and dispatch an action to set the title.
4. Run and test. You should experience no errors, but nothing happens on the UI when you change the title.

Decomposing our existing reducer

5. Open reducers.js. Write a subreducer for the name property. Call it nameReducer. It'll have the same shape as the parent reducer: (oldState, action) => newState.
6. Pull any name logic out of the root reducer and put it in the new *name* sub-reducer. For example, you probably have user.name.first and user.name.last in the root reducer. Move those.

Don't forget that the input state of this sub-reducer will be <u>just</u> the name portion of state and not the entire state itself. As far as nameReducer is concerned, state is nothing more than the name object; it has a *first*, a *last*, and a *title* property. So the good news here is that your reducer is super-simple! It only has three cases and only deals with one level, not sub-objects and sub-sub-objects and so on. As an example, your first name case would look like this:

```
  case actionTypes.SET_NAME_FIRST:
    return { ...state, first: action.first};
```

Simpler, right?

7. With that in mind, adjust your first and last cases.
8. Add a case to handle a CHANGE_NAME_TITLE action.
9. Run and test. Still nothing changes yet but there should be no errors.

Combining the root reducer and the name reducer

You're going to want to combine these reducers, while retaining the basic shape of (oldState,action) => newState. Feel free to accomplish this any way you prefer, but here is an example using object spreading:

```
const reducer = (state, action) => ({
  ...state, user: {...userReducer(state.user, action)}
});
```

The combined reducer is a function that receives state and action and returns state

The userReducer is given only the slice of state regarding the user object

This new reducer -- the combined reducer -- is what will be passed into the createStore method. But what is userReducer?

```
const userReducer = (state, action) => ({
  ...rootUserReducer(state, action),
  name: nameReducer(state.name, action),
});
```

The idea is this: Break down state into smaller and smaller slices with each slice having its own reducer. Once you go through the pain of combining them, each reducer is very very simple!

10. Add something like the above to your reducers.js file. Don't forget to export reducer because it's already imported it into store.js.
11. One more thing. If you decide to follow the code above, your old *reducer* will now have to be called *rootUserReducer*. And since it is now receiving only the slice of the state dealing with a user, you'll need to "flatten" the objects. (Hint: add a console.log(state) to it and examine the state here. You'll see that it doesn't have a user, it is a user.) Fix it according to that.
12. Run and test. You'll know it works when changing first and last begin to work again and that title can now be changed also. Keep adjusting and debugging until it is working.

Implementing the location

13. Now that you've had some experience combining reducers and setting up sub reducers, do that for the location.
14. Create a locationReducer in reducers.js. Remember, state to this function should be only the location slice of the user.
15. Combine that reducer with the others. (Hint: it will be part of the userReducer).
16. Run and test. At this point your entire Registration form should work.