# Refactoring to Set Initial State from the Reducer

In Redux, the initial state of an app should be returned to us by the reducer. Let's see how we can apply this rule on Skypey to return 'contacts' in the initial state.

Firstly, please have a look at the creation of the store in store/index.js. In particular, consider this line of code:

```
const store = createStore(reducer, { contacts });
```

**The initial state object is passed directly into createStore**. Remember that the store is created with the signature, `createStore(reducer, initialState)`. In this case, the initial state has been set to the object, {contacts: contacts}.

Even though this approach works, this is typically used for server side rendering (don't bother if you don't know what this means). For now, understand that this approach of setting an initial state in createStore is more used in the real world for server side rendering.

Right now, remove the initial state in the createStore method.

We'll have the initial state of the application set solely by the reducer. Trust me, you'll get the hang of this.

Here's what the **store/index.js** file will look like once you remove the initial state from createStore:

```
import { createStore } from "redux";
import reducer from "../reducers";

const store = createStore(reducer);

export default store;
```

store/index.js

And here's the current content of the **reducer/index.js** file:

```
export default (state, action) => {
  return state;
};
```

Please change that to this:

```
import { contacts } from "../static-data";

export default (state = { contacts }, action) => {
  return state;
};
```

So, what's happening here?

Using ES6 default parameters, we have set the state parameter to an initial value of **{contacts}**.

This is essentially the same as **{contacts: contacts}**.

Hence, the **return state** statement within the reducer will return this value, {contacts: contacts} as the initial state of the application.

At this point, the app now works - just like before. The only difference here is that the initial state of the application is now managed by the Reducer.

Is it possible to manage the complete state with one reducer? Let's keep refactoring.