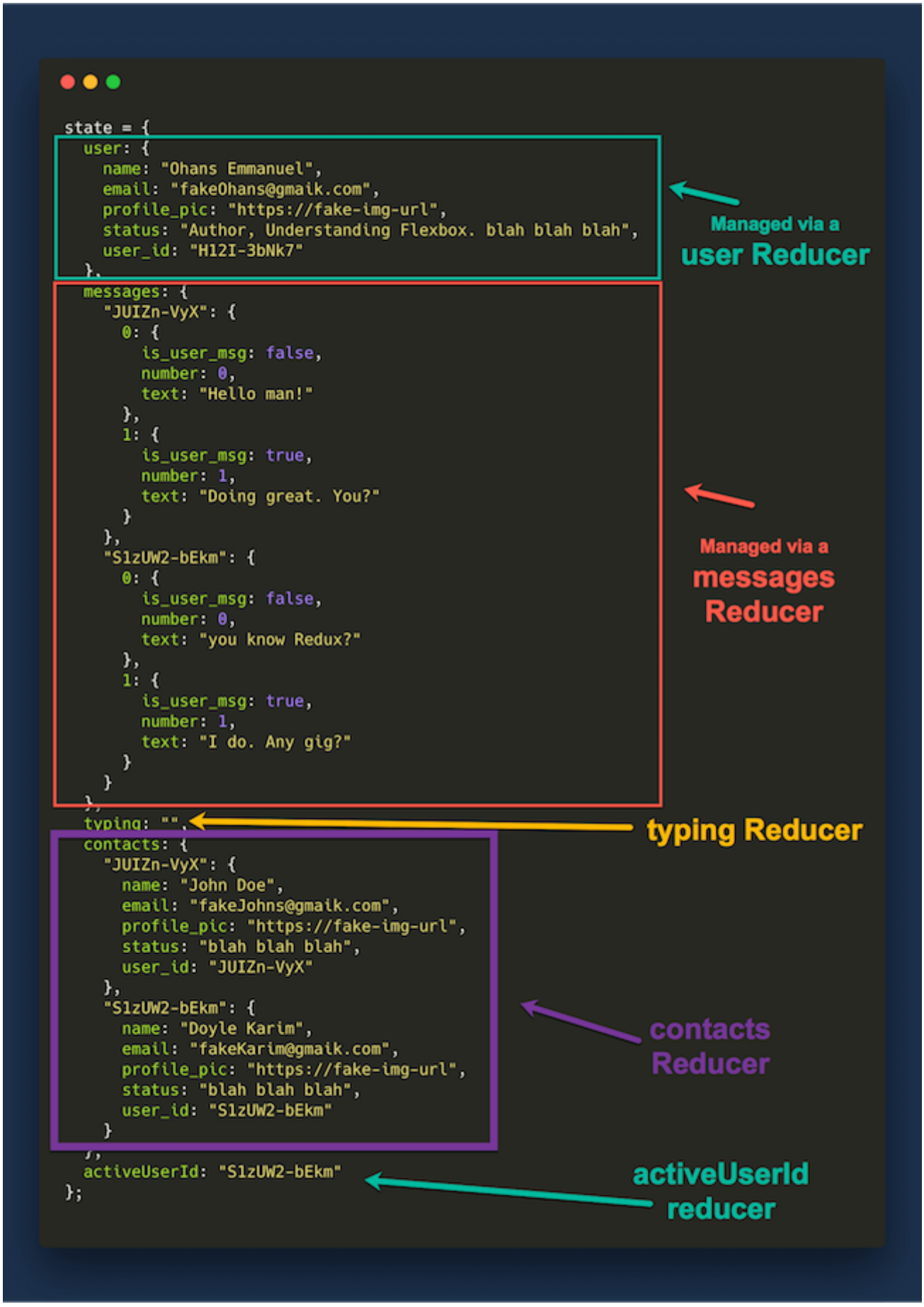
Refactoring Skypey to Use Multiple Reducers

Here, we'll implement the use of multiple reducers by making separate files for each field e.g. contacts, user etc. and then merging them into one combined reducer.

> In the last lesson, we decided to make use of multiple reducers to handle all the fields in the state object:



Now, for every field in the state object, we will create a corresponding reducer. The current ones at this stage are, contacts and user.

Let’s go over how this affects our code first, then I’ll take a step back to explain

how it works again.

Take a look at reducer/index.js:

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

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

reducer/index.js

Rename this file to **contacts.js**

This will become the **contacts reducer**. Create a user.js file within the reducers directory.

This will be the user reducer.

Here's the content:

```
import { generateUser } from "../static-data";
export default function user(state = generateUser(), action) {
  return state;
}
```

Again, I have created a **generateUser** function to generate some static user information.

Using ES6 default parameters, the initial state is set to the result of invoking this function.

Therefore return state will now return a user object.

Right now, we have 2 different reducers. Let's combine them for the greater good :)

- Create an index.js file within the reducers directory.

Firstly, import the two reducers, user and contacts.

```
import user from "./user";
import contacts from "./contacts";
```

To combine these reducers, we need the helper function `combineReducers` from redux.

Import it like this:

```
import { combineReducers } from "redux";
```

Now, index.js will export the combination of both reducers like this:

```
export default combineReducers({
     user,
     contacts,
});
```

Notice that the **combineReducers** function takes in an object. An object whose shape is exactly like the state object of the application.

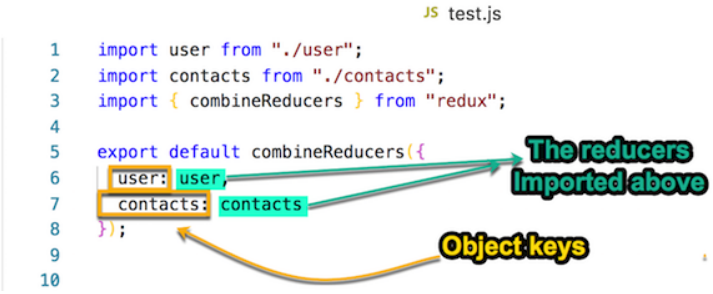
The code block is the same as this:

```
export default combineReducers({
     user: user,
     contacts: contacts
})
```

The object has keys user and contacts, just like the state object we've got in mind.

What about the values of these keys?

The values come from the reducers!



It is very important to understand this, but it is quite alright if you feel a little lost.

In the next lesson we'll re-examine how reducers work so that we can put this all into perspective.