Reducer Composition Revisited

In this lesson, we will refactor our reducers into a more generic format. This will help us combine them in the combinedReducer (Implementation below).

Let me take a step back and explain how reducer composition works again. This time, from a different perspective.

Consider the javascript object below:

```
const state = {
  user: "me",
  messages: "hello",
  contacts: ["no one", "khalid"],
  activeUserId: 1234
}
```

Now, assume that instead of having the values of the keys hardcoded, we wanted it to be represented by function calls. That may look like this:

```
const state = {
  user:  getUser(),
  messages: getMsg(),
  contacts: getContacts(),
  activeUserId: getID()
}
```

This assumes that `getUser()` will also return the previous value, “**me**”. The same goes for the other replaced functions.

Still following?

Now, let’s rename these functions.

```
const state = {
  user:  user(),
  messages: messages(),
  contacts: contacts(),
  activeUserId: activeUserId()
}
```

Now, the functions have names identical to their corresponding object keys. Instead of `getUser()`, we now have `user()`.

Let’s get imaginative.

Imagine that there existed a certain utility function imported from some library.

Let’s call this function, **killerFunction**.

Now, killerFunction makes it possible to do this:

```
const state = killerFunction({
  user: user,
  messages: messages,
  contacts: contacts,
  activeUserId: activeUserId
})
```

What has changed?

Instead of invoking each of the functions, you just write the function names. killerFunction will take care of invoking the functions.

Now using ES6, we can simplify the code further:

```
const state = killerFunction({
  user,
  messages,
  contacts,
  activeUserId
})
```

This is the same as the previous code block. Assuming the functions are in scope, and have the same name (identifier) as the object key.

Got that?

Now, this is kind of how combineReducer from Redux works.

The values of every key in your state object will be gotten from the reducer. Do not forget that a reducer is just a function.

Just like killerFunction, combineReducers is capable of making sure the values are gotten from invoking the passed functions.
All the key and values put together will then result in the state object of the application.

That is it!

An important point to ALWAYS remember is that when using combineReducers, **the value returned from each reducer is NOT the state of the application**.

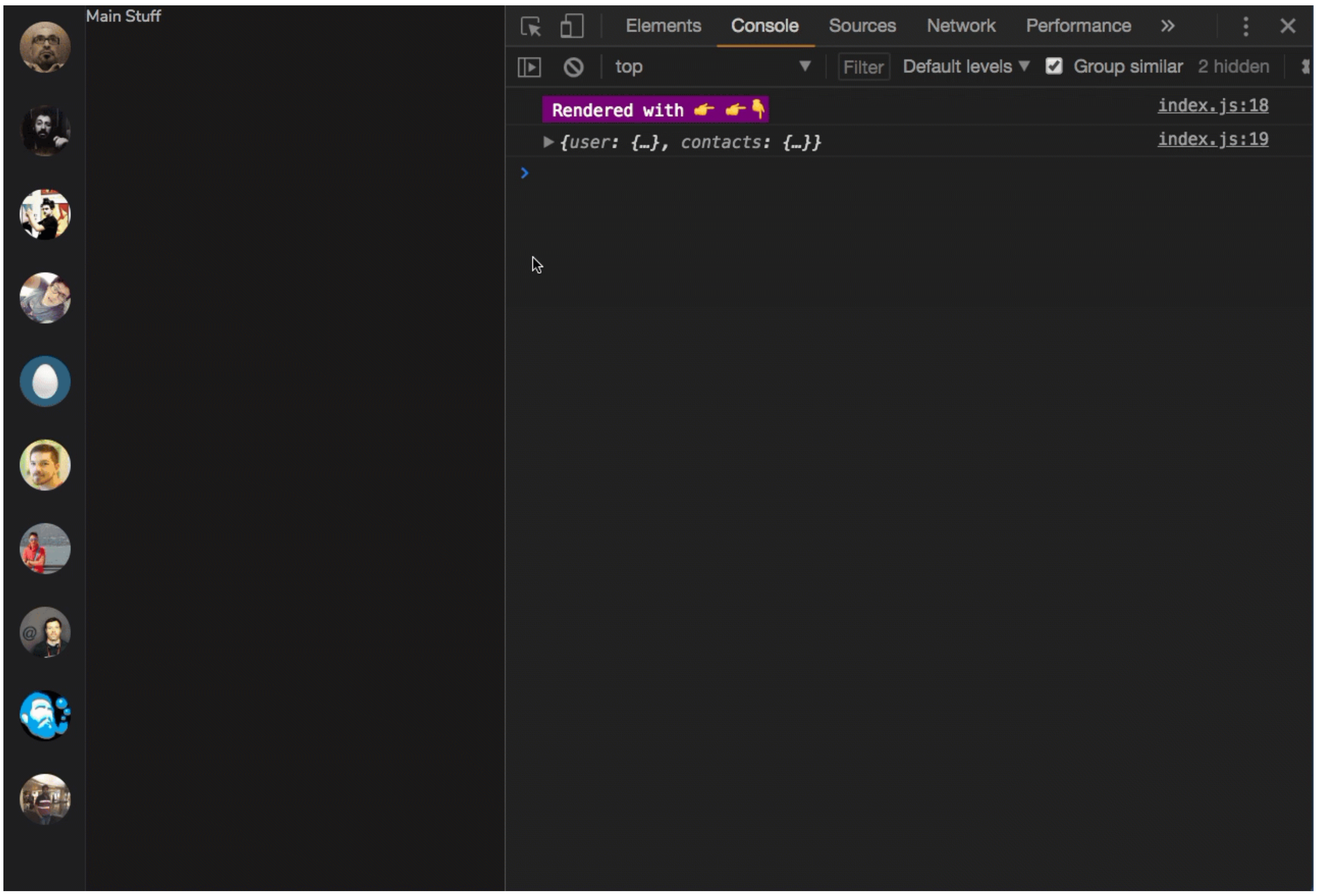
It is only the VALUE of the particular key they represent in the state object!

For example, the user reducer returns the value for the user key in the state. Likewise, the messages reducer returns the value for the messages key in the state. e.t.c.

Now, here's the complete content of reducers/index.js

```
import { combineReducers } from "redux";
import user from "./user";
import contacts from "./contacts";
export default combineReducers({
  user,
  contacts
});
```

Now if you inspect the logs, you'll find user and contacts right there in the state object.



```
Rendered with 👉 👉👇                                    index.js:18
▶{user: {…}, contacts: {…}}                              index.js:19
```

Now that the **User** and **Contacts** functionalities are set in place, we'll move on to the **Main** component of the App.