Exercise Solution

Our objective was to delete a contact. Let's see how we can delete a contact.

If you figured it out on your own, you should be impressed with your grasp on Redux.

Nevertheless, it's good to look at different ways to approach the problem. This is how I did it.

The first step in doing something new is always creating a new action. In the `constants/action-types.js`, we'll add our "DELETE_CONTACT" string. Again, this optional, but a safe practice:

```
export const SET_ACTIVE_USER_ID = "SET_ACTIVE_USER_ID";
export const DELETE_CONTACT = "DELETE_CONTACT";
```

action-types.js

With that out of the way, we'll create the action in `actions/index.js`. This is very simple, as our payload consists of just the user ID:

```
import { SET_ACTIVE_USER_ID, DELETE_CONTACT } from "../constants/action-types";

export const setActiveUserId = id => ({
  type: SET_ACTIVE_USER_ID,
  payload: id
});

// the Delete Chat Action
export const deleteContact = (user_id) => ({
        type: DELETE_CONTACT,
        payload: user_id
})
```

Our `deleteContact` method creates an action of type `DELETE_CONTACT` for every `user_id`.

As you already know, the action is being dispatched from `User.js`. We also

had to set the activeUserId to null. Let's get done with this:

```
function handleCrossClick(e, user_id) {
  e.stopPropagation();
  store.dispatch(setActiveUserId(null));
  store.dispatch(deleteContact(user_id));
}
```

User.js

Two actions are being dispatched in the code above. But will this do anything? If you said no, you guessed right. We need to define the reducer for this task.

In the `reducers` directory, where do you think this action should be handled? Take a good look. We are dealing with contacts, and so the `contacts.js` reducer would be most appropriate for the job.

Here is the code for our reducer:

```
import { contacts as userContacts } from "../static-data";
import { DELETE_CONTACT } from "../constants/action-types";

export default function contacts(state = userContacts, action) {
  switch(action.type){
        case DELETE_CONTACT:

      const contactId = action.payload;
      const { [contactId]: deleted, ...newState } = state
      return newState;

    default:
      return state;
  }
}
```

reducers/contacts.js

Nothing too fancy here. The important part was to avoid mutating the state of the app. This is handled by the lines

```
const contactId = action.payload;
const { [contactId]: deleted, ...newState } = state
return newState;
```

A new state `newState` is returned, and it excludes the contact we wanted to delete.

And that's it for this one. Go to the console on check how the activeUserId is set to null after every contact deletion. This will prove useful as we'll be able to go back to the empty screen if we deleted the chat that was rendered.

Play around with this solution if you want. I'll see you in the next section.