Dispatch Recognition

A dispatched action must be recognized by the reducer so that it can perform further actions. For our setActiveUserID, let's use the reducer to return the payload from the setUserID action i. e. the user id.

Previously, we dispatched the action to the `activeUserID` reducer, but the reducer had not been told what to do with it.

Let’s fix this, but don’t forget to remove the `console.log(user_id)` after inspecting the logs.

Have a look at the `activeUserId` reducer:

```
export default function activeUserId(state = null, action) {
  return state;
}
```

Right now, we are ignoring every action that passes through this reducer. We return null despite the dispatch of any actions. To handle this, we’ll write a switch statement as shown below.

reducer/activeUserId.js:

```
import { SET_ACTIVE_USER_ID } from "../constants/action-types";
export default function activeUserId(state = null, action) {
  switch (action.type) {
    case SET_ACTIVE_USER_ID:
      return action.payload;
    default:
      return state;
  }
}
```

reducer/activeUserId.js:

You should understand what’s going on here.

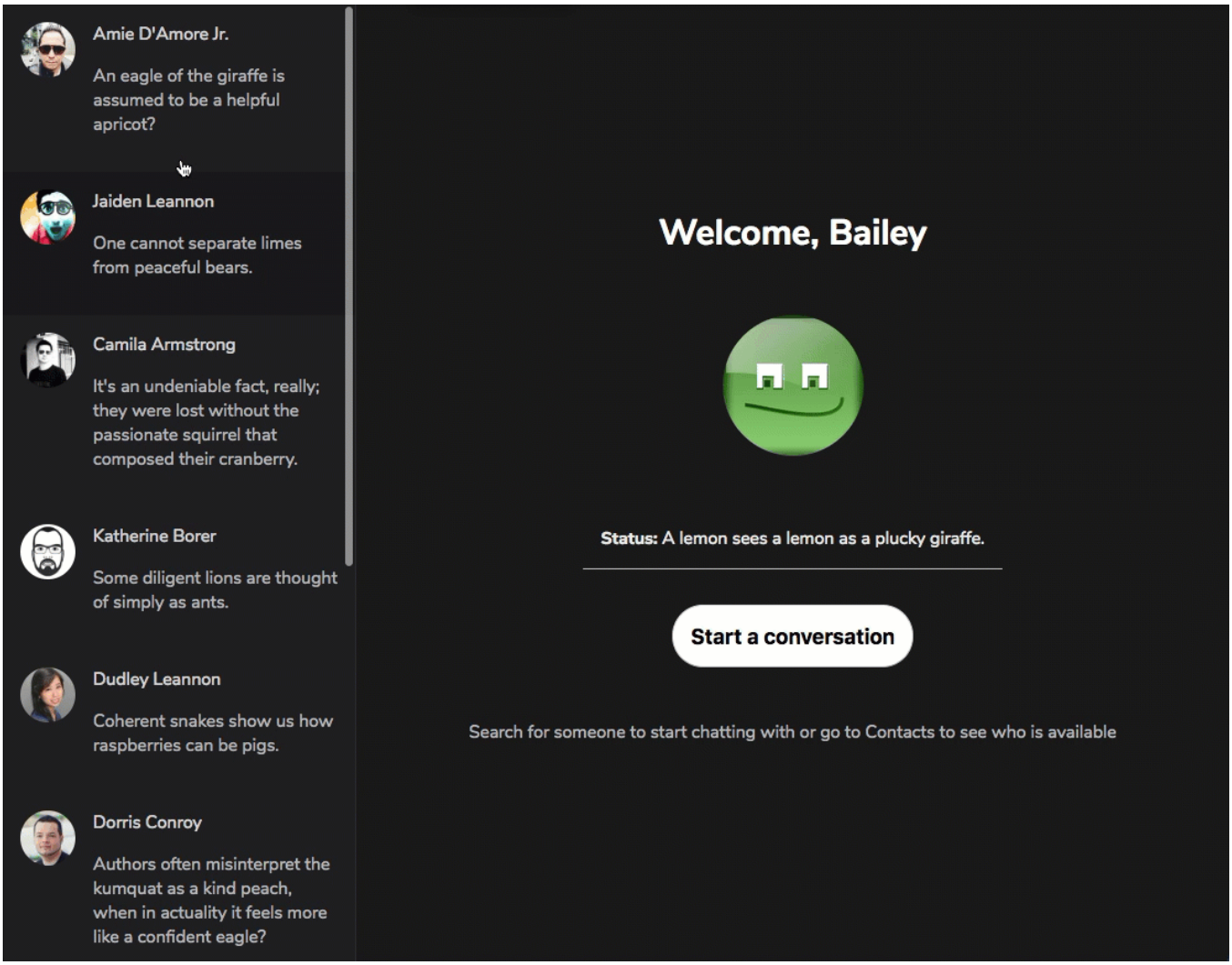
The first line imports the string, `SET_ACTIVE_USER_ID` We then check if the action passed in is of type `SET_ACTIVE_USER_ID` . If yes, then the new value of

activeUserId is set to action.payload

Don't forget that the action payload contains the user_id of the user contact.

Let's see this in action. Does it work as expected?

Yes!



Now, the ChatWindow component is rendered with the right `activeUserId` set.

As a reminder, it is important to remember that **with reducer composition, the returned value of each reducer is the value of the state field they represent, and NOT the entire state object**.

With the user ID returned to us, it's time to deconstruct our chat window into its components so that we can work on them one by one using this ID.