Javascript Implementation For Chats

This lesson gives us the full implementation of ChatWindow.js and Chats.js. We will delve into the full explanation for both in the next lesson. The primary observation is that we will access the store to get the state.messages and manipulate them for the current active user.

If you haven't already, create the files, Chats.js and Chats.css in the components directory.

Now, import Chats and render it below the <Header /> component in ChatWindow.

containers/ChatWindow.js:

```
...
import Chats from "../components/Chats";
...
return (
  <div className="ChatWindow">
    <Header user={activeUser} />
    <Chats />
  </div>
);
```

containers/ChatWindow.js

The <Chats/> component will take the list of messages from the state object, map over these, and then render them beautifully.

Remember that the messages passed into Chats are specifically the messages for the active user!

Whereas **state.messages** holds all the messages for every user contact, **state.messages[activeUserId]** will fetch the messages for the active user.

This is why every conversation is mapped to the user id of the user - for easy retrieval as we have done.

Grab the active user's messages and pass them as props in Chats.

containers/ChatWindow.js:

```
...
import Chats from "../components/Chats";
...
const activeMsgs = state.messages[activeUserId];
return (
  <div className="ChatWindow">
    <Header user={activeUser} />
    <Chats messages={activeMsgs} />
  </div>
);
```

Now, remember that the messages of each user is a giant object with each message having a number field:

For easier iteration and rendering, we'll convert this to an array. Just like we did with the list of users in the Sidebar.

For that, we'll need Lodash.

containers/ChatWindow.js:

```
...
import _ from "lodash";
import Chats from "../components/Chats";
...
const activeMsgs = state.messages[activeUserId];
return (
  <div className="ChatWindow">
    <Header user={activeUser} />
    <Chats messages={_.values(activeMsgs)} />
  </div>
);
```

containers/ChatWindow.js

Now, instead of passing **activeMsgs**, we pass in **_.values(activeMsgs)**

There's one more important step before we view the results.

The component, Chats has not been created.

In Chats.js, write the following

components/Chats.js

```
import React, { Component } from "react";
import "./Chats.css";
const Chat = ({ message }) => {
  const { text, is_user_msg } = message;
  return (
    <span className={`Chat ${is_user_msg ? "is-user-msg" : ""}`}
      >{text}</span>
  );
};
class Chats extends Component {
  render() {
    return (
      <div className="Chats">
        {this.props.messages.map(message => (
          <Chat message={message} key={message.number} />
        ))} </div>
    );
  }
}
export default Chats;
```

containers/Chat.js

Take some time closely examine the implementation above. It isn't too much

to comprehend, but I'll explain what's going on in the next lesson.