Chats Implementation Explained

Chats.js is used to handle each message itself. The text and is_user_msg is obtained as these are the two things we need to know for displaying each message.

In the last lesson we wrote this implementation for Chats.js

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

Let's go through this, step by step.

Firstly, have a look at the the Chats component. You'll notice that I have used a class based component here. You'll see why later on.

In the render function, we map over the messages props and for each message , we return a Chat component.

The Chat component is super simple:

```
const Chat = ({ message }) => {
  const { text, is_user_msg } = message;
  return (
    <span className={`Chat ${is_user_msg ? "is-user-msg" : ""}`}
      >{text}</span>
  );
};
```

For each message that's passed in, the text content of the message and the is_user_msg flag are both grabbed using the ES6 destructuring syntax,

```
const { text, is_user_msg } = message;
```

The return statement is more interesting.

A simple span tag is rendered.

Strip out some of the JSX magic, and here's the simple form of what is rendered:

```
<span> {text} </span>
```

The text content of the message is wrapped in a span element. Simple.

However, we need to differentiate between the application user's message, and the contact's message.

Don't forget that a conversation happens with at least 2 people sending messages back and forth.

If the message being rendered is the user's message, we want the rendered markup to be this:

```
<span className="Chat  is-user-msg"> {text} </span>
```

And if not, we want this:

```
<span className="Chat"> {text} </span>
```

Note that what's changed is the is-user-msg class being toggled.

This way we can specifically style the user's message using the css selector shown below:

```
.Chat.is-user-msg {}
```

So, this is why we have some fancy JSX for rendering the class names based on the presence or absence of the is_user_msg flag.

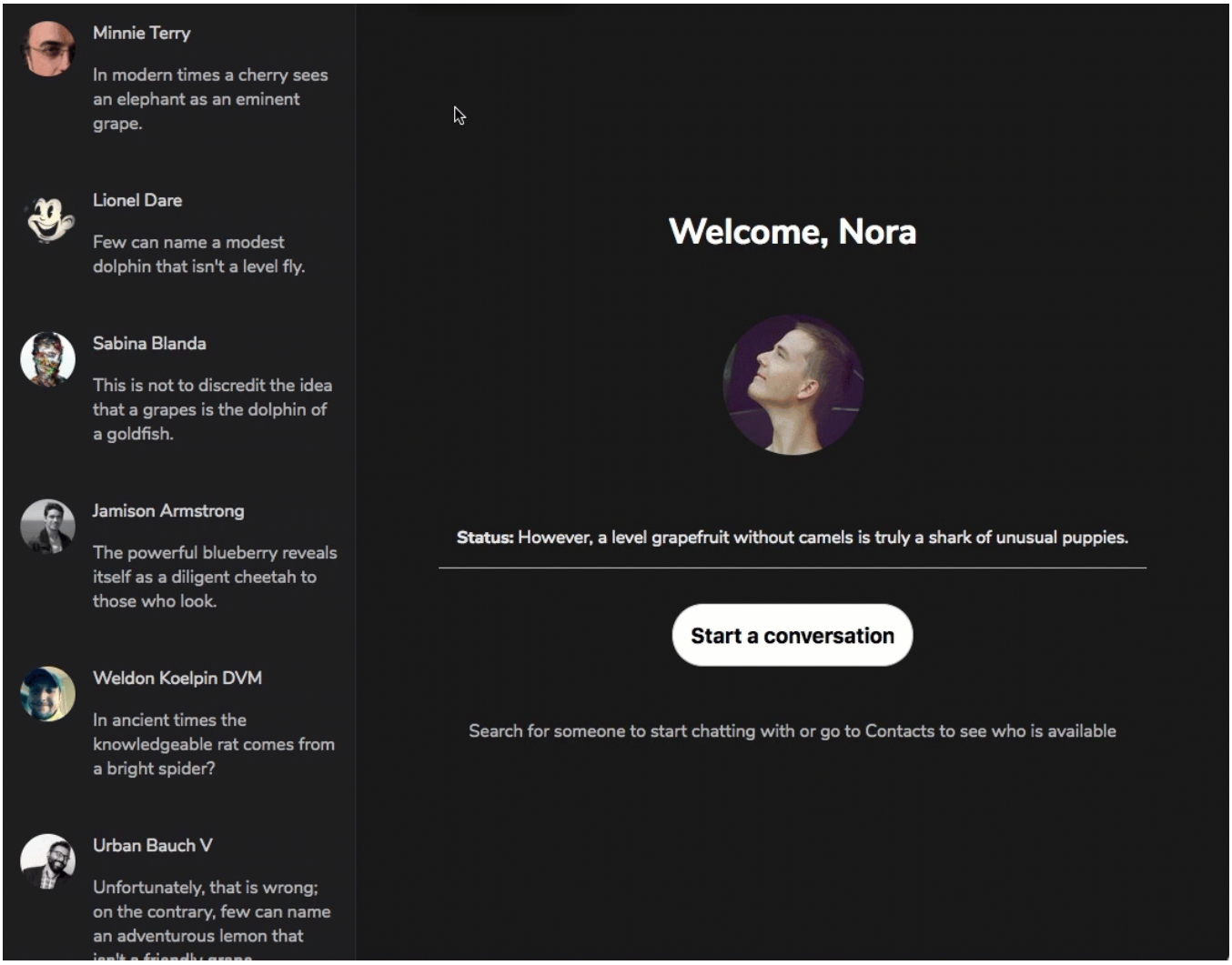
```
<span className={`Chat ${is_user_msg ? "is-user-msg" : ""}`}>{text}</
```

span>

The real sauce is this:

```
${is_user_msg ? "is-user-msg" : “”
```

That’s the ternary operator right there! Now, you can make sense of all the code within containers/Chats.js now, huh? Here’s the result so far.



The current result of our iterations. Still needs some work.

The messages are rendered but it doesn’t look so good.

This is because all the messages are rendered in span tags.

Since span tags are inline elements, all the messages just render in a continuous line - looking squashed.

This is where CSS shines.

We'll style the chat threads in the following lesson.