The Messages Field

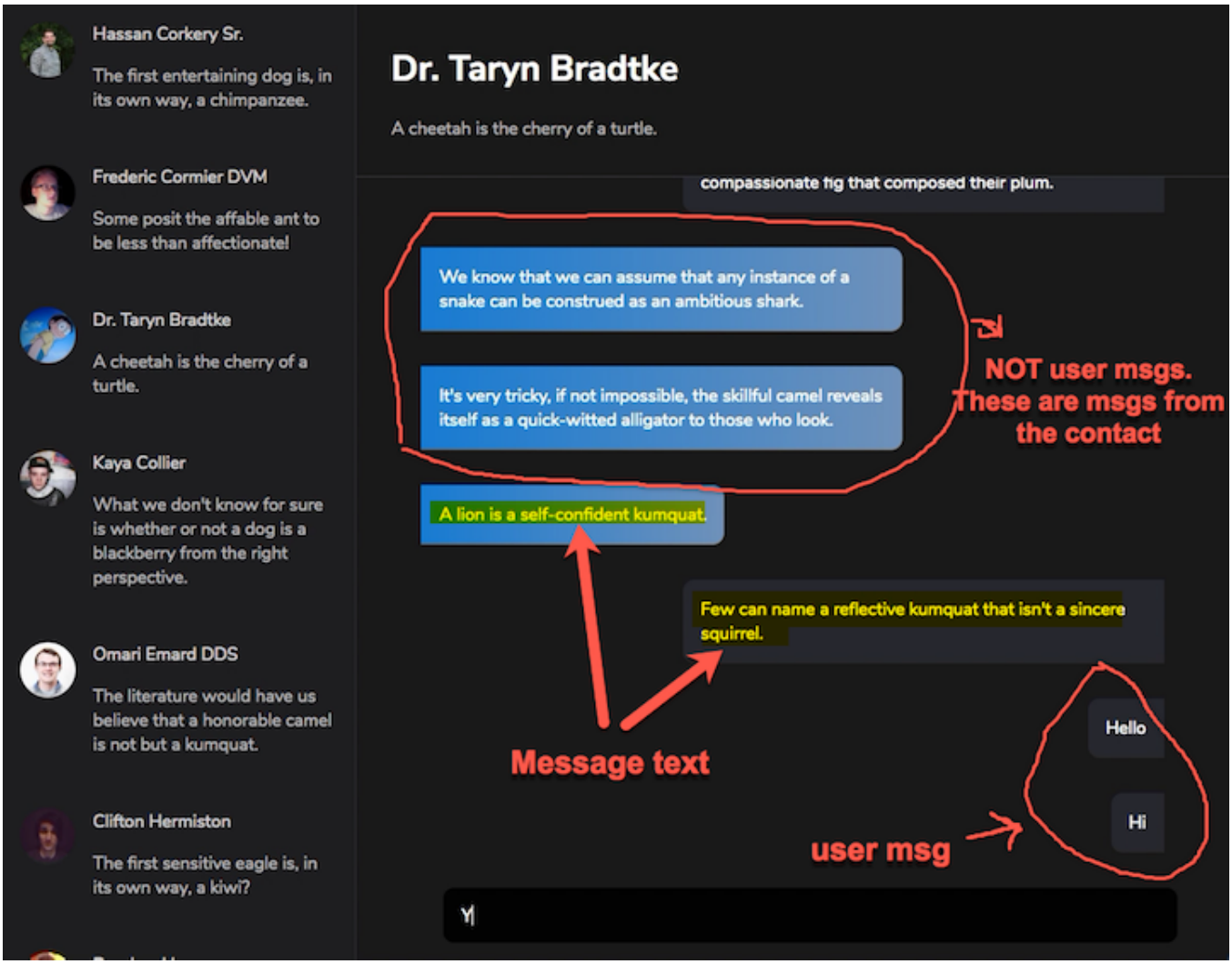
Now, we will restructure our Messages component to use objects in of arrays.

We currently have the messages as an array with message objects.

```
messages: [
  {
    messageTo: 'contact1',
    text: "Hello"
  },
  {
    messageTo: 'contact2',
    text: "Hey!"
  }
]
```

We will now define a more appropriate shape for the message objects. A message object will be represented by the message object below:

```
{
    text,
    is_user_msg
};
```



The text is the displayed text within the chat bubble. However, **is_user_msg** will be a Boolean - true or false. This is important to differentiate if a message is from a contact or the default app user.

Looking at the graphic above, you'll notice that the user's messages and those of a contact are styled differently in the chat window. The user's messages stay on the right, and the contact, on the left. One is blue, the other is dark.

You now see why the boolean, is_user_msg is important. We need it to render the messages appropriately.

For example, the message object may look like this:

```
{
  text: "Hello there. U good?",
  is_user_msg: false
}
```

Now, representing the messages field within the state with an object, we

should have something like this:

```
messages: {
  user_id:
  {
    text,
      is_user_msg
  },
  user_id_2:
  {
    text,
      is_user_msg
  }
}
```

Notice how I'm also using an object instead of an array again. Also, we're going to map each message to the unique key, **user_id** of the contact.

This is because a user can have different conversations with different contacts, and it is important to show this representation within the state object. For example, when a contact is clicked, we need to know which was clicked! How do we do this?

Yes, with their user_id.

The representation above is incomplete but we've made a whole lot of progress! The messages field we've represented here assumes that each contact (represented by their unique user id) has only one message.

But, that's not always the case. A user can have many messages sent back and forth within a conversation.

So how do we do this?

The easiest way is to have an array of messages, but instead, I'll represent this with objects.

```
messages: {
     user_id: {
        0: { text,
           is_user_msg
        },
        1: { text,
          is_user_msg
        }
     },
     user_id_2: {
        0: { text,
```

```
          is_user_msg
        }


    }
}
```

Now, we are taking into consideration whatever amount of messages are sent within a conversation. One message, two messages or more, they are now represented in the messages representation above.

You may be wondering why I have used numbers, 0, 1 etc. to create a mapping for each contact message.

I explain that next.

For what it's worth, the process of removing nested entities from your state object and designing it like we've done here is called **Normalizing the State Object**. I don't want you confused incase you see that somewhere else.

But is using objects instead of arrays really a good idea? Let's discuss that in the next lesson.