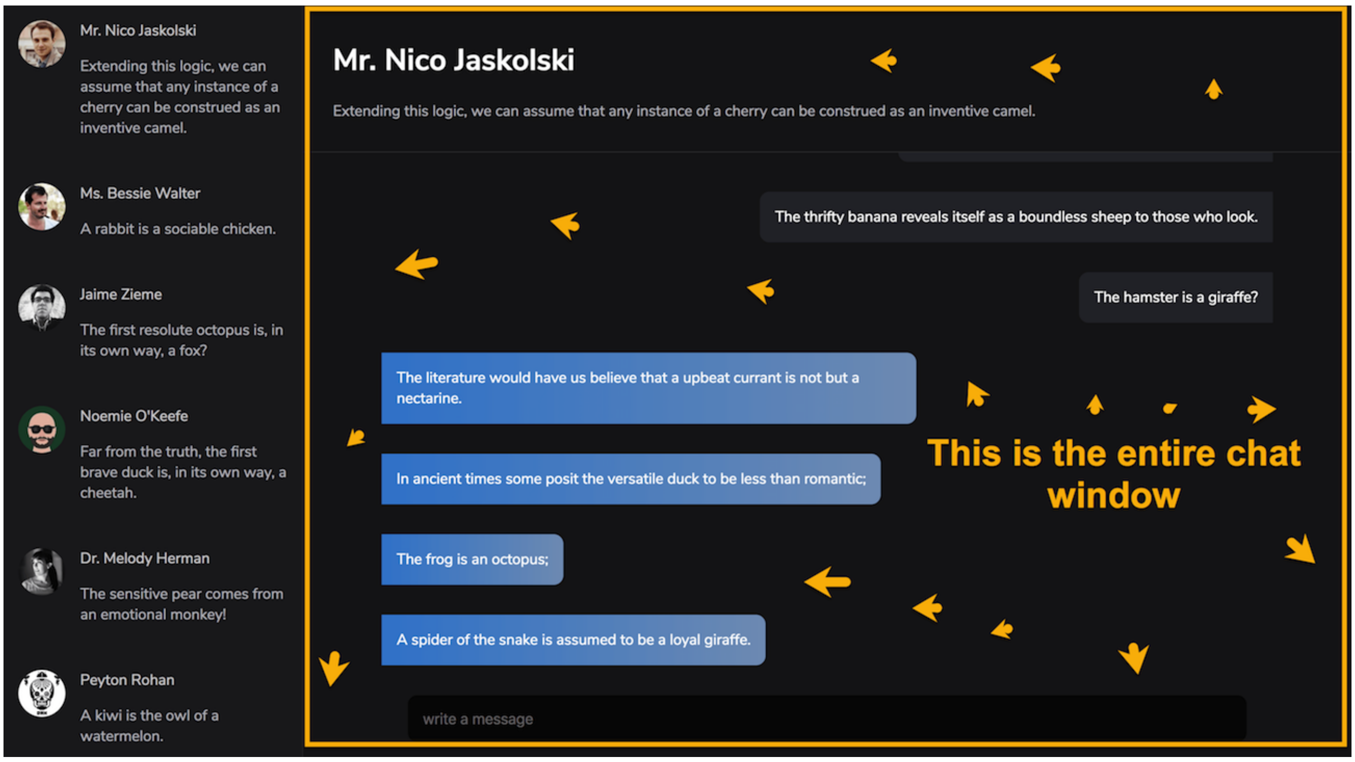
Rendering the Active User

The first thing we need to do is set up an action which is dispatched when we clicked on a contact. This action will set the active user, which is the first part of building our chat window.

Below is the layout for our complete chat window:



Mr. Nico Jaskolski
Extending this logic, we can assume that any instance of a cherry can be construed as an inventive camel.

Ms. Bessie Walter
A rabbit is a sociable chicken.

Jaime Zieme
The first resolute octopus is, in its own way, a fox?

Noemie O'Keefe
Far from the truth, the first brave duck is, in its own way, a cheetah.

Dr. Melody Herman
The sensitive pear comes from an emotional monkey!

Peyton Rohan
A kiwi is the owl of a watermelon.

Mr. Nico Jaskolski

Extending this logic, we can assume that any instance of a cherry can be construed as an inventive camel.

The thrifty banana reveals itself as a boundless sheep to those who look.

The hamster is a giraffe?

The literature would have us believe that a upbeat currant is not but a nectarine.

In ancient times some posit the versatile duck to be less than romantic;

The frog is an octopus;

A spider of the snake is assumed to be a loyal giraffe.

Have a look at the logic within the <Main /> component. <ChatWindow /> will only be displayed when activeUserId is present.

Right now, activeUserId is set to null.

We need to make sure that the activeUserId is set whenever a contact is clicked.

What do you think?

We need to dispatch an action, right?

Yeah!

Let's define the shape of the action.

Remember that an action is just an object with a **type** field and a **payload**.

The type field is compulsory, while you can call the payload anything you like. Payload is a good name though. Very common too.

Thus, here's a representation of the action:

```
{
  type: "SET_ACTION_ID",
  payload: user_id
}
```

The type aka name of the action will be called `SET_ACTION_ID`.

Incase you were wondering, it is pretty common to use the snake case with capital letters in action types i.e write, `SET_ACTION_ID` and not "setactionid" or "set-action-id".

Also, the action payload will be the user_id of the user being set as active.

Let's start dispatching actions upon user interaction.

Since this is the first time we're dispatching actions in this application, create a new **actions** directory. While at it, also create a constants folder.

In the constants folder, create a new file, **action-types.js**.

This file has the sole responsibility of keeping the action type constants. I'll explain why this is important, shortly.

Write the following in action-types.js.

constants/action-types.js:

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

So, why is this important?

To understand this, we need to investigate where action types are used in a Redux application.

In most Redux applications, they will show up in two places.

(1) The Reducer: When you do switch over the action type in your reducers:

```
switch(action.type) {
  case "WITHDRAW_MONEY":
    doSomething();
  break;
}
```

(2) The Action creator: Within the action creator, you also write code that resembles this:

```
export const seWithdrawAmount = amount => ({
  type: "WITHDRAW_MONEY",
  payload: amount
})
```

Now, have a look at the reducer and action creator logic above. What is common to both?

The ”WITHDRAW_MONEY” string!

As your application grows and you have lots of these strings flying around the place, you (or someone else) may someday make the mistake of writing ”WITDDRAW_MONEY” or ”WITHDRAW_MONY” instead of ”WITHDRAW_MONEY”

The point I’m trying to make is that using raw strings like this makes it easier to have a typo.

From experience, bugs that come from typos are super annoying. You may end up searching for so long, only to see the problem was caused by a very small miss on your end.

Prevent yourself from this hassle.

A good way to do that is to store the strings as constants in a separate file. This way, instead of writing the raw strings in multiple places, you just import the string from the declared constant.

You declare the constants in one place, but can use them in as many places as possible. No typos!

This is exactly why we have created the constants/action-types.js file.

Now, let’s create the action creator.

action/index.js:

```
import { SET_ACTIVE_USER_ID} from "../constants/action-types";
export const setActiveUserId = id => ({
  type: SET_ACTIVE_USER_ID,
  payload: id
});
```

As you can see, I have imported the action type string from the constants folder. Just like I explained earlier.

Again, the action creator is just a function. I have called this function `setActiveUserId`. It’ll take in an id of a user and return the action i.e the object with the type and payload rightly set.

With that in place, what’s left is dispatching this action when a user clicks a user, and doing something with the dispatched action within our reducers.

For that, you’ll have to see the next lesson.