Building Message Input

This lesson contains the dispatch functionality for the text the user sends. The onChange event occurs when a user types, and the event further activates the handleChange function which dispatches the data to the store reducer.

Let's now create the actual `MessageInput` component.

Since this component will talk directly to the Redux store for setting and getting its typing value, it should be created in the containers directory.

While at it, also create a **MessageInput.css** file as well.

containers/MessageInput.js:

```
import React from "react";
import store from "../store";
import { setTypingValue } from "../actions";
import "./MessageInput.css";
const MessageInput = ({ value }) => {

  const handleChange = e => {
    store.dispatch(setTypingValue(e.target.value));
  };
  return (
    <form className="Message">
      <input
        className="Message__input"
        onChange={handleChange}
        value={value}
        placeholder="write a message"
        /> </form>
  ); };
export default MessageInput;
```

containers/MessageInput.js

Nothing magical happening up there.

Whenever the user types into the input box, the **onChange** event is fired. This in turn fires the **handleChange** function. handleChange in turn dispatches the **setTypingValue** action we created earlier. This time, passing the required payload, **e.target.value**

We’ve created the component, but to show up in the chat window we need to include it in the return statement of ChatWindow.js.

```
...
import MessageInput from "./MessageInput";
const { typing } = state;
return (
  <div className="ChatWindow">
    <Header user={activeUser} />
    <Chats messages={_.values(activeMsgs)} />
    <MessageInput value={typing} />
  </div> );
```

And now, we’ve got this working!

Uh, but the aesthetics need a lot of work. Let’s style it in the next lesson.