

ChatRoom Offline: Entrada de texto

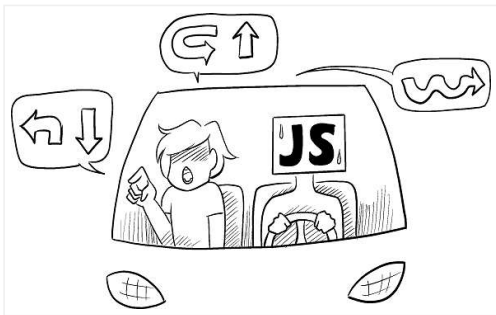
Resumen

- Escribiremos un componente `Input.js` que contiene un `<form>` y permite al usuario teclear un texto, almacenado en un estado
- Lo mostrará por consola al darle a Aceptar
- Añadiremos http://localhost:3000/chat_room y mostraremos dicho componente

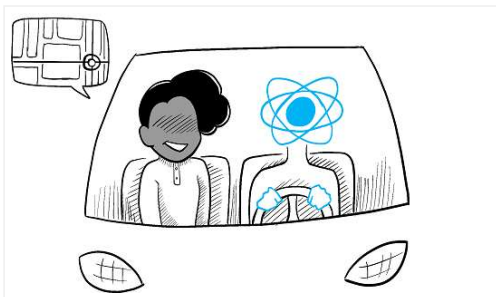
Descripción

Nuestras interacciones con el usuario de momento sólo han tenido lugar a través de botones. Pero las entradas de texto son algo muy habitual que, simplemente... ¡No podemos pasar por alto!

[Reaccionar a la entrada de texto del usuario](#) es un problema muy interesante en React. ¿Recuerdas el trabajo que nos daba *gestionar un formulario* desde puro JavaScript? Al hacerlo de manera imperativa, hay que explicar *cada cambio* que se produce. Manejar *a mano* el API del DOM del navegador supone una dificultad que crece exponencialmente con la complejidad de la interfaz de usuario (UI):



Con React, [plasmamos la interfaz de forma declarativa](#):



La tarea

Añade un nuevo componente `src/components/chat_room/Input.js` :

```
const Input = (props) => {  
  
  return <>  
    <form>  
      <input type="text" data-cy="input_chatroom" placeholder="Escribe algo..." />  
      <input type="submit" data-cy="chatroom_submit" value="Aceptar" />  
    </form>  
  </>  
}  
  
export default Input;
```

Ahora, **crea** un nuevo componente en `src/pages/ChatRoomPage.js` para una nueva página. De momento, sólo mostrará el `<Input />` :

```
import Input from "../components/chat_room/Input";  
  
const ChatRoomPage = (props) => {  
  return <div>  
    <Input />  
  </div>  
}
```

```
    </div>
  }

  export default ChatRoomPage;
```

Y, para poder verlo, **añade** en `App.js` la línea `<Route />` necesaria para *mapear* la ruta `"/chat_room"` al componente `<ChatRoomPage />`.

¿Lo has hecho?

¿Puedes ver el campo de texto y el botón desde http://localhost:3000/chat_room?

¡Genial! Continuemos.

Un estado... ¡cómo no!

El *texto* que el usuario introduce es algo único y relevante para nuestra interfaz. ¡Almacenémoslo en un estado!

Crea el siguiente estado en `Input.js`:

```
const [message, setMessage] = useState("");
```

A continuación, **crea** una función para *cada vez* que el usuario pulsa una tecla:

```
const customOnChange = (event) => {
  setMessage(event.target.value);
}
```

Y **modifica** el elemento `<input>` del `<form>` de la siguiente manera:

```
-   <input type="text" data-cy="input_chatroom" placeholder="Escribe algo..." />
+   <input type="text" data-cy="input_chatroom" placeholder="Escribe algo..." onChange={customOnChange} value={message} />
```

Como ves, hemos añadido `onChange` y `value`:

- `onChange` indica *qué* método se ejecutará cada vez que *cambia* el campo (con cada tecla tecleada). En `customOnChange`, `event.target` representa el propio elemento `<input>`. Por ello, `event.target.value` es el texto tecleado.
- `value` se emplea para *enlazar* el *valor* del formulario al estado `message`. Así, está *atado* "bidireccionalmente". Si en el futuro cambiamos el estado (`setMessage()`), entonces se reflejará en la interfaz. Pista: Lo haremos 😊

Y `onSubmit`

Ahora, cuando pulsas Aceptar en http://localhost:3000/chat_room la página se refresca, ¿verdad?

¡Es el comportamiento por defecto de los formularios!

Añade:

```
const customOnSubmit = (event) => {
  event.preventDefault(); // Cancelar comportamiento por defecto
  console.log("Enhorabuena, has enviado el mensaje:");
  console.log(message);
  setMessage(""); // Vaciar el input
}
```

Y **asócialo** a `onSubmit` del formulario:

```
-   <form>
+   <form onSubmit={customOnSubmit}>
```

¡Enhorabuena! Ya has implementado un comportamiento *customizado*. De momento, no es gran cosa. ¡Sólo un `console.log`!

Pero, ¿puedes verificar que lo ves en http://localhost:3000/chat_room?



Por último

Sube tus cambios al repositorio en un nuevo *commit*.



Rubén Montero [@ruben.montero](#) changed milestone to [%Sprint 2](#) 3 weeks ago