ImmutableJS es una librería que nos proporciona distintas estructuras de datos inmutables. Como hemos visto, el estado de Redux es inmutable, por lo que esta librería sirve de gran ayuda a la hora de trabajar con los `reducers`.

En el ejemplo del apartado anterior hemos visto que para actualiza el estado debíamos de utilizar `Object.assign`:

```
let messages = state.messages.slice();
messages.push(action.message);
// Necesitamos crear una copia del store!
return Object.assign({}, state, { messages: messages });
```

Este caso es muy sencillo, pero ¿qué ocurre si quisiéramos actualizar el email del usuario del siguiente estado?

```
const initialState = {
  messages: [],
  user: {
    username: 'test',
    attributes: {
      email: 'test@test2.com',
      created_at: 'now'
    }
  }
}
```

La lógica se complica pues necesitamos crear copias de los objetos de manera recursiva:

```
switch(action.type) {
  case 'UPDATE_EMAIL': {
    let email = action.email;

    return Object.assign({}, state, {
      user: Object.assign({}, state.user, {
        attributes: Object.assign({}, state.user.attributes, {
          email
        })
      })
    });
  }
}
```

Gracias a ImmutableJS podemos simplificar esta lógica definiendo el estado como un `Map`. `Map` es un tipo muy similar a los objetos de JavaScript. De hecho, el método `fromJS` recibe como parámetro un objeto y retorna un `Map`. Además, todos los atributos del objeto que recibe se convierten a tipos de ImmutableJS.

Vamos a reescribir nuestro dispatcher agregando esta nueva acción y utilizando ImmutableJS:

```
// Estado inicial de; store de redux
const initialState = Immutable.fromJS({
  messages: [],
  user: {
    username: 'test',
    attributes: {
      email: 'test@test2.com',
      created_at: 'now'
    }
  }
});

// Reducer de nuestra aplicación
const messageReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'MESSAGE_SEND':
      // Set y push generan nuevos objetos
      return state.set('messages', state.get('messages').push(action.message));
    case 'UPDATE_EMAIL':
      // Con mergeDeep mantenemos los elementos ya existentes
      return state.mergeDeep({
        user: { attributes: { email: action.email } }
      });
    default:
      return state;
  }
}
```

Código disponible en Codepen.