Vamos a ver como funciona ReduxJS con un ejemplo. Tenéis todo el código disponible en Codepen.

```
// Creamos la accion "Mandar un mensaje"
const sendMessage = (message) => {
  return {
    type: 'MESSAGE_SEND',
    message: message
  }
}

// Estado inicial de; store de redux
const initialState = {
  messages: []
}

// Reducer de nuestra aplicación
const messageReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'MESSAGE_SEND':
      let messages = state.messages.slice();
      messages.push(action.message);
      // Necesitamos crear una copia del store!
      return Object.assign({}, state, { messages: messages });
    default:
      return state;
  }
}

// Funcion con la que nos suscribimos al store
function newState(newState) {
  console.log('new State!');
}

// Creamos el store
const { createStore } = Redux;
const store = createStore(messageReducer);
```

En este ejemplo, hemos creado la acción `sendMessage` que guarda el mensaje que le pasemos como parámetro en el `store`. También hemos definido el `reducer`, que recibe el estado actual y la acción que queremos ejecutar contra el `store`. Si el estado aún no está definido, utilizaremos `initialState`.

Siempre debemos de crear una copia del state. Este es inmutable, de ahí la necesidad de actualizar esta variable utilizando el método `Object.assign`.**Además, siempre retornamos por defecto el `state` actual**.

Por último, el método `createStore` nos devuelve una instancia del `store`. Ya podemos lanzar acciones contra el store utilizando el método `dispatch`.

```
store.dispatch(sendMessage({ text: 'Hello World!' }));
```

Esta línea de código lanza la acción contra el `store` actualizando el estado actual. Si queremos ver el estado actual, lo obtenemos con el método `getState`.

Para suscribirnos al `store` utilizamos el método `subscribe`.

```
store.subscribe(function() {
  console.log('El store ha sido actualizado!!!');
});
```