Redux

SemEs2-08:
Jesús Pérez Noriega UO252137
Ana Mª García Sánchez UO264030

Índice

Introducción

El estado de las aplicaciones varía mucho.

Librerías de front-end que se desentienden del estado de la aplicación.

Redux es un contenedor predecible del estado de aplicaciones JavaScript.

Permite escribir aplicaciones:

- Consistentes
- Adaptables a distintos ambientes
- Fáciles de probar

Se puede usar combinado con cualquier librería de vistas.

Es muy pequeño y sin dependencias.

Conceptos básicos

Redux es muy simple.

El estado es un objeto sin setters.

Se realizan acciones.

Una función toma el estado y la acción realizada y devuelve el estado resultante, el nuevo estado de la aplicación.

Utiliza elementos ya existentes en JavaScript, sin librerías ni dependencias.

```
{ type: 'ADD_TODO', text: 'Ir a nadar a la piscina' }
{ type: 'TOGGLE_TODO', index: 1 }
{ type: 'SET_VISIBILITY_FILTER', filter: 'SHOW_ALL' }
```

```
{
  todos: [{
    text: 'Comer',
    completed: true
  }, {
    text: 'Hacer ejercicio',
    completed: false
  }],
  visibilityFilter: 'SHOW_COMPLETED'
}
```

Restricciones: Tres principios

```
type Store = {
  dispatch: Dispatch
  getState: () => State
  subscribe: (listener: () => void) => () => void
  replaceReducer: (reducer: Reducer) => void
}
```

Única fuente de la verdad

El estado de toda la aplicación está almacenado en un único objeto llamado store.

El estado es de solo lectura

Hay que emitir una acción, que es un objeto describiendo que ocurrió.

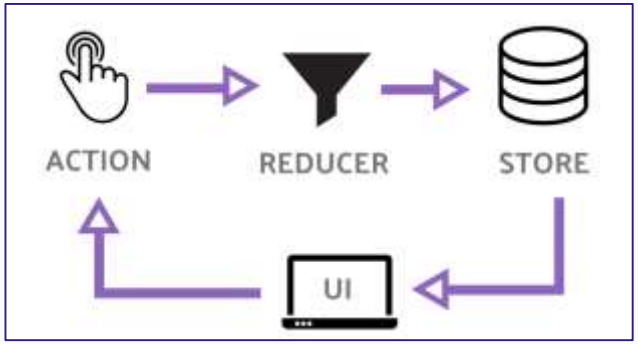
Funciones puras

Para transformar store por las acciones, se utilizan reducers puros.

Patrones, arquitectura y aspectos de desarrollo

Librería de JavaScript:

- YARN
- NPM
- Link facilitado por el creador



Reduce la interacción entre componentes simplificando su complejidad y facilitando depuración, mantenimiento y escalabilidad.

Patrones: Flux, Command y Flummox

Agnóstica al framework

Atributos de calidad

Sencillez

Se puede incluir a un proyecto ya hecho de forma rápida y sin conflictos

Rendimiento

Librería muy ligera que facilita la renderización de elementos

Mantenibilidad

Se basa en el principio de única responsabilidad, entre otros

Constante actualización y resolución de dudas en su repositorio

Stakeholders

¿Quién afecta a REDUX?

- Dan Abramov.
- Andrew Clark.
- Tim Dorr y Mark Erikson.
- Jaime Paton.
- Patrocinadores
- Facebook

Pueden contribuir:

- Programadores de JS
- Gente con GitHub

¿A quién afecta REDUX?

- Gente que hace el front-end en apps web JS.
- Personas que programan en JS
- Desarrolladores de React
- Programadores con proyectos ya empezados
- Gente buscando como almacenar el estado de una app
- La comunidad de Redux
- Personas que hayan usado Flux o Elm

Ventajas y Desventajas

Ventajas

-Sencillez de incorporación

-Redux dev tools

-Principio de única responsabilidad

-Unit Testing/TDD

-Información y tutoriales

-Librería ligera

-No tiene dependencias

Desventajas

-Muchos ficheros y carpetas.

-Patrones difíciles de entender

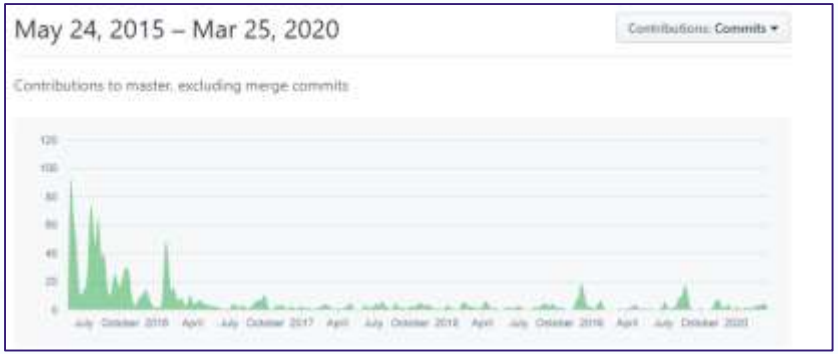
-Documentación en proceso

-Ocupación de memoria

-Muchos switches y constantes que son Strings

-Dificultad de distinguir cuándo es necesario

Comunidad



723 contribuyentes, 61 releases y casi 3.000 commits

Casi 2.000 pull requests cerradas y 11 abiertas

Secciones para aprender a usar Redux, FAQ, ejemplos...

Issues

Proyecto en constante evolución.

28 abiertos actualmente y más de 1700 cerrados.

Los primeros issues trataban sobre errores y debates.

Desde hace unos años se están usando para consultar dudas

A día de hoy, la mayor parte de los issues abiertos son referentes a la documentación de Redux.

Ejemplo del uso de Redux

uo252137 commented 7 hours ago

I would like to know if it would be possible to receive, as a parameter in a reducer, an arrow function instead of an action, so the switch can be omitted and it would be only necesary to execute the mentioned function.

If this were possible, switch cases could be avoided, resulting in improved performance of reducers with many different actions.

Example:

```
Before:
        function reducer(previousState, action) {
                switch(action.type) {
                        case 'ACTION1':
                          return { ..1.. };

                        case 'ACTION2':
                          return { ..2.. };

                        case 'ACTION3':
                          return { ..3.. };
                }
        }

After:
        action1(previousState) => { ..1.. }
        action2(previousState) => { ..2.. }
        action3(previousState) => { ..3.. }

        function reducer(previousState, arrowFunction) {
                return arrowFunction(previousState);
        }
```

Nuestro Issue

markerikson commented 1 hour ago — Contributor

Please see https://redux.js.org/recipes/reducing-boilerplate#generating-reducers and https://redux-toolkit.js.org/api/createReducer .

Conclusión

¿Preguntas?

FIN