4-Redux

Redux: flujo reactivo unidireccional con Angular y RxJs

```
# 1. Arquitectura del patrón Redux

# 2. Implementación de un Store con RxJs
```

> Redux no hace rápido lo simple, sino mantenible lo complejo

1 Arquitectura del patrón Redux

```
As a: devoloper,
  I want: to know what actions can be done
  so that: I can control the functionality

As a: developer,
  I want: to know what changes have been done
  so that: I can debug and predict behaviour
```

1.1 Principios de Redux

- **Single Source Of Truth**: Cada pieza de información se almacena en un único lugar, independientemente de dónde sea creada, modificada o requerida.
- **Read Only State**: La información será de sólo lectura y sólo podrá modificarse a través de conductos oficiales.
- **Changes By Pure Functions**: Los cambios tienen que poder ser replicados, cancelados y auditados; lo mejor, usar una función pura.

1.2 Elementos de Redux

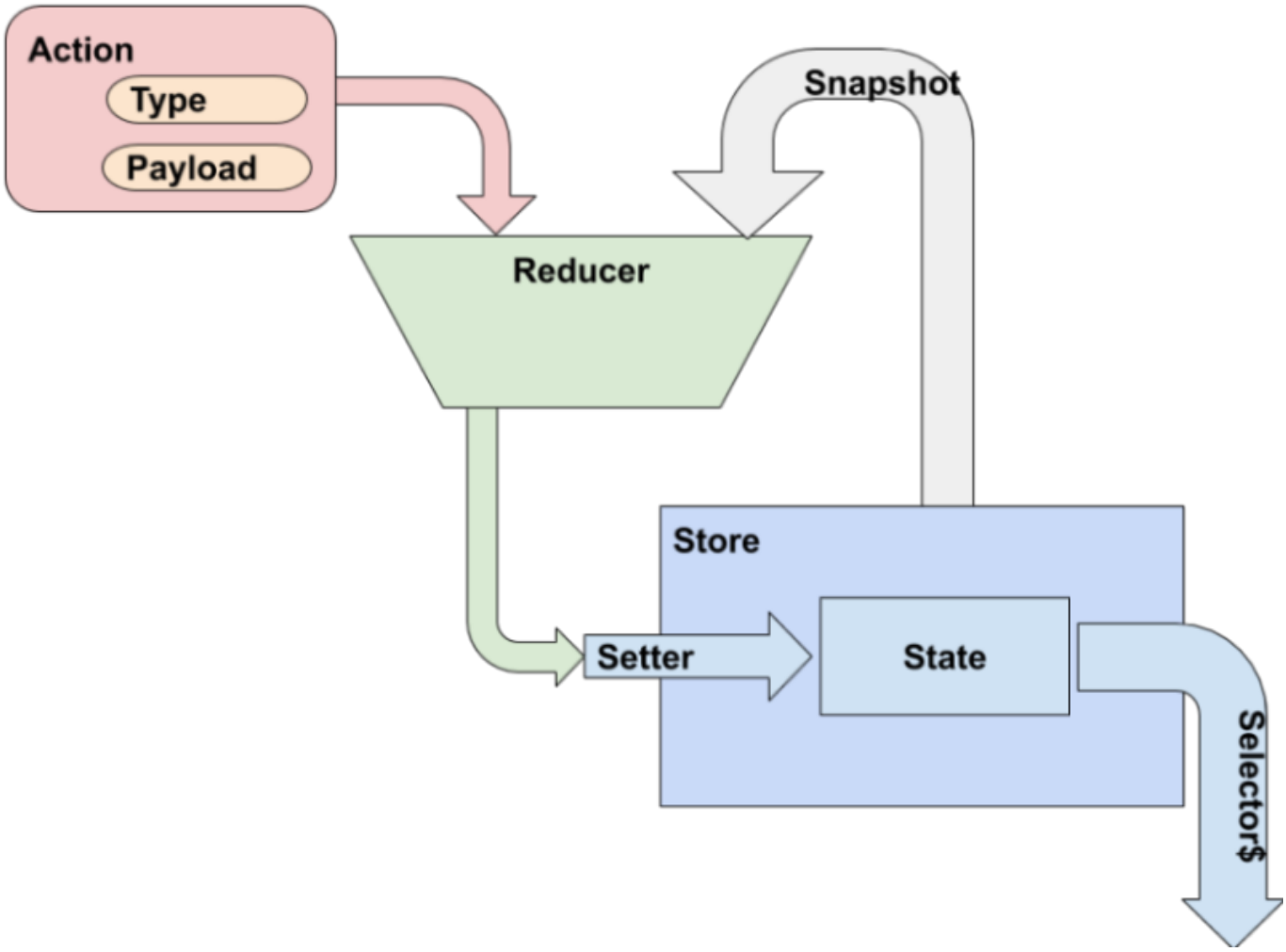
- **Store**: El sistema que mantiene el estado. Despacha acciones de mutado sobre el mismo y comunica cambios enviando copias de sólo lectura a los subscriptores.
- **State**: Árbol de objetos que contienen la única copia válida de la información. Representa el valor del almacén en un momento determinado.Nunca expondremos un puntero a este dato privado.

Acceso al estado

- **Setters** : Métodos que asignan y notifican un nuevo cambio. Clonan la información recibida para que el llamante no tenga un puntero al estado.
- **Selectors** : Métodos para consulta del estado. Devuelven un observable al que suscribirse para obtener notificaciones de cambio o una instantánea. En cualquier caso siempre emitirá o devolverá un clon del estado.

Mutaciones del estado

- **Actions**: Objetos identificados por un tipo y cargados con un *payload*. Transmiten una intención de mutación sobre el estado del *store*.
- **Reducers** : Son funciones puras, que ostentan la exclusividad de poder mutar el estado. Reciben dos argumentos: el estado actual y una acción con su tipo y su carga. Clonan el estado, realizan los cambios oportunos y devuelven el estado mutado.



---

2 Implementación de un Store con RxJs

```
As a: seller,
  I want: to know how many products are out of stock
```

```
  so that: I can refill them
```

```
ng g @nrwl/workspace:library rx-store
```

2.1 El Store observable mínimo

libs\rx-store\src\lib\mini-store.ts

```
import { BehaviorSubject, Observable } from 'rxjs';

export class MiniStore<T> {
  private state: T;
  private subject$ = new BehaviorSubject(this.get());

  constructor(initialState: T) {
    this.set(initialState);
  }
  public set(newSate: T) {
    this.state = { ...newSate };
    this.subject$.next(this.get());
  }
  public select$(): Observable<T> {
    return this.subject$.asObservable();
  }
  private get(): T {
    return { ...this.state };
  }
}
```

2.2 El envío de acciones

libs\rx-store\src\lib\rx-store.ts

```
import { BehaviorSubject, Observable } from 'rxjs';

export interface Action {
  type: string;
  payload: any;
}

export type reducerFunction<T> = (state: T, action: Action) => T;
```

```
export class RxStore<T> {
  private state: T;
  private subject$ = new BehaviorSubject<T>(this.get());

  constructor(initialState: T, private reducer: reducerFunction<T>) {
    this.set(initialState);
  }
  public select$(): Observable<T> {
    return this.subject$.asObservable();
  }
  public dispatch(action: Action) {
    const curretState = this.get();
    const newState = this.reducer(curretState, action);
    this.set(newState);
  }
  private get(): T {
    return { ...this.state };
  }
  private set(newSate: T) {
    this.state = { ...newSate };
    this.subject$.next(this.get());
  }
}
```

2.3 La función reductora de estado

libs\rx-store\src\lib\rx-store.spec.ts

```
const stockReducer: reducerFunction<ProductStock> = function(
    state: ProductStock,
    action: Action
  ): ProductStock {
    const clonedState = { ...state };
    switch (action.type) {
      case 'set':
        clonedState.stock = action.payload;
        break;
      case 'increment':
        clonedState.stock += action.payload;
        break;
      default:
        break;
    }
    return clonedState;
  };
```

```
describe('WHEN: I get an increment ', () => {
  const stockRxStore = new RxStore<ProductStock>(initial, stockReducer);
  const incrementAction: Action = { type: 'increment', payload: 5 };
  stockRxStore.dispatch(incrementAction);
  it('THEN: it should raise the stock', done => {
    stockRxStore.select$().subscribe(res => {
      expect(res).toEqual({ stock: 30 });
      done();
    });
  });
});
```

> **Blog de apoyo:** Flujo reactivo unidireccional con Angular y RxJs
>
> > By Alberto Basalo

> Next:

Redux con NgRx

Instalación y configuración

Actions

State reducer

Selectors

Effects