5-NgRx

Redux con NgRx

```
# 1. Instalación y configuración
# 2. Actions
# 3. Reducers
# 4. Selectors
# 5. Effects
```

```
As a: developer,
  I want: to use logging tools
  so that: I can debug the application better

As a: devoloper,
  I want: to have a set of allowed actions
  so that: I can get help while developing

As a: developer,
  I want: to query my state in a reactive way
  so that: I can be notified of any changes nay time

As a: developer,
  I want: to use send, chain and forget actions
  so that: I can make asynchronous calls
```

NgRx: Reactive State for Angular

Inspirada en Redux

Basada en RxJS

Estándar de facto,

Herramientas de generación de código

Modular y extensible.

Mantra para Redux:

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

--

Mantra Redux con NgRx:

> NgRx no hace rápido a Redux, sino mantenible el boilerplate

1 Instalación y configuración

1.1 Instalación de NgRx

```
ng g @nrwl/angular:ngrx app --module=apps/shop/src/app/app.module.ts --root --
minimal
```

1.2 DevTools

```
# https://chrome.google.com/webstore/detail/redux-
devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=en
```

1.3 Router

```
StoreRouterConnectingModule.forRoot({ routerState: RouterState.Minimal })
StoreModule.forRoot(
      {
        router: routerReducer
      },...
```

2 Actions

```
As a: customer,
  I want: to add payment methods
  so that: I can pay with them

As a: customer,
  I want: to select one as preferred
  so that: I can make fewer clicks

As a: customer,
  I want: to change de expiration date
  so that: I get my cards up to date
```

2.1 Create

```
ng g m payments --project=shop --module=app.module.ts --routing --route=payments
ng g @ngrx/schematics:feature payments/store/paymentMethod --project=shop --
module=payments/payments.module.ts --no-flat --no-spec --creators
```

apps\shop\src\app\payments\store\payment-method\payment-method.model.ts

```
export interface PaymentMethod {
  id: string;
  expiration: Date;
}

export interface PaymentMethods {
  list: PaymentMethod[];
  preferred: string;
}
```

apps\shop\src\app\payments\store\payment-method\payment-method.actions.ts

```
export const loadPaymentMethods = createAction(
  '[PaymentMethod] Load PaymentMethods'
);

export const addPaymentMethod = createAction(
  '[PaymentMethod] Add PaymentMethod',
  props<{ newPaymentMethod: PaymentMethod }>()
);

export const selectPreferredPaymentMethod = createAction(
  '[PaymentMethod] Select preferred PaymentMethod',
  props<{ preferredId: string }>()
);

export const setExpirationPaymentMethod = createAction(
  '[PaymentMethod] Set Expiration Date on PaymentMethod',
  props<{ updatedPaymentMethod: PaymentMethod }>()
);
```

2.2 Dispatch

apps\shop\src\app\payments\store\payment-method\payment-method.service.ts

```
import { Injectable } from '@angular/core';
import { Store } from '@ngrx/store';
import * as PaymentMethodActions from './payment-method.actions';
import {
  PaymentMethod,
```

```
  PaymentMethods
} from './payment-method.model';

@Injectable({
  providedIn: 'root'
})
export class PaymentMethodFacade {
  constructor(private store: Store<PaymentMethods>) {}
}
```

```
public loadPaymentMethods() {
  this.store.dispatch(PaymentMethodActions.loadPaymentMethods());
}
public addPaymentMethod(newPaymentMethod: PaymentMethod) {
    this.store.dispatch(
      PaymentMethodActions.addPaymentMethod({
        newPaymentMethod: { ...newPaymentMethod }
      })
    );
  }
public selectPreferredPaymentMethod(preferredId: string) {
  this.store.dispatch(
    PaymentMethodActions.selectPreferredPaymentMethod({ preferredId })
  );
}
public setExpirationPaymentMethod(updatedPaymentMethod: PaymentMethod) {
  this.store.dispatch(
    PaymentMethodActions.setExpirationPaymentMethod({
      updatedPaymentMethod: { ...updatedPaymentMethod }
    })
  );
}
```

apps\shop\src\app\payments\payments.component.ts

```
export class PaymentsComponent implements OnInit {
  constructor(private paymentMethodService: PaymentMethodService) {}

  ngOnInit() {
    this.paymentMethodService.loadPaymentMethods();
    const visa: PaymentMethod = {
      id: '1234 7896 3214 6549',
      expiration: new Date(2020, 6-1, 30)
    };
    this.paymentMethodService.addPaymentMethod(visa);
    this.paymentMethodService.selectPreferredPaymentMethod(visa.id);
    visa.expiration = new Date(2021, 12-1, 31);
    this.paymentMethodService.setExpirationPaymentMethod(visa);
```

```
  }
}
```

3 State reducer

3.1 State

apps\shop\src\app\payments\store\payment-method\payment-method.reducer.ts

```
export interface State {
  paymentMethods: PaymentMethods;
}

export const initialState: State = {
  paymentMethods: { list: [], preferred: null }
};
```

3.2 Create function

```
const paymentMethodReducer = createReducer(
  initialState,
  on(PaymentMethodActions.loadPaymentMethods, state => state)
);
```

```
on(PaymentMethodActions.addPaymentMethod, (state, { newPaymentMethod }) => {
  return {
    ...state,
    paymentMethods: {
      ...state.paymentMethods,
      list: [...state.paymentMethods.list, newPaymentMethod]
    }
  };
})
```

```
on(
  PaymentMethodActions.selectPreferredPaymentMethod,
  (state, { preferredId }) => {
    return {
      ...state,
      paymentMethods: { ...state.paymentMethods, preferred: preferredId }
```

```
    };
  }
)
```

```
on(
  PaymentMethodActions.setExpirationPaymentMethod,
  (state, { updatedPaymentMethod }) => {
    const list = state.paymentMethods.list;
    const updatedlist = list.map(pM =>
      pM.id === updatedPaymentMethod.id ? updatedPaymentMethod : pM
    );
    return {
      ...state,
      paymentMethods: {
        ...state.paymentMethods,
        list: updatedlist
      }
    };
  }
)
```

3.3 Register in Store

apps\shop\src\app\payments\store\payment-method\payment-method.reducer.ts

```
export function reducer(state: State | undefined, action: Action) {
  return paymentMethodReducer(state, action);
}
```

apps\shop\src\app\payments\payments.module.ts

```
import * as fromPaymentMethod from './store/payment-method/payment-
method.reducer';
@NgModule({
  imports: [
    StoreModule.forFeature(
      fromPaymentMethod.paymentMethodFeatureKey,
      fromPaymentMethod.reducer
    )]
```

4 Selectors

4.1 Create selector

apps\shop\src\app\payments\store\payment-method\payment-method.selectors.ts

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { paymentMethodFeatureKey, State } from './payment-method.reducer';

export const getPaymentMethodState = createFeatureSelector<State>(
  paymentMethodFeatureKey
);

export const getPaymentMethodsList = createSelector(
  getPaymentMethodState,
  (state: State) => state.paymentMethods.list
);

export const getPreferredPaymentMethod = createSelector(
  getPaymentMethodState,
  (state: State) => state.paymentMethods.preferred
);
```

4.2 Selecting data

```
  public getPaymentMethodsList$(): Observable<PaymentMethod[]> {
    return this.store.select(PaymentMethodSelectors.getPaymentMethodsList);
  }

  public getPreferredPaymentMethod$(): Observable<string> {
    return this.store.select(PaymentMethodSelectors.getPreferredPaymentMethod);
  }
```

4.3 Showing data

```
export class PaymentsComponent implements OnInit {
  public paymentMethodsList$: Observable<PaymentMethod[]>;
  public preferredPaymentMethod$: Observable<string>;
  constructor(private paymentMethodService: PaymentMethodService) {}

  ngOnInit() {
    this.paymentMethodsList$ = this.paymentMethodService.getPaymentMethodsList$();
    this.preferredPaymentMethod$ =
this.paymentMethodService.getPreferredPaymentMethod$();
  }
}
```

```
<p>Payment Methods List:</p>
<pre>{{ paymentMethodsList$ | async | json }}</pre>
```

```
<p>Preferred Payment Method:</p>
<pre>{{ preferredPaymentMethod$ | async | json }}</pre>
```

5 Effects

5.1 Efecto básico

Acciones

apps\shop\src\app\payments\store\payment-method\payment-method.actions.ts

```
export const loadPaymentMethods = createAction(
  '[PaymentMethod] Load Payment Methods'
);

export const loadPaymentMethodsSucess = createAction(
  '[PaymentMethod] Load Payment Methods Success',
  props<{ paymentMethodList: PaymentMethod[] }>()
);

export const loadPaymentMethodsError = createAction(
  '[PaymentMethod] Load Payment Methods Error'
);
```

Definición

- Los reductores son funciones puras...
- ... sin efectos colaterales.
- Las actividades que los reductores no pueden...
- ... las realizan los efectos.

apps\shop\src\app\payments\store\payment-method\payment-method.effects.ts

```
@Injectable()
export class PaymentMethodEffects {
  public miEfecto$ : Observable<Action>;

  constructor(private actions$: Actions) {}
}
```

```
public loadPaymentMethods$ = createEffect(() =>
  this.actions$.pipe(
    ofType(PaymentMethodActions.loadPaymentMethods),
```

```
      concatMap(() => {
        try {
          let storedList = JSON.parse(
            window.localStorage.getItem(this.storeKey)
          );
          if (!storedList) {
            storedList = initialState.paymentMethods.list;
            window.localStorage.setItem(
              this.storeKey,
              JSON.stringify(storedList)
            );
          }
          return of(
            PaymentMethodActions.loadPaymentMethodsSucess({
              paymentMethodList: storedList
            })
          );
        } catch (e) {
          return of(PaymentMethodActions.loadPaymentMethodsError);
        }
      })
    )
  );
```

Reducer y Register

apps\shop\src\app\payments\store\payment-method\payment-method.reducer.ts

```
    on(
      PaymentMethodActions.loadPaymentMethodsSucess,
      (state, { paymentMethodList }) => {
        return {
          ...state,
          paymentMethods: { ...state.paymentMethods, list: paymentMethodList }
        };
      }
    ),
    on(PaymentMethodActions.loadPaymentMethodsError, state => state),
```

apps\shop\src\app\payments\payments.module.ts

```
  EffectsModule.forFeature([PaymentMethodEffects])
```

Otro más sin reacciones

apps\shop\src\app\payments\store\payment-method\payment-method.effects.ts

```
public addPaymentMethod$ = createEffect(() =>
  this.actions$.pipe(
    ofType(PaymentMethodActions.addPaymentMethod),
    concatMap(action => {
      try {
        let storedList = JSON.parse(
          window.localStorage.getItem(this.storeKey)
        );
        storedList = [...storedList, action.newPaymentMethod];
        window.localStorage.setItem(
          this.storeKey,
          JSON.stringify(storedList)
        );
        return EMPTY;
      } catch (e) {
        return EMPTY;
      }
    })
  )
);
```

5.2 Efecto asíncrono

```
As a: customer,
  I want: to see the current exchange rate in several currencies
  so that: I can decide
```

```
ng g m rates --project=shop --module=app.module.ts --routing --route=rates
ng g @ngrx/schematics:feature rates/store/exchange-rate --project=shop --
module=rates/rates.module.ts --no-flat --no-spec --creators
```

Actions

apps\shop\src\app\rates\store\exchange-rate\exchange-rate.actions.ts

```
export const loadExchangeRates = createAction(
  '[ExchangeRate] Load ExchangeRates'
);

export const loadExchangeRatesSuccess = createAction(
  '[ExchangeRate] Load ExchangeRates Success',
  props<{ rates: any }>()
);

export const loadExchangeRatesError = createAction(
  '[ExchangeRate] Load ExchangeRates Error',
```

```
    props<{ rates: any }>()
  );
```

Effect

apps\shop\src\app\rates\store\exchange-rate\exchange-rate.effects.ts

```
export class ExchangeRateEffects {
  public loadExchangeRates$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ExchangeRateActions.loadExchangeRates),
      concatMap(() =>
        this.http.get<any>('https://api.exchangeratesapi.io/latest').pipe(
          map(res =>
            ExchangeRateActions.loadExchangeRatesSuccess({ rates: res.rates })
          ),
          catchError(err =>
            of(ExchangeRateActions.loadExchangeRatesError({ rates: err }))
          )
        )))
  );
  constructor(private actions$: Actions, private http: HttpClient) {}
}
```

Reducer

apps\shop\src\app\rates\store\exchange-rate\exchange-rate.reducer.ts

```
export interface ExchangeState {
  rates: any;
}

export const initialState: ExchangeState = {
  rates: {}
};
```

```
  on(ExchangeRateActions.loadExchangeRates, state => state),
  on(ExchangeRateActions.loadExchangeRatesSuccess, (state, payload) => ({
    ...state,
    rates: payload.rates
  })),
  on(ExchangeRateActions.loadExchangeRatesError, (state, payload) => ({
    ...state,
    rates: payload.rates
  }))
```

Component

apps\shop\src\app\rates\rates.component.ts

```
export class RatesComponent implements OnInit {
  public rates$ = this.store.select(exchangeRateFeatureKey, 'rates');

  constructor(private store: Store<ExchangeState>) {}

  ngOnInit() {
    this.store.dispatch(ExchangeRateActions.loadExchangeRates());
  }
}
```

```
<p>rates :</p>
<pre> {{ rates$ | async  | json  }} </pre>
```

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

> Next:

Deploy Progressive Web Apps

Angular Service Worker con el CLI

Configuración de caché

Actualizaciones y notificaciones