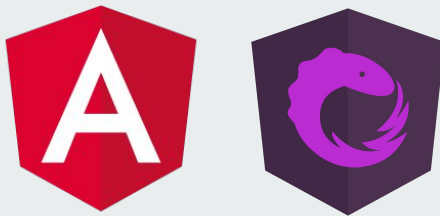




Building angular apps with NgRx





Agenda



- About me
- What is NgRx?
- How it works?
- When to use NgRx
- When NOT to use NgRx
- Conclusions
- What is next for NgRx?
- Who to follow
- References

About me

- Full Stack developer (10 xp years)
- Working currently @ Mobomo as FE Team Lead
- Stack: net., .net core, angular, ionic, react.
- ng-rosario co-organizer
- @salimchemes  
- www.salimchemes.com



What is NgRx?



- NgRx is a framework for building reactive applications in Angular.
- Provides
 - state management
 - isolation of side effects
 - entity collection management
 - router bindings
 - code generation
 - developer tools

What is NgRx?

ngrx supercharges the redux pattern with RxJS



=



+



+



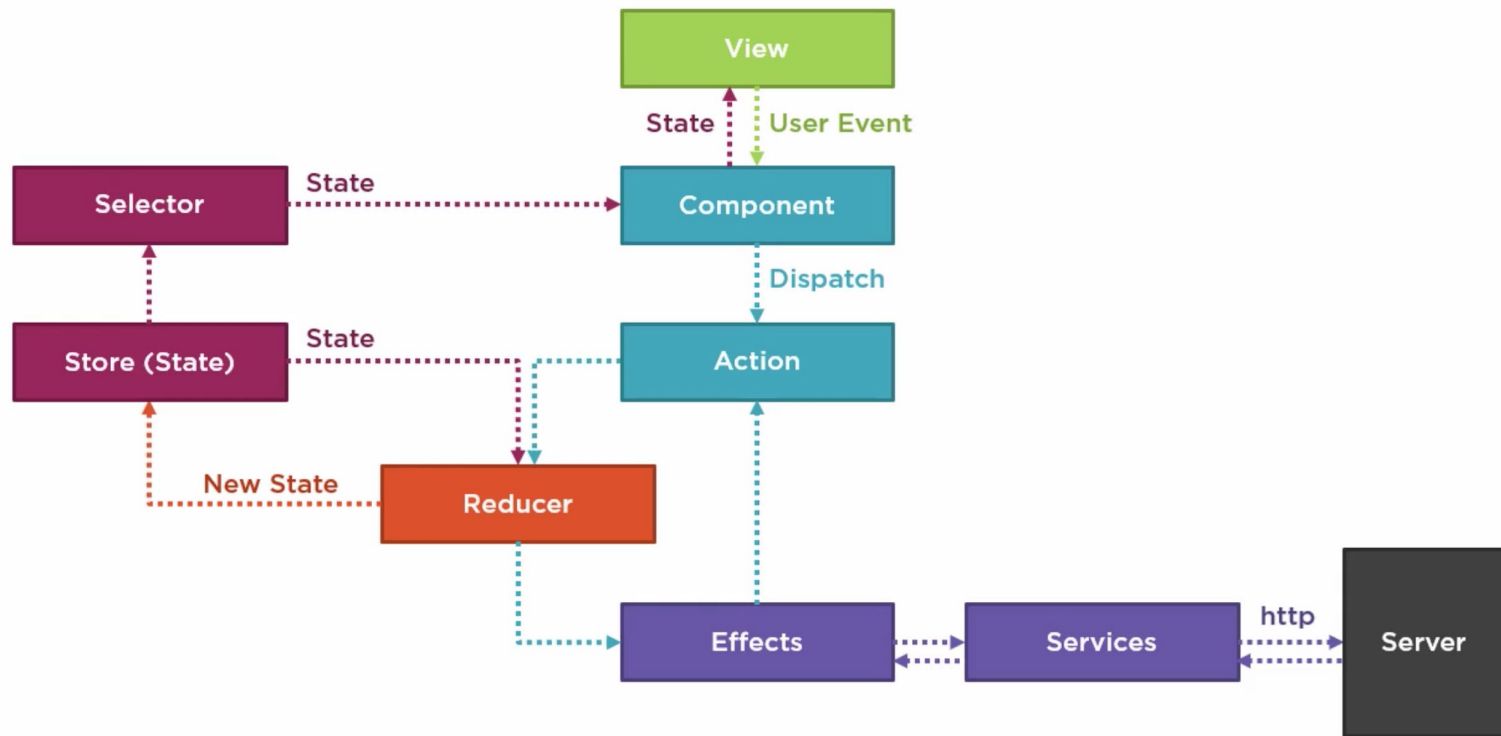
ngrx

redux
Pattern

Angular

RxJS

How it works?



How it works?

Redux is not great for making simple things quickly. It's great for making really hard things simple.

Jani Evakallio

Store



- Represents the state of our app
- It's typically typed with an interface
- Needs to be initialized
- Single source of truth (SST)

src/app/counter.reducer.ts

```
1. import { createReducer, on } from '@ngrx/store';
2. import { increment, decrement, reset } from
   './counter.actions';
3.
4. export const initialState = 0;
5.
6. const _counterReducer = createReducer(initialState,
7.   on(increment, state => state + 1),
8.   on(decrement, state => state - 1),
9.   on(reset, state => 0),
10. );
11.
12. export function counterReducer(state, action) {
13.   return _counterReducer(state, action);
14. }
```


Actions



- Describes unique events that are dispatched from components and services

src/app/counter.actions.ts

```
import { createAction } from '@ngrx/store';

export const increment = createAction('[Counter Component]  
Increment');
export const decrement = createAction('[Counter Component]  
Decrement');
export const reset = createAction('[Counter Component]  
Reset');
```



Reducers

- Pure function that takes the previous state and an action, and returns the next state

scoreboard.reducer.ts

```
export const initialState: State = {  
  home: 0,  
  away: 0,  
};
```

scoreboard.reducer.ts

```
const scoreboardReducer = createReducer(  
  initialState,  
  on(ScoreboardPageActions.homeScore, state => ({ ...state,  
    home: state.home + 1 })),  
  on(ScoreboardPageActions.awayScore, state => ({ ...state,  
    away: state.away + 1 })),  
  on(ScoreboardPageActions.resetScore, state => ({ home: 0,  
    away: 0 })),  
  on(ScoreboardPageActions.setScores, (state, { game }) =>  
    ({ home: game.home, away: game.away })),  
);  
  
export function reducer(state: State | undefined, action:  
  Action) {  
  return scoreboardReducer(state, action);  
}
```

Selectors

- Pure functions to perform queries to Store (like Linq)
- We can select one or multiple slices from the Store
- Please, don't get state from the Store directly

index.ts

```
1. import { createSelector } from '@ngrx/store';
2.
3. export interface FeatureState {
4.   counter: number;
5. }
6.
7. export interface AppState {
8.   feature: FeatureState;
9. }
10.
11. export const selectFeature = (state: AppState) =>
    state.feature;
12.
13. export const selectFeatureCount = createSelector(
14.   selectFeature,
15.   (state: FeatureState) => state.counter
16. );
```

Effects



- Isolates side effects from components
- Listen actions
- Performs tasks, which are sync or async and return a new action
- Filter these actions based on the type of action they are interested in

```
7. @Injectable()
8. export class MovieEffects {
9.
10.   loadMovies$ = createEffect(() =>
11.     this.actions$.pipe(
12.       ofType('[Movies Page] Load Movies'),
13.       mergeMap(() => this.moviesService.getAll()
14.         .pipe(
15.           map(movies => ({ type: '[Movies API] Movies
16.             Loaded Success', payload: movies })),
17.           catchError(() => EMPTY)
18.         ))
19.     );
```

When to use NgRx



- When your app has a complex state
- Fully understand the redux pattern
- You spike an implementation to run experiments if its needed
- Your team understands the pattern

When NOT to use NgRx



- Your team is new in Angular (angular + observables first)
- Your app is simple (avoid overkill)
- Your team already has a good pattern implemented
- Your team does not understand the pattern

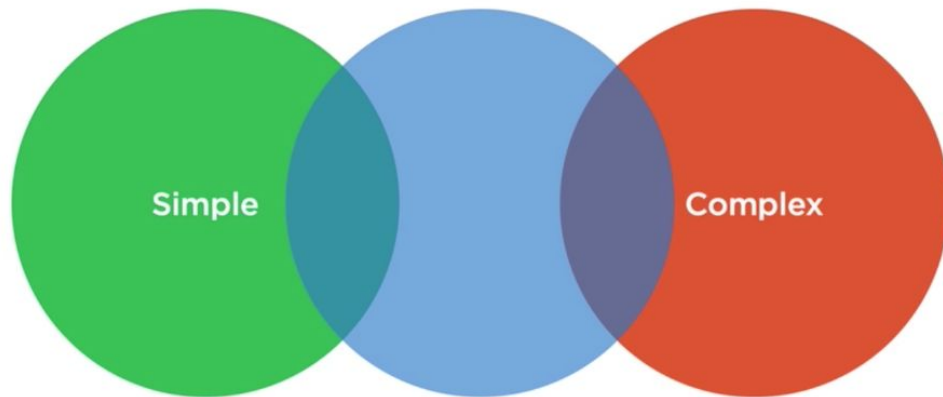
When and when not to use NgRx



	What is?	Pros	Cons
services	Observables to communicate components. Based on rxjs. No libraries here	<ul style="list-style-type: none">* easy to communicate components	<ul style="list-style-type: none">* hard to maintain when having multiple services implemented on the same component (not having single source for truth)
ngrx	NgRx Store provides reactive state management for Angular apps inspired by Redux. Unify the events in your application and derive state using RxJS.	<ul style="list-style-type: none">* immutable, uses a store as single source of truth* redux dev tools	<ul style="list-style-type: none">* a lot of code to add into the app (big boilerplate)* all the devs need to understand well the pattern (actions, reducers, selectors, effect, server)
ngrx-data	NgRx Data is an extension that offers a gentle introduction to NgRx by simplifying management of entity data while reducing the amount of explicitness.	<ul style="list-style-type: none">* ngrx simplified* less boilerplate* redux dev tools	<ul style="list-style-type: none">* need to understand the pattern to know whats happening there (kind of magic)
observable-store	Tiny front-end state management library that provides a lot of functionality	<ul style="list-style-type: none">* SSOT (single source of truth)* immutable* less boilerplate* change notification to subscribersworks with any fwk/lib	<ul style="list-style-type: none">* no redux dev tools

When and when not to use NgRx

Choosing a State Management Option



Services

ngrx-data

NgRx

Observable Store



Who to follow

- [@wesgrimes](#) - ngrx team (check this [post](#))
- [@mikeryandev](#) - ngrx co-creator
- [@nrwl_io](#) - dev experts and cool tools
- [@DeboraKuratah](#) - angular, rxjs, ngrx, etc
- [@aaronFrost](#) - angular + AiA podcast

References



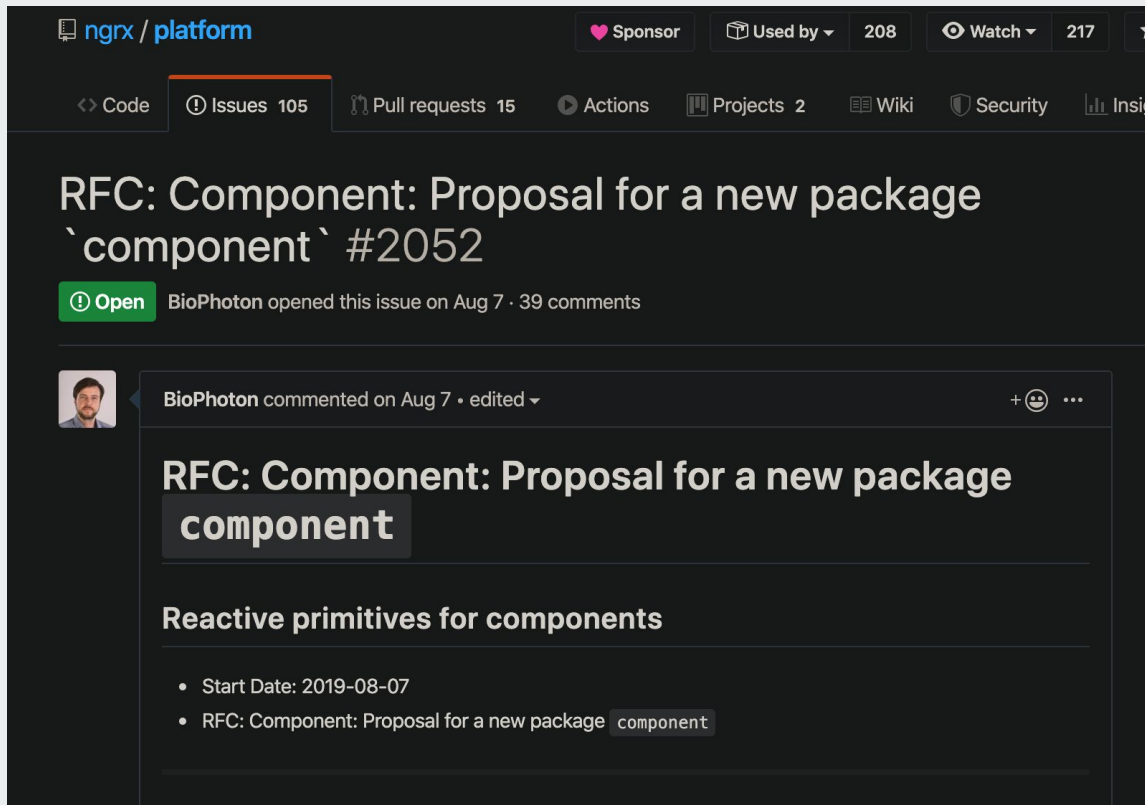
- [“Adventures in Angular: Angular state w/ ngrx with Mike Ryan”](#)
- [“Adventures in Angular: NgRx, The Mystical Machine, with Wes Grimes”](#)
- [“Play by Play: Angular and ngrx - by Lars Klint and Duncan Hunter”](#)
- [“Angular NgRx: Getting Started - by Deborah Kurata and Duncan Hunter”](#)
- [“Angular Architecture and Best Practices” - Dan Wahlin](#)

Conclusions



- NgRx is a robust solution to handle complex and large applications
- It provides a high learning curve
- If you need it, you will notice it
- It's important to understand the framework before implementation
- Evaluate different options before making your decision (ngrx-data, ngxs, services, akita, etc) about State Management

What is next on NgRx?



The screenshot shows the GitHub interface for the `ngrx/platform` repository. The issue title is "RFC: Component: Proposal for a new package `component` #2052". It is marked as "Open" and was created by BioPhoton on August 7, with 39 comments. The issue content includes the title "RFC: Component: Proposal for a new package component" and a section titled "Reactive primitives for components". Below this, there is a list of bullet points: "Start Date: 2019-08-07" and "RFC: Component: Proposal for a new package component".

ngrx / platform

Sponsor Used by 208 Watch 217

<> Code Issues 105 Pull requests 15 Actions Projects 2 Wiki Security Insights

RFC: Component: Proposal for a new package `component` #2052

Open BioPhoton opened this issue on Aug 7 · 39 comments

BioPhoton commented on Aug 7 · edited

RFC: Component: Proposal for a new package component

Reactive primitives for components

- Start Date: 2019-08-07
- RFC: Component: Proposal for a new package component



Q&A



Thanks



[ng-rosario](#)

@salimchemes

