State management with NgRx

Overview

- What is it?
- How do all parts of NgRx fit together?
- Why use it?
- Why not use it?
- Why do we use it?
- Resources
- Summary
- Credits

NgRx - What is it?

NgRx is a reactive state management framework

- **Reactive** change-based flow: either cause change or expect change
- **State management** entity data, loading indicator visible, active route

NgRx - What is it?

Store

```
{
  activeRoute: "/items",
  filters: {
    "propA": "valA"
  },
  items: [
    { "name": "item1", "propA": "valA" },
    { "name": "item2", "propA": "valA" }
  ],
  loading: false
}
```

Actions log

[Route] Navigate payload: { "route": "/items" }

[Filters] Apply payload: { "propA": "valA" }

[Items] Load payload: { "propA": "valA" }

[Items] Load Success payload: { "items": [...] }

NgRx - What is it?

Store before “[Filters] Apply”

```
{
  activeRoute: "/items",
  filters: {},
  items: [
    { "name": "item1", "propA": "valA" },
    { "name": "item2", "propA": "valA" },
    { "name": "item3", "propA": "valB" },
    { "name": "item4", "propA": "valC" },
    { "name": "item5", "propA": "valD" },
  ],
  loading: false
}
```

Actions log

[Route] Navigate payload: { "route": "/items" }

[Filters] Apply payload: { "propA": "valA" }

[Items] Load payload: { "propA": "valA" }

[Items] Load Success payload: { "items": [...] }

NgRx - What is it?

Store after "[Items] Load"

```
{
  activeRoute: "/items",
  filters: {
    propA: "valA"
  },
  items: [
    { "name": "item1", "propA": "valA" },
    { "name": "item2", "propA": "valA" },
    { "name": "item3", "propA": "valB" },
    { "name": "item4", "propA": "valC" },
    { "name": "item5", "propA": "valD" },
  ],
  loading: true
}
```

Actions log

[Route] Navigate payload: { "route": "/items" }

[Filters] Apply payload: { "propA": "valA" }

[Items] Load payload: { "propA": "valA" }

[Items] Load Success payload: { "items": [...] }

NgRx - What is it? - Redux

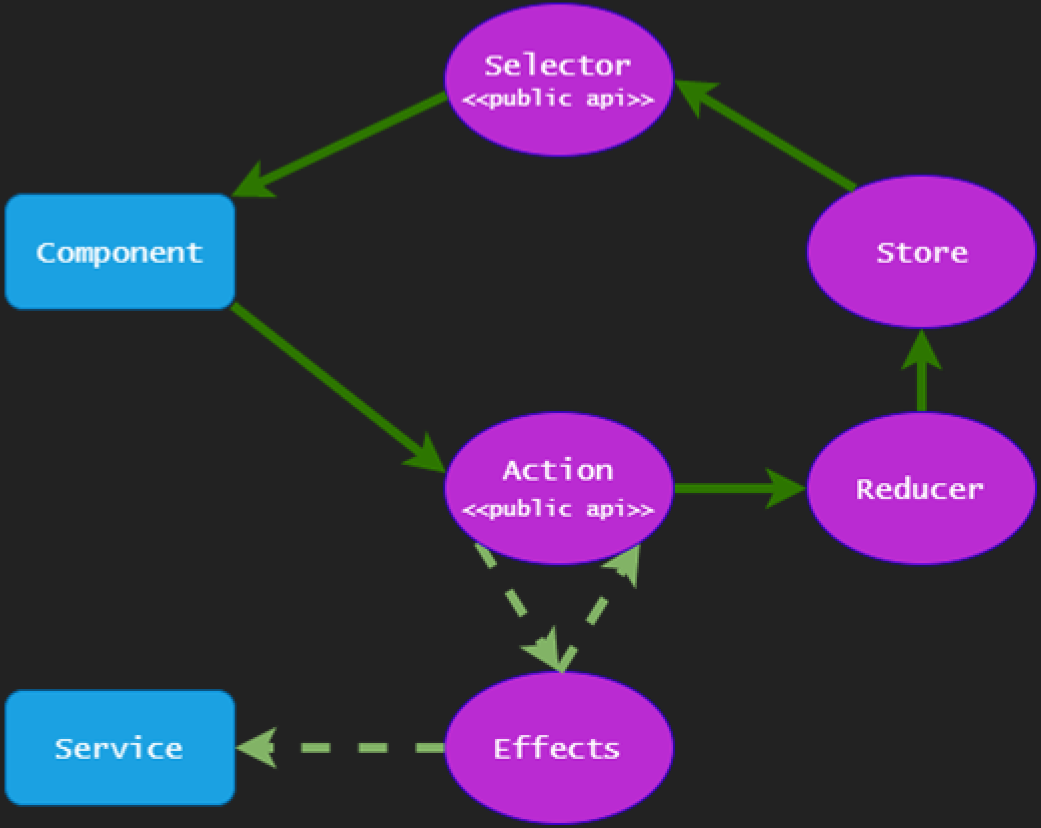
An open source JavaScript library to manage application state

- Created by Dan Abramov and Andrew Clark
- The initial release was in 2015
- It is licensed under MIT
- Abramov began writing the first Redux implementation, while preparing for a conference talk at React Europe on hot reloading

The three principles

- Single source of truth
- State is read-only
- Changes are made with pure functions

NgRx - How do all parts of NgRx fit together?



NgRx - How do all parts fit together? - Example 0/10

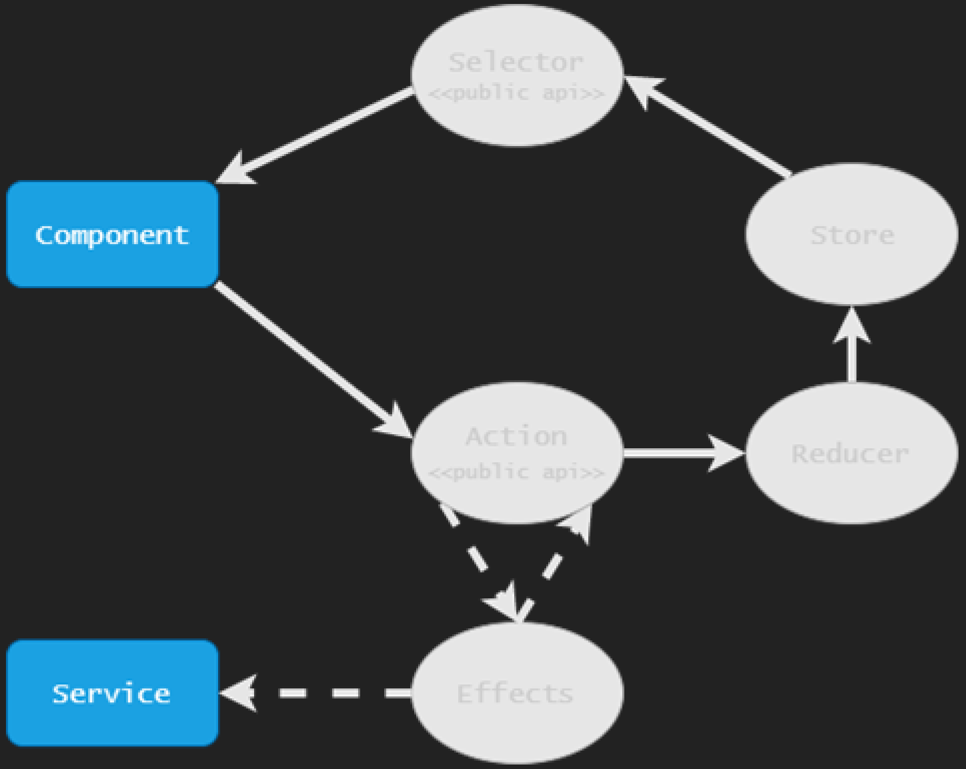
Store

```
{
  items: [],
  loading: false
}
```

Actions

UI

Flow description



NgRx - How do all parts fit together? - Example 1/10

Store

```
{
  items: [],
  loading: false
}
```

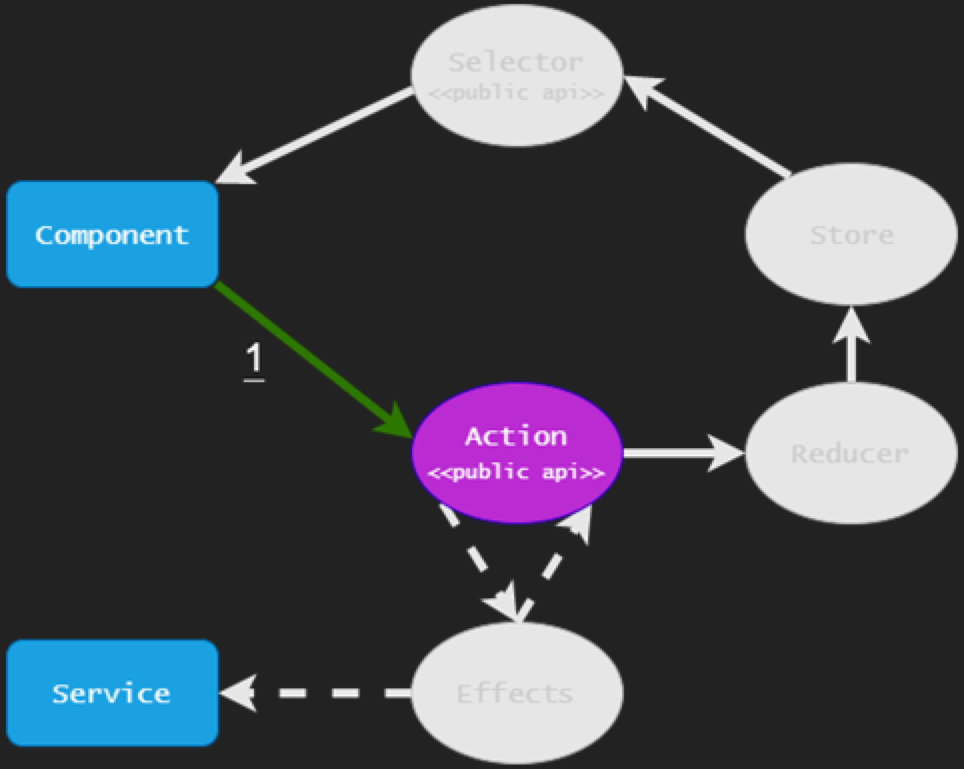
Actions

- [Users] Load

UI

Flow description

1 the component dispatches a “[Users] Load” action



NgRx - How do all parts fit together? - Example 2/10

Store

```
{
  items: [],
  loading: false
}
```

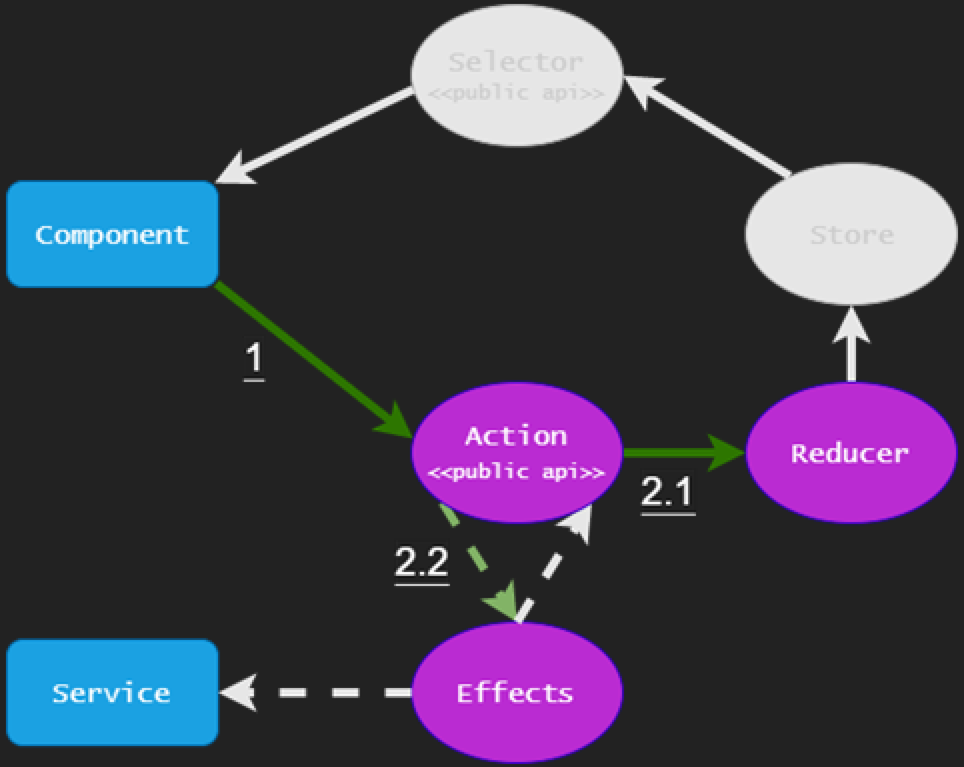
Actions

- [Users] Load

UI

Flow description

2.1 the reducer receives the “[Users] Load” action
2.2 the effect receives the “[Users] Load” action



NgRx - How do all parts fit together? - Example 3/10

Store

```
{
  items: [],
  loading: true
}
```

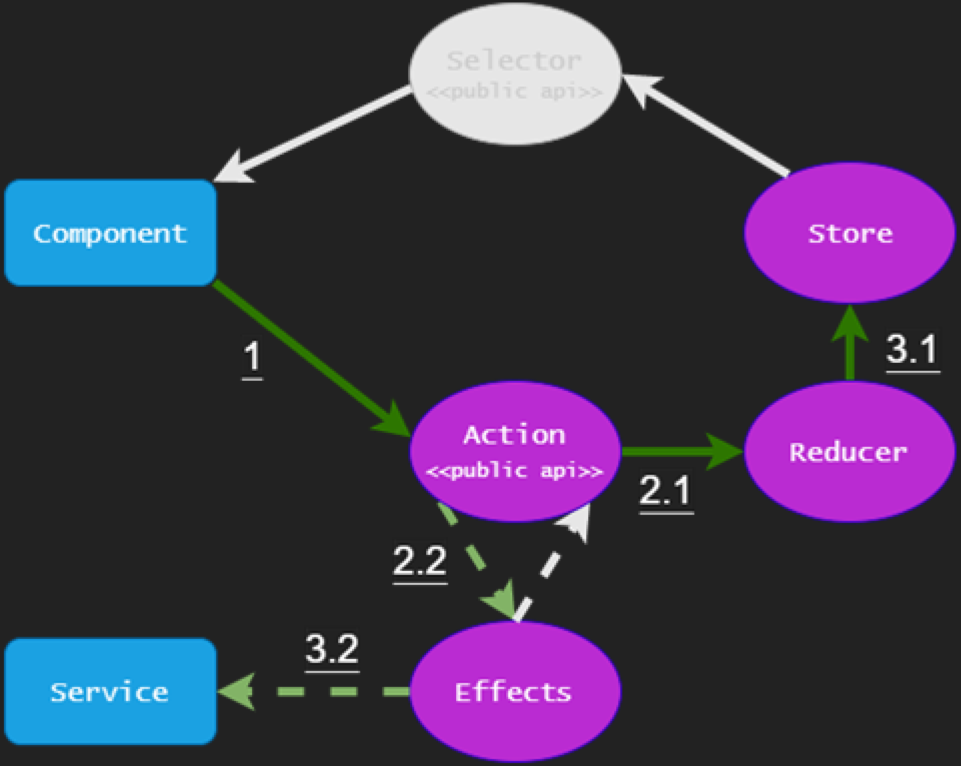
Actions

- [Users] Load

UI

Flow description

3.1 the reducer interprets the action and updates the state accordingly
3.2 the effect interprets the action and requests the users from the API through a service



NgRx - How do all parts fit together? - Example 4/10

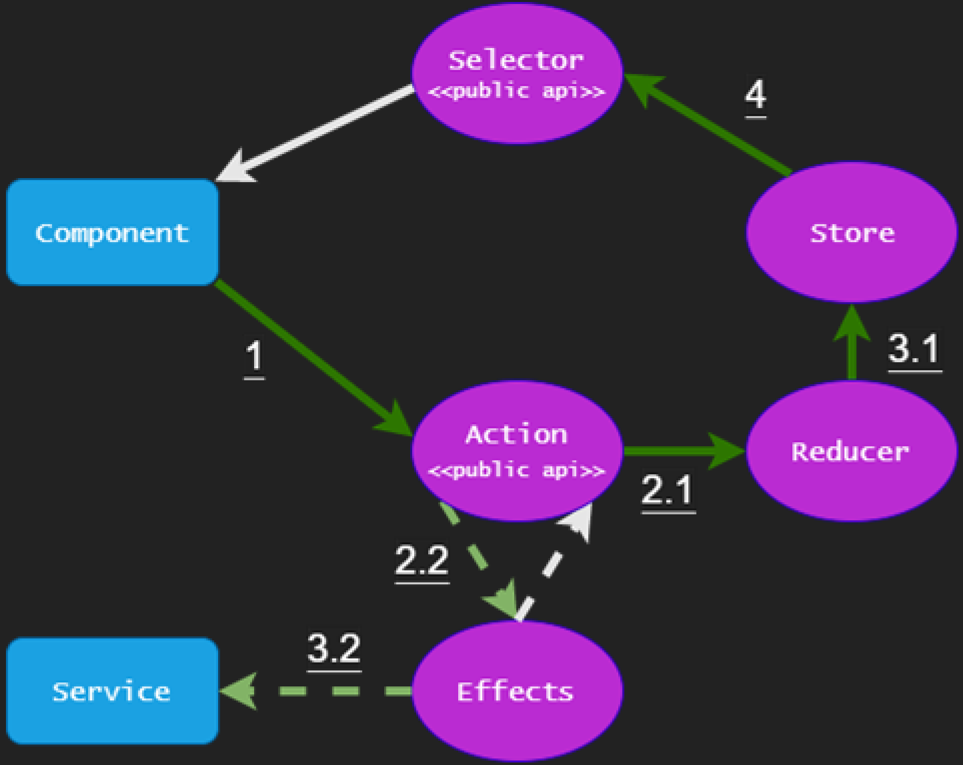
Store
```
{
  items: [],
  loading: true
}
```

Actions
- [Users] Load

UI

Flow description

4 the store triggers the "loading" selector

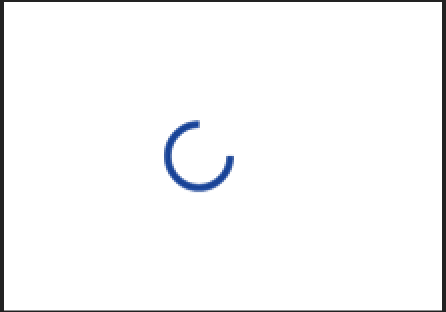


NgRx - How do all parts fit together? - Example 5/10

Store
```
{
  items: [],
  loading: true
}
```

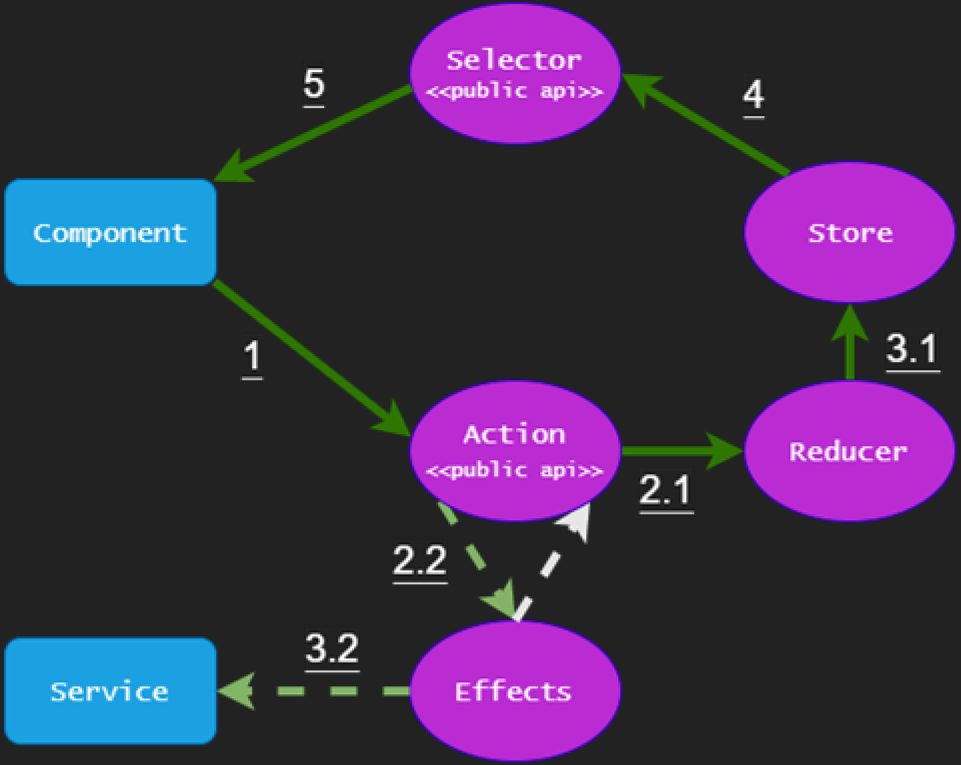
Actions
- [Users] Load

UI



Flow description

5 the “loading” selector notifies the component and the component displays a loading indicator since “loading” is true



NgRx - How do all parts fit together? - Example 6/10

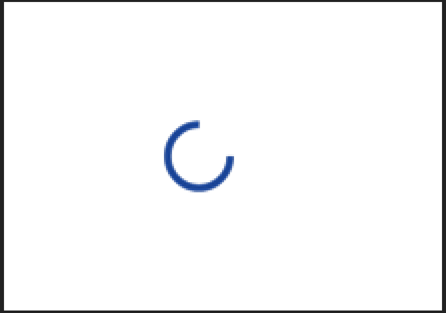
Store

```
{
  items: [],
  loading: true
}
```

Actions

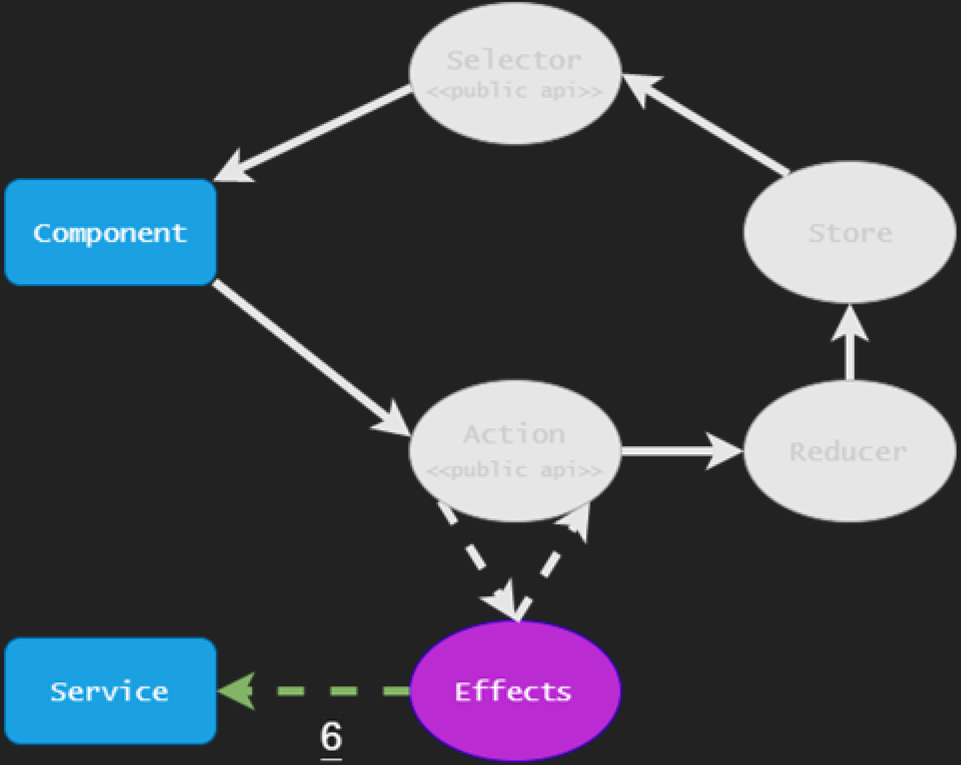
- [Users] Load

UI



Flow description

6 the service returns the API response



NgRx - How do all parts fit together? - Example 7/10

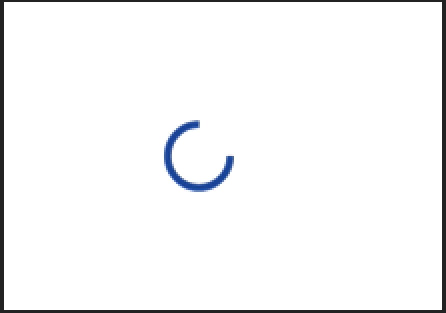
Store

```
{
  items: [],
  loading: true
}
```

Actions

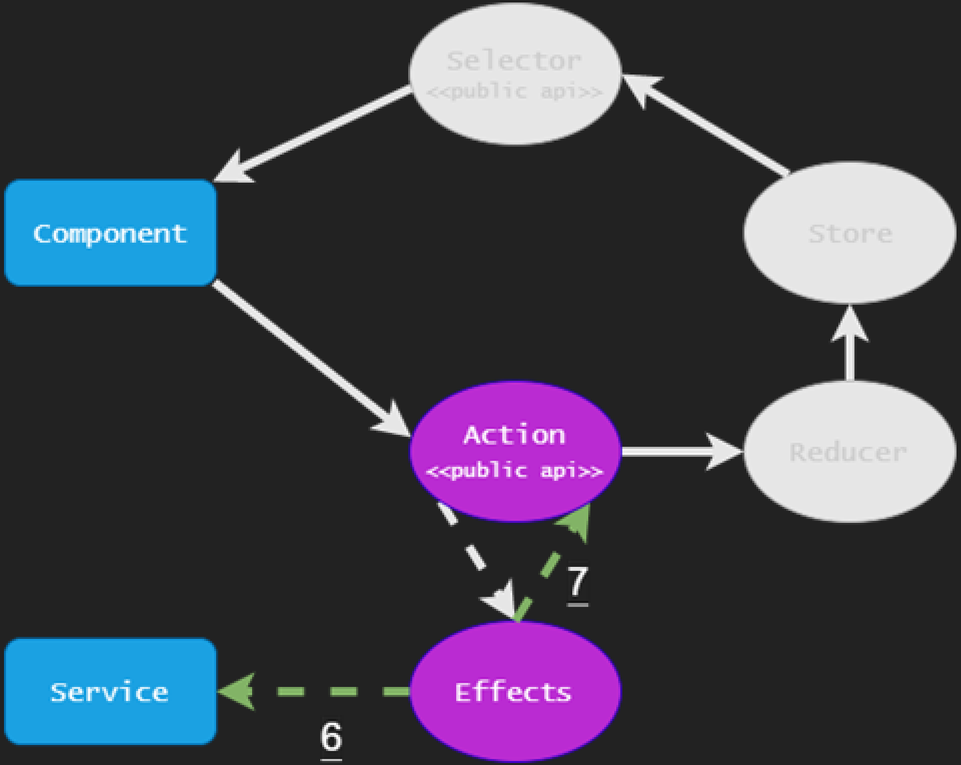
```
- [Users] Load
- [Users] Load Success
  {
    payload: [
      { id: 1, name: "a" },
      { id: 2, name: "b" },
    ]
  }
```

UI



Flow description

7 the effect dispatches a “[Users] Load Success” action that contains the users list returned by the API



NgRx - How do all parts fit together? - Example 8/10

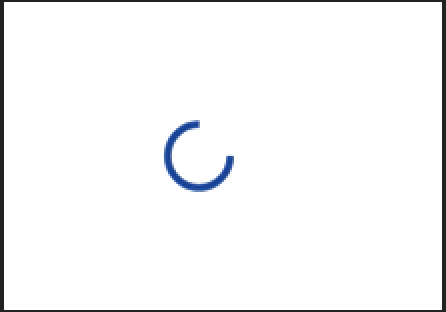
Store

```
{
  items: [],
  loading: true
}
```

Actions

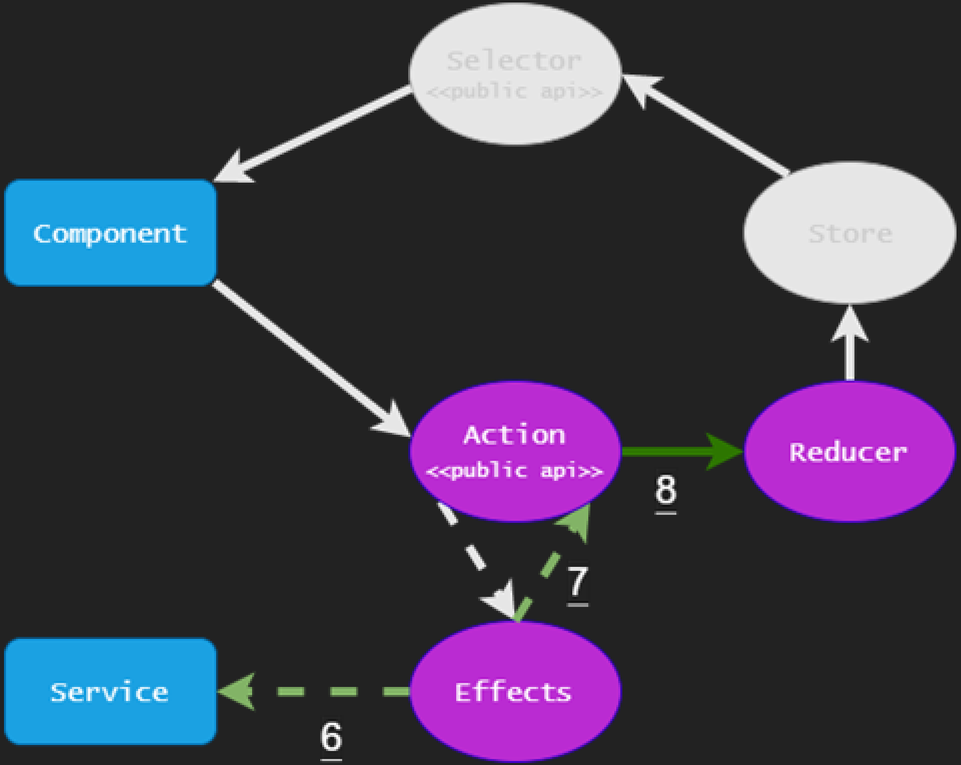
```
- [Users] Load
- [Users] Load Success
  {
    payload: [
      { id: 1, name: "a" },
      { id: 2, name: "b" },
    ]
  }
```

UI



Flow description

8 the reducer receives the “[Users] Load Success” action

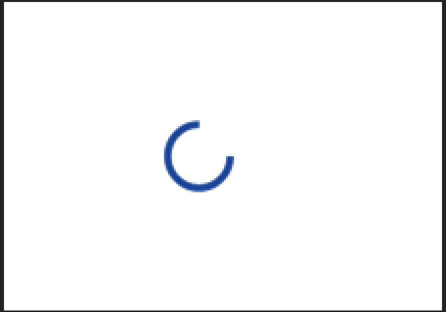


NgRx - How do all parts fit together? - Example 9/10

Store
```
{
  items: [
    { id: 1, name: "a" },
    { id: 2, name: "b" },
  ],
  loading: false
}
```

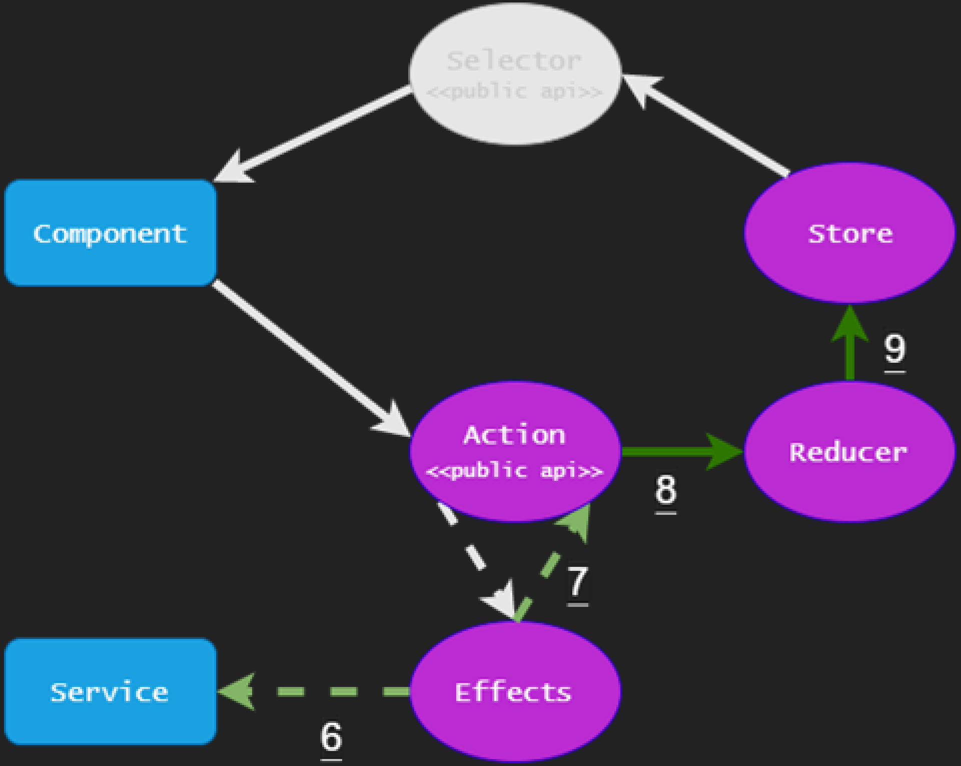
Actions
```
- [Users] Load
- [Users] Load Success
  {
    payload: [
      { id: 1, name: "a" },
      { id: 2, name: "b" },
    ]
  }
```

UI



Flow description

9 the reducer interprets the “[Users] Load Success” action and updates the state accordingly



NgRx - How do all parts fit together? - Example 10/10

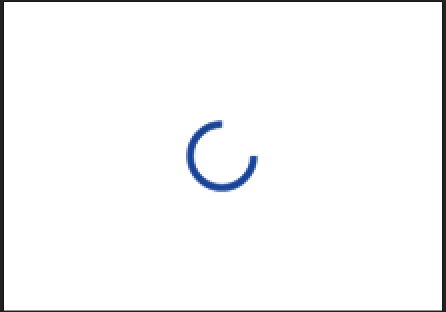
Store

```
{
  items: [
    { id: 1, name: "a" },
    { id: 2, name: "b" },
  ],
  loading: false
}
```

Actions

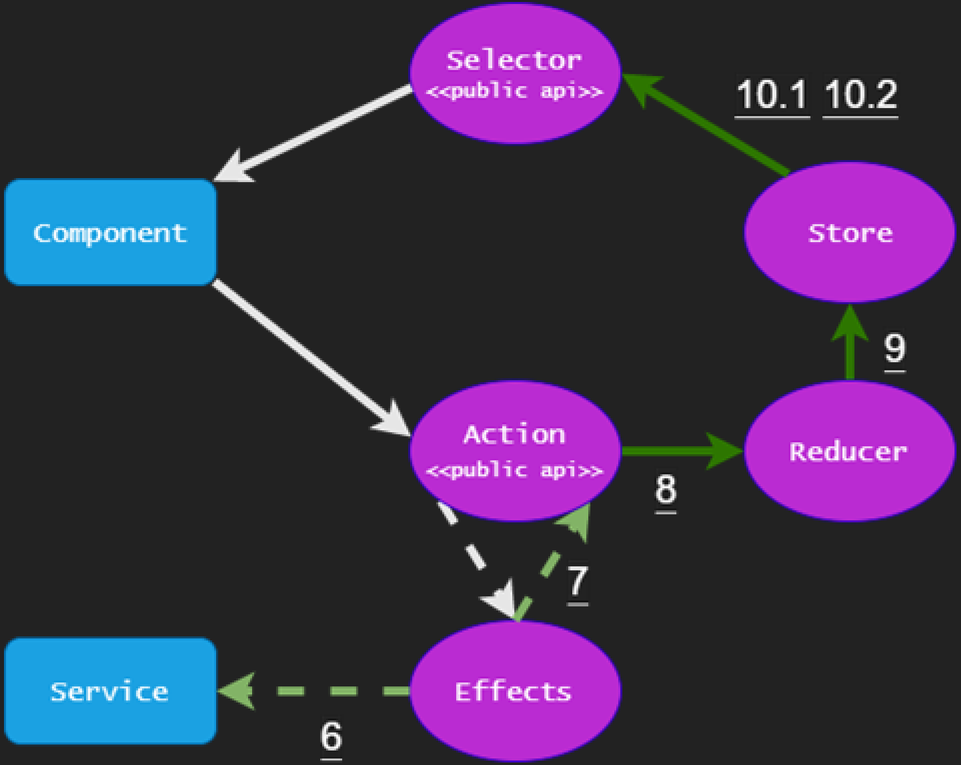
```
- [Users] Load
- [Users] Load Success
  {
    payload: [
      { id: 1, name: "a" },
      { id: 2, name: "b" },
    ]
  }
```

UI



Flow description

10.1 the store triggers the “users” selector
10.2 the store triggers the “loading” selector



NgRx - How do all parts fit together? - Example 10/10

Store

```
{
  items: [
    { id: 1, name: "a" },
    { id: 2, name: "b" },
  ],
  loading: false
}
```

Actions

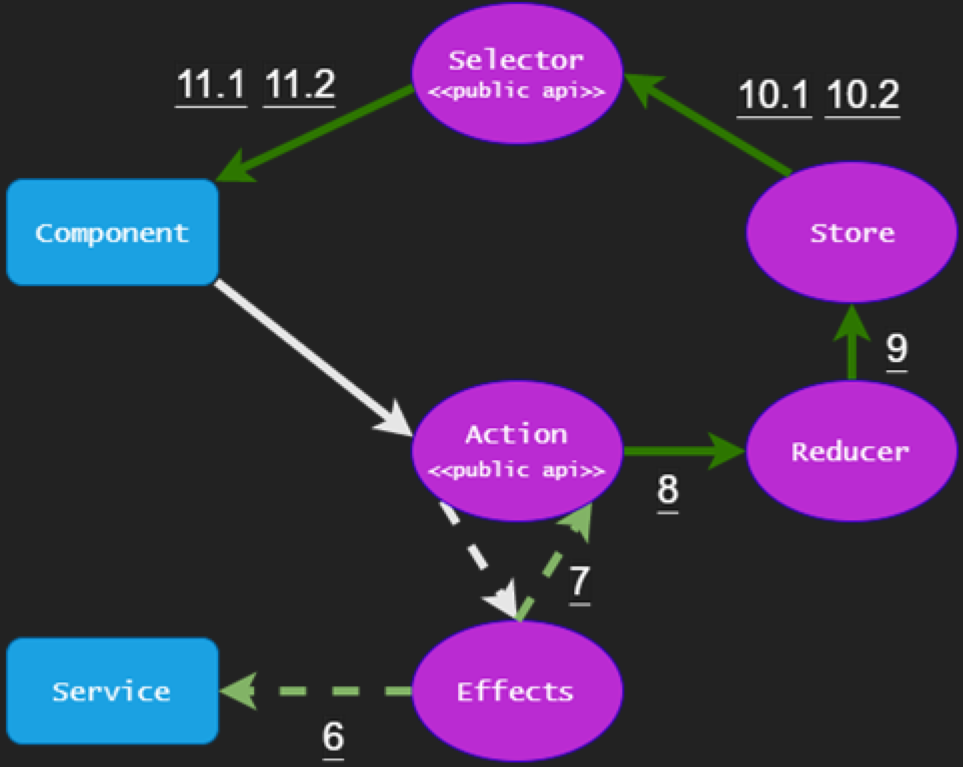
```
- [Users] Load
- [Users] Load Success
  {
    payload: [
      { id: 1, name: "a" },
      { id: 2, name: "b" },
    ]
  }
```

UI

id	name
1	a
2	b

Flow description

11.1 the "items" selector notifies the component and the component displays the users list
11.2 the "loading" selector notifies the component and the component hides the loading indicator



NgRx - Why use it? - Predictability

- **Structured changes** only actions cause state changes, only reducers update state, only selectors read state, only effects handle side-effects
- **No update race conditions** actions served in dispatch order
- **Unidirectional flow**

NgRx - Why use it? - Maintainability

- **Single source of truth**
- **Cleaner code** separation of concerns
- **Debugging tools** (store-devtools) actions trail, state diffs, time travel
- **Unit testing tools** provideMockStore, provideMockActions, selector projector()
- **Declarative style**

NgRx - Why use it? - Performance

- **Immutability + Angular OnPush** minimize redraws
- **Selector memoization and rendered models** country flag, is action available
- **Shared data among components**

NgRx - Why use it? - SHARI

- **Shared** state that is accessed by many components and services
- **Hydrated** state that is persisted and rehydrated from external storage
- **Available** state that needs to be available when re-entering routes
- **Retrieved** state that must be retrieved with a side-effect
- **Impacted** state that is impacted by actions from other sources

NgRx - Why not use it?

- **Framework complexity** a lot of steps for setup, boilerplate code
- **RxJS** prior knowledge is not required, but lack of can contribute to slower integration
- **Need fully object-oriented solution** selectors and reducers promote a functional paradigm

NgRx - Why do we use it?

- **Technology** Angular
- **Data sharing** multiple components need same piece of data
- **Effects** encapsulation of external interactions and business logic
- **Testable**

Resources

- **NgRx** ngrx.io
- **Redux** redux.js.org
- **Redux DevTools** extension.remotedev.io
- **Flux** facebook.github.io/flux
- **ReactiveX** reactivex.io
- **RxJS** rxjs-dev.firebaseapp.com
- **CQRS** martinfowler.com/bliki/CQRS.html, docs.microsoft.com/en-us/azure/architecture/patterns/cqrs
- **Event sourcing** martinfowler.com/eaaDev/EventSourcing.html, docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing

NgRx - Summary

- **A state management framework**
- **Predictable**
- **Provides clear separation of concerns** leads to cleaner application code
- **Integrates easily with Angular's OnPush strategy**
- **Testable**

NgRx - Credits

Credits to my very good friend and colleague Stratos Vetsos for his help on store setup and great usage insights during our .