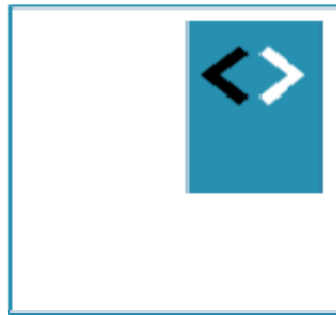


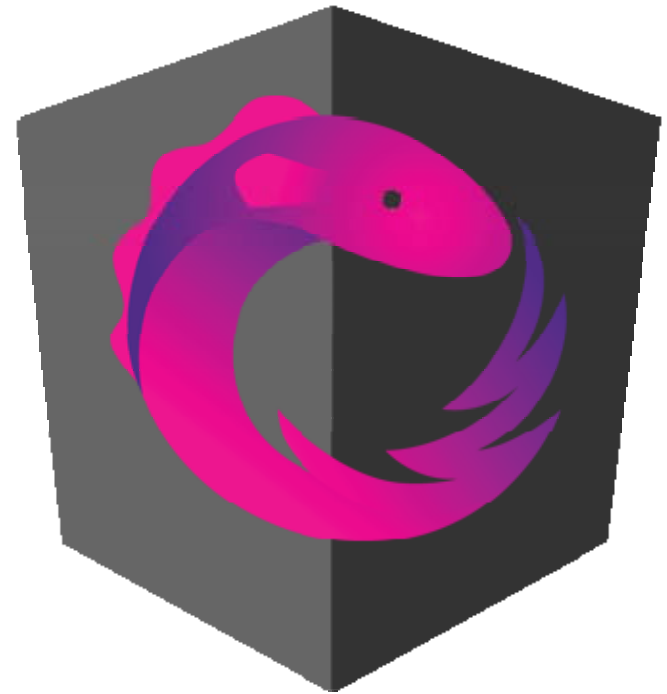
Angular Advanced State management with `@ngrx/store`



Peter Kassenaar –
info@kassenaar.com

What is State Management?

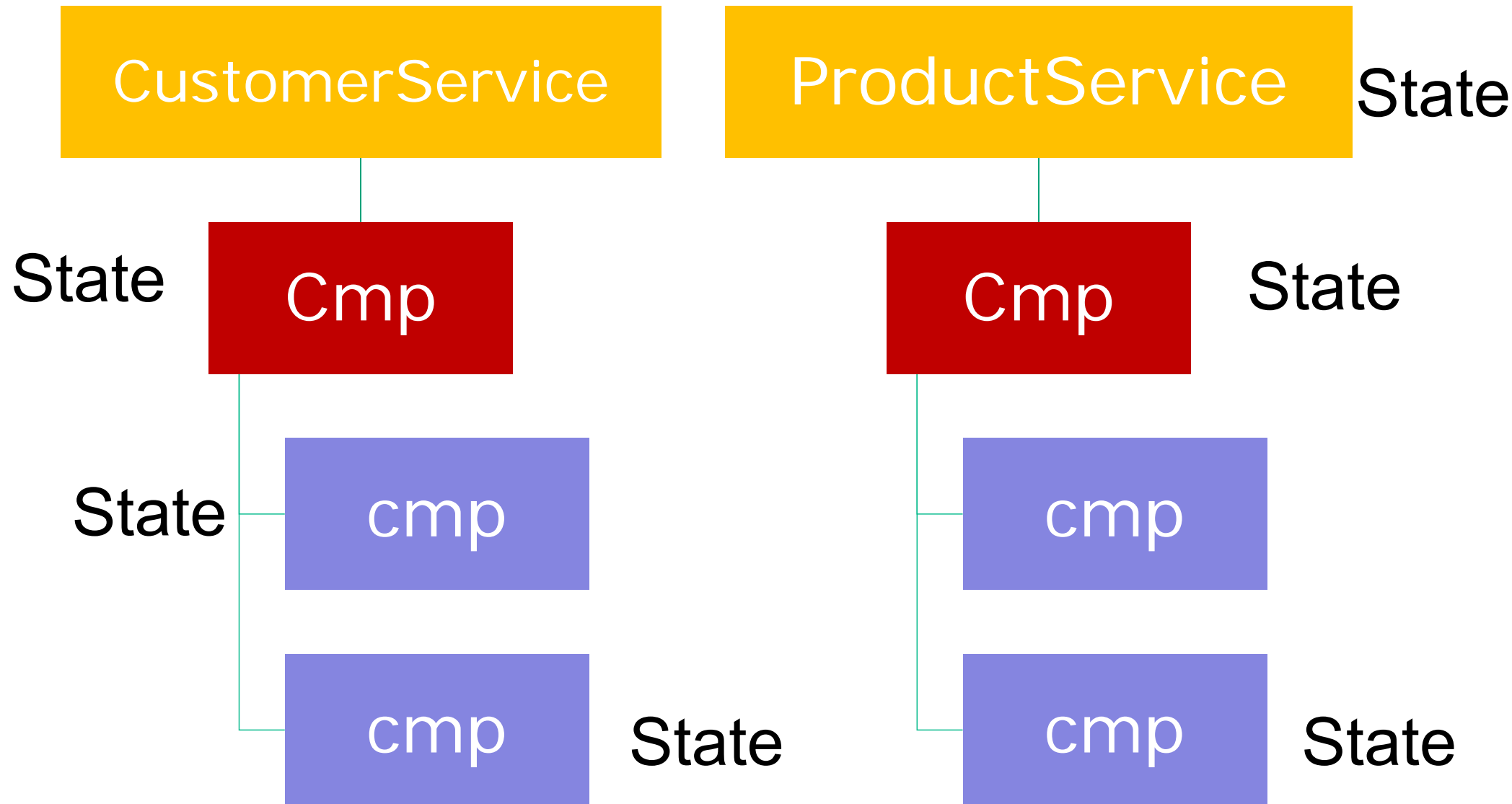
- Various design patterns, used for managing *state* (data in its broadest sense!) in your application.
- Multiple solutions possible – depends on application & framework



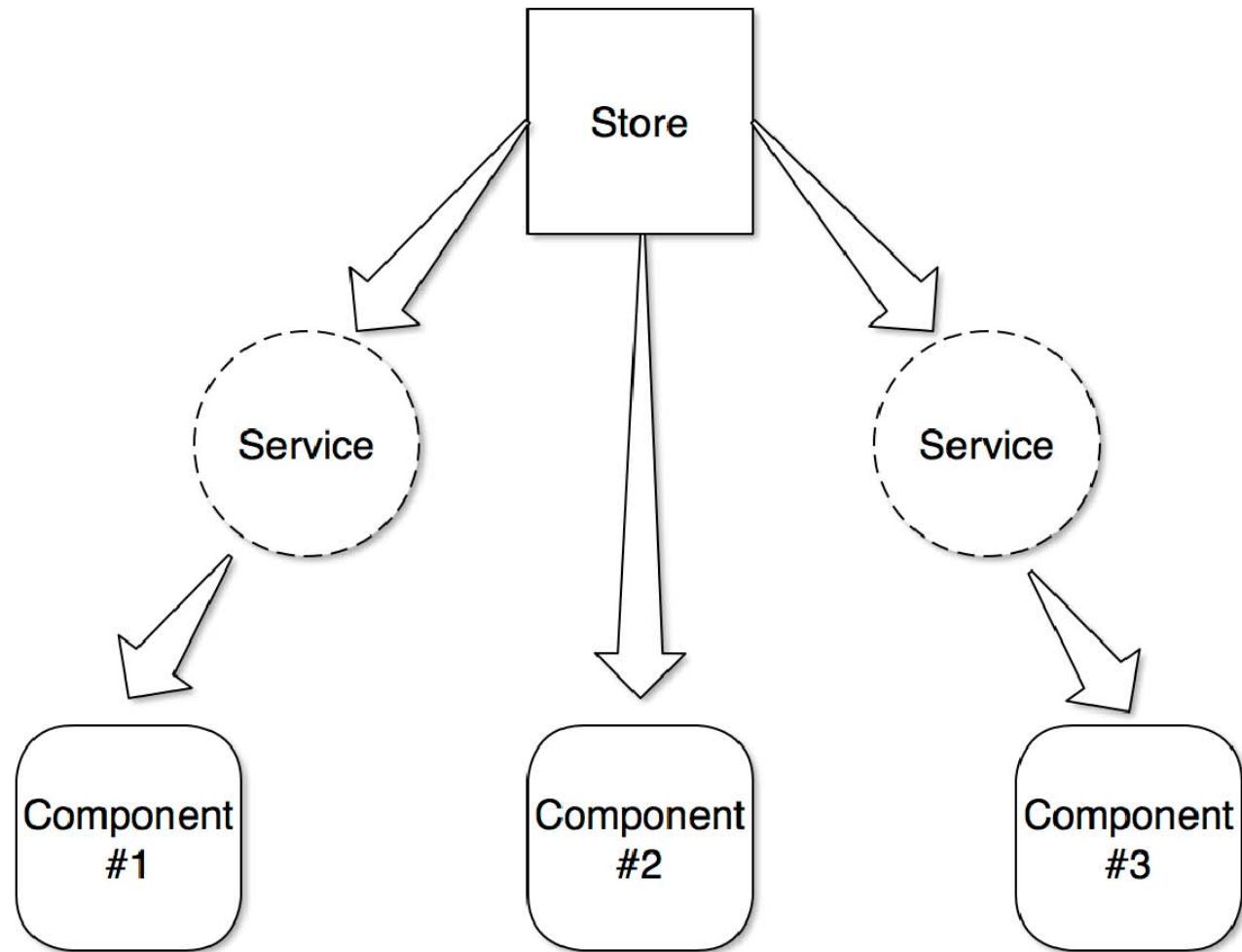
Benefits of using a store

- State is only changed in a controlled way
- Component state is also driven from the store
- Based on immutable objects – b/c they are predictable
- In Angular – immutability is fast
 - Because no changes can appear, no change detection is needed!
- Developer tools available to debug and see how the store changes over time
 - “Time travelling Developer tools”

State management without a store

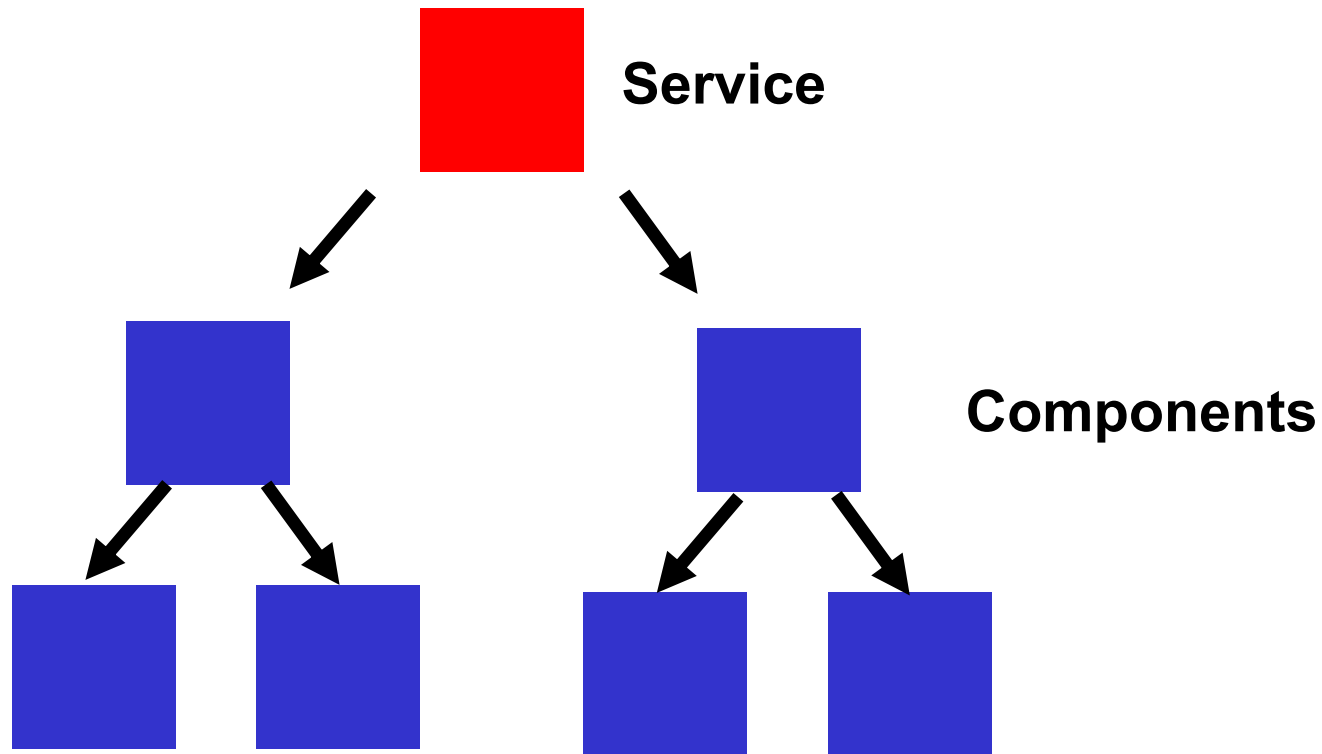


Store architecture - #1

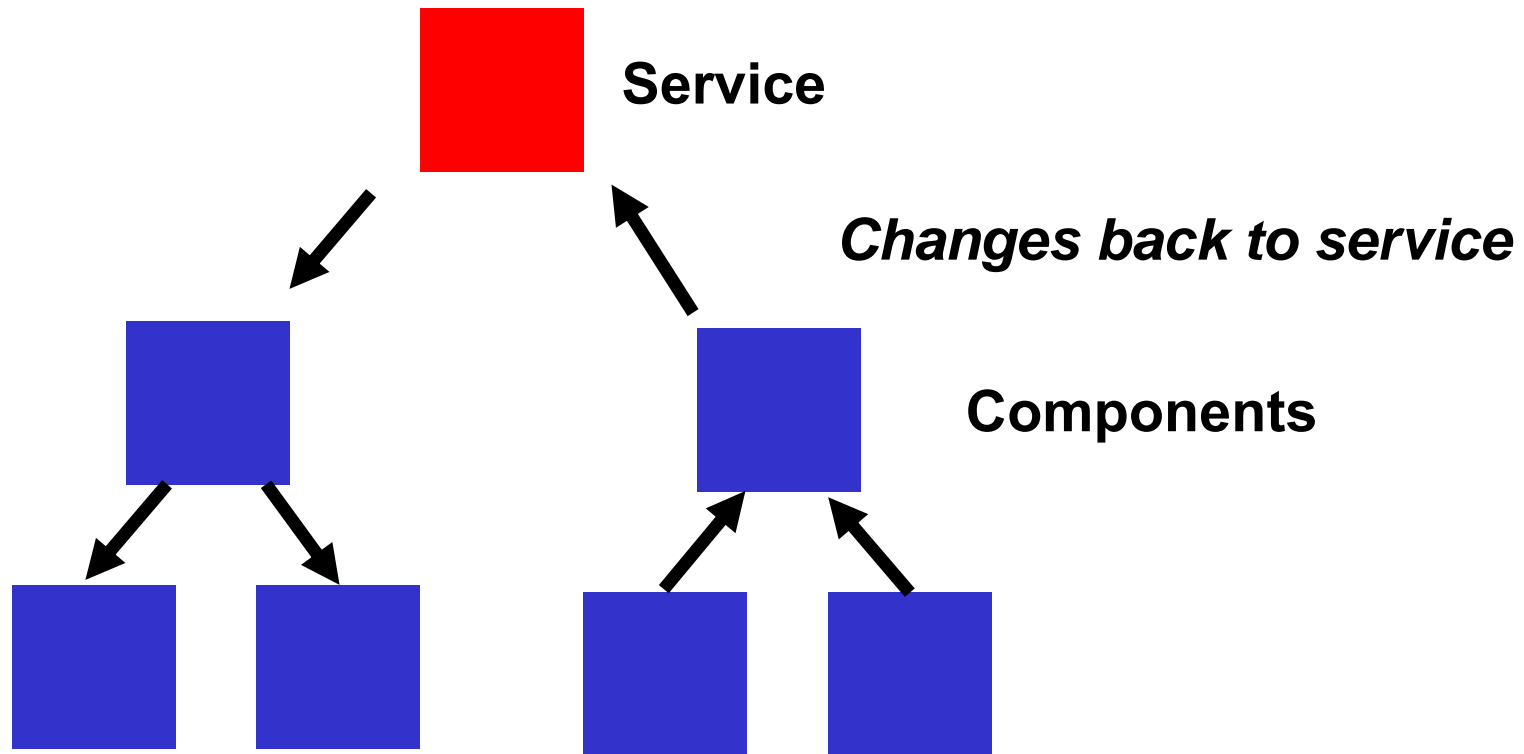


<https://gist.github.com/btroncone/a6e4347326749f938510>

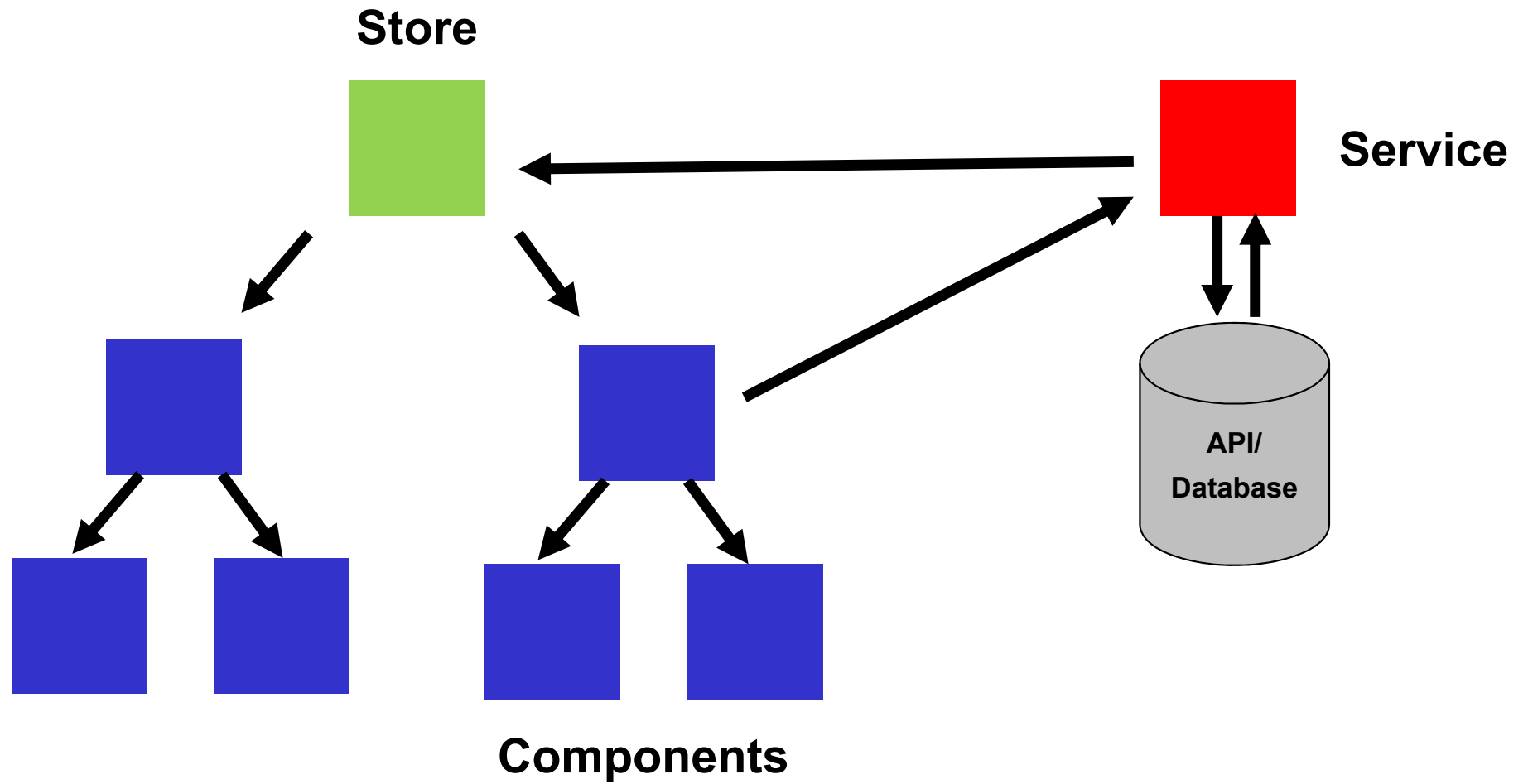
Store architecture - #2 - traditional



Store architecture - #2



Store architecture - #2 with a store



Angular State Management

- Simple applications - In the component
 - `counter : number = 0;`
 - `this.counter += 1;`
- Intermediate applications - In a service
 - `counter : number = 0;`
 - `this.counter = this.counterService.increment(1);`
 - Cache counter value in the service

- Larger applications - In a *data store* – all based on *observables*

```
counter$: Observable<number>;
```

```
constructor(private store: Store<AppState>){  
    this.counter$ = store.select('counter');  
}
```

```
increment(){  
    this.store.dispatch({ type: INCREMENT });  
}
```



Store Terminology and concepts

Important Store terminology / concepts

Store

"The store can be seen as your client side database. But more importantly, it reflects the state of your application.

You can see it as the single source of truth."

*"The store holds all the data. You modify it by dispatching **actions** to it."*

Reducer

“Reducers are functions that know what to do with a given action and the previous state of your app.

Reducers will take the previous state from your store and apply a pure function to it. From the result of that pure function, you will have a new state. The new state is put in the store.”

Actions

*“Actions are the payload that contains needed information to alter your store. Basically, an action has a **type** and a **payload** that your reducer function will take to alter the state.”*

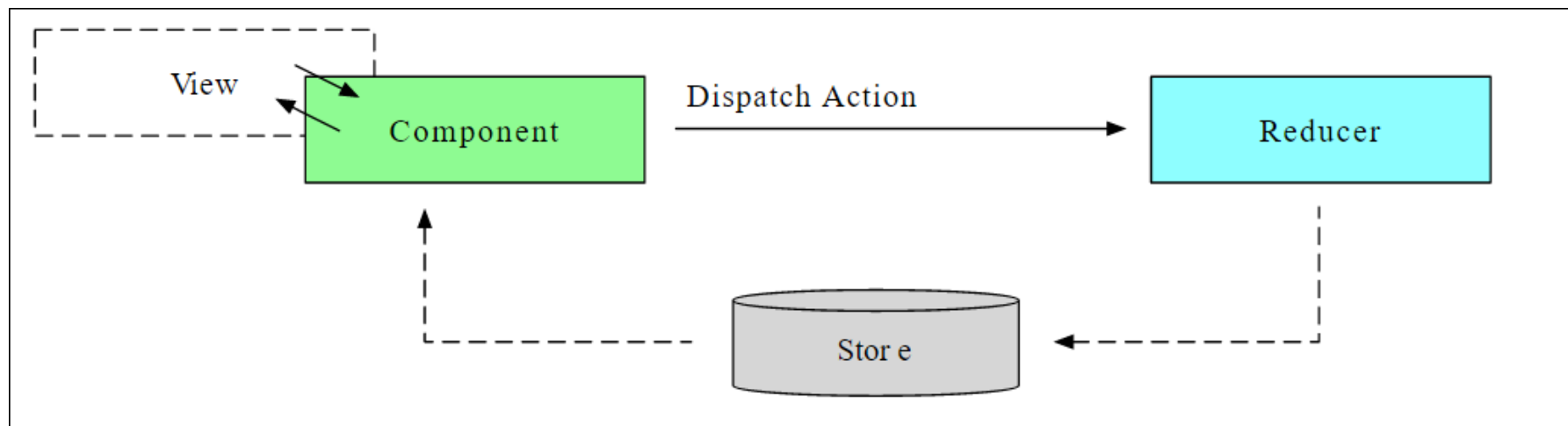
Dispatcher

"Dispatchers are simply an entry point for you to dispatch your action. In NgRx, there is a dispatch method directly on the store. I.e., you call `this.store.dispatch({...})`"

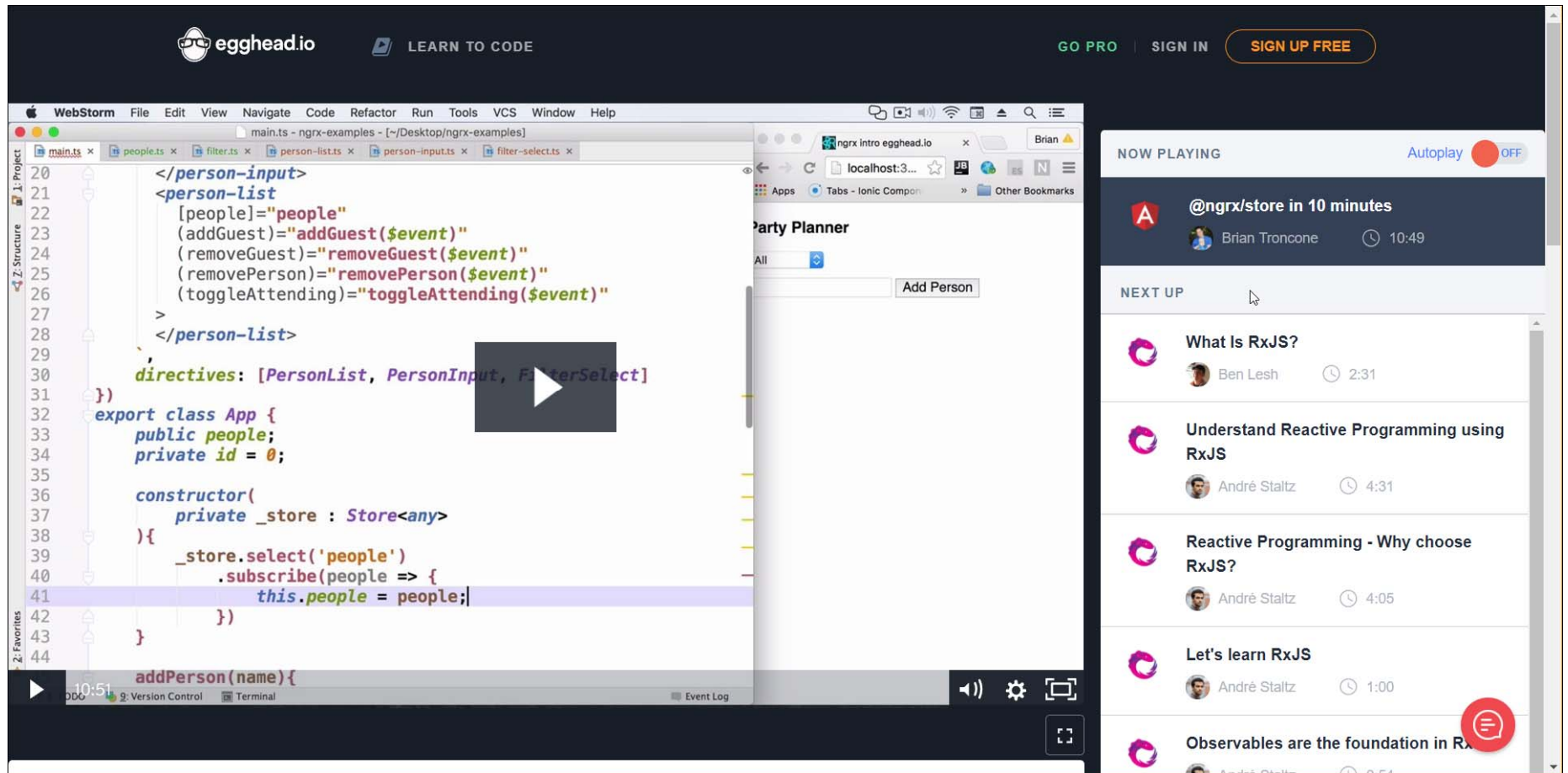
Reducers, Store and components

The **Component** first dispatch an Action. When the **Reducer** gets the Action, it will update the state(s) in the **Store**.

The Store has been injected to the Component, so the View will update based on the store state change (it is subscribed).



Store concepts in a video



The screenshot displays a video player interface for a tutorial on NgRx Store concepts. The video is titled "@ngrx/store in 10 minutes" by Brian Troncone, with a duration of 10:49. The main content area shows a code editor with TypeScript code for an Angular application using NgRx. The code includes a component with a list of people and a service that manages the state using the NgRx Store. A play button is overlaid on the code editor.

```
20 </person-input>
21 <person-list
22   [people]="people"
23   (addGuest)="addGuest($event)"
24   (removeGuest)="removeGuest($event)"
25   (removePerson)="removePerson($event)"
26   (toggleAttending)="toggleAttending($event)"
27 >
28 </person-list>
29
30 directives: [PersonList, PersonInput, FilterSelect]
31
32 export class App {
33   public people;
34   private id = 0;
35
36   constructor(
37     private _store : Store<any>
38   ){
39     _store.select('people')
40       .subscribe(people => {
41         this.people = people;
42       })
43   }
44
45   addPerson(name){
```

The video player interface includes a "NOW PLAYING" section with the video title and author, and a "NEXT UP" section with a list of recommended videos:

- What Is RxJS? (Ben Lesh, 2:31)
- Understand Reactive Programming using RxJS (André Staltz, 4:31)
- Reactive Programming - Why choose RxJS? (André Staltz, 4:05)
- Let's learn RxJS (André Staltz, 1:00)
- Observables are the foundation in Rx (André Staltz, 0:54)

<https://egghead.io/lessons/angular-2-ngrx-store-in-10-minutes>

Setting up @ngrx/store

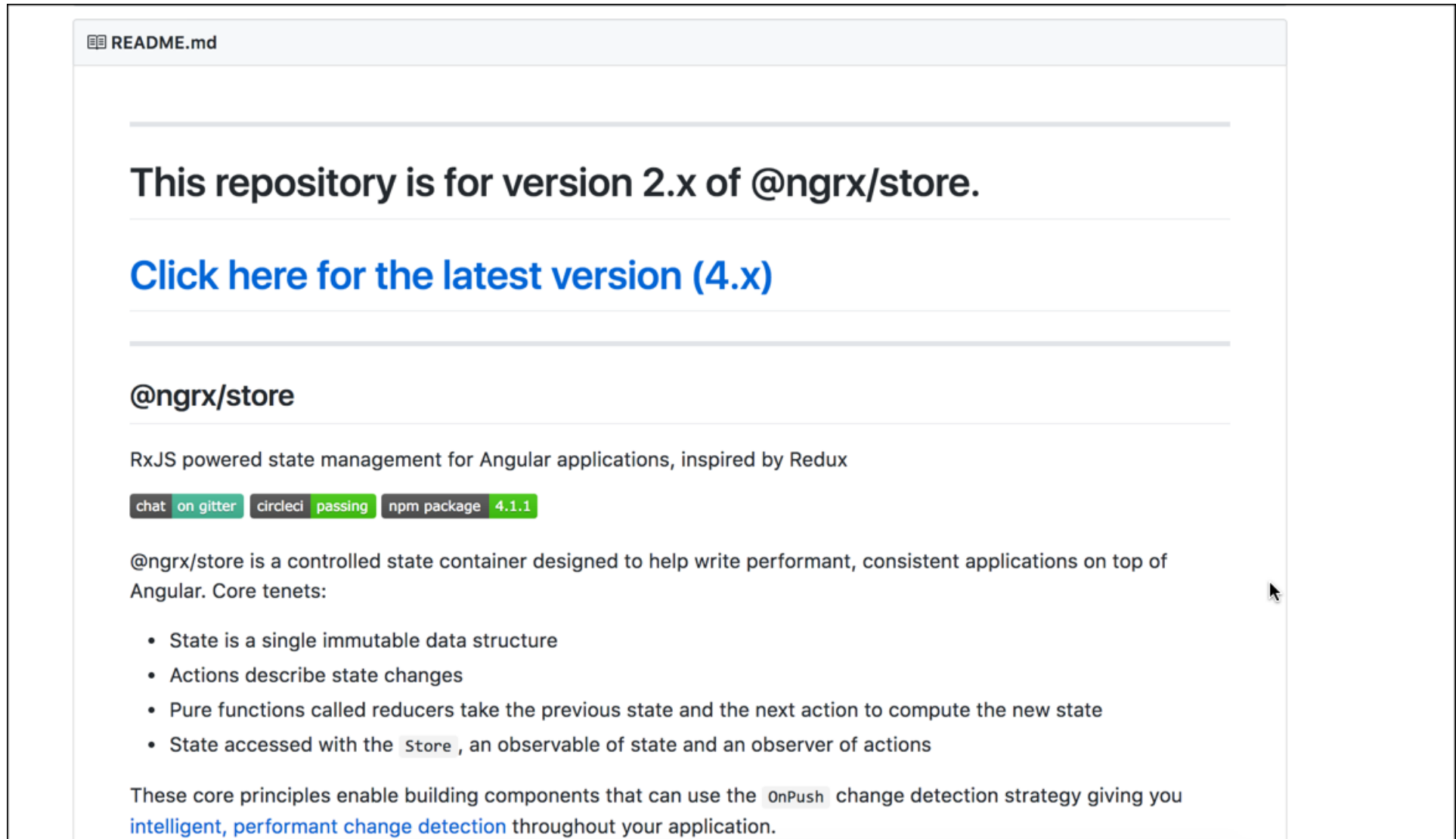
- Install core files & store files
- Create new project or add to existing project
- v2.x:

```
npm install @ngrx/core @ngrx/store --save
```

- V4.x +:

```
npm install @ngrx/store --save
```

Different versions, different docs...



The screenshot shows the README.md file for the @ngrx/store repository. It features a title, a link to the latest version, a description of the library, and a list of core tenets.

README.md

This repository is for version 2.x of @ngrx/store.

[Click here for the latest version \(4.x\)](#)

@ngrx/store

RxJS powered state management for Angular applications, inspired by Redux

chat on gitter circleci passing npm package 4.1.1

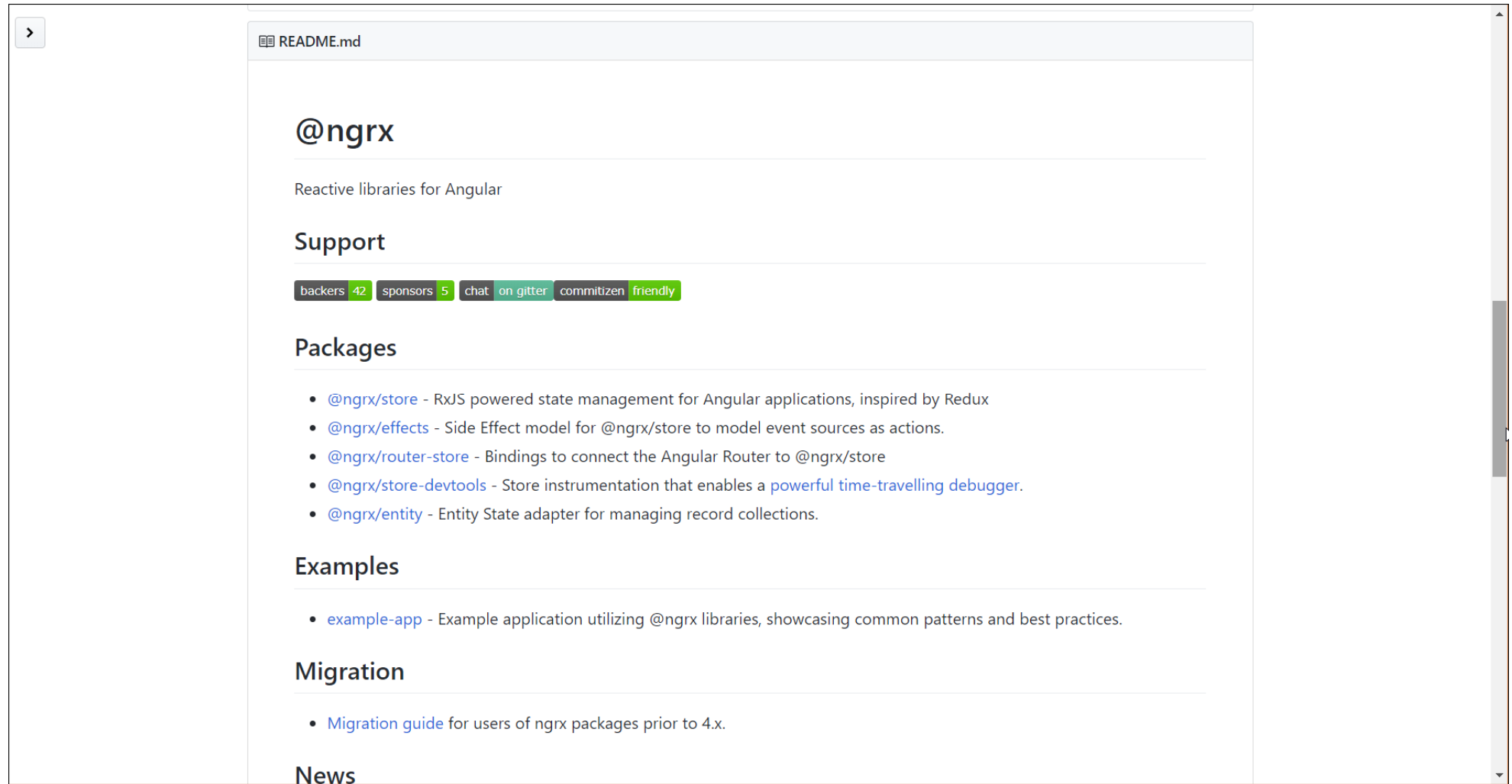
@ngrx/store is a controlled state container designed to help write performant, consistent applications on top of Angular. Core tenets:

- State is a single immutable data structure
- Actions describe state changes
- Pure functions called reducers take the previous state and the next action to compute the new state
- State accessed with the `Store`, an observable of state and an observer of actions

These core principles enable building components that can use the `OnPush` change detection strategy giving you [intelligent, performant change detection](#) throughout your application.

<https://github.com/ngrx/store>

Different versions, different docs...



<https://github.com/ngrx/platform>

Migration guide (but Google is more useful):

Dependencies

You need to have the latest versions of TypeScript and RxJS to use ngrx V4 libraries.

TypeScript 2.4.x

RxJS 5.4.x

@ngrx/core

@ngrx/core is no longer needed, and can conflict with @ngrx/store. You should remove it from your project.

BEFORE:

```
import { compose } from '@ngrx/core/compose';
```

AFTER:

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

@ngrx/store

Action interface

The `payload` property has been removed from the `Action` interface. It was a source of type-safety issues, especially when used with `@ngrx/effects`. If your interface/class has a payload, you need to provide the type.

<https://github.com/ngrx/platform/blob/master/MIGRATION.md>

1. Create your first reducer

- A reducer is simply an exported function with a name.
- It takes two parameters:
 - Current `state`, or otherwise empty object/initial state
 - `action`, of type `Action`
- `action` is of shape `{type : string , payload? : any}`.
- NO DEFAULT payload in `@ngrx/store >4.0.0`
- Switch on `action.type`
 - Later: use abstraction. For now – a simple string
 - Always provide a `default` action which returns the unaltered `state`.

New file – counter.ts

// counter.ts - a simple reducer

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

```
export function counterReducer(state: number = 0, action: Action) {  
  switch (action.type) {  
    case 'INCREMENT':  
      return state + 1;  
  
    case 'DECREMENT':  
      return state - 1;  
  
    case 'RESET':  
      return 0;  
  
    default:  
      return state;  
  }  
}
```

2. Adding store and reducer

- Register the state container with your application.
- Import reducers
- V.2.0.0: Use `StoreModule.forRoot()` to add it to the module

...

// Store stuff

import {StoreModule} from '@ngrx/store';

import {counterReducer} from '../reducers/counter';

@NgModule({

...,

imports : [

BrowserModule,

StoreModule.forRoot({counter: counterReducer})

],

...

})

export class AppModule {}

3. Using the Store Service

- Import and inject the `Store` service to components
- Create an interface (or class) for your `AppState`
- Initialize the store with `AppState` Type
- Use `store.select()` to select slice(s) of the state
- Add methods to call reducer functions
 - `increment()`
 - `decrement()`
 - `etc..`

```

import {Component} from '@angular/core';
import {Store} from '@ngrx/store';
import {Observable} from 'rxjs/Observable';

interface AppState {
  counter: number;
}

@Component({
  selector    : 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  counter$: Observable<number>;

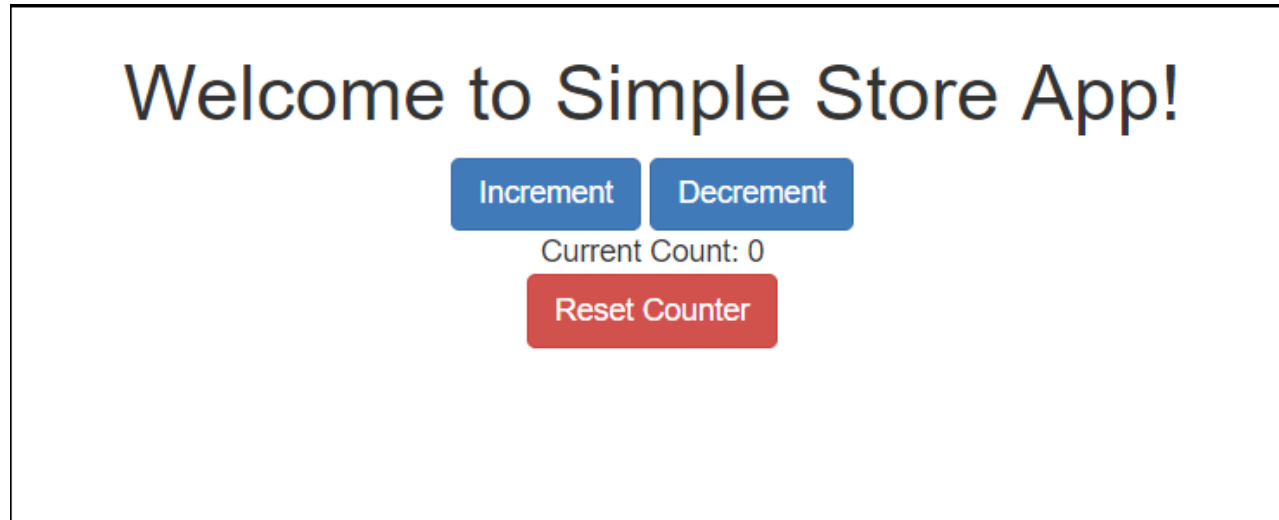
  constructor(private store: Store<AppState>) {}

  ngOnInit() {
    this.counter$ = this.store.select('counter')
  }

  increment() {
    this.store.dispatch({type: 'INCREMENT'})
  }
  ...
}

```

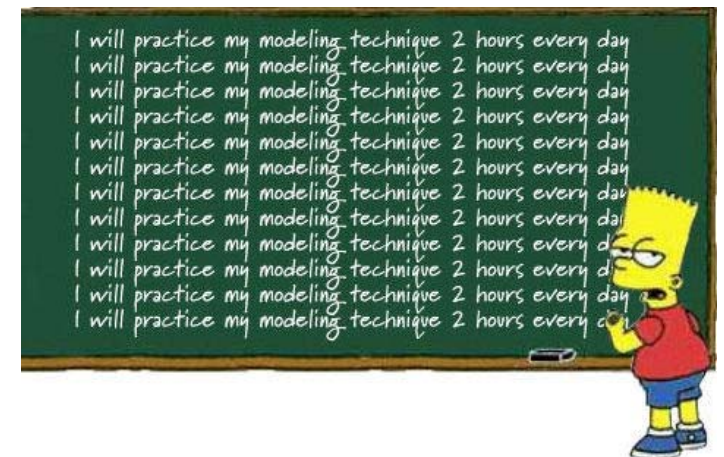
Next steps



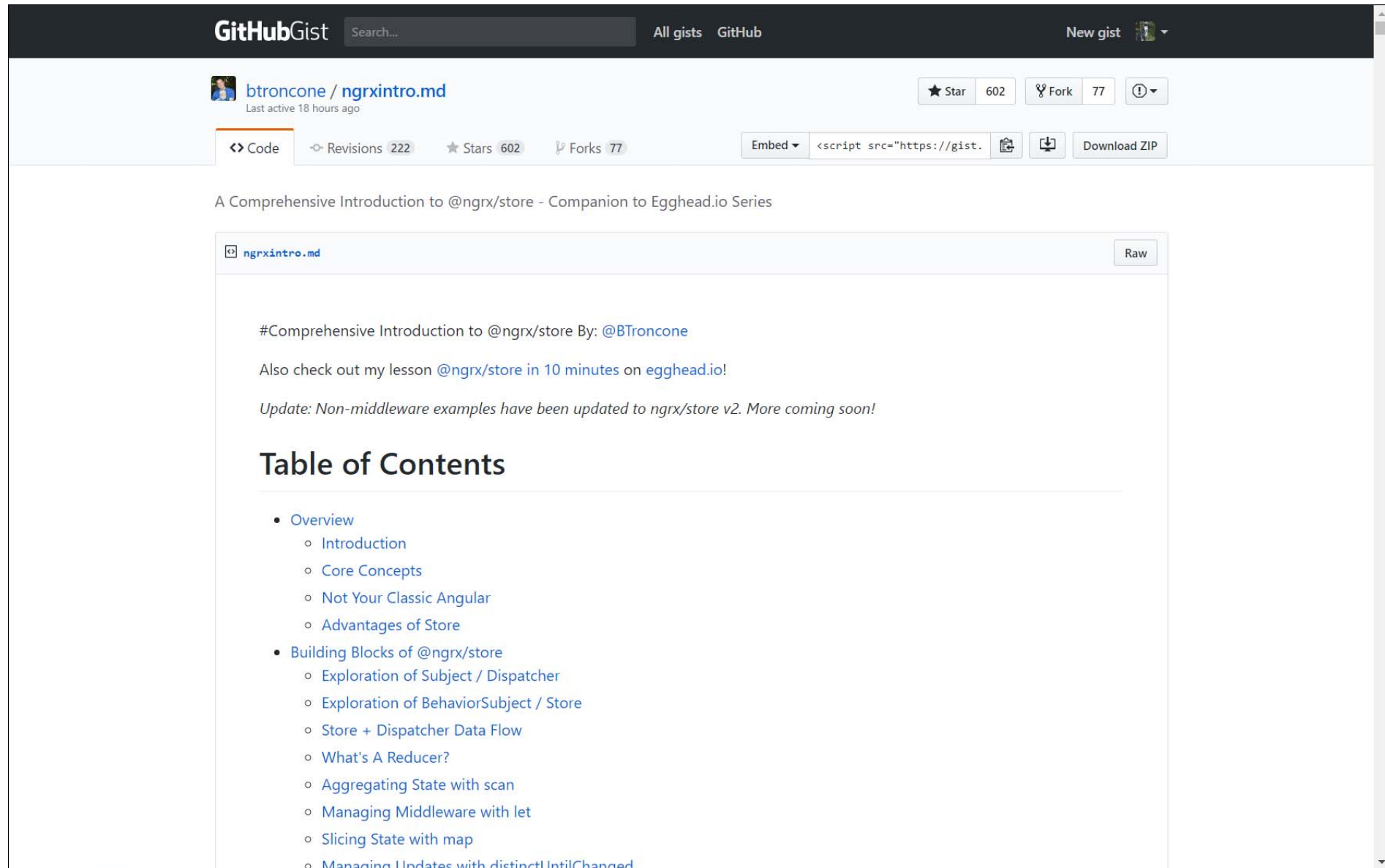
Add new components, subscribe to store,
enhance store, etc.

Workshop

- Create a new app, follow the previous steps to add a Store
- OR: Start from `../200-ngrx-simple-store`
- Make yourself familiar with the store concepts and data flow. Study the example code.
- Create some extra actions on the reducer. For example:
 - Add +5 with one click
 - Subtract -5 with one click
 - Reset counter to 0 if `counter >= 10;`



More info on @ngrx/store



The screenshot shows a GitHub Gist page for a file named `ngrxintro.md` by user `btroncone`. The page has 602 stars and 77 forks. The content of the gist is a comprehensive introduction to @ngrx/store, including a table of contents.

GitHubGist Search... All gists GitHub New gist

btroncone / ngrxintro.md
Last active 18 hours ago

★ Star 602 🍴 Fork 77 ⓘ

Code Revisions 222 Stars 602 Forks 77 Embed <script src="https://gist.> Download ZIP

A Comprehensive Introduction to @ngrx/store - Companion to Egghead.io Series

`ngrxintro.md` Raw

#Comprehensive Introduction to @ngrx/store By: @BTroncone

Also check out my lesson @ngrx/store in 10 minutes on egghead.io!

Update: Non-middleware examples have been updated to ngrx/store v2. More coming soon!

Table of Contents

- Overview
 - Introduction
 - Core Concepts
 - Not Your Classic Angular
 - Advantages of Store
- Building Blocks of @ngrx/store
 - Exploration of Subject / Dispatcher
 - Exploration of BehaviorSubject / Store
 - Store + Dispatcher Data Flow
 - What's A Reducer?
 - Aggregating State with scan
 - Managing Middleware with let
 - Slicing State with map
 - Managing Updates with distinctUntilChanged

<https://gist.github.com/btroncone/a6e4347326749f938510>

Gitbooks

Overview

Create our first app

Add Component

Component 101

Add ChatComponent

Add Pipe

Add Form

Add Service

Update ChatComponent

Add ChatService

Smart and Dumb Component

ChangeDetectionStrategy

Add Router

Add Routes

Add ProfileComponent and Ro...

Finish the app

Part III RxJS 5

Part IV Add ngRx

Add ngRx/store

Add ngRx/effects

Finish the rest part

Part V Add REST

Part VI Update to Apollo

Published with GitBook

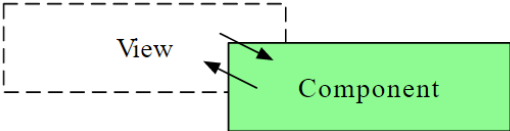
A

Add ngRx/store

Dataflow

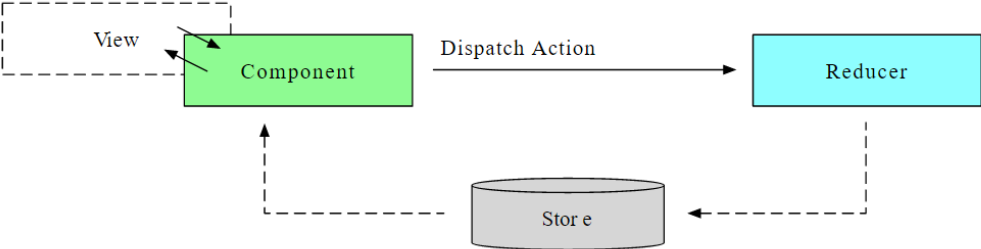
ngRx/store is RxJS powered state management for Angular 2 apps, inspired by Redux. It helps us manage all states in one place (Store).

Without ngRx/store, the **Component** updates the View directly.



With ngRx/store, the dataflow looks like this. The **Component** first dispatch an Action. When the **Reducer** gets the Action, it will update the states in the **Store**.

The Store has been injected to the Component, so the View will update based on the store state change.



Add ngRx/store

<https://hongbo-miao.gitbooks.io/angular2-server/content/part4/add-ngrx-store.html>

OneHungryMind – Lukas Ruebbelke



ABOUT

Build a Better Angular 2 Application with Redux and ngrx


👤 Lukas Ruebbelke




The Evolution of Angular State Management






<http://onehungrymind.com/build-better-angular-2-application-redux-ngrx/>

Dzone article

 **DZone** / Web Dev Zone


Over a million developers have joined DZone. [Sign In](#) / [Join](#) 






[REFCARDZ](#) [GUIDES](#) [ZONES](#) | [AGILE](#) [BIG DATA](#) [CLOUD](#) [DATABASE](#) [DEVOPS](#) [INTEGRATION](#) [IOT](#) [JAVA](#) [MOBILE](#) [PERFORMANCE](#) [SECURITY](#) [WEB DEV](#)

Managing State in Angular with ngrx/store

In this article, we'll explore managing state with an immutable data store in an Angular application using ngrx/store: reactive Redux for Angular.

 by Kim Maida 兕 MVB · Mar. 15, 17 · Web Dev Zone


 Like (3)  Comment (1)  Save  Tweet  4,711 Views

Join the DZone community and get the full member experience. [JOIN FOR FREE](#)

Learn how to build modern digital experience apps with Crafter CMS. Download this eBook now. Brought to you in partnership with [Crafter Software](#).

Managing State in Angular Apps

State management in large, complex applications has been a headache plaguing AngularJS

Subscribe 

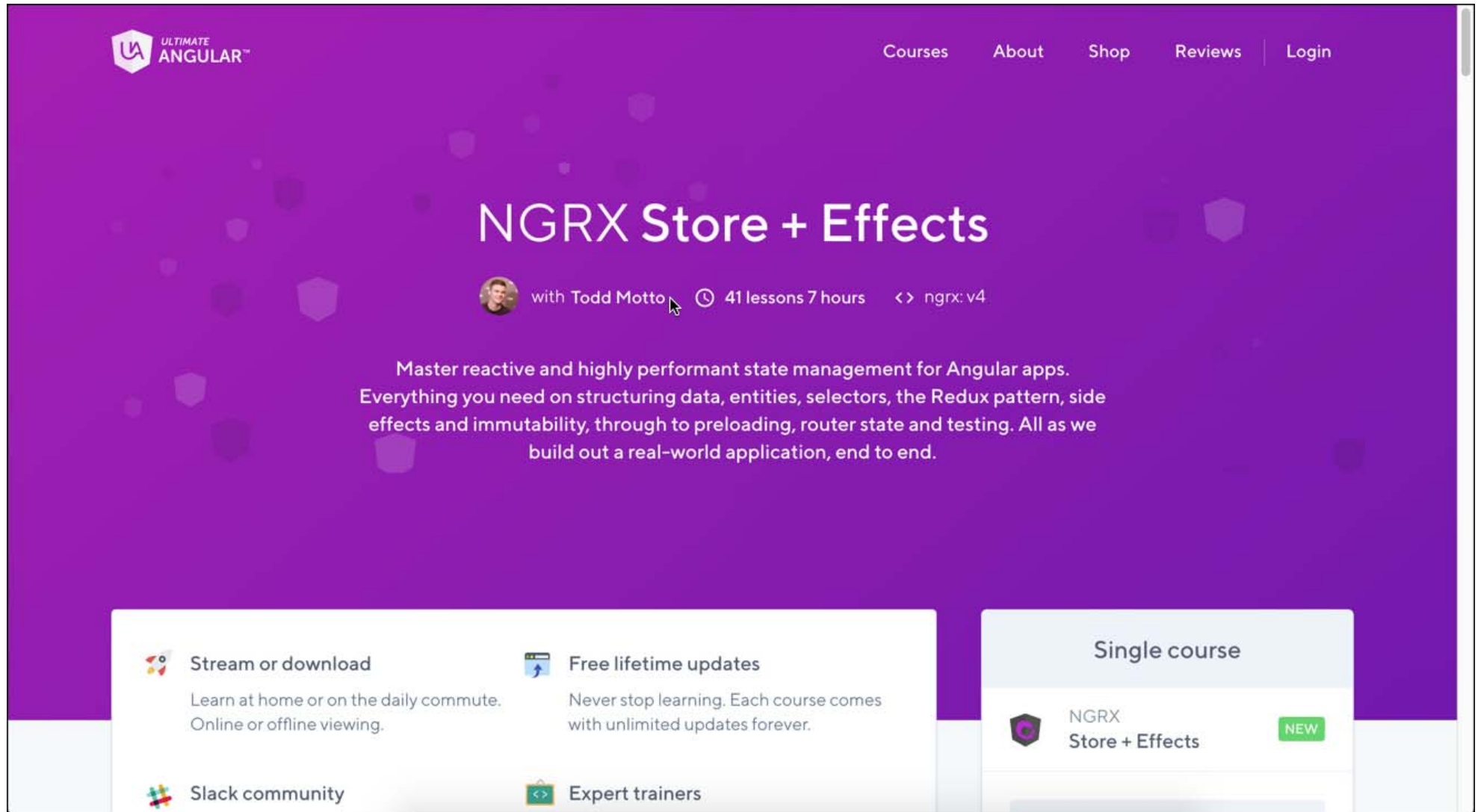
<https://dzone.com/articles/managing-state-in-angular-with-ngrxstore>

Rob Wormald - co-created @ngrx/store



<https://www.youtube.com/watch?v=mhA7zZ23Odw> - and more

NEW! As per Dec. 2017



The screenshot shows the 'Ultimate Angular' website with a purple background. The top navigation bar includes links for 'Courses', 'About', 'Shop', 'Reviews', and 'Login'. The main heading is 'NGRX Store + Effects'. Below it, it says 'with Todd Motto' next to a profile picture, followed by '41 lessons 7 hours' and 'ngrx: v4'. A descriptive paragraph states: 'Master reactive and highly performant state management for Angular apps. Everything you need on structuring data, entities, selectors, the Redux pattern, side effects and immutability, through to preloading, router state and testing. All as we build out a real-world application, end to end.' At the bottom, there are four features: 'Stream or download' (Learn at home or on the daily commute. Online or offline viewing.), 'Free lifetime updates' (Never stop learning. Each course comes with unlimited updates forever.), 'Slack community', and 'Expert trainers'. On the right, a 'Single course' section lists 'NGRX Store + Effects' with a 'NEW' badge.





ULTIMATE ANGULAR™

Courses About Shop Reviews Login


NGRX Store + Effects

with Todd Motto 41 lessons 7 hours <> ngrx: v4

Master reactive and highly performant state management for Angular apps. Everything you need on structuring data, entities, selectors, the Redux pattern, side effects and immutability, through to preloading, router state and testing. All as we build out a real-world application, end to end.

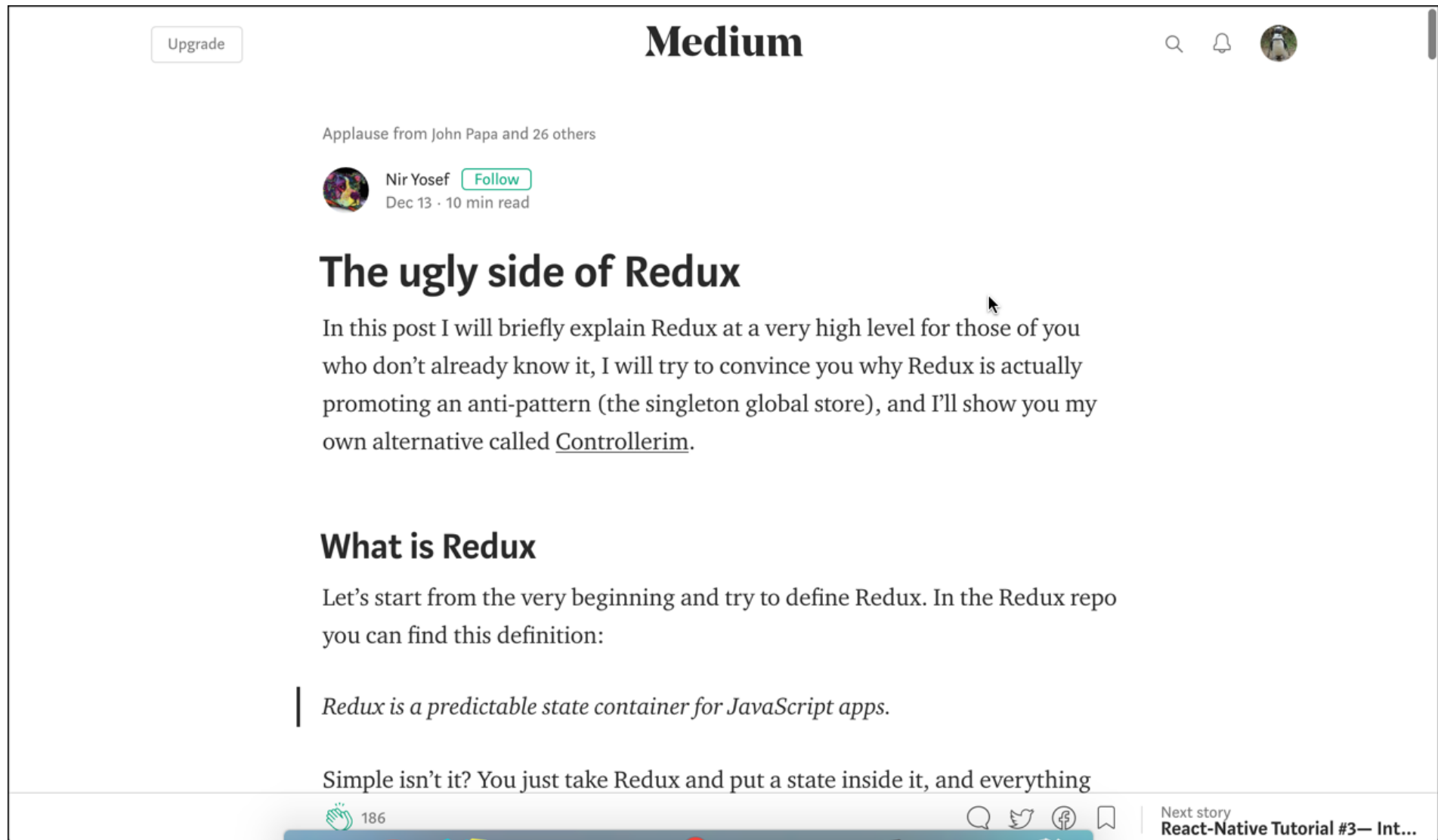
-  **Stream or download**
Learn at home or on the daily commute. Online or offline viewing.
-  **Free lifetime updates**
Never stop learning. Each course comes with unlimited updates forever.
-  **Slack community**
-  **Expert trainers**

Single course

 **NGRX Store + Effects** **NEW**

<https://ultimateangular.com/ngrx-store-effects>

Think about this – “The Ugly side of Redux”



The screenshot shows a Medium article page. At the top, there's a navigation bar with the Medium logo, a search icon, a notification bell, and a user profile picture. Below the navigation bar, there's a section for the article's author and engagement. The author is Nir Yosef, with a 'Follow' button. The article title is 'The ugly side of Redux'. The first paragraph explains that the post will discuss Redux at a high level, its anti-patterns, and the author's alternative, 'Controllerim'. The second section is titled 'What is Redux' and starts with a definition from the Redux repo. The article is partially visible, showing the beginning of a quote about Redux being a predictable state container.

Upgrade

Medium

Applause from John Papa and 26 others

Nir Yosef [Follow](#)

Dec 13 · 10 min read

The ugly side of Redux

In this post I will briefly explain Redux at a very high level for those of you who don't already know it, I will try to convince you why Redux is actually promoting an anti-pattern (the singleton global store), and I'll show you my own alternative called Controllerim.

What is Redux

Let's start from the very beginning and try to define Redux. In the Redux repo you can find this definition:

Redux is a predictable state container for JavaScript apps.

Simple isn't it? You just take Redux and put a state inside it, and everything

186

Next story
React-Native Tutorial #3— Int...

<https://medium.com/@niryo/the-ugly-side-of-redux-6591fde68200>