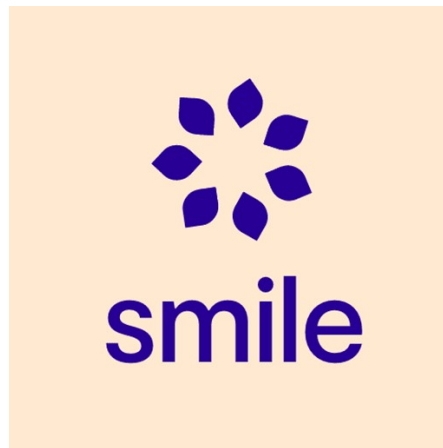




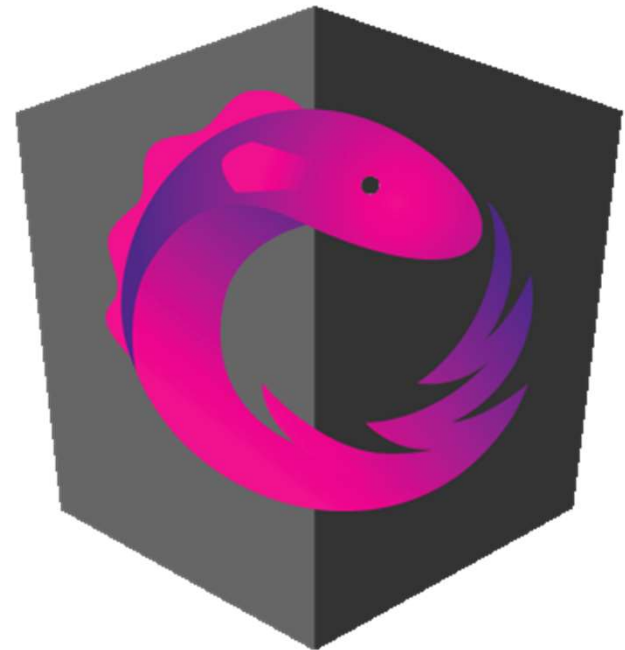
Angular Advanced State management and @ngrx/store



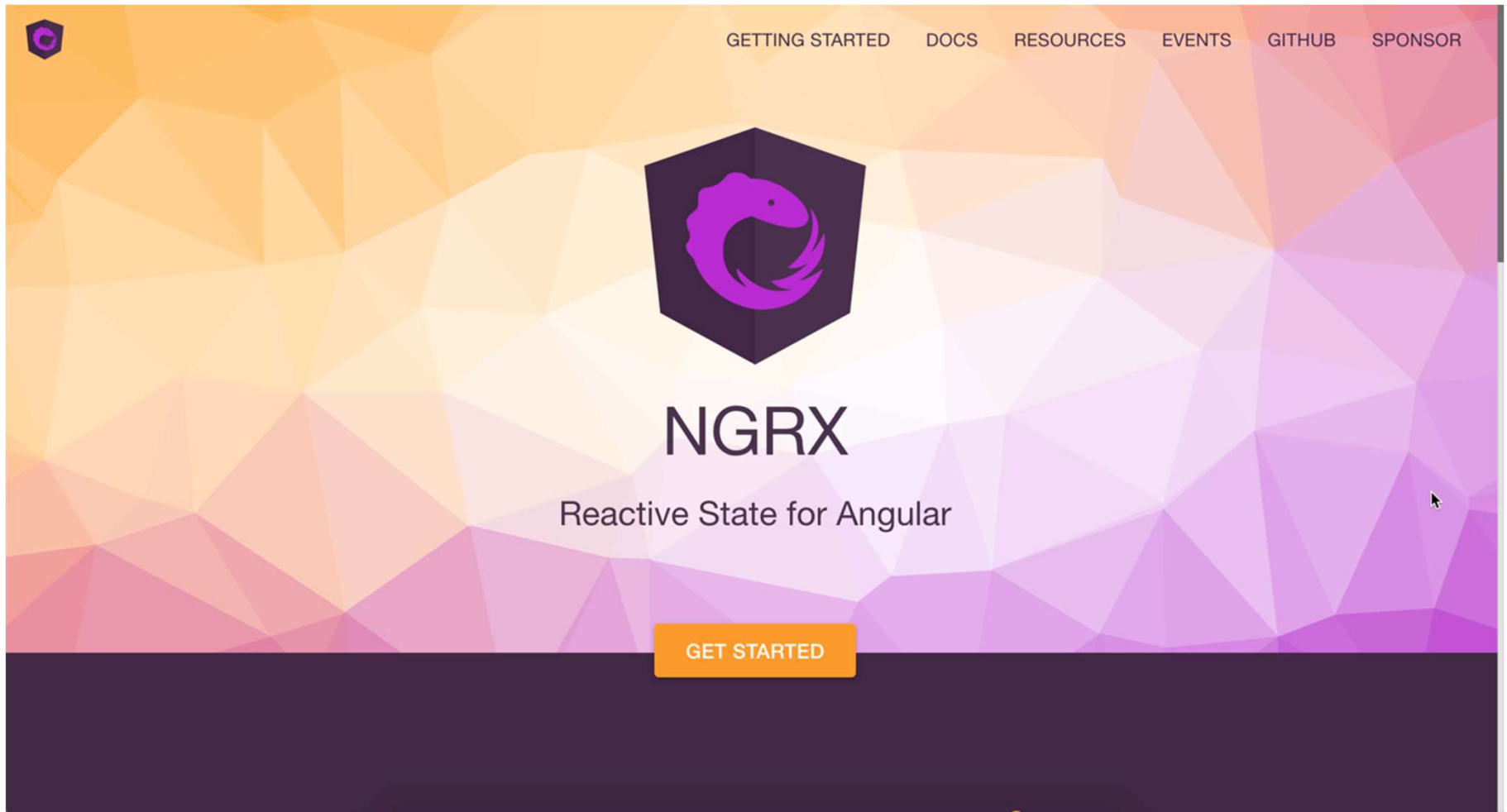
Peter Kassenaar
info@kassenaar.com

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



<https://ngrx.io/>

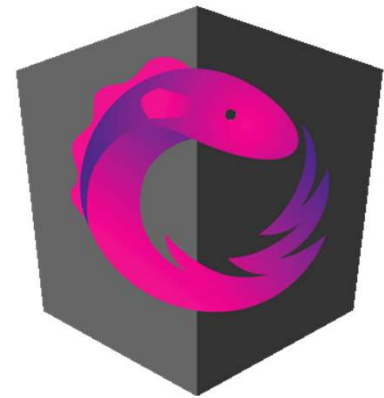


What is ngrx?

*"NgRx provides **reactive state management** for Angular apps inspired by Redux"*


@ngrx/store – 3 generations




- **Generation #1** – Angular 2
 - Creator: Rob Wormald
 - Simple implementation, (almost) all hand coded
- **Generation #2** – Angular 4-7
 - Action Creators, custom payload
 - `@Effects`
- **Generation #3** – Angular 8+
 - `createAction()`, `createReducer()` and more
 - (they try to make it) less complex...
 - ...if you know the principles and where to look




Maybe you don't need a store...

- <https://medium.com/@rmcavin/my-favorite-state-management-technique-in-angular-rxjs-behavior-subjects-49f18daa31a7>


 Javascript

  Upgrade 

My favorite state management technique in Angular — RxJS Behavior Subjects

 Rachel Cavin [Follow](#)
Dec 5, 2018 · 4 min read

Most of the apps I build in Angular are fairly small, we build many small front end apps instead of a few larger ones. Historically, my team and I had always just relied on the standard input/emitter Angular way of component interaction, which worked well most of the time but could lead to the occasional excessive passing between sibling components. We had looked into NgRx and other flux implementations but they felt a bit overkill for the size of our applications. Recently, we discovered the solution to our state management needs—the RxJS Behavior Subject!



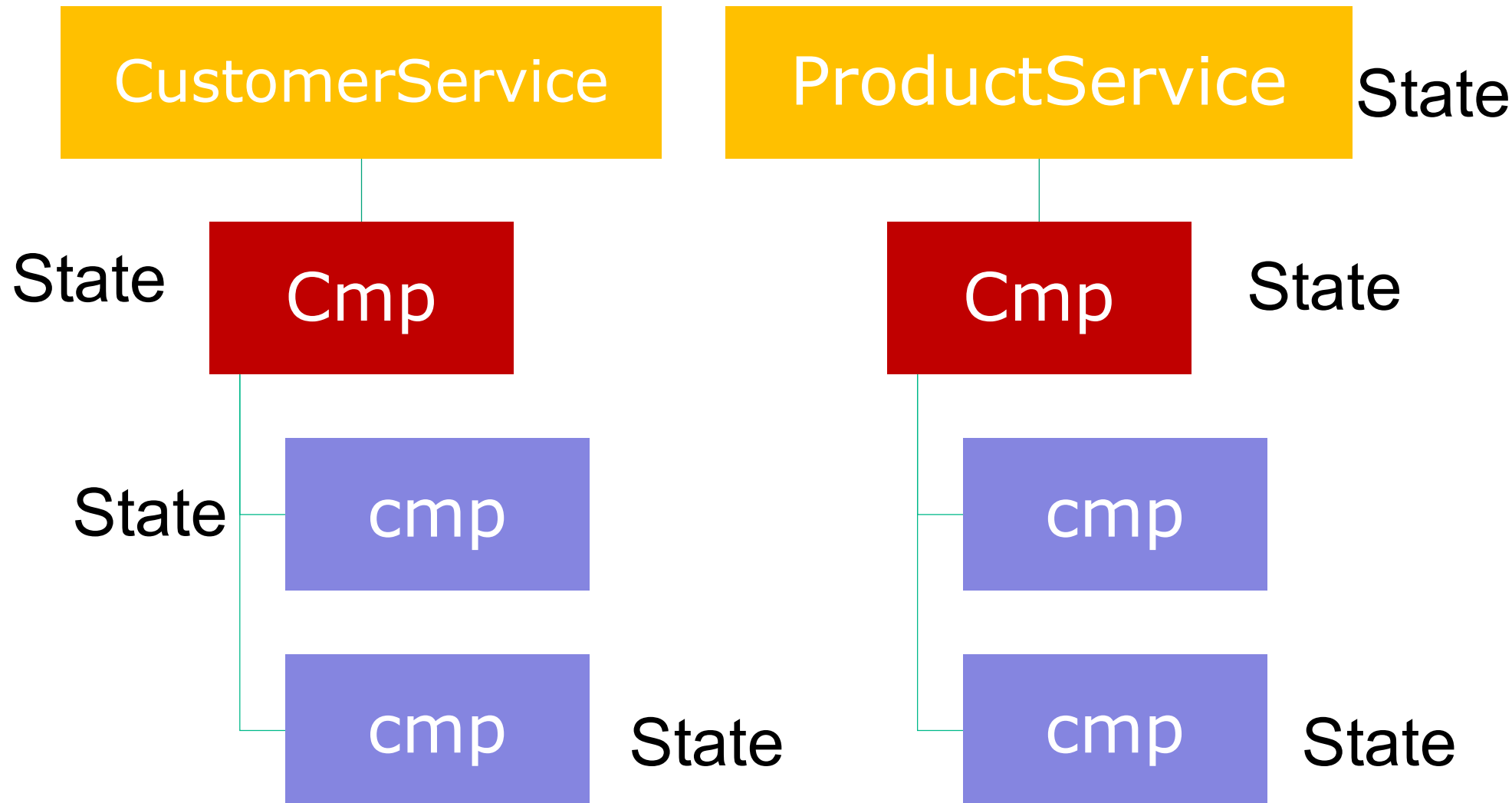
Behavior subjects are similar to regular subjects in RxJS, except



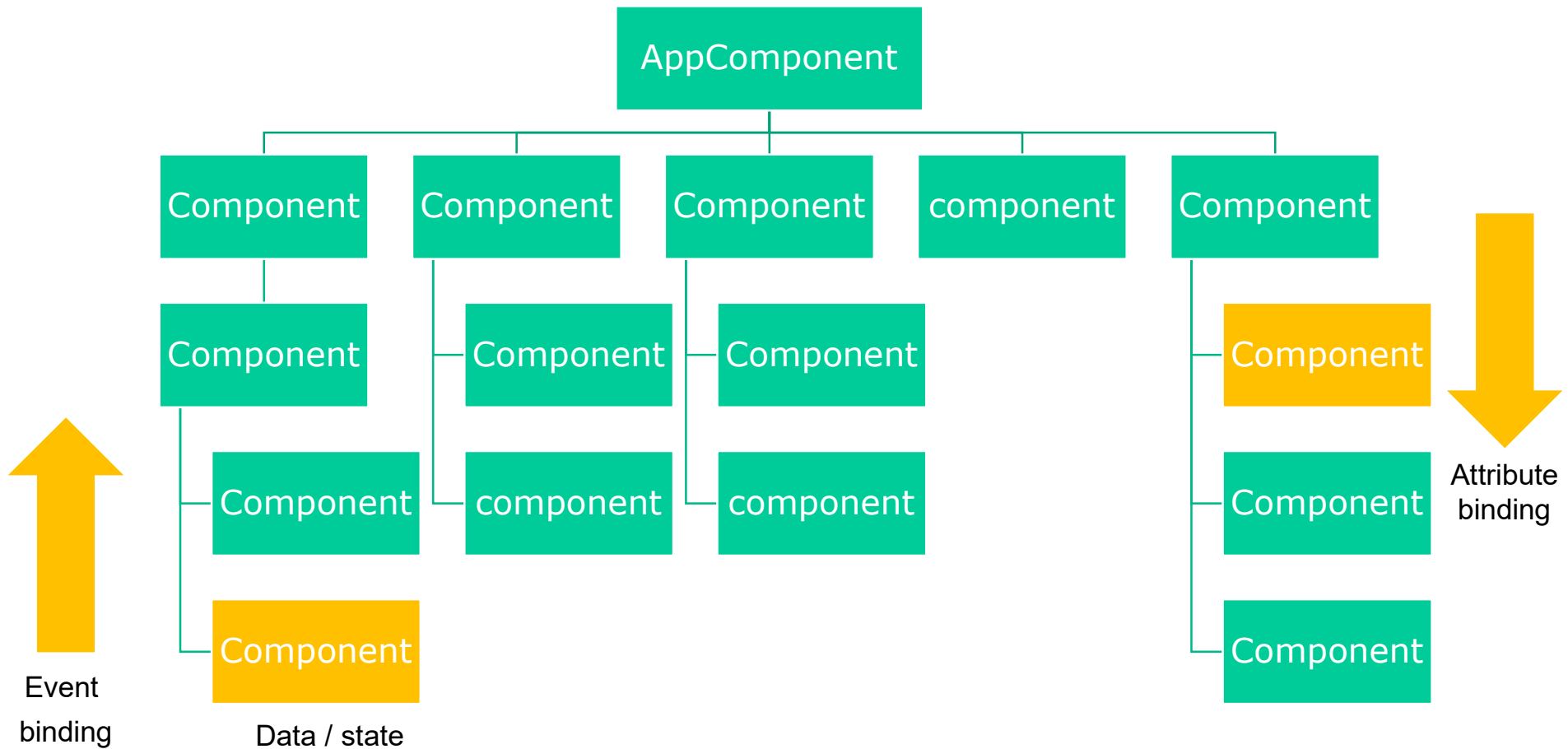
Why state management?

Why on earth would you need/want a Store ?

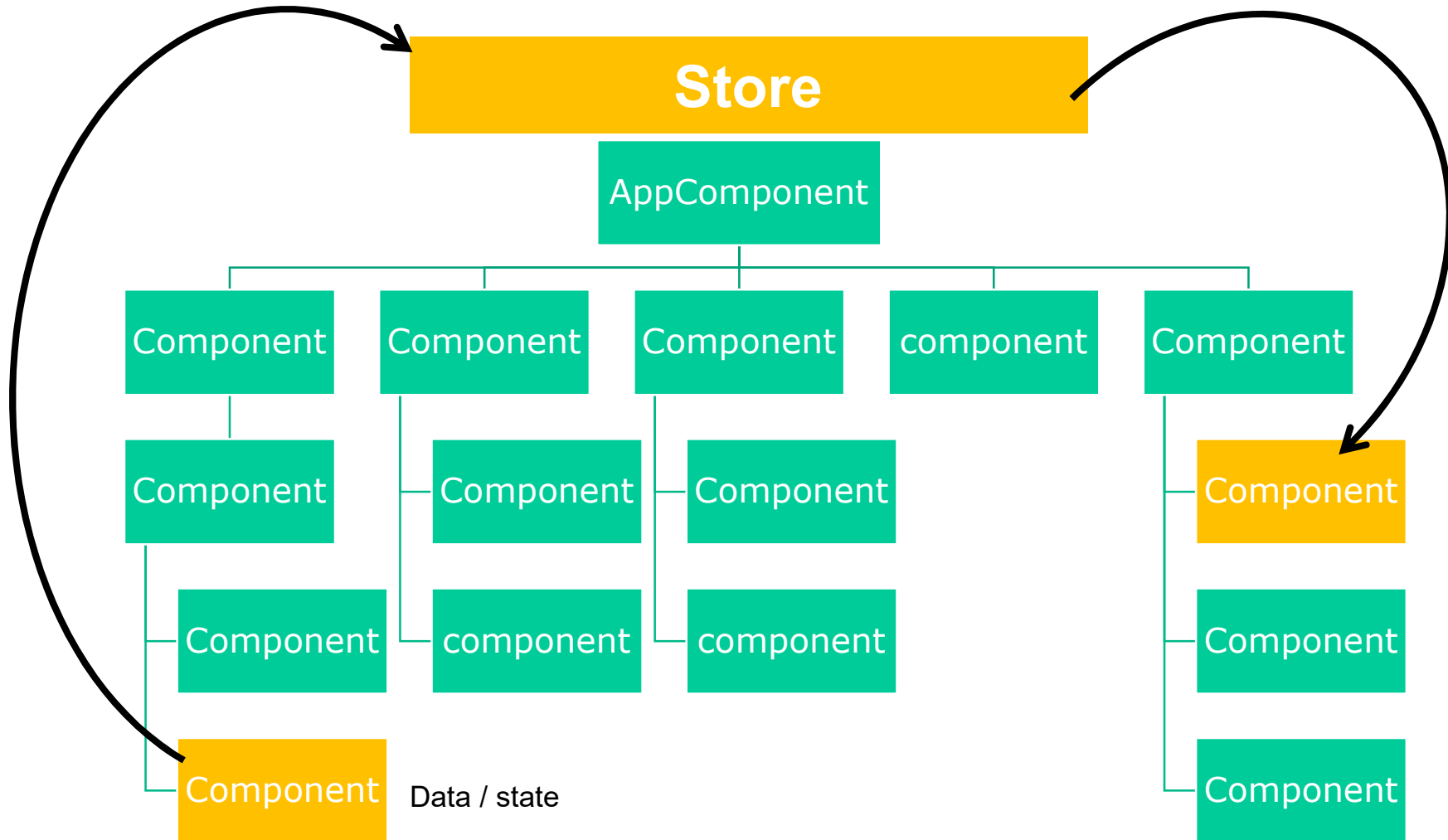
State management **without** a store



Data flow in complex applications



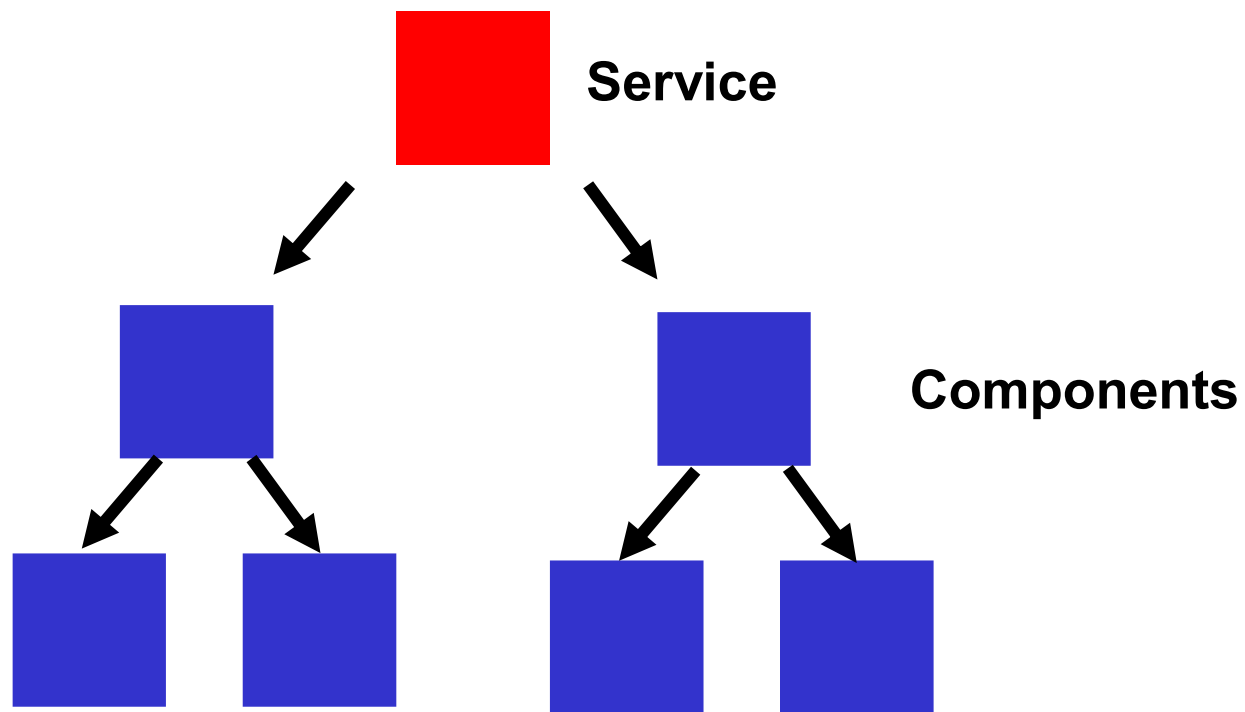
State management *with* a store



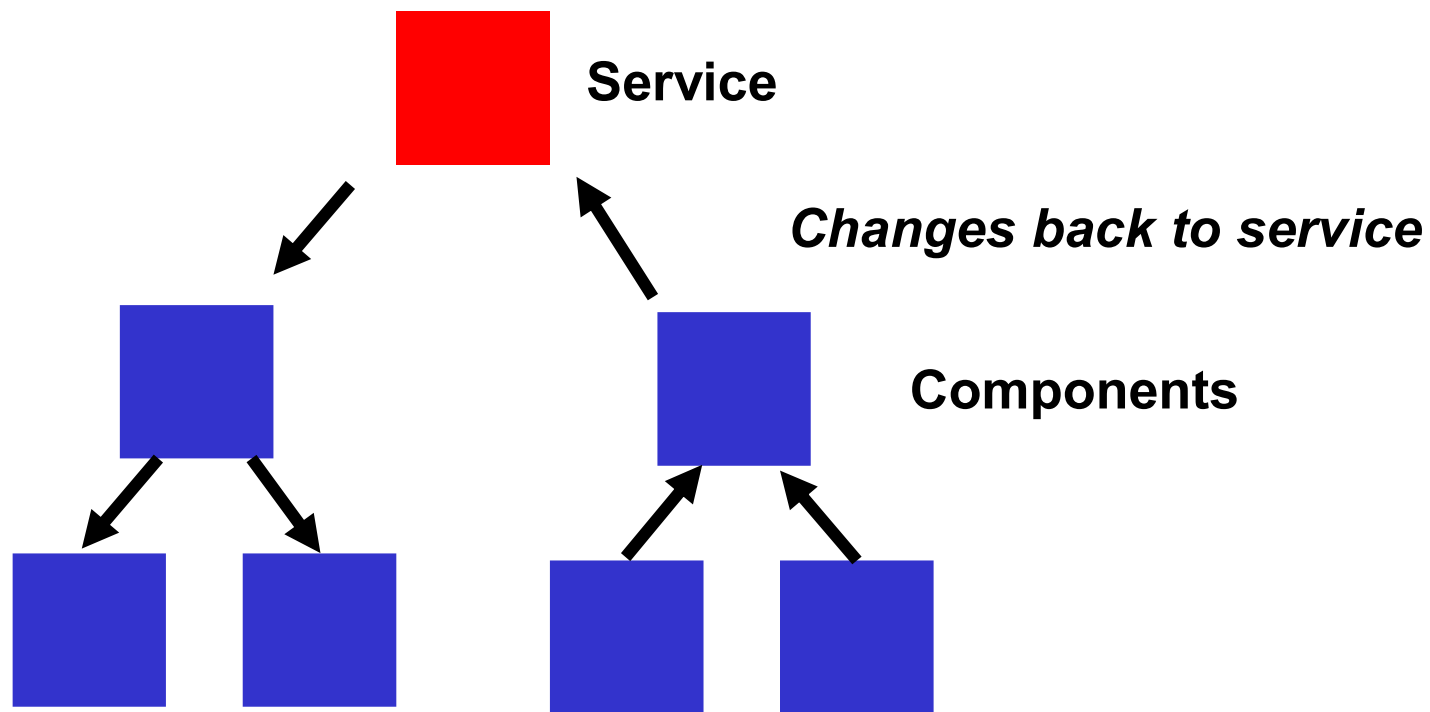
Benefits of using a store

- 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”

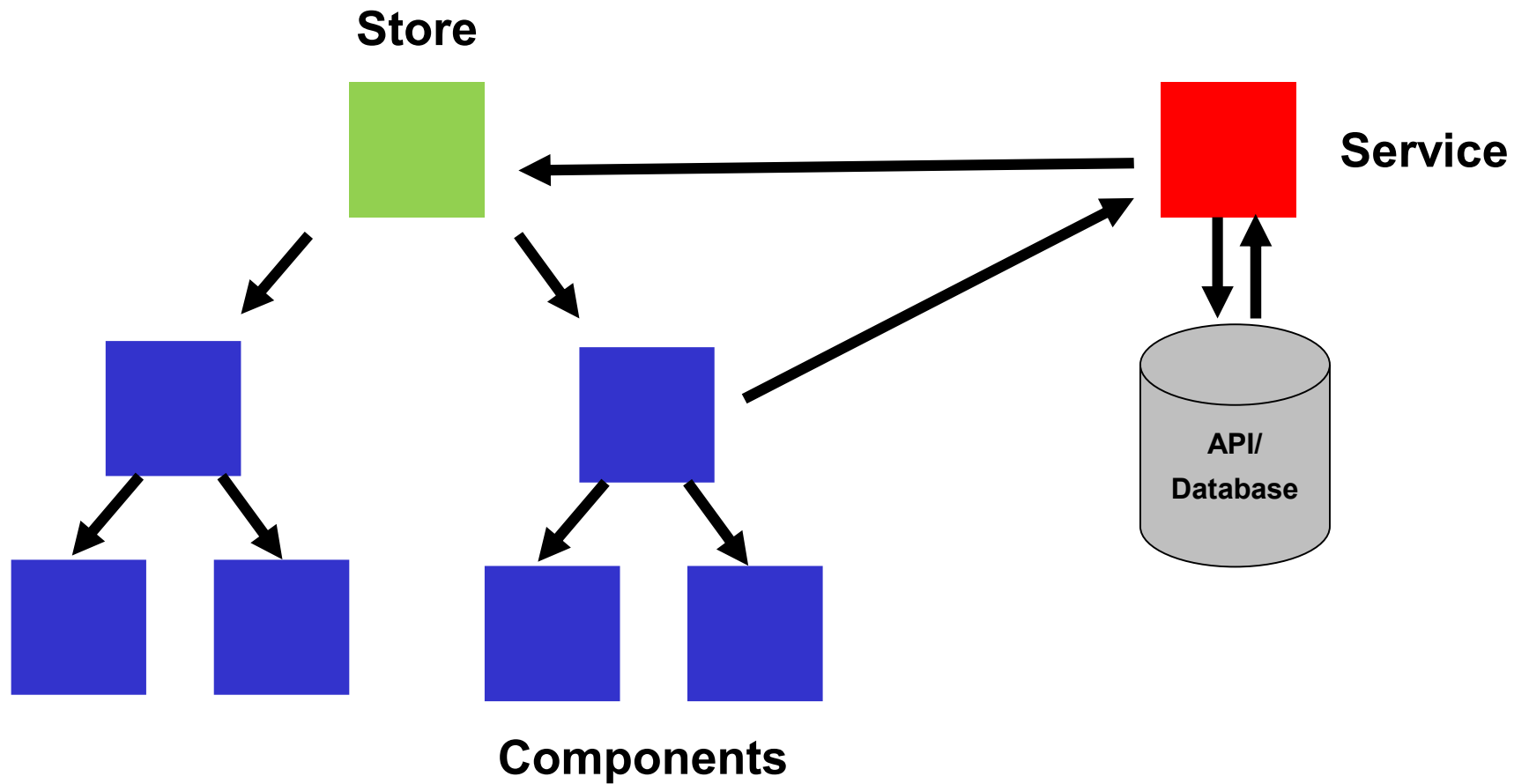
Store architecture - #2 - traditional



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<State>){
    this.counter$ = store.pipe(
        select('counter')
    );
}

increment(){
    this.store.dispatch(counterIncrement());
}
```




@ngrx/store Terminology and concepts

Working with `@ngrx/store`, the officially endorsed state management library for Angular

<https://ngrx.io/>



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."*

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."*

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."

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({...})`"

Source: <https://www.toptal.com/angular-js/ngrx-angular-reaction-application>

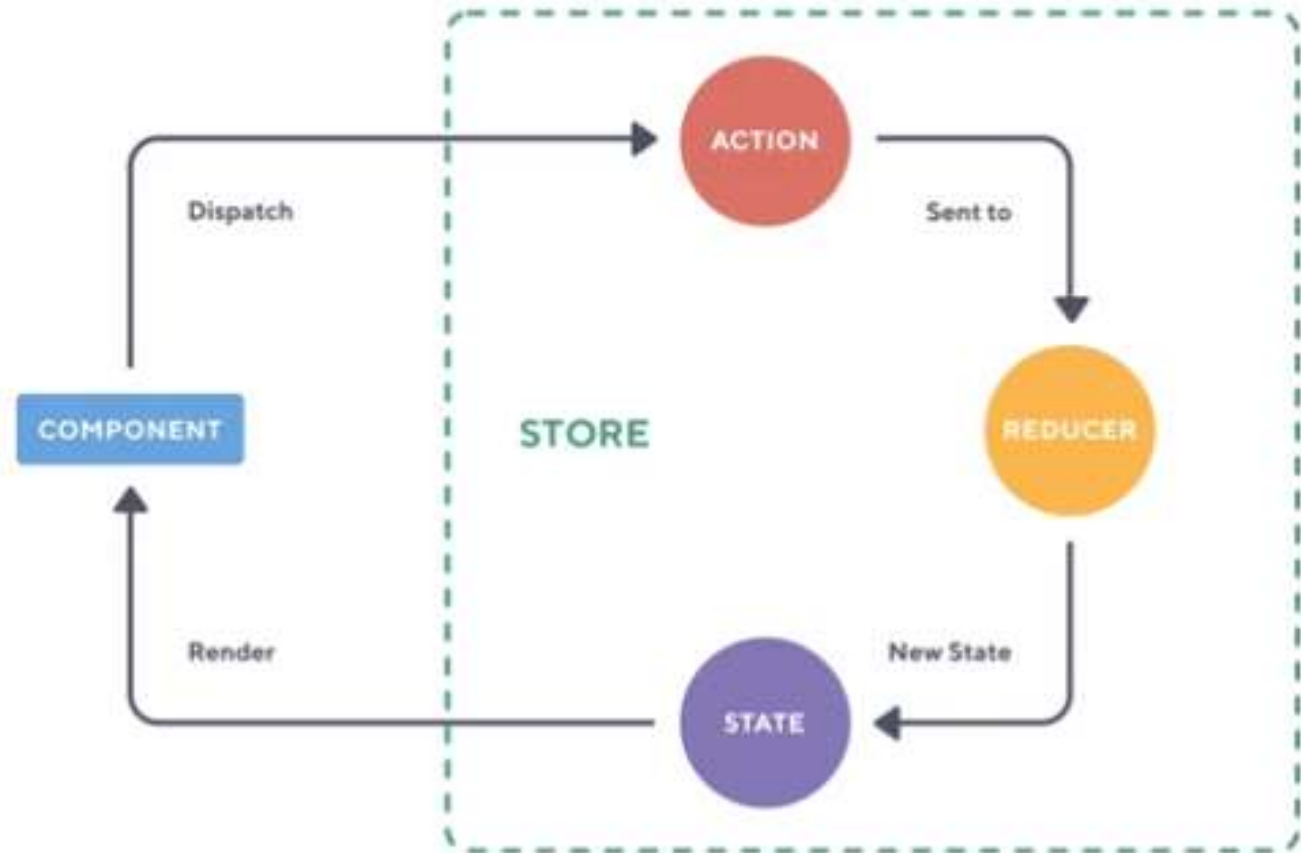
Reducers, Store and Components - The complete picture

The **Component** first dispatches 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`).

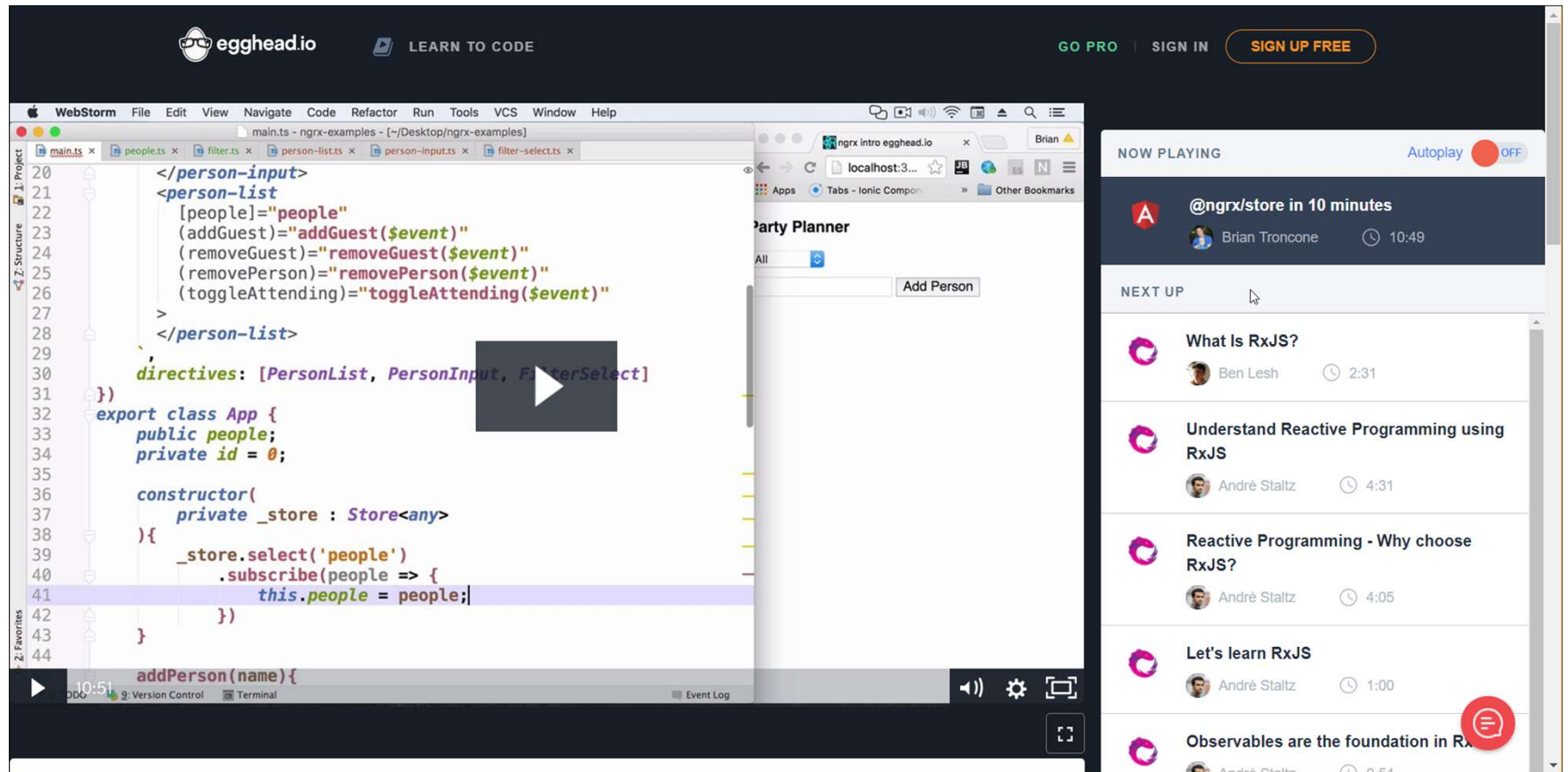
REDUX ARCHITECTURE

One-way dataflow



<https://platform.ultimateangular.com/courses/ngrx-store-effects/lectures/3788532>

Store concepts in a video (a little bit old now)



<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
- Via `npm install` or `ng add`
- Older versions have different installations!

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

OR

```
ng add @ngrx/store
```

Adding via Angular CLI

- **`ng add @ngrx/store`**
- Option flags, see <https://ngrx.io/guide/store/install>
- Adding via Angular CLI will do the following
 - Update dependencies in `package.json` and `npm install`
 - Create `src/app/reducers` folder.
 - Create `src/app/reducers/index.ts` file with an empty `State` interface, an empty reducers map, and an empty metaReducers array.
 - Update `src/app/app.module.ts`.

Installation docs

The screenshot shows the official @ngrx/store installation documentation page. The page has a purple header with navigation links: GETTING STARTED, DOCS, BLOG, RESOURCES, EVENTS, GITHUB, and SPONSOR. A search bar and social media icons are also present. On the left, a sidebar lists various sections of the documentation, with 'Installation' highlighted. The main content area is titled 'Installation' and contains three sections: 'Installing with npm', 'Installing with yarn', and 'Installing with ng add'. Each section includes a code block with the respective command and a copy icon. The 'Installing with ng add' section also includes a list of optional flags. A version selector at the bottom left shows 'stable (v8.4.0)'.

Installation

Installing with npm

For more information on using `npm` check out the docs [here](#).

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

Installing with yarn

For more information on using `yarn` check out the docs [here](#).

```
yarn add @ngrx/store
```

Installing with ng add

If your project is using the Angular CLI 6+ then you can install the Store to your project with the following `ng add` command ([details here](#)):

```
ng add @ngrx/store
```

Optional ng add flags

- `path` - path to the module that you wish to add the import for the `StoreModule` to.
- `project` - name of the project defined in your `angular.json` to help locating the module to add the `StoreModule` to.

<https://ngrx.io/guide/store/install>



Creating your first store

Set up a simple store – explaining all the concepts

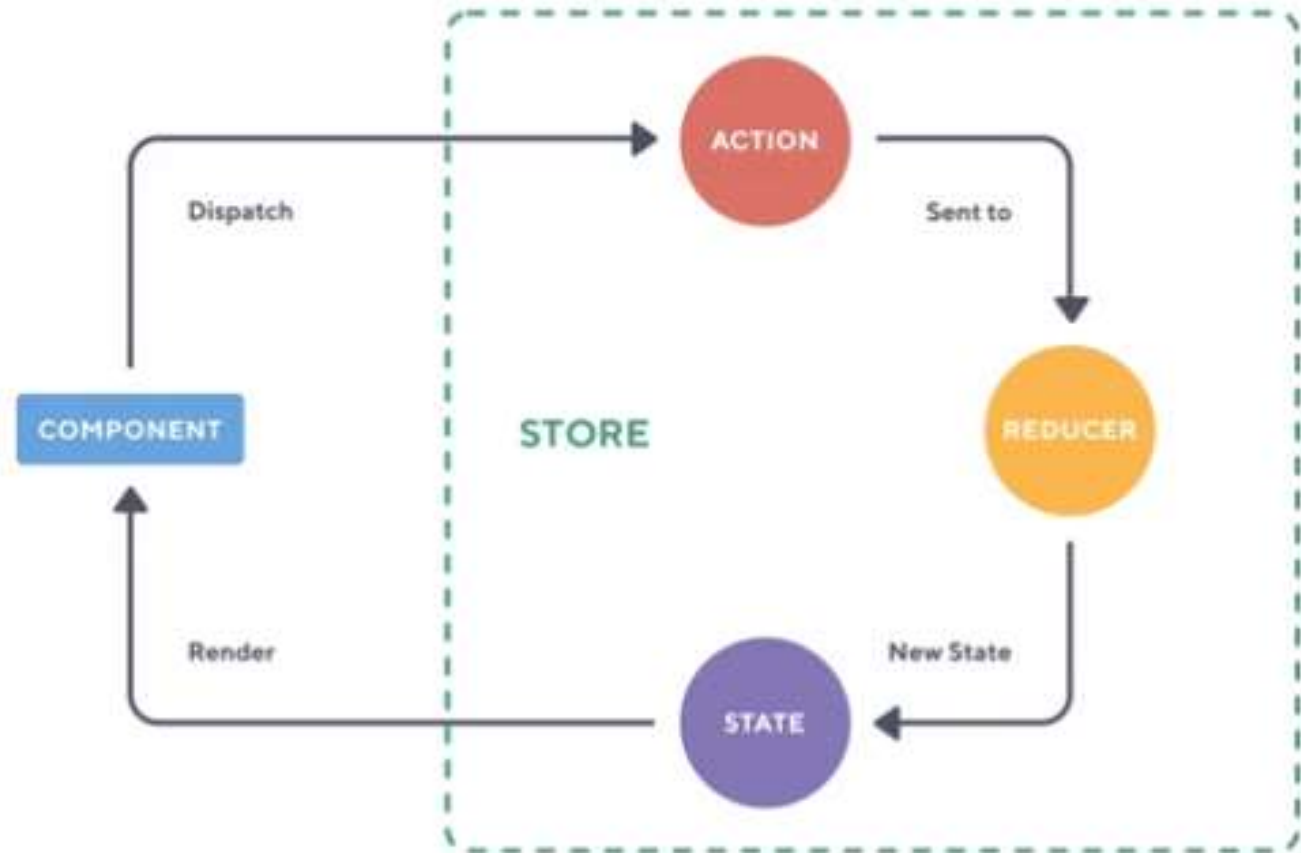
Step 0 – install core files

- We're adding the store manually to explain all concepts

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

REDUX ARCHITECTURE

One-way dataflow



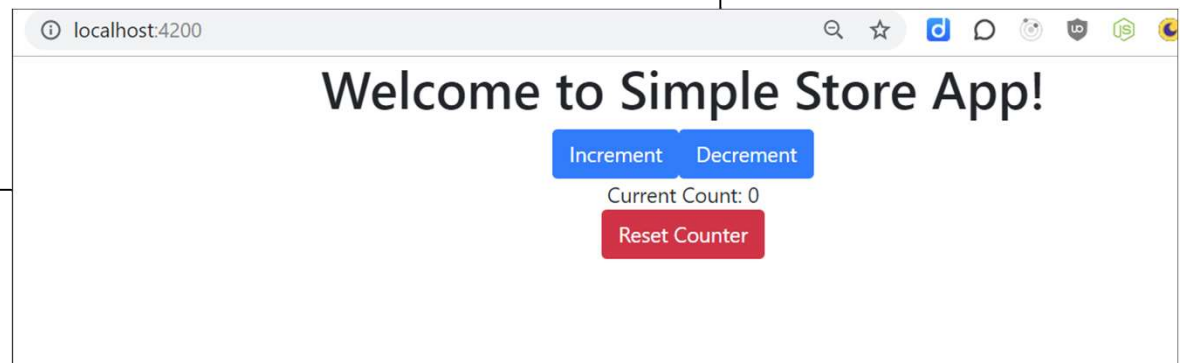
<https://platform.ultimateangular.com/courses/ngrx-store-effects/lectures/3788532>

Start somewhere, then work clockwise

- 1. For instance, first create a **component**

```
<!-- Simple Component, holding a counter store -->
<div>
  <h1>
    Welcome to {{ title }}!
  </h1>
  <button (click)="increment()">Increment</button>
  <button (click)="decrement()">Decrement</button>
  <div>Current Count: {{ count$ | async }}</div>

  <button class="btn btn-danger" (click)="reset()">
    Reset Counter
  </button>
</div>
```



2. Create your actions

- Create a new file, `../store/counter.actions.ts`
- The architecture can be complex, with nested (sub) folders etc, but it doesn't matter for the internals

```
// counter.actions.ts - the Actions for our counter  
import {createAction} from '@ngrx/store';  
  
// export our actions as constants  
export const increment = createAction('COUNTER - increment');  
export const decrement = createAction('COUNTER - decrement');  
export const reset = createAction('COUNTER - reset');
```

3. Create your reducers

- 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`
- We're going to create more complex actions, with `payload` later on
- You'll need the exported reducer function to support AOT-compiling
- <https://ngrx.io/guide/store/reducers>

```
// Import store stuff and available actions
import {Action, createReducer, on} from '@ngrx/store';
import {decrement, increment, reset} from './counter.actions';

// Initial state: counter=0
export const initialState = 0;

// Internal variable/function with reducers. It receives a state from
// the actual (exported) counterReducer below
const reducer = createReducer(initialState,
  on(increment, state => state + 1),
  on(decrement, state => state - 1),
  on(reset, state => 0)
);


// The exported reducer function is necessary
// as function calls are not supported by the AOT compiler.
export const counterReducer = (state = 0, action: Action) => {
  return reducer(state, action);
};
```

4. Adding store and reducer to module

- Register the state container with your application.
- Import reducers
- Use `StoreModule.forRoot()` to add it to the module
- More complex: we can have a *map* of reducers, or child modules holding their own stores
 - `metaReducer`: <https://ngrx.io/guide/store/metareducers>

```
...
// 1. import store stuff
import {StoreModule} from '@ngrx/store';
import {counterReducer} from '../store/counter.reducer';

@NgModule({
  declarations: [
    AppComponent,
    ...
  ],
  imports: [
    BrowserModule,
    // 2. Add the StoreModule to the AppModule,
    // to make the store known inside the application
    StoreModule.forRoot({count: counterReducer}),
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```



5. Using/calling the Store in component

- Import and inject the `Store` service to components
- Initialize the store with correct Type
 - More complex: create a custom `AppState` interface
- Use `store.pipe(select())` to select slice(s) of the state
- Add methods to dispatch actions
 - `increment()`
 - `decrement()`
 - `etc..`

```

// app.component.ts
import {Component, OnInit} from '@angular/core';
import {Observable} from 'rxjs';
import {Store, select} from '@ngrx/store';

// Import all possible actions
import {increment, decrement, reset} from './store/counter.actions';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  title = 'Simple Store App';
  count$: Observable<number>;

  constructor(private store: Store<{ count: number }>) {}

  ngOnInit() {
    // Select the 'count' property from the store and
    // assign it to count$ variable.
    this.count$ = this.store.pipe(
      select('count')
    );
  }

  // dispatch actions for the store. They are imported above
  increment() {
    this.store.dispatch(increment());
  }
  ...
}

```



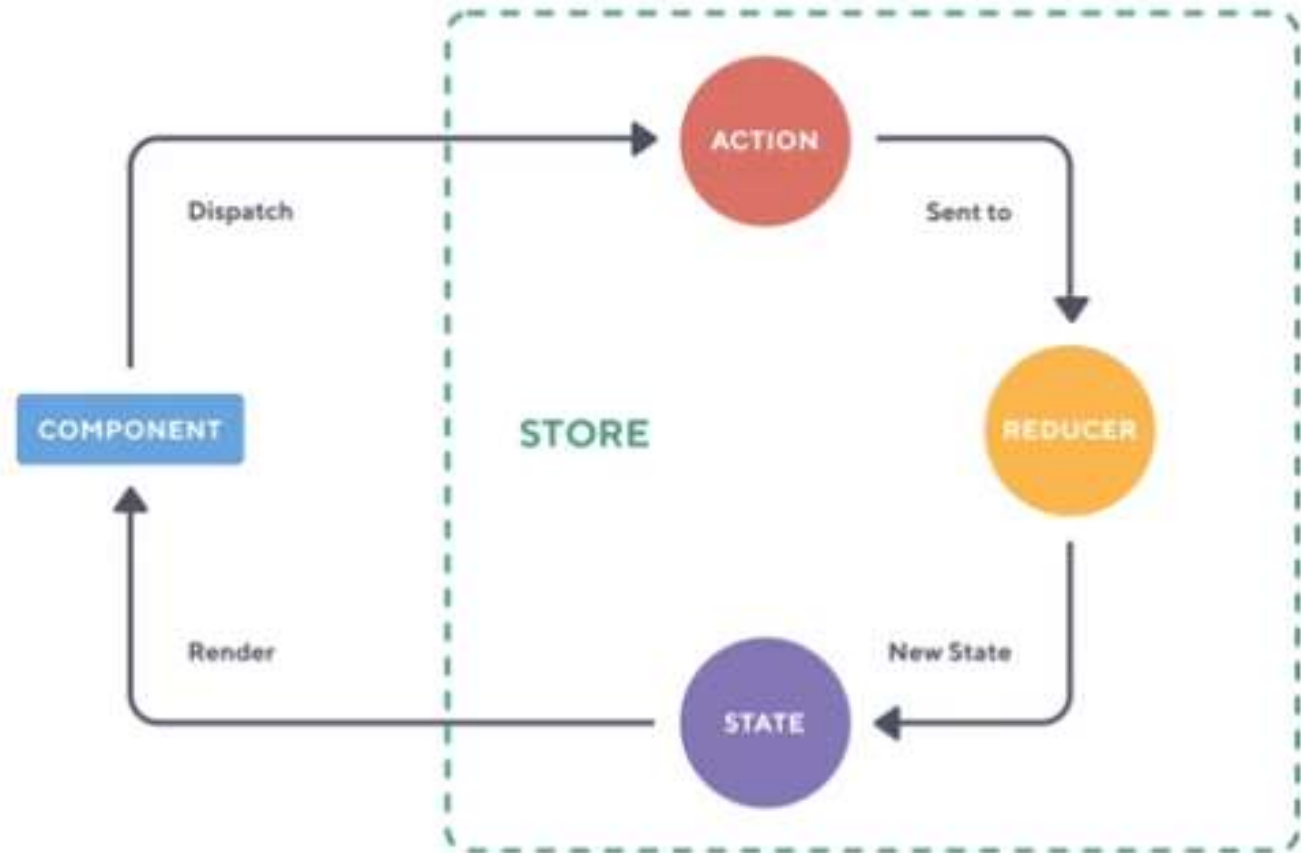
Run the app



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

REDUX ARCHITECTURE

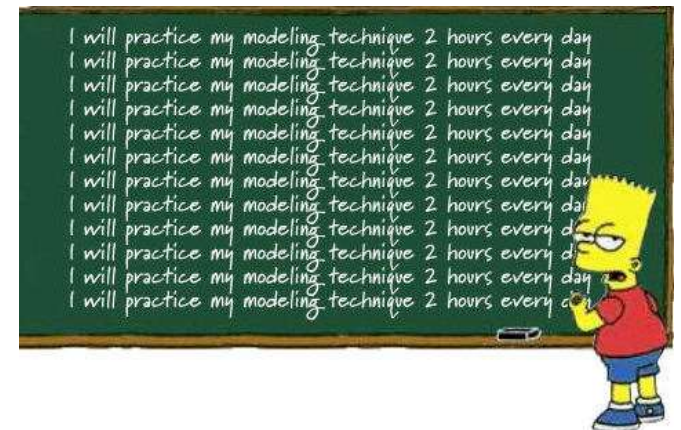
One-way dataflow



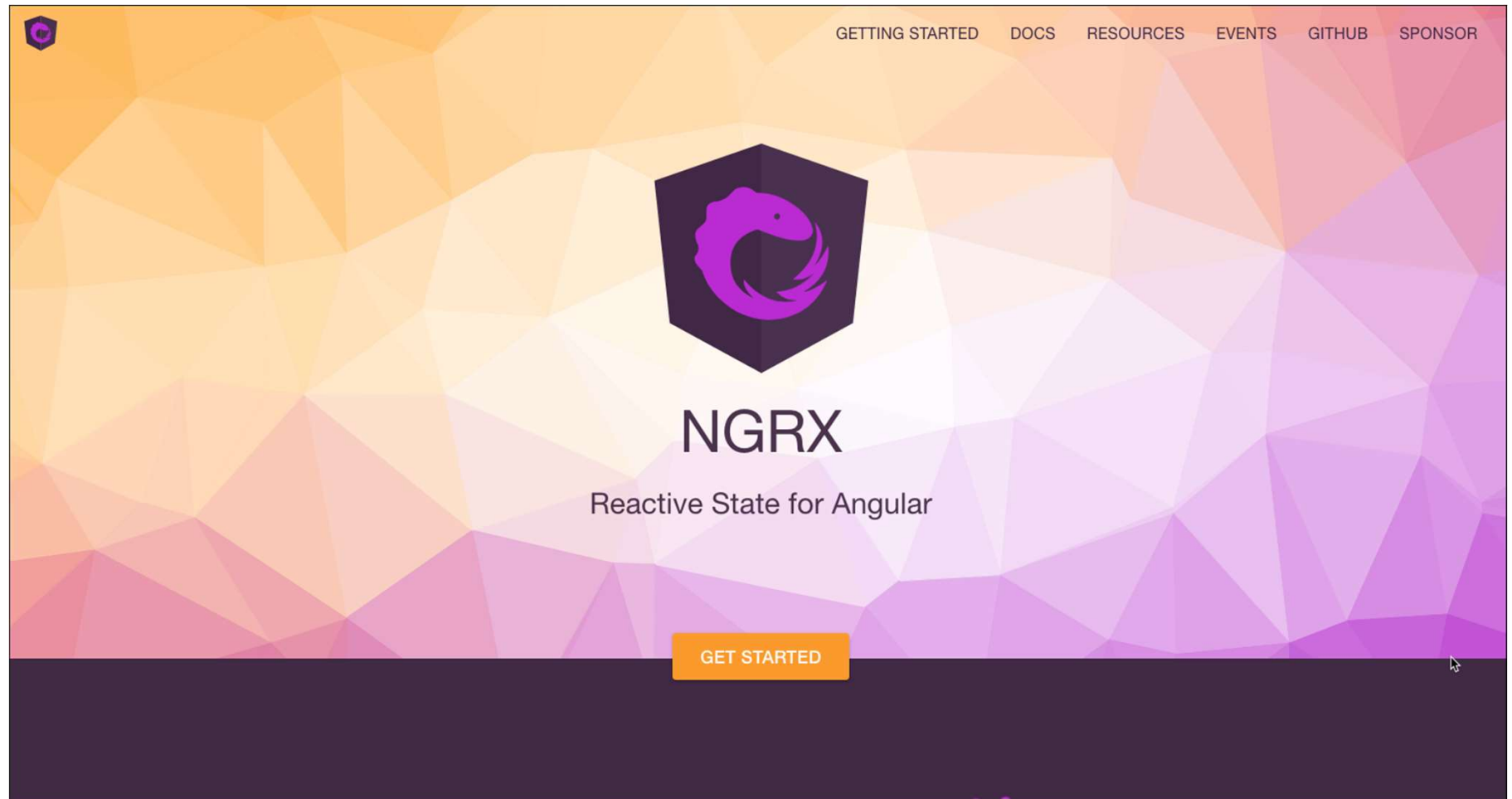
<https://platform.ultimateangular.com/courses/ngrx-store-effects/lectures/3788532>

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 >= 25;`

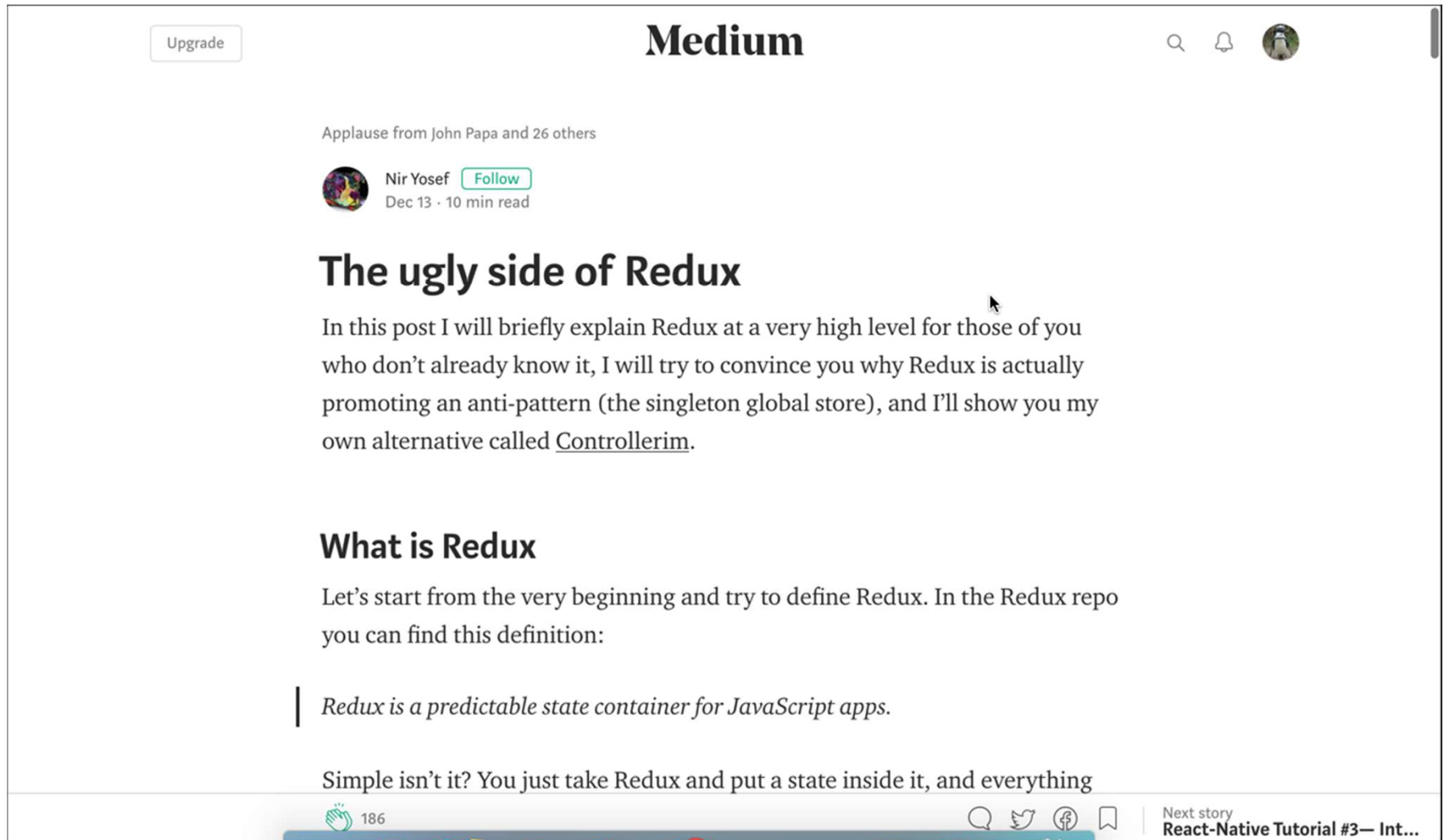


Official site



<https://ngrx.io/>

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 profile picture. Below the navigation bar, there's a section for 'Applause from John Papa and 26 others'. The author's profile is shown as 'Nir Yosef' with a 'Follow' button and a timestamp 'Dec 13 · 10 min read'. The article title is 'The ugly side of Redux'. The main text begins with '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.' Below this, there's a sub-header 'What is Redux' followed by the text 'Let's start from the very beginning and try to define Redux. In the Redux repo you can find this definition:'. A quote is displayed: 'Redux is a predictable state container for JavaScript apps.' The text continues with 'Simple isn't it? You just take Redux and put a state inside it, and everything'. At the bottom of the article preview, there's a social sharing bar with icons for clapping (186), commenting, tweeting, Facebook, and bookmarking. On the right side of the bottom bar, there's a 'Next story' link titled 'React-Native Tutorial #3— Int...'. The entire screenshot is framed within a browser window.

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.

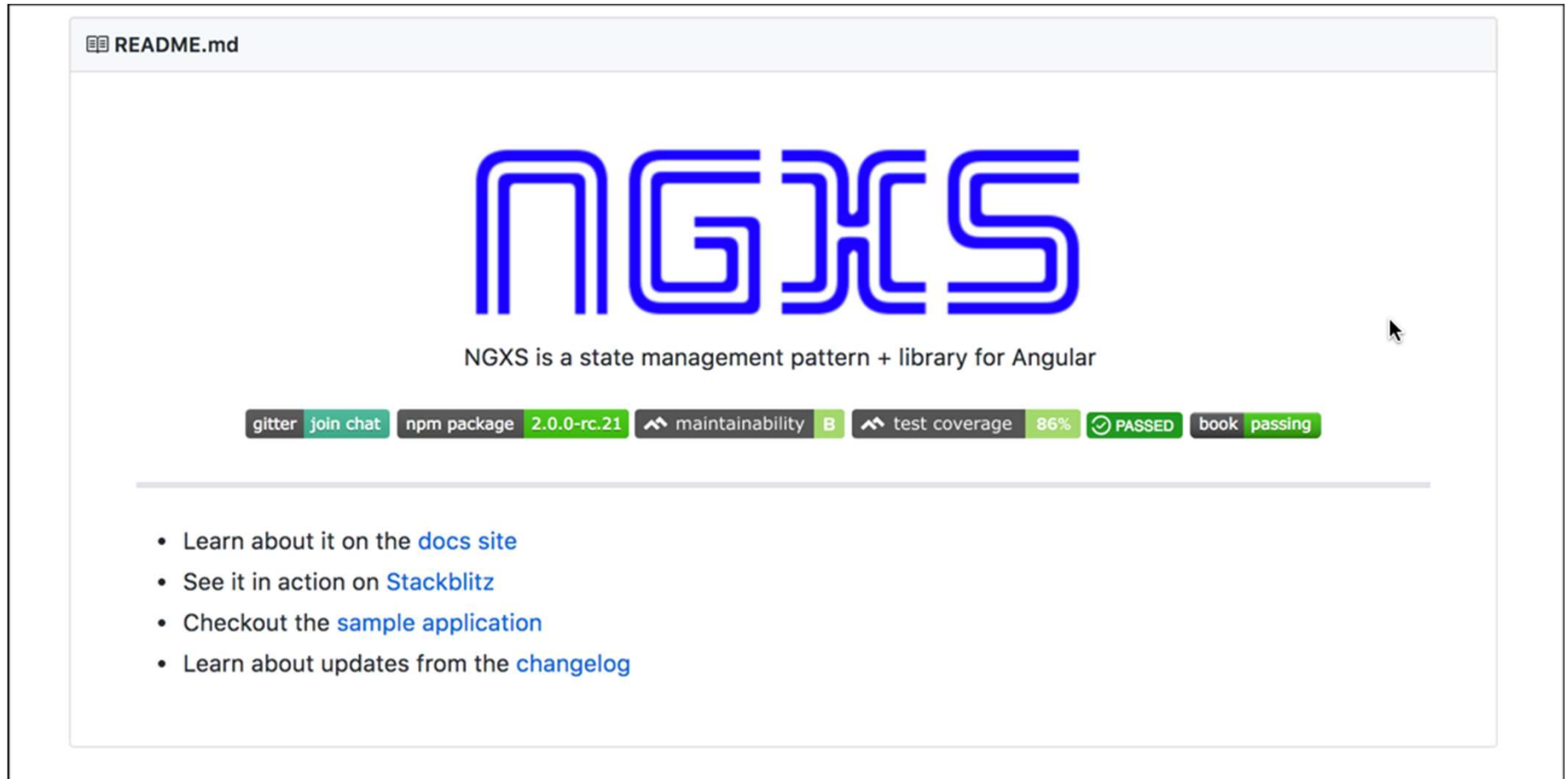
Simple isn't it? You just take Redux and put a state inside it, and everything

186

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

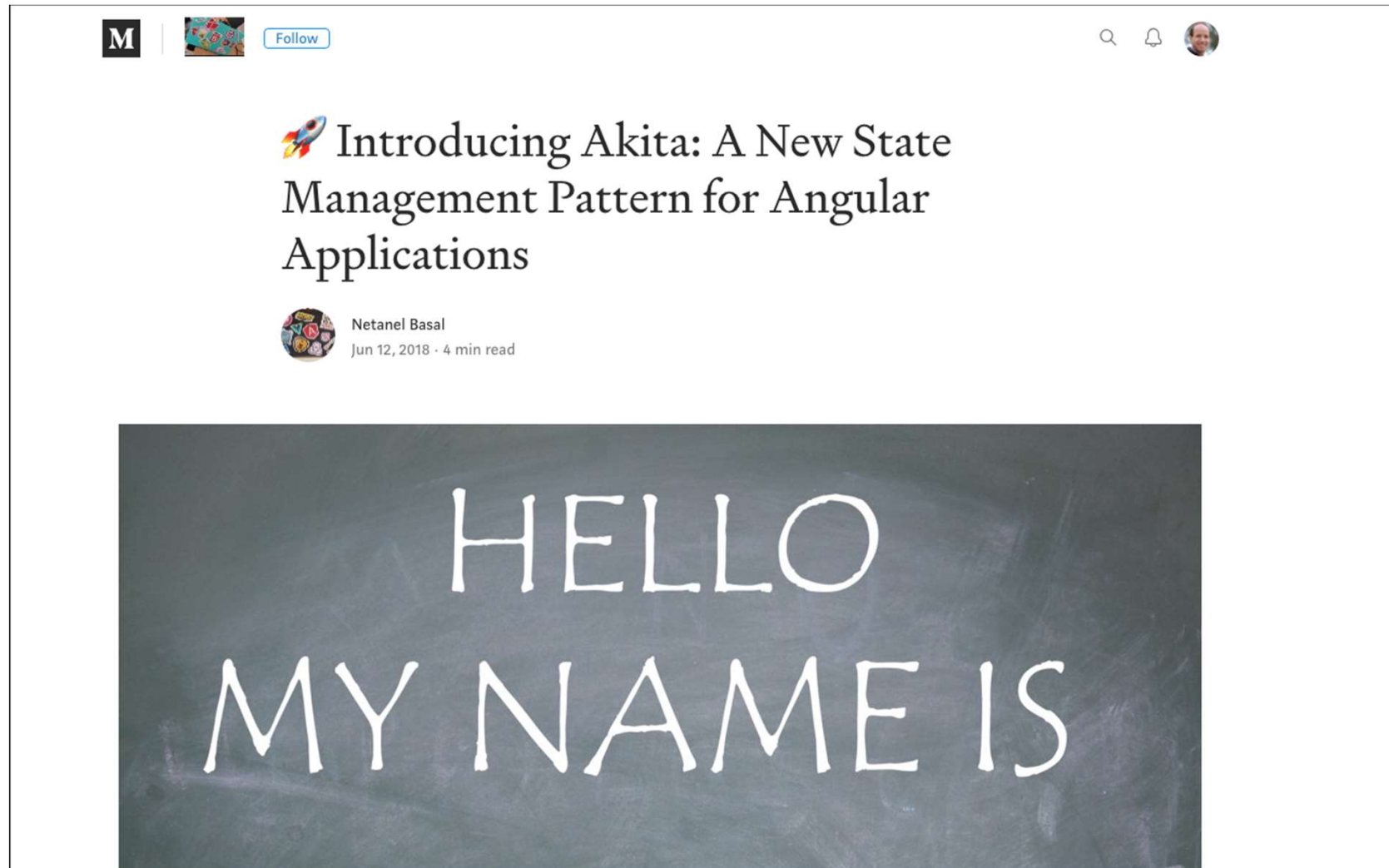
<https://medium.com/@niryo/the-ugly-side-of-redux-6591fde68200>

Alternative State Management solution



<https://github.com/amcdnl/ngxs>

Akita – another state management alternative



<https://netbasal.com/introducing-akita-a-new-state-management-pattern-for-angular-applications-f2f0fab5a8>

Redux-Observable – yet another option

Type to search

README

1.1. Introduction

BASICS

2.1. Epics

2.2. Setting Up The Middleware

RECIPES

3.1. Cancellation

3.2. Error Handling

3.3. Injecting Dependencies Into Epics

3.4. Writing Tests

3.5. Usage with UI Frameworks

3.6. Adding New Epics Asynchronously

3.7. Hot Module Replacement

HELP

4.1. Troubleshooting

API REFERENCE

5.1. createEpicMiddleware

5.2. combineEpics

5.3. EpicMiddleware

6.1. MIGRATION



redux-observable

chat on gitter build error npm v2.0.0 downloads 1M/month

RxJS-based middleware for [Redux](#). Compose and cancel async actions to create side effects and more.

<https://redux-observable.js.org>

Note: this project is quite stable, but is currently in maintenance mode. Critical fixes will still be released, but for now no additional features or changes will be considered as the maintainer [@jayphelps](#) is not actively working on any apps that use redux-observable (or any UI apps at all, actually.) If you would like to become a maintainer, please reach out to [@jayphelps](#). There is some [ongoing discussion](#) around a potential 2.0.0 release (based on the current alpha) if you'd like to join in.

Install

This has peer dependencies of `rxjs@6.x.x` and `redux@4.x.x`, which will have to be installed as well.

```
npm install --save redux-observable
```

Note: current stable version `redux-observable` doesn't work with `rxjs@7.x.x`, it works with `rxjs@6.x.x`. If you still want to use `rxjs@7.x.x`, upgrade to `redux-observable@2.0.0-rc.2`.

UMD

We publish a UMD build inside our npm package. You can use it via the [unpkg](#) CDN:

<https://unpkg.com/redux-observable@latest/dist/redux-observable.min.js>

<https://redux-observable.js.org/>

Or...ngRedux (a little bit older now, prev. generation)

The screenshot displays the GitHub repository page for `angular-redux / ng-redux`. The repository is public and has 20 watchers, 186 forks, and 1.2k stars. The main content area shows a list of files and folders, including `examples`, `src`, `test`, `.babelrc`, `.editorconfig`, `.gitignore`, `.travis.yml`, `CHANGELOG.md`, `CONTRIBUTING.md`, `LICENSE.md`, `README.md`, `commitlint.config.js`, `index.d.ts`, `package-lock.json`, `package.json`, and `rollup.config.js`. The right sidebar contains information about the repository, including the README, MIT license, 1.2k stars, 20 watching, 186 forks, 13 releases (with v4.4.5 being the latest), no published packages, 716 users using the repository, and 47 contributors.

angular-redux / ng-redux Public

Watch 20 Fork 186 Star 1.2k

Code Issues 27 Pull requests 14 Actions Wiki Security Insights

master 18 branches 22 tags

Go to file Add file > Code

oliviernt chore(): updating contribution guideline 0bce20e on May 4 363 commits

File	Commit Message	Time Ago
examples	Resolution of problem with some versions of npm/node	5 years ago
src	chore(): fixing tests	last year
test	chore(): fixing tests	last year
.babelrc	configure transform runtime for dist	5 years ago
.editorconfig	Add .editorconfig	7 years ago
.gitignore	chore(): fixing tests	last year
.travis.yml	remove semantic release	7 years ago
CHANGELOG.md	chore(release): 4.4.3	2 years ago
CONTRIBUTING.md	chore(): updating contribution guideline	last year
LICENSE.md	add LICENCE.md file	7 years ago
README.md	feat(store creator callback): provide callback for creating store	4 years ago
commitlint.config.js	chore: lint commit messages	5 years ago
index.d.ts	feat(store creator callback): provide callback for creating store	4 years ago
package-lock.json	chore(): updating contribution guideline	last year
package.json	chore(release): 4.4.5	last year
rollup.config.js	chore(build): makes redux a peerDependency	5 years ago

About

Angular bindings for Redux

- Readme
- MIT license
- 1.2k stars
- 20 watching
- 186 forks

Releases 13

v4.4.5 Latest on May 4

+ 12 releases

Packages

No packages published

Used by 716

Contributors 47

<https://github.com/angular-redux/ng-redux>

Or – Dan Wahlin, Observable Store

The video player shows a presentation by Dan Wahlin. The main content is a diagram illustrating the concept of an Observable Store. The diagram consists of several boxes: blue boxes labeled 'Component' and colored boxes labeled 'Service' (red, purple, green, orange, and grey). Arrows indicate dependencies: components depend on services, and some services depend on other services. For example, a red 'Service' box is depended on by two blue 'Component' boxes, and it in turn depends on a blue 'Component' box. This blue 'Component' box then depends on a green 'Service' box, which is depended on by two more blue 'Component' boxes. Other similar structures are shown with purple, orange, and grey service boxes. The video player interface includes the YouTube logo, a search bar, and a video player with a progress bar. The video title is 'Stepping Up: Observable Services to Observable Store' by Dan Wahlin. The channel is 'ng-conf' with 61K subscribers. The video has 133 likes and a 'Share' button. The video player also shows a 'Subscribe' button and a 'Download' button. The video player is part of a YouTube page with a sidebar on the right showing related videos.

<https://github.com/DanWahlin/Observable-Store>

Next Steps

- [@ngrx/effects](#) - Side Effect model for @ngrx/store to model event sources as actions.
- [@ngrx/router-store](#) - Bindings to connect the Angular Router to @ngrx/store
- [@ngrx/store-devtools](#) - Store instrumentation that enables a powerful time-travelling debugger
- [@ngrx/entity](#) - Entity State adapter for managing record collections.
- [@ngrx/schematics](#) - Scaffolding library for Angular applications using NgRx libraries

<https://ngrx.io/docs>



Sample Store apps

Some study material

Ngrx store platform sample app

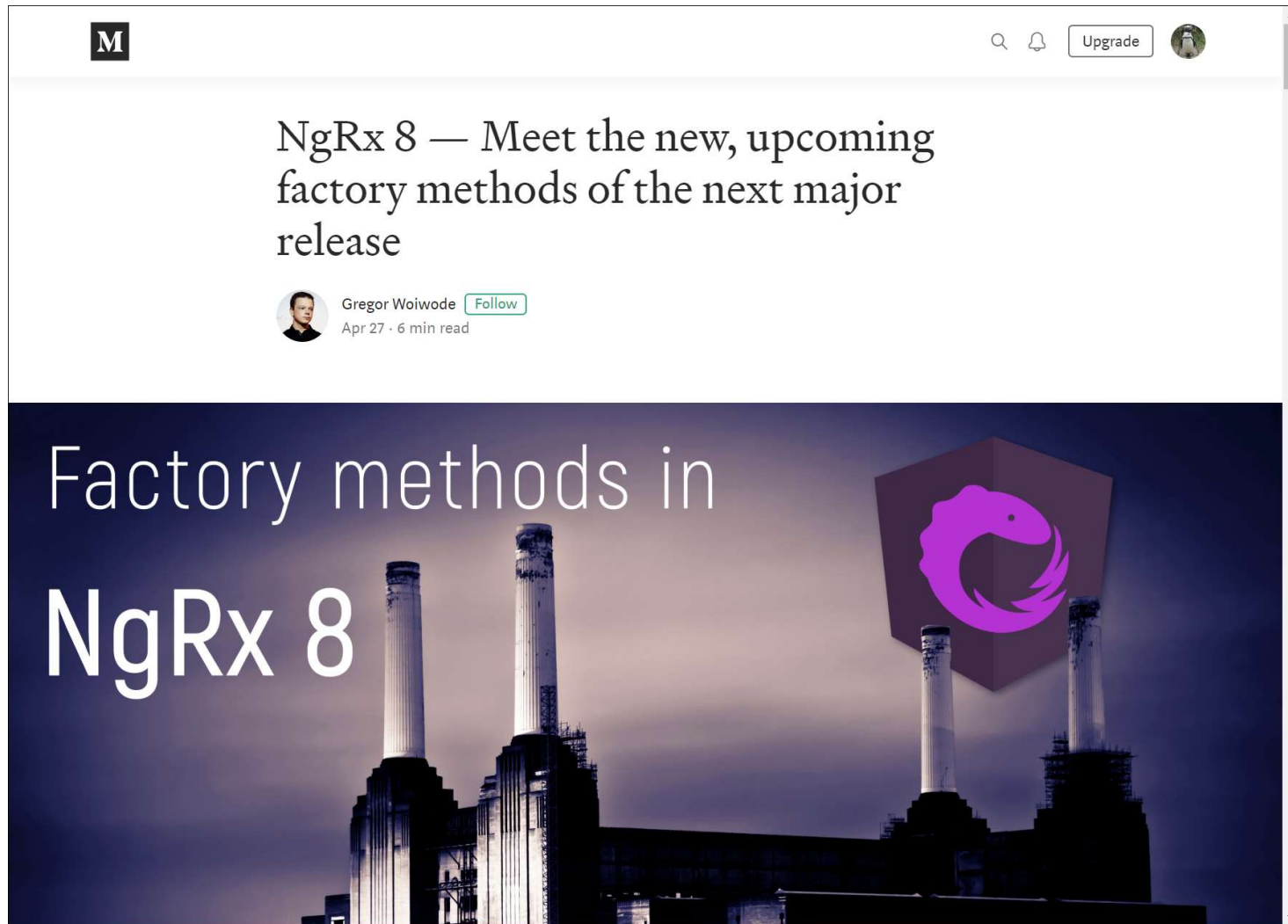
The screenshot shows the GitHub repository page for `ngrx/platform`. The repository has 220 watchers, 4,448 stars, and 1,142 forks. The current view is the `Code` tab, showing the file structure of the `example-app` project. The branch is `master`. The file structure is as follows:

File/Folder	Commit Message	Time Ago
..		
src	feat(example): make the example app more user friendly (#1508)	2 months ago
README.md	docs(example): update stackblitz link (#1277)	6 months ago
browserslist	chore(example): move example app into projects folder (#1242)	6 months ago
jest.config.js	feat: update angular dependencies to V7	4 months ago
karma.conf.js	chore(example): move example app into projects folder (#1242)	6 months ago
tsconfig.app.json	feat: update angular dependencies to V7	4 months ago
tsconfig.spec.json	chore(example): move example app into projects folder (#1242)	6 months ago
tslint.json	chore(example): move example app into projects folder (#1242)	6 months ago

Below the file structure, there is a `README.md` file. The content of the `README.md` file is partially visible, showing the text `@ngrx example application`.

<https://github.com/ngrx/platform/tree/master/projects/example-app>

More info



<https://medium.com/@gregor.woiwode/ngrx-8-meet-the-new-upcoming-factory-methods-of-the-next-major-release-a97a079cc089>