Introduction State management
Building a custom RxJS Store

Peter Kassenaar
info@kassenaar.com

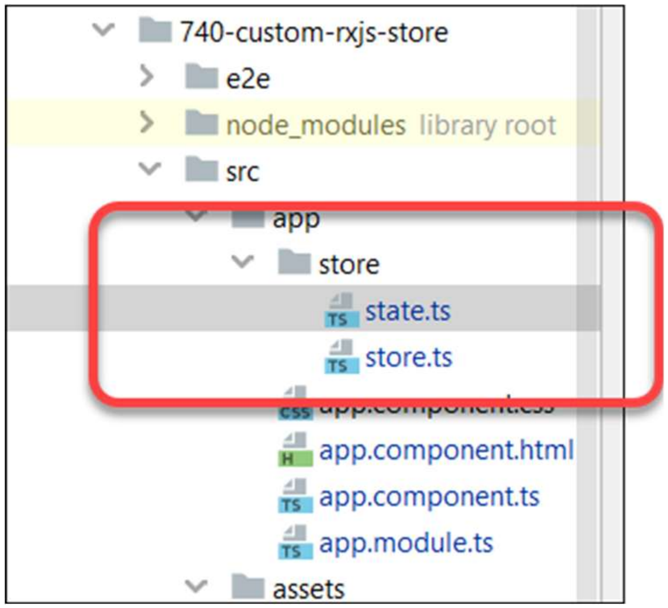
Building a store from scratch

Using observables, standard RxJS techniques and custom code, without a library

Steps

1. Create a `../store` folder and `store.ts` file, add it to the module
2. Create interface `State` for the data you want to 'store' (duh…)
3. Create a constant `state` of type `State`
4. Create a subject of type `BehaviorSubject` with type `State`, initialize it with inital `state`.
5. Expose the subject as an observable
6. Create `.set()` method and `.select()` methods

Step 1 – create store, and Step 2) state



We are now creating a *simple store*, for one application, with one (1) module.

With `@ngrx/store` things can get – way – more complex. This example is to demonstrate how store concept works.

```
// state.ts
export interface Todo {
  id: number;
  name: string;
  done: boolean;
}
export interface State {
  todos: Todo[];
  // other slices of the store
}
```

Step 3 – create state, and Step 4) `BehaviorSubject`

```
// store.ts
import {State} from './state';

const state: State = {
  todos: undefined
};

export class Store {
  // use behaviorsubject to create a subject with initial state
  // the last value is also passed to new subscribers.
  // The behaviorsubject holds the data (i.e. state)
  private subject = new BehaviorSubject<State>(state);
}
```

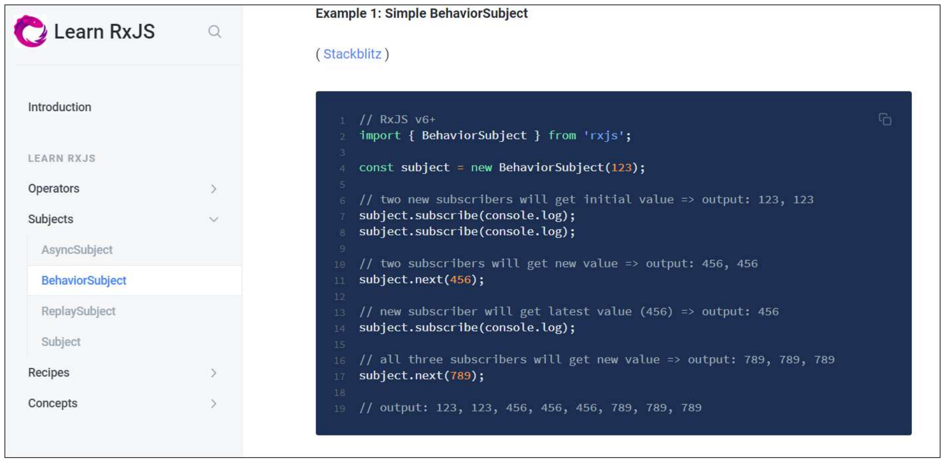
So the state is initially a list of undefined `Todo`'s.

We're going to set them later from the code. Of course you can fetch them from a backend and so on.

We use `BehaviorSubject` to create initial state. A `Subject` cannot do that.

On `BehaviorSubject<Type>()`

- `BehaviorSubject` can hold a variable (i.e. state), where a `Subject` can not.
- New subscribers get a copy of that data, i.e. the last emitted state, which of course is very useful in a store scenario.
- You pass a new piece of data to the `BehaviorSubject` with the `.next()` method.



Example 1: Simple BehaviorSubject

(Stackblitz)

```
// RxJS v6+
import { BehaviorSubject } from 'rxjs';

const subject = new BehaviorSubject(123);

// two new subscribers will get initial value => output: 123, 123
subject.subscribe(console.log);
subject.subscribe(console.log);

// two subscribers will get new value => output: 456, 456
subject.next(456);

// new subscriber will get latest value (456) => output: 456
subject.subscribe(console.log);

// all three subscribers will get new value => output: 789, 789, 789
subject.next(789);

// output: 123, 123, 456, 456, 456, 789, 789, 789
```

https://www.learnrxjs.io/learn-rxjs/subjects/behaviorsubject

Step 5 - Expose the subject as an observable

```
// store.ts
export class Store {
  // use behaviorsubject to create a subject with initial state
  // the last value is also passed to new subscribers.
  // The behaviorsubject holds the data (i.e. state)
  private subject = new BehaviorSubject<State>(state);
  private store = this.subject.asObservable()
    .pipe(
      distinctUntilChanged() // make it a little bit smoother, don't overnotify the subscribers
    );
}
```

The `store` is the variable we expose to the outer world later on, so components and services deal with an observable instead of a subject directly

Step 6 – create `.set()` and `.select()` methods

- Also create a helper `get` property that returns the current value of the state

```
// store.ts
export class Store {
   …
  // internal helper function, return the current
  // value of the subject.
  get value(): any {
    return this.subject.value;
  }

  // set a new piece in the store. Update the
  // current store, using the spread operator (favor immutability)
  set(name: string, payload: any): void {
    this.subject.next({
      ...this.value, [name]: payload
    });
  }

  // select a slice from the store, use pluck to only fetch the
  // requested branch of the json-tree from the store
  select<T>(name: string): Observable<T> {
    return this.store.pipe(
      pluck(name),
    );
  }
}
```

Getter (internal)

Setter, using .next()

Dynamically set the name of the property in the store. If it doesn't exists, it creates it

Selector, using .pluck()

Access our store, only return the selected slice

This is all we need to do to create a reactive store!

Use the store in the component

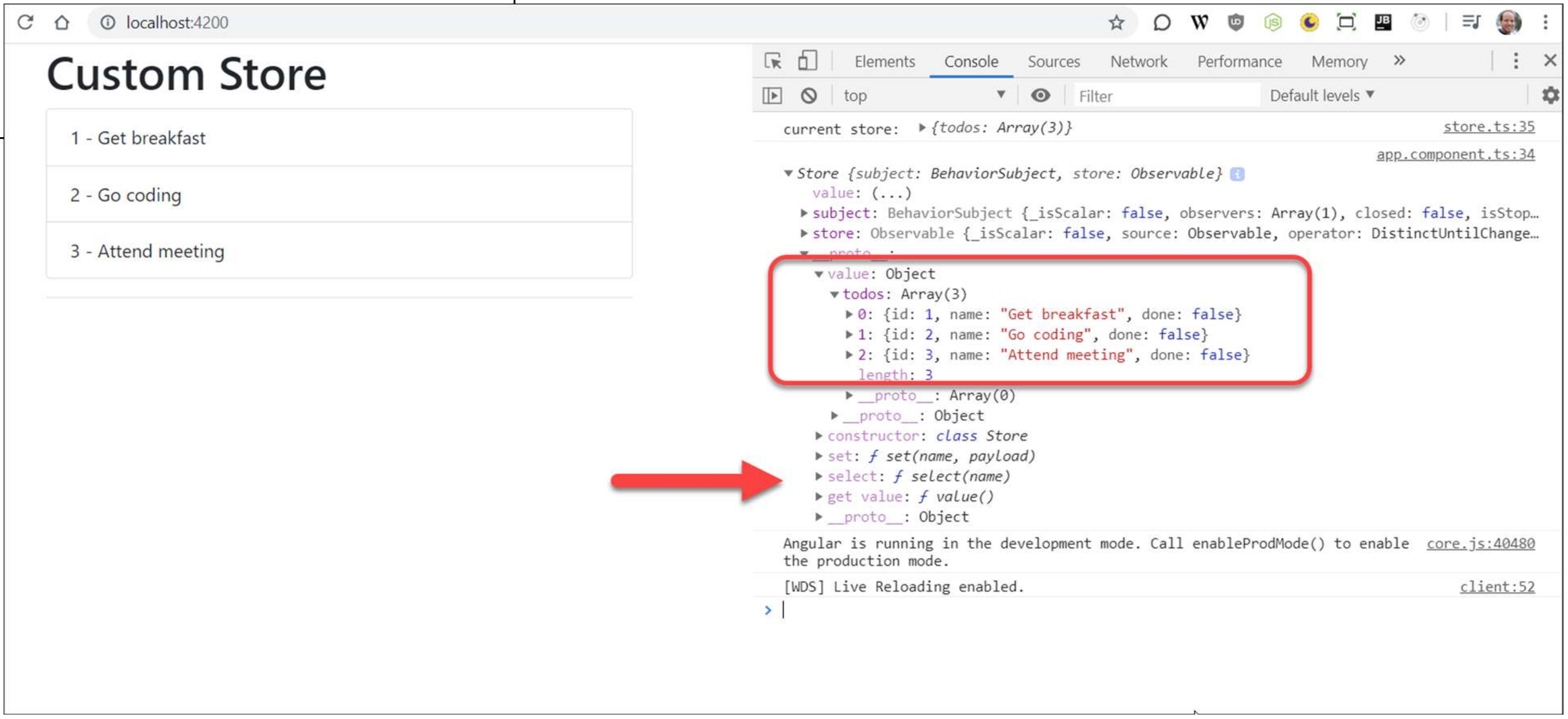
- Import the store in the component
  - set data, retrieve that data and bind it to the UI

```
// app.component.ts
export class AppComponent implements OnInit {
  todo$: Observable<any>;

  constructor(private store: Store) {        Import store
  }

  ngOnInit(): void {
    // just some dummy data in the store, you can fetch this from a backend of course
    const someTodos: Todo[] = [
      {id: 1, name: 'Get breakfast', done: false},
      {id: 2, name: 'Go coding', done: false},
      {id: 3, name: 'Attend meeting', done: false},
    ];
    this.store.set('todos', someTodos);   // 1. don't set a component property directly! Instead, set data in the store
    this.todo$ = this.store.select<Todo[]>('todos'); // 2. Fetch data from the store, assign it to local property
    console.log(this.store);  // 3. Just some logging, to see if the store works
  }
}
```

Note: no callbacks here. Everything is reactive (as one would expect from a reactive store)

Result

```html
<div class="container">
  <h1>Custom Store</h1>
  <ul class="list-group">
    <li class="list-group-item"
      *ngFor="let todo of todo$ | async">
      {{ todo.id }} - {{ todo.name }}
    </li>
  </ul>
  <hr>
</div>
```



Custom Store

1 - Get breakfast

2 - Go coding

3 - Attend meeting

```
current store:  ▸{todos: Array(3)}                                    store.ts:35
                                                               app.component.ts:34
▾Store {subject: BehaviorSubject, store: Observable}
   value: (...)
  ▸subject: BehaviorSubject {_isScalar: false, observers: Array(1), closed: false, isStop…
  ▸store: Observable {_isScalar: false, source: Observable, operator: DistinctUntilChange…
  ▾__proto__:
    ▾value: Object
      ▾todos: Array(3)
        ▸0: {id: 1, name: "Get breakfast", done: false}
        ▸1: {id: 2, name: "Go coding", done: false}
        ▸2: {id: 3, name: "Attend meeting", done: false}
         length: 3
        ▸__proto__: Array(0)
      ▸__proto__: Object
    ▸constructor: class Store
    ▸set: ƒ set(name, payload)
    ▸select: ƒ select(name)
    ▸get value: ƒ value()
    ▸__proto__: Object
Angular is running in the development mode. Call enableProdMode() to enable  core.js:40480
the production mode.
[WDS] Live Reloading enabled.                                                client:52
>
```

Updating the store

Writing new values in the store by writing a custom `.update()` method

Update the list of Todo’s in the store

```
// Update the store, in this case a list of todos
updateTodo(name: string, payload: Todo): void {
  // 1. fetch the correct slice from the store (even if we only have one)
  const value = this.value[name];
  const newTodos: Todo[] = value.map((todo: Todo) => {
    // 2. loop over our todos and update the given item
    if (payload.id === todo.id) {
      return {...todo, ...payload};
    } else {
      return todo;
    }
  });
  // 3. Set the store with the new value of newTodos
  this.set(name, newTodos);
  // 4. Optional - write state to localStorage, save todos in backend, etc.
}
```

Get correct slice

Loop over items, use array mapping (!)

Return updated item…

Or simply return if not applicable

Write new array in the store

Update the UI and logic for component

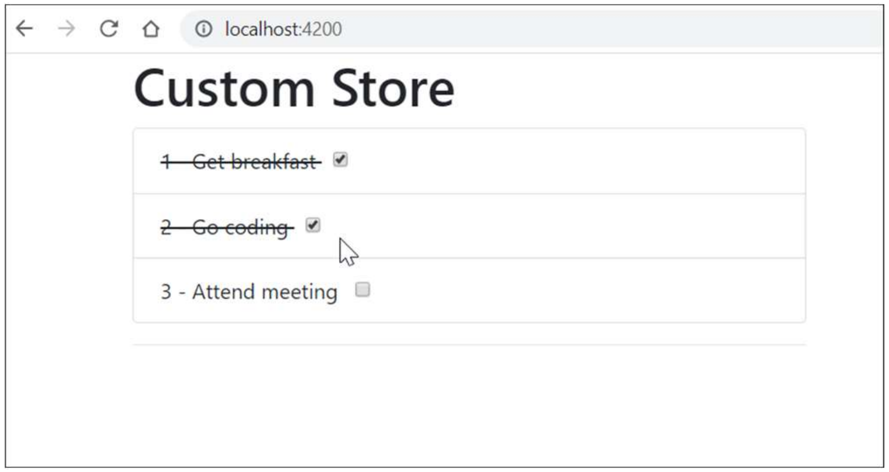
```
<ul class="list-group">
  <li class="list-group-item"
    *ngFor="let todo of todo$ | async">
    <span [ngClass]="{'todo-done': todo.done}">
      {{ todo.id }} - {{ todo.name}}
    </span>
    &nbsp;
    <input type="checkbox" [checked]="todo.done" (change)="updateTodo(todo)">
  </li>
</ul>
```

custom CSS class

Add behavior

```
// update the state of a todo item
updateTodo(todo: Todo) {
  // toggle the state of item
  todo.done = !todo.done;
  this.store.updateTodo('todos', todo);
}
```

Toggle state and update the store



Custom Store

~~1 - Get breakfast~~ ✓

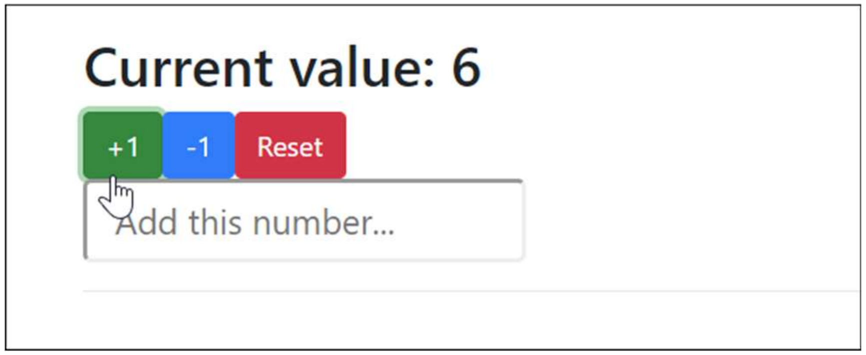
~~2 - Go coding~~ ✓

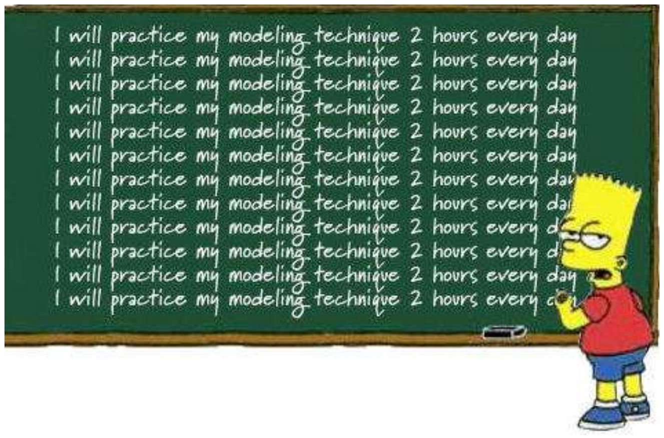
3 - Attend meeting ☐

Result

Workshop - 1

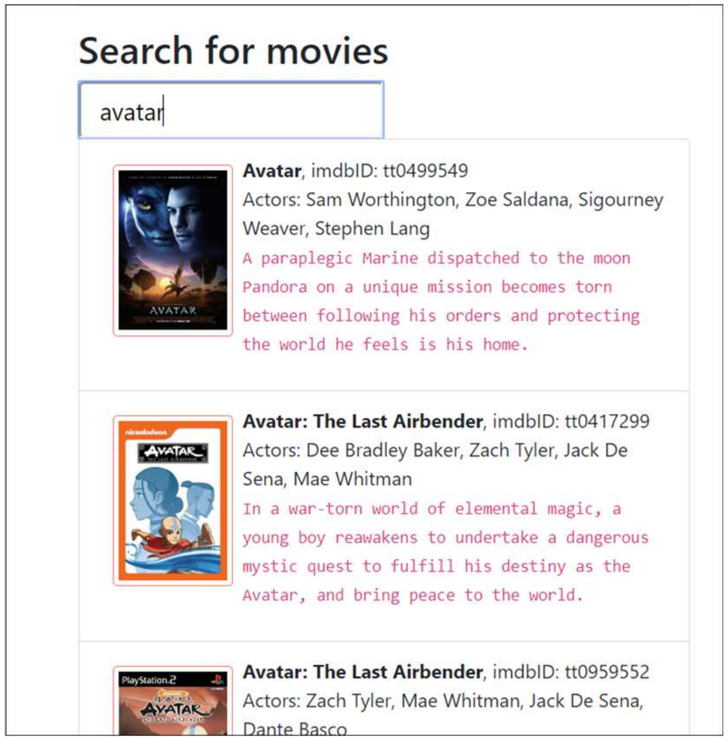
- Create a new app, Create a custom store, as described in the slides
- OR: Start from `../740-custom-rxjs-store`
- Create a `counter$` property and add it to the store.
  - In your component: show buttons to `increment()`, `decrement()` and `reset()` the `counter` in the store
  - Add it –for now – to the *same* component, for simplicity
- Some UI and logic is already available in the example, but first try it yourself!





Workshop - 2

- Create a `movies$` property and add it to the store.
- Add a textbox to search for movies, put movies in the store.
- Search for movie details, based on the imdbID which is now available.
- Some UI and logic is already available in the example, but first try it yourself!



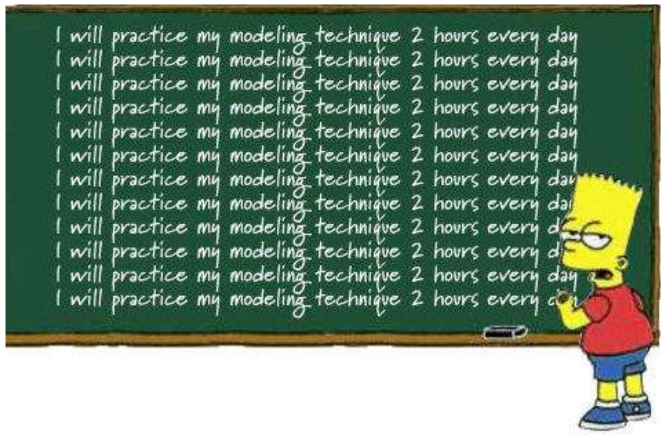
Search for movies

avatar

Avatar, imdbID: tt0499549
Actors: Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang
A paraplegic Marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.

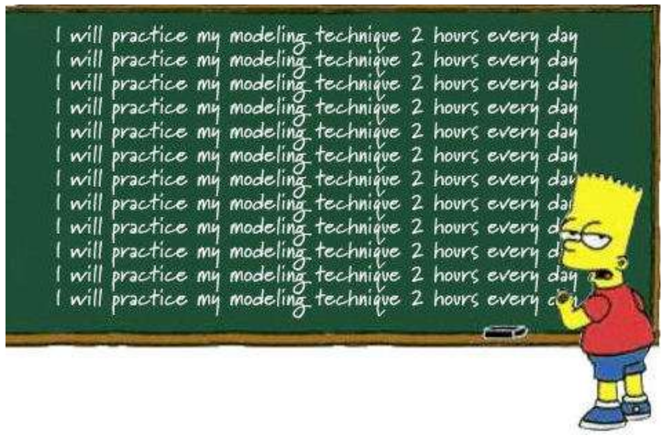
Avatar: The Last Airbender, imdbID: tt0417299
Actors: Dee Bradley Baker, Zach Tyler, Jack De Sena, Mae Whitman
In a war-torn world of elemental magic, a young boy reawakens to undertake a dangerous mystic quest to fulfill his destiny as the Avatar, and bring peace to the world.

Avatar: The Last Airbender, imdbID: tt0959552
Actors: Zach Tyler, Mae Whitman, Jack De Sena, Dante Basco

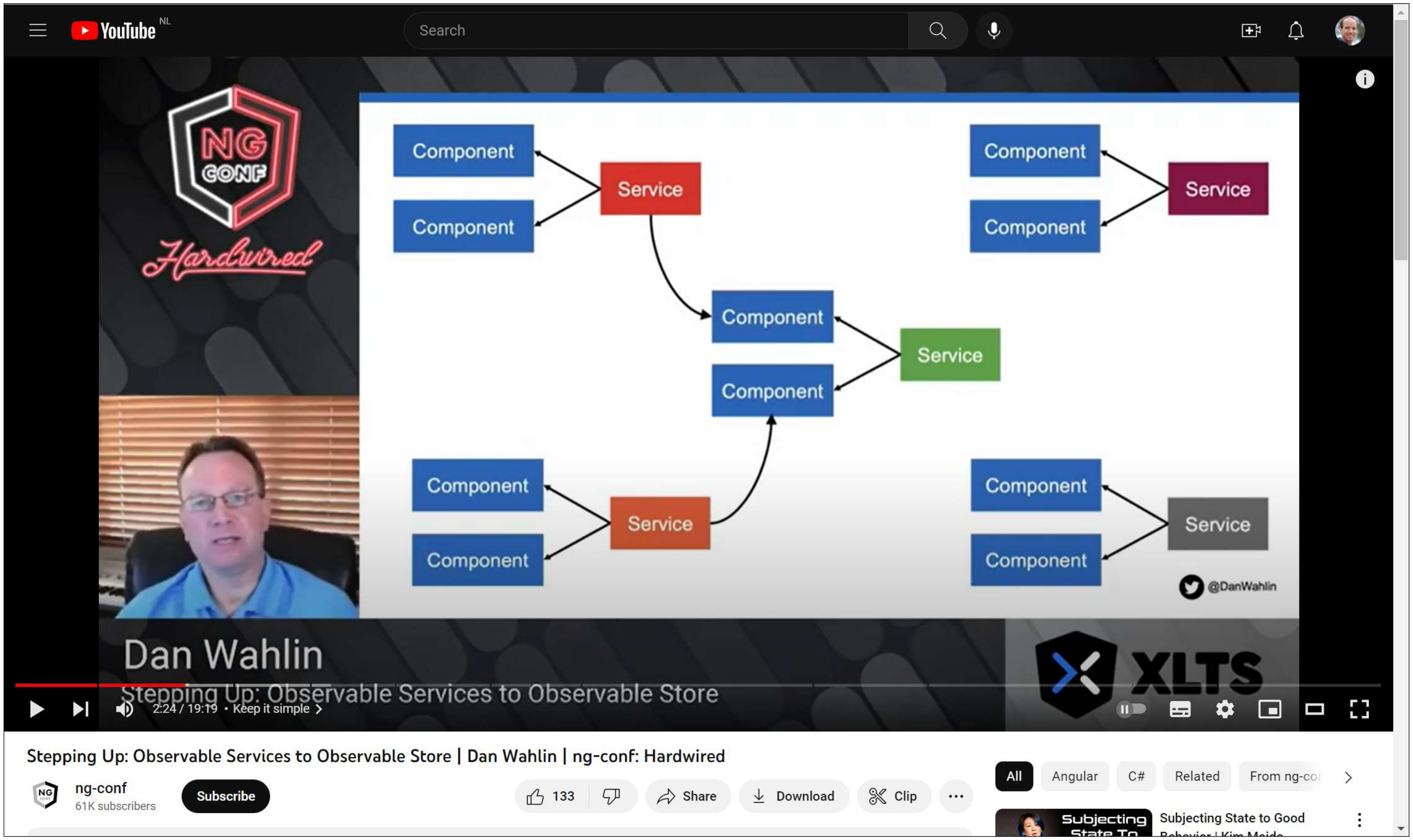

I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every

Optional workshop - 3

- Add the router, (like in `../740-custom-rxjs-store`)
- Make sure that the store contents survive a switch in components.
  - E.g. Movies retain in the store, the counter value is preserved, and so on.
- Tip: *don't reinitialize* the store in the `ngOnInit()` of every component, instead, do it once in app.component.ts and work from there.

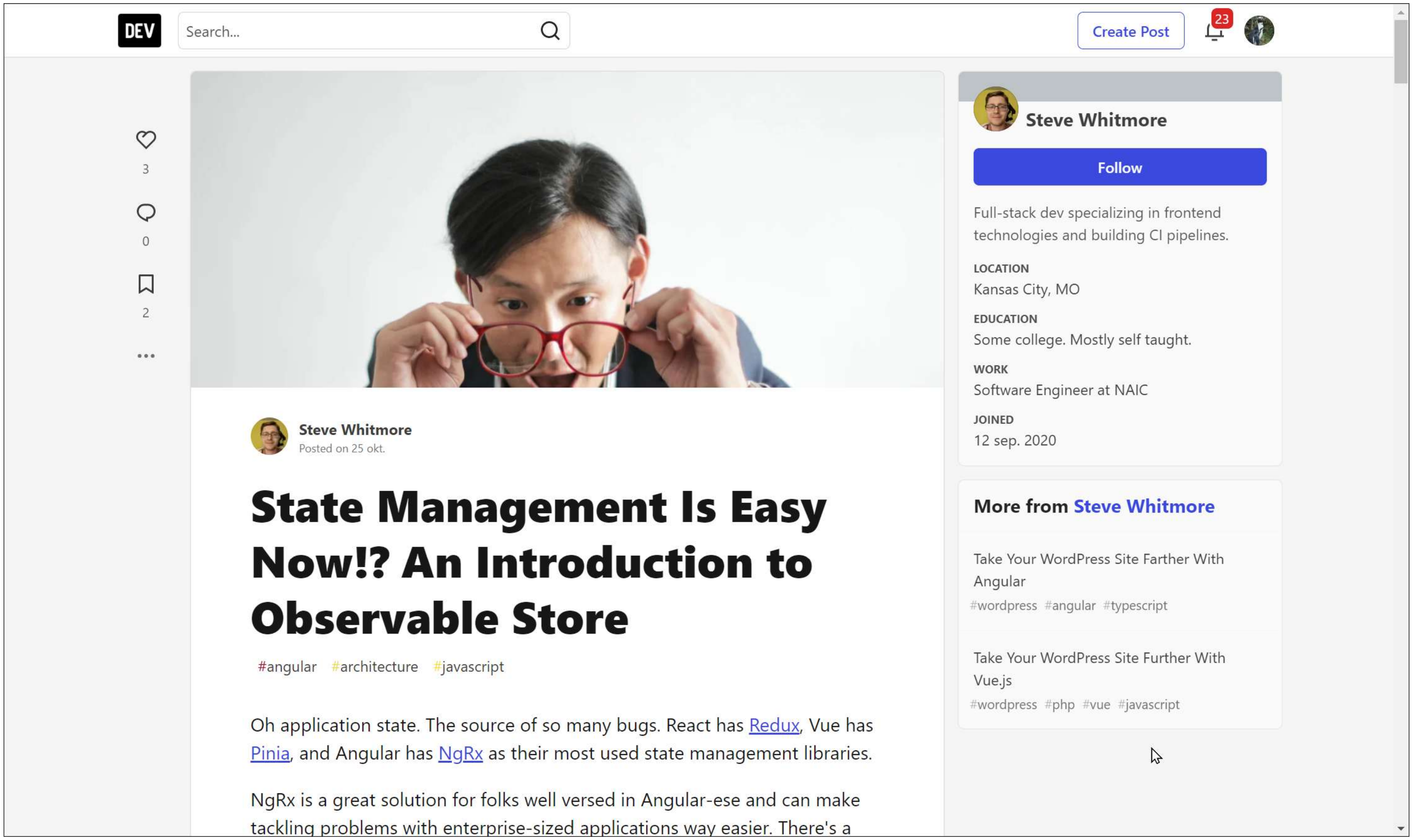


Simple (KISS) Observable Store by Dan Wahlin


YouTube NL
Search
NG CONF
Hardwired
Component
Service
@DanWahlin
Dan Wahlin
Stepping Up: Observable Services to Observable Store
XLTS
2:24 / 19:19 • Keep it simple
Stepping Up: Observable Services to Observable Store | Dan Wahlin | ng-conf: Hardwired
ng-conf
61K subscribers
Subscribe
133
Share
Download
Clip
All
Angular
C#
Related
From ng-co
Subjecting State to Good Behavior | Kim Maida

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

Blogpost on Observable Store



DEV Search... Create Post 23

3 0 2

Steve Whitmore
Follow

Full-stack dev specializing in frontend technologies and building CI pipelines.

LOCATION
Kansas City, MO

EDUCATION
Some college. Mostly self taught.

WORK
Software Engineer at NAIC

JOINED
12 sep. 2020

Steve Whitmore
Posted on 25 okt.

State Management Is Easy Now!? An Introduction to Observable Store

#angular #architecture #javascript

Oh application state. The source of so many bugs. React has Redux, Vue has Pinia, and Angular has NgRx as their most used state management libraries.

NgRx is a great solution for folks well versed in Angular-ese and can make tackling problems with enterprise-sized applications way easier. There's a

More from Steve Whitmore

Take Your WordPress Site Farther With Angular
#wordpress #angular #typescript

Take Your WordPress Site Further With Vue.js
#wordpress #php #vue #javascript

https://dev.to/stevewhitmore/state-management-is-easy-now-an-introduction-to-observable-store-15ij