

# Reactive Programming with RxJS and Angular

Osnabrück, 6. Juni 2018

## Thomas Cybulski



tho\_cyb



ThomasCybulski

t.cybulski@salt-and-pepper.eu

# Agenda

- Basics & History
- Reactive Extensions
- Observables and Subjects
- Operator - Code Examples
- Lessons learned

# 01

## Basics & History

## It can be so easy ...

**`Rx.Observable.prototype.flatMapLatest(selector, [thisArg])`**

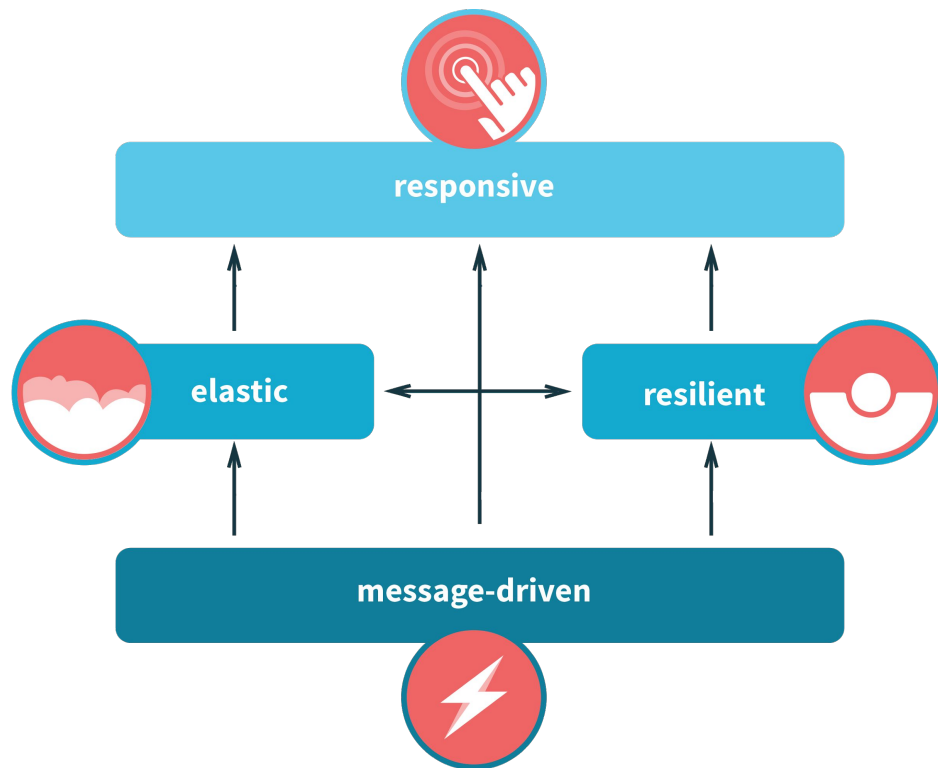
Projects each element of an observable sequence into a new sequence of observable sequences by incorporating the element's index and then transforms an observable sequence of observable sequences into an observable sequence producing values only from the most recent observable sequence.



# Motivation

```
search(term:string) {  
  let promise = new Promise((resolve, reject) => {  
    this.http.get('my_URL')  
      .toPromise()  
      .then(  
        res => { // Success  
          this.results = res.json().results;  
          resolve();  
        },  
        msg => { // Error  
          reject(msg);  
        }  
      );  
  });  
  return promise;  
}
```

# The Reactive Manifesto





# Responsive

*React to user*



# Resilient

*React to failure*



# Elastic

*React to load*



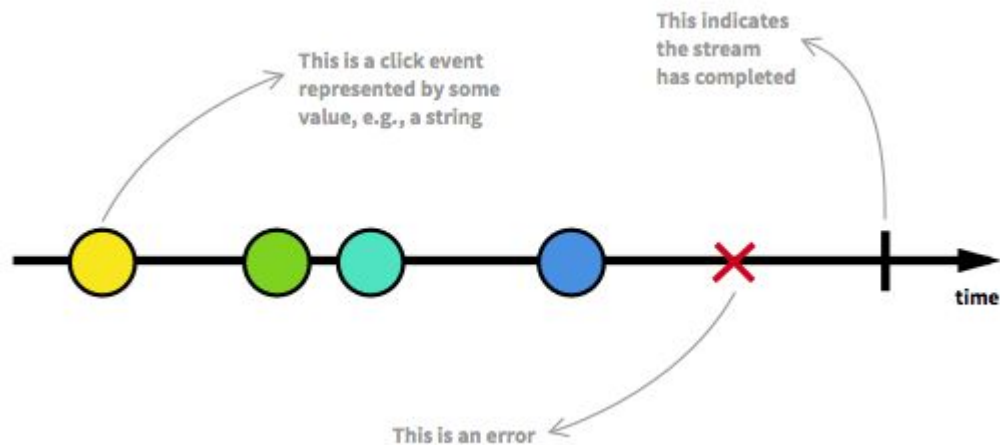
# Message Driven

*React to event*



**“Reactive programming is programming with asynchronous data streams.”**

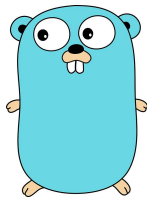
# Reactive Streams



# 02

## Reactive Extensions

## Reactive Extensions



Not only for JavaScript





# 03

## Observables and Subjects

## What are Observables?

- Data producer
- Observer - has `next()`, `error()` and `complete()` methods
- You can subscribe to him
- When it's done, it's done! No reuse after `unsubscribe()`, `error()` or `complete()`

## What are Subjects?

- Like Observable
- Difference:
  - Data producer and consumer
  - It's multicast
  - Has a state

## Observables

**Hot and cold  
Observables**

## Cold Observables

```
// Does not run until someone USES it  
let names = http.get('/contacts.json');
```

# Cold Observables

| `async`

## Hot Observables

```
const socket = new WebSocket('ws://myUrl');

const source = new Observable((observer) => {
  socket.addEventListener('message', (e) => observer.next(e));
});
```

Subscribe only once!

# 04

## Operators



# Operators

## RxJS 5 Operators By Example

A complete list of RxJS 5 operators with clear explanations, relevant resources, and executable examples.

*Prefer a split by operator type?*

### Contents (In Alphabetical Order)

- audit
- auditTime
- buffer
- bufferCount
- bufferTime ★
- bufferToggle
- bufferWhen
- catch / catchError ★
- combineAll
- combineLatest ★
- concat ★
- concatAll
- concatMap ★
- concatMapTo
- create
- debounce
- debounceTime ★
- defaultIfEmpty
- delay
- delayWhen
- distinctUntilChanged ★

- do / tap ★
- empty
- every
- exhaustMap
- expand
- filter ★
- finalize / finally
- first
- forkJoin
- from ★
- fromEvent
- fromPromise ★
- groupBy
- ignoreElements
- interval
- last
- let
- map ★
- mapTo
- merge ★
- mergeAll
- mergeMap / flatMap ★
- multicast
- of ★
- partition
- pluck
- publish
- race
- range

- retry
- retryWhen
- sample
- scan ★
- share ★
- shareReplay ★
- single
- skip
- skipUntil
- skipWhile
- startWith ★
- switchMap ★
- take ★
- takeUntil ★
- takeWhile
- throttle
- throttleTime
- throw
- timeout
- timer
- toPromise
- window
- windowCount
- windowTime
- windowToggle
- windowWhen
- withLatestFrom ★
- zip

★ - commonly used

# Operators



## Operators

- Don't be scared!
- Use known operators!
- When no operator fits:
  - Develop imperative in the `subscribe()` function



# Operators

```
let names = data.map( data =>  
  data.name  
);
```

# Operators

```
// Try 5 times to get the data  
let resilientNames = names.retry(5);
```

# Operators

*// LearnRxJS*

*<https://www.learnrxjs.io/>*

# Operators

*// Which operator do I use?*

***<https://xgrommx.github.io>***

# Operators

*// RX Marbles*

*<http://rxmarbles.com/>*



*// Show us some examples!!*

## Lessons learned - “Simple Films”

- `HttpClient` API is observable
- `subscribe()` to execute
- `pipe` to operators
- `map` to transform
- `catchError`



## Lessons learned - “Wikipedia & Error Isolation”

- Reactive Forms API is observable
- `valueChanges`
- `pipe` many operators to transform



## Lessons learned - “takeUntil & async pipe”

- `unsubscribe()` otherwise MemoryLeaks
- `async` cares to `subscribe()` & `unsubscribe()`

