

Reactive Angular with RxJS

01 - Introduction

Peter Kassenaar –
info@kassenaar.com

Peter Kassenaar

- Trainer, author, developer – since 1996
- Specialty: “*Everything JavaScript*”
- JavaScript, ES6, Angular, NodeJS, TypeScript, jQuery, PhoneGap, Ionic

www.kassenaar.com

info@kassenaar.com

Twitter: [@PeterKassenaar](https://twitter.com/@PeterKassenaar)



Belastingdienst



Angulartraining.nl

Home Training Dates Information Contact

2018 dates now available!

```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: './home/home.module#HomeModule' },
  { path: 'training', loadChildren: './training/training.module#TrainingModule' },
];
const config: ExtraOptions = {
  enableTracing: false,
  preloadingStrategy: PreloadAllModules
};
@NgModule({
  imports: [RouterModule.forRoot(routes, config)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



World-class Angular training in Dutch and English

Live classrooms - focused on today's developers

LEARN MORE SIGN UP!

www.angulartraining.nl

Github repo – slides & example code

The screenshot shows a GitHub repository page for 'PeterKassenaar/reactive'. The repository has 1 branch and 0 tags. It contains files: .gitignore, LICENSE, and README.md. The README.md file is open, showing the content: 'Slides en voorbeeldcode bij de training Reactive Angular - maart 2021'. The repository has 1 commit by PeterKassenaar, made 3 minutes ago. The commit message is 'Initial commit'. The repository has 0 stars, 0 forks, and 1 unwatcher. The 'About' section mentions 'Slides en voorbeeldcode bij de training Reactive Angular - maart 2021'. The 'Releases' section says 'No releases published' and 'Create a new release'. The 'Packages' section says 'No packages published' and 'Publish your first package'. The bottom navigation bar includes links for GitHub, Inc., Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

PeterKassenaar/reactive: Slides & example code

github.com/PeterKassenaar/reactive

PeterKassenaar / reactive

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

PeterKassenaar Initial commit d008d81 3 minutes ago 1 commit

.gitignore Initial commit 3 minutes ago

LICENSE Initial commit 3 minutes ago

README.md Initial commit 3 minutes ago

README.md

reactive

Slides en voorbeeldcode bij de training Reactive Angular - maart 2021

About

Slides en voorbeeldcode bij de training
Reactive Angular - maart 2021

Readme

MIT License

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

<https://github.com/PeterKassenaar/reactive>

About you...



About you...

Knowledge of **Angular**, (mobile/web-) apps?

How long have you worked with Angular yet?

Tell us a little bit about your **projects**.

What are your **expectations** of this course?

On the website...

Training Reactive Angular met RxJS

3 dagen | € 2.199 | Eerstvolgende startdatum 6 februari

[Overzicht](#) [Inleiding](#) **Modulen** [Extra](#) [Startdata & Locaties](#) [Tarieven](#) [Virtuele training](#)

Training Reactive Angular met RxJS: Modulen

Tijdens de Training Reactive Angular met RxJS komen de volgende onderwerpen aan bod:

- Reactive Extensions for Angular
 - Introductie Reactive Extensions
 - Observable en Observer patronen
 - Subjects
 - Asynchrone Calls
 - Observables combineren
 - Error Handling
- Change Detection
 - Zones
 - Werking van Change Detection
 - Immutables and Observables
- State management met Redux
 - Introductie Redux
 - Store, Actions en Reducers
- RxJS
 - RxJS gebruiken in Redux
 - Async Pipes
 - Tools
- Module Loading en Bundling
 - Dynamic Module Loading

Agenda - 3 days

- Inventarization...
- The observable design pattern / observable streams
 - `.subscribe()`
 - Working with `HttpClient` and `async pipe`
 - Various operators
- Managing state
 - Concepts
 - Building a custom store from scratch
 - Using `@ngrx/store`
- Change Detection
- Advanced routing / routing events / routing guards

Material

- Software (Node.js + Angular + Editor + libraries)
- Handouts (PDF, Github)
- Workshops (Github)
- Websites (online)



angular.io/

Labs and example code

- 'Reactive Angular', not *one technique*:
various Repos
 - <https://github.com/PeterKassenaar/voorbeeldenAngular2>
 - <https://github.com/PeterKassenaar/AngularAdvanced>
 - <https://github.com/PeterKassenaar/ng-rxjs-operators>
 - ...
- Running the labs/examples, small projects
 - npm install, npm start)

Agenda

22, 23, 24 March 2021 – Mo – Wed.

~09:30 start

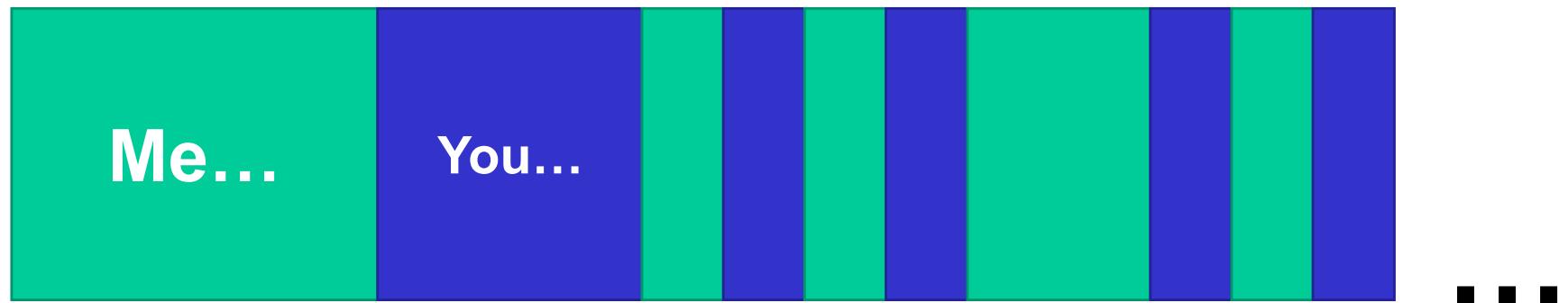
~ 11:00 coffee break

~ 12:30 Lunch Break

~ 15:00 coffee/tea break

~16:00-16:15 wrap-up

Overall process



Questions?



RxJS Observables

Writing an Observable from scratch – using Angular

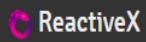
In this module:

- Introduction to the observable design pattern
- Creating observables from scratch
- Using observables for Http communication
- The async pipe
- Communicating with live API's
- More on RxJS Operators

What is RxJS?

- Lots of actions now work **asynchronous**
- Other frameworks – **Promises**
- Angular – **Observables**

“Programming with observable streams”



An API for asynchronous
with observable streams

Choose your platform

<http://reactivex.io/>



Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Ruby: Rx.rb
- Python: RxPY
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift

ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

DOCUMENTATION

Observable

Operators

Single

Combining

LANGUAGES

RxJava[®]

RxJS[®]

Rx.NET[®]

RxCPP

RESOURCES

Tutorials

COMMUNITY

GitHub[®]

Twitter[®]

Others

Why Observables?

"We can do much more with observables than with promises.

With observables, we have a whole bunch of operators to pull from, which let us customize our streams in nearly any way we want."

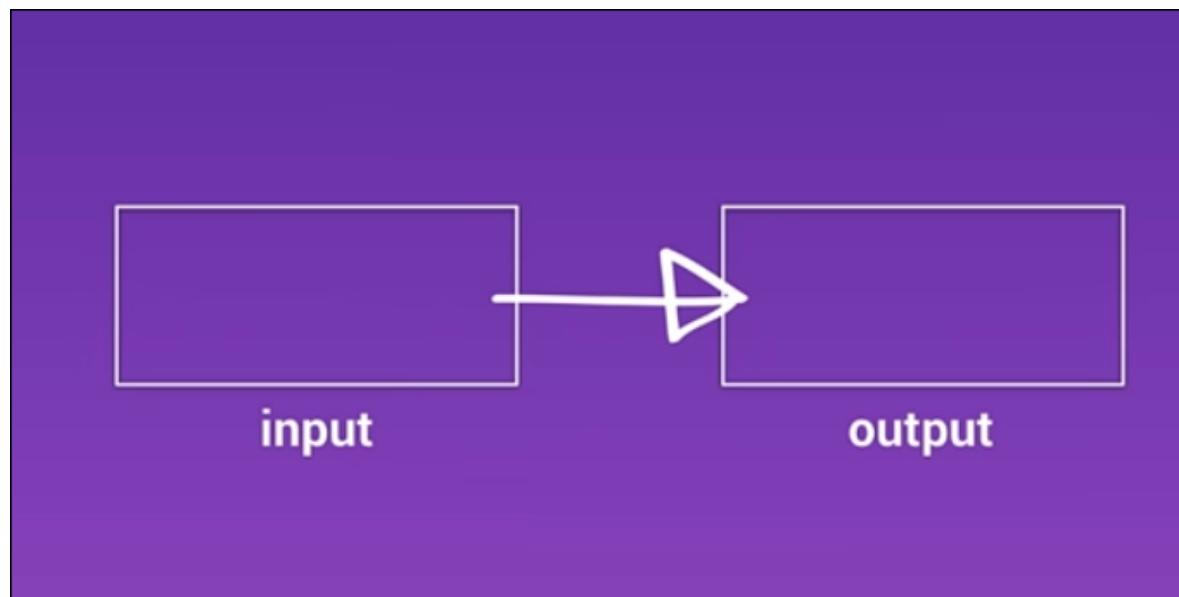
<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>

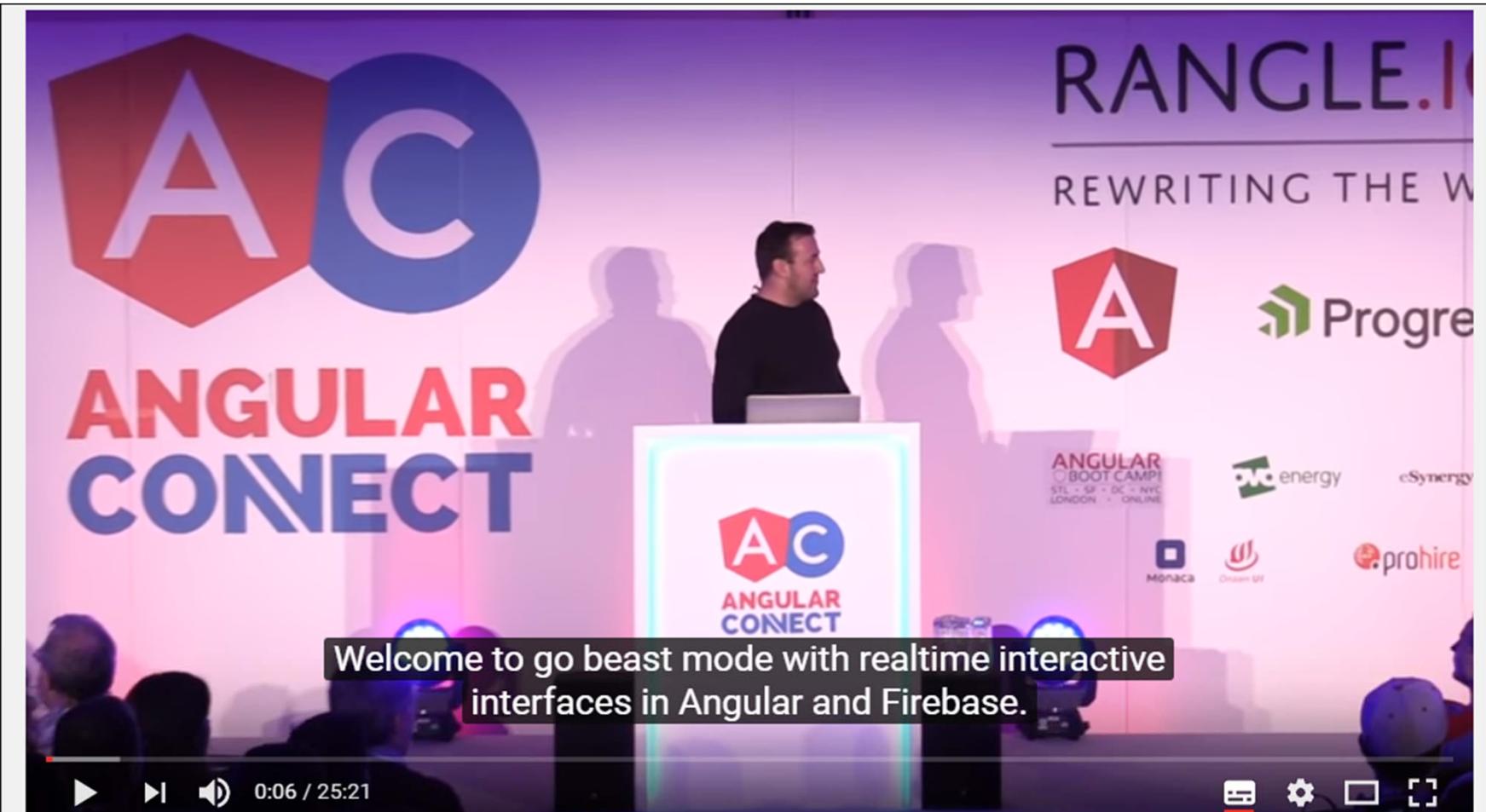
Observables en RxJs

- “Reactive Programming”
 - *“Reactive programming is programming with asynchronous data streams.”*
 - <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- Observables provide extra options, as opposed to Promises
 - Mapping
 - Filtering
 - Combining
 - Cancel
 - Retry
 - ...
- Therefore: no more `.success()`, `.error()` and `.then()` chaining!

How do observables work

- First - *The Observable Stream*
- Later - all 10.000 operators...
- Traditionally:



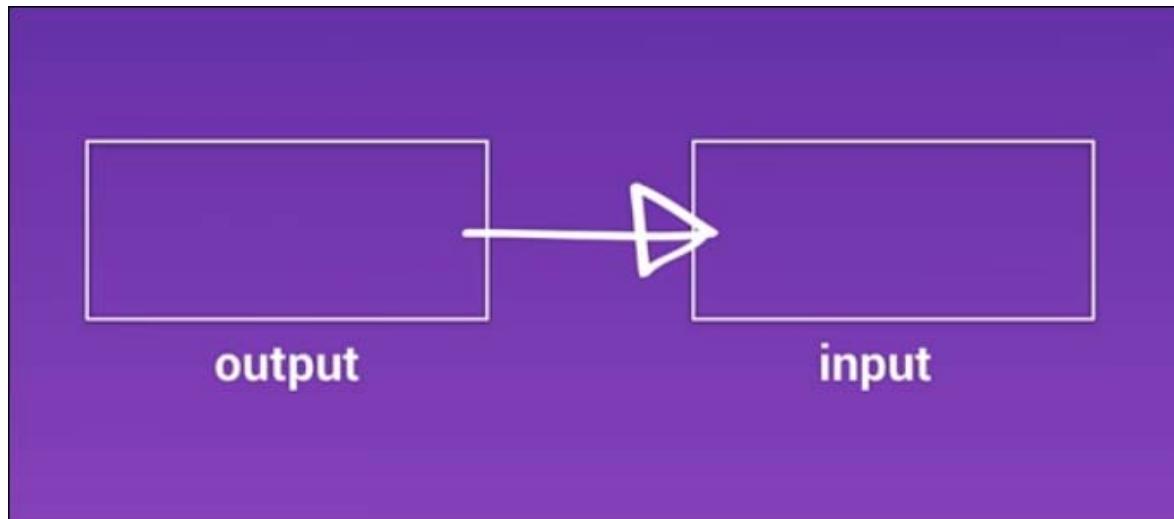


Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

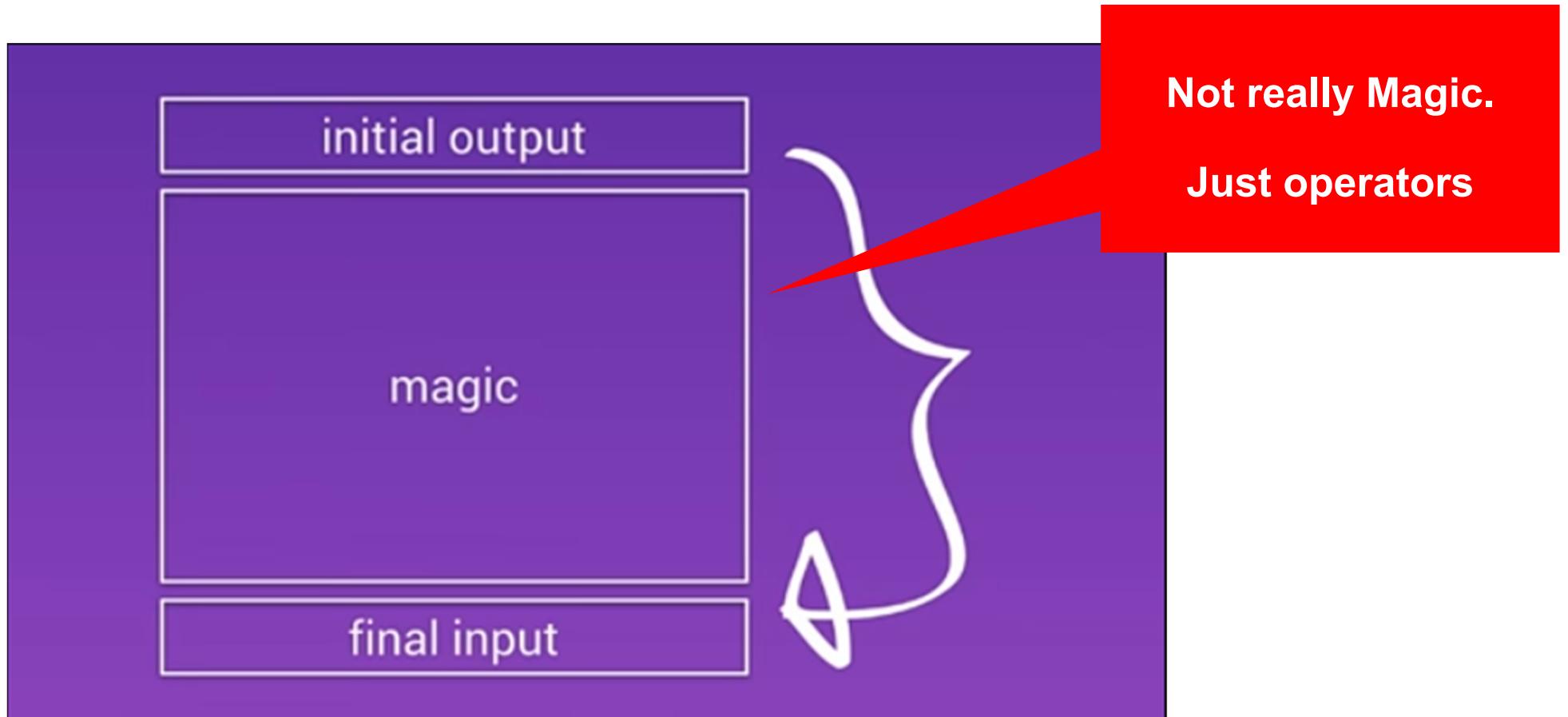
<https://www.youtube.com/watch?v=5CTL7aqSvJU>

<https://youtu.be/5CTL7aqSvJU?t=4m31s>

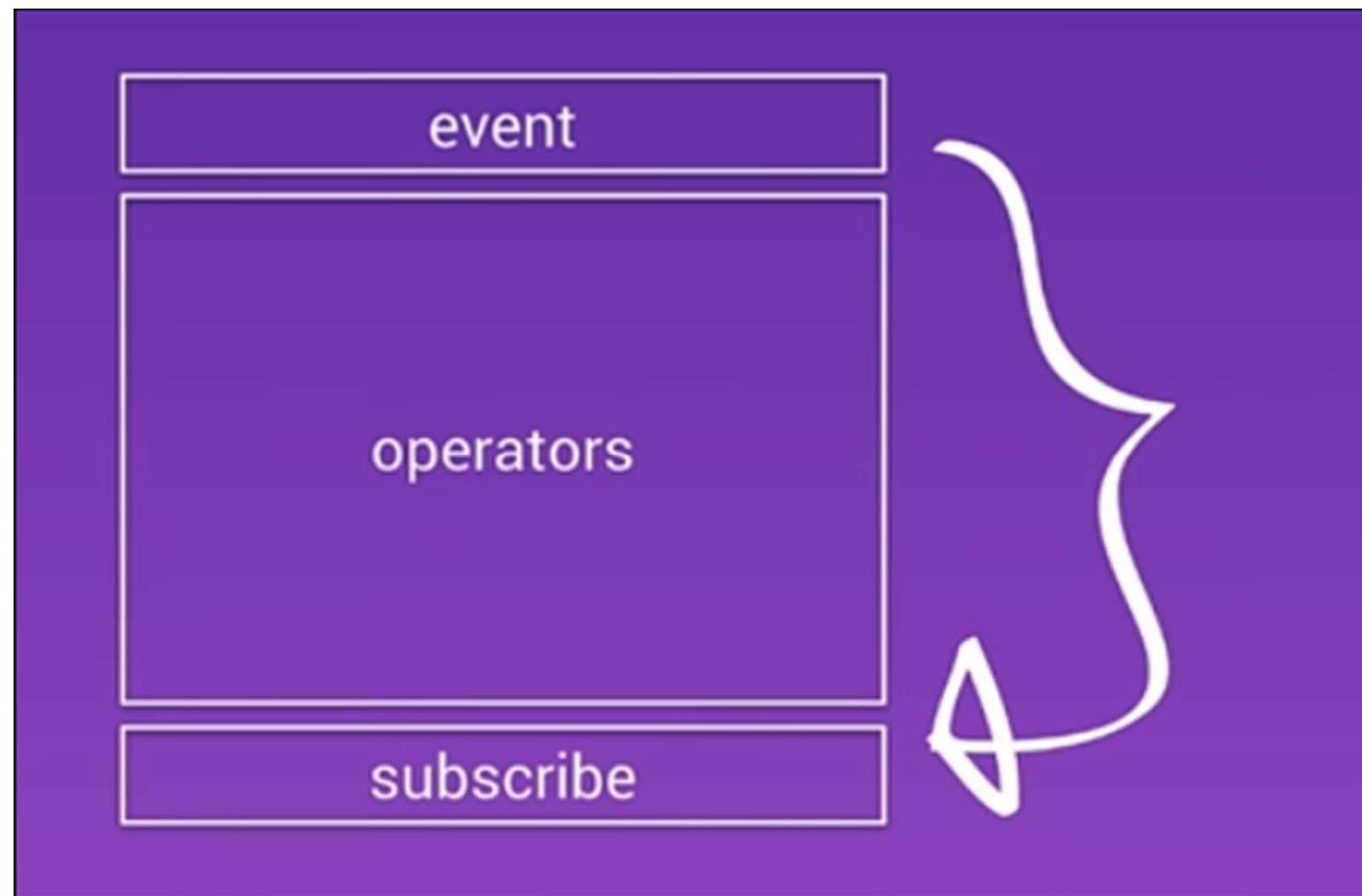
- With Observables -
 - a system, already outputting data,
 - Subscribe to that data
- "trade Output for Input"
- "Push vs. Pull"



"The observable sandwich"



Subscribe to events

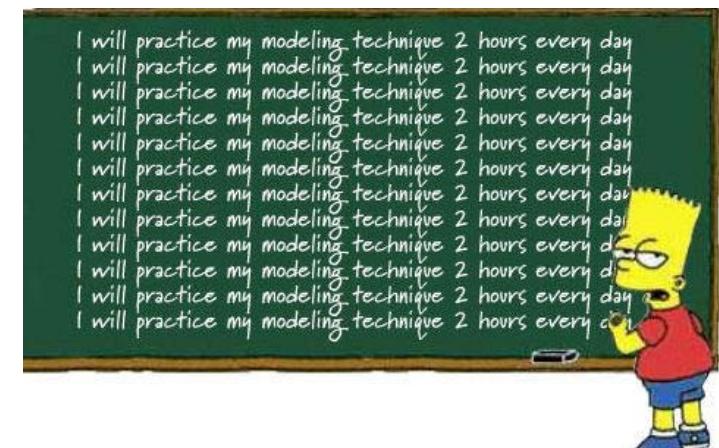


Project: ../180-observables-from-scratch



Workshop

- Create your own Angular app
- Give it some UI, add for example a textbox and a button
- Create observables, based on the DOM-elements
 - For example – handle button clicks, handle text input
- Subscribe to the observables, updating the UI on changes
- Generic example `../180-observable-from-scratch.`
 - In the `\advanced` repo



Extra workshop

- Pass the text from the textbox to *another* component and show it there
 - Working with observables as you now know
 - Working with `@Input()` parameters



Async Http-services with RxJS/Observables

Reactive programming with asynchronous streams

Async Services

- Getting static data: *synchronous* action
- Working with HttpClient: *asynchronous* action
- Angular 1/Other: Promises
- Angular 2: Observables

Even though Http always only delivers one result, they went with RxJS !

Example - ../201-services-http

- In the /VoorbeeldenAngular2 repo
- Other examples:
 - ../203-services-cache
 - ../204-services-apiService
 - ../205-services-http-CRUD

In code:

```
this.http.get<City[]>('assets/data/cities.json')
  .pipe(
    delay(...),
    map(...)
  )
  .subscribe((result) => {
    //... Do something
  });

```

Initial Output

Optioneel:
operator(s)

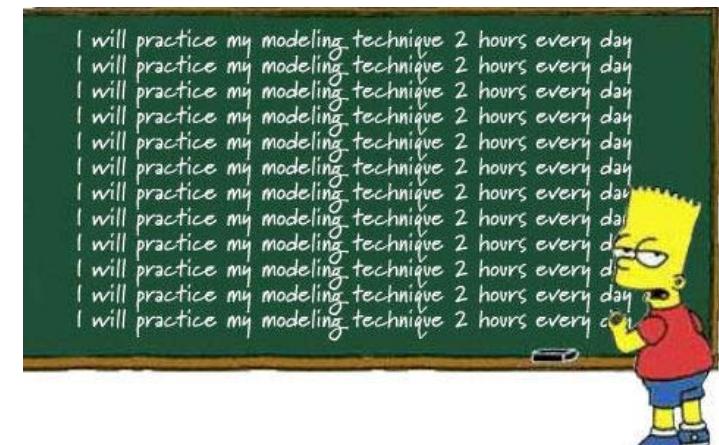
Final Input

Also: import HttpClientModule in @NgModule

- *// Angular Modules*
...
• **import {HttpClientModule} from '@angular/common/http';**
// Module declaration
`@NgModule({
 imports : [BrowserModule, HttpClientModule],
 declarations: [AppComponent],
 bootstrap : [AppComponent],
 ...
})
export class AppModule {
}`

Workshop

- Bekijk het voorbeeld in /201_services_http
 - In de fundamentals-repo
- Maak een eigen .json-bestand en importeer dit in je applicatie.
- Subscribe je op het resultaat uit de JSON-file en toon dit in de UI
- Denk aan het importeren van HttpClientModule in je applicatie
- Optional: verdeel de logica over een service/component





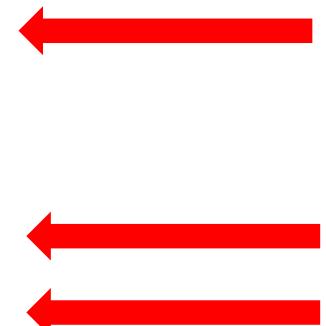
More on .subscribe()

subscribe() in your observable – lots of options

Subscribe - only once per block!

- Three parameters:
 - success()
 - error() – Optional!
 - complete() – Optiona!

```
this.cityService.getCities()  
  
.subscribe(cityData => {  
    this.cities = cityData;  
},  
err => console.log(err),  
()=> console.log('Getting cities complete...')  
)
```



Best practices

- *Specific* solution?
 - Use the `.subscribe()` parameters **in the component**
- *Generic* solution?
 - Use the `.pipe()` option **in the service**
- Mapping, Error Handling, Complete handling?
 - Use operators in the service

RxJS-operators in the service

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {map, delay, takeUntil, ...} from "rxjs/operators";

@Injectable()
export class CityService {

    constructor(private http: HttpClient) {

    }

    // return all cities
    getCities(): Observable<Response> {
        return this.http.get('shared/data/cities.json')
            .pipe(...);
    }
}
```

Useful operators

- RxJS operators are (mostly) like Array operators
- Perform actions on a stream of objects
- Grouped by subject
 - Creation operators
 - Transforming
 - Filtering
 - Combining
 - Error Handling
 - Conditional and Boolean
 - Mathematical
 - ...

Ben Lesh on observables in RxJS 6.0

The two you care about

- rxjs
 - **Types:** Observable, Subject, BehaviorSubject, etc.
 - **Creation methods:** fromEvent, timer, interval, delay, concat, etc.
 - **Schedulers:** asapScheduler, asyncScheduler, etc.
 - **Helpers:** pipe, noop, identity, etc
- rxjs/operators
 - **All operators:** map, mergeMap, takeUntil, scan, and so one.

@benlesh



Introducing RxJS6! - Ben Lesh

<https://www.youtube.com/watch?v=JCXZhe6KsxQ>

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

The screenshot shows the 'Operators' page of the Learn RxJS website. The left sidebar has a navigation menu with sections like Introduction, LEARN RXJS (Operators, Combination, Conditional, Creation, Error Handling, Multicasting, Filtering, Transformation, Utility, Full Listing, Subjects, Recipes, Concepts), and a footer powered by GitBook. The main content area has a title 'Operators' and a subtitle 'A complete list of RxJS operators with clear explanations, relevant resources, and executable examples.' It includes a link 'Prefer a complete list in alphabetical order?'. Below this is a section titled 'Contents (By Operator Type)' with two lists: 'Combination' (with items like combineAll, combineLatest, concat, concatAll, endWith, forkJoin, merge, mergeAll, pairwise, race, startWith, withLatestFrom, zip) and 'Conditional' (with items like defaultIfEmpty, every, iif). Yellow stars are placed next to some operator names.

Operators

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

Prefer a complete list in alphabetical order?

Contents (By Operator Type)

- Combination
 - [combineAll](#)
 - [combineLatest](#) ★
 - [concat](#) ★
 - [concatAll](#)
 - [endWith](#)
 - [forkJoin](#)
 - [merge](#) ★
 - [mergeAll](#)
 - [pairwise](#)
 - [race](#)
 - [startWith](#) ★
 - [withLatestFrom](#) ★
 - [zip](#)
- Conditional
 - [defaultIfEmpty](#)
 - [every](#)
 - [iif](#)



Async pipe

Automatische `.subscribe()` en `.unsubscribe()`

Async Pipe

- Bij `.subscribe()`, eigenlijk ook `.unsubscribe()` aanroepen.
 - Netjes!
 - Bij HTTP-requests niet beslist nodig, bij andere subscriptions wel, in verband met memory leaks.
- Niet meer zelf `.subscribe()` en `.unsubscribe()` aanroepen:
 - **Gebruik async pipe van Angular**

- In de component:

```
Cities$: Observable<City[ ]>; // Nu: Observable naar Type
```

```
...
```

```
ngOnInit() {  
    // Call naar de service, retourneert Observable  
    this.cities$ = this.cityService.getCities()  
}
```

- In de view:

```
<li *ngFor="let city of cities$ | async">
```

Werken met Live API's

- MovieApp
- examples\210-services-live

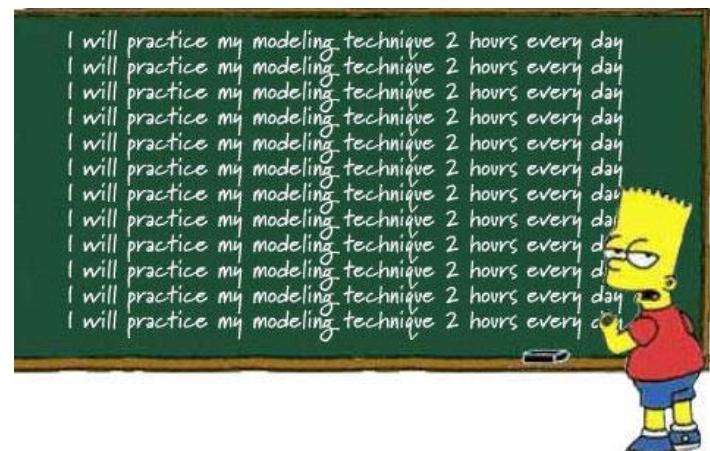


Voorbeeld API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weerbericht)
- <http://randomuser.me/> (random NAW-gegevens)
- <http://ergast.com/mrd/> - Ergast Motor (F1) API
- <http://www.omdbapi.com/> - Open Movie Database
- <http://swapi.co/> - Star Wars API
- Zie ook JavaScript APIs.txt met meer voorbeelden

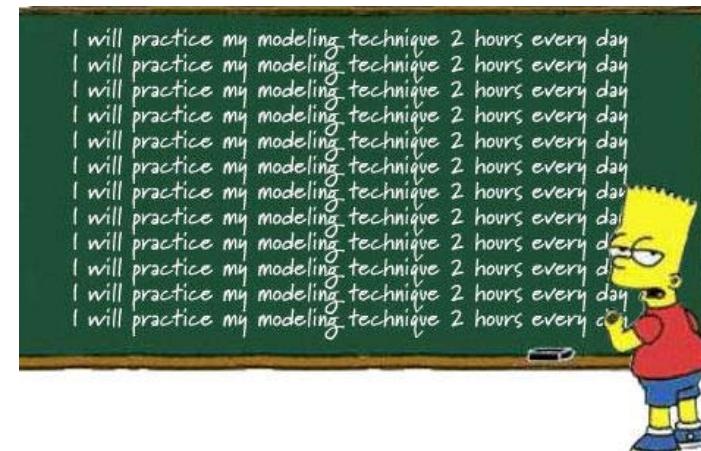
Workshop

- Pick one of your own projects, or see for instance:
 - `./210-services-live`
- Create a small application using one of the API's in the file `JavaScript API's.txt`, using RxJS-calls, for example
 - Pokemon API
 - Kenteken API
 - OpenWeatherMap API
 - ...



Example debounceTime

- debounceTime() – operator, used to delay the execution of the observable
- <https://www.learnrxjs.io/operators/filtering/debounceTime.html>
- Example: .../502-forms-typeahead





More on operators

Transforming your stream in the `.pipe()` function

The Problem with RxJS Operators...

- There are so many of them...

The screenshot shows a website titled "Learn RxJS". The sidebar on the left contains a navigation menu with the following items:

- LEARN RXJS
- Introduction
- Operators (highlighted with a red border)
- Combination
 - combineAll
 - combineLatest
 - concat
 - concatAll
 - forkJoin
 - merge
 - mergeAll
 - race
 - startWith
 - withLatestFrom
 - zip
- Conditional

The main content area features the title "Learn RxJS" and a subtitle "Clear examples, explanations, and resources for RxJS.". Below this is a section titled "Introduction" which contains text about RxJS being one of the hottest libraries in web development and its applications across various frameworks and languages. It also mentions the challenges of learning reactive programming. A "But..." section follows, discussing the difficulty of learning RxJS due to its complexity and the shift from imperative to declarative style.

[Introduction](#)[Operators](#)[Combination](#)[combineAll](#)[combineLatest](#)[concat](#)[concatAll](#)[forkJoin](#)[merge](#)[mergeAll](#)[race](#)[startWith](#)[withLatestFrom](#)[zip](#)[Conditional](#)

Learn RxJS

Clear examples, explanations, and resources for RxJS.

Introduction

RxJS is one of the hottest libraries in web development today. Offering a powerful, functional API dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to spread your knowledge across nearly any language, having a solid grasp on reactive programming and what it can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is hard. There's the multitude of concepts, large API surface area, and fundamental shift in mindset from an imperative to declarative style. This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the official documentation and existing learning material while offering a new, fresh narrative to clear any hurdles and to help you learn RxJS.

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

Start With These

- map
- filter
- scan
- mergeMap
- switchMap
- combineLatest
- concat
- do



10:14 / 39:03



RxJS 5 Thinking Reactively | Ben Lesh



AngularConnect

<https://www.youtube.com/watch?v=3LKMwkuK0ZE>

*"Don't try to "Rx everything". Start with
the operators you know, and go
imperatively from there. There's
nothing wrong with that."*

- Ben Lesh

Example code

PeterKassenaar / **ng-rxjs-operators**

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Examples of using different RxJS operators in an Angular CLI-project

Add topics Edit

3 commits 1 branch 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File	Description	Time
.gitignore	Updated .gitignore. Added yarn support	Latest commit 9ac806d 22 hours ago
src	Added first version of code.	22 hours ago
.angular-cli.json	Added first version of code.	22 hours ago
.editorconfig	Added first version of code.	22 hours ago
.gitignore	Updated .gitignore. Added yarn support	22 hours ago

<https://github.com/PeterKassenaar/ng-rxjs-operators>

This presentation - 2 sections

1. Basic Streams

- Create a stream, or multiple streams from scratch, based on a UI-element
- Subscribe to that stream(s) and do something

2. Operators

Demo the purpose and inner workings of some often used operators

Helpful resource - rxMarbles

RxJS Marbles Interactive diagrams of Rx Observables

Fork me on GitHub

sequenceEqual

COMBINATION OPERATORS

combineLatest

concat

merge

race

startWith

withLatestFrom

zip

FILTERING OPERATORS

debounceTime

debounce

distinct

distinctUntilChanged

elementAt

filter

find

findIndex

first

ignoreElements

last

The diagram illustrates the `combineLatest` operator. It shows two parallel streams: one with elements 1, 2, 3, 4, 5 and another with elements A, B, C, D. The resulting stream contains pairs of elements: (1A, 2B), (2C, 2D), (3D, 4D), and (5D). The label below the streams is `combineLatest((x, y) => "" + x + y)`.

combineLatest($(x, y) \Rightarrow "" + x + y$)

Built on RxJS v5.0.3

<https://rxmarbles.com/>



Basic streams

Creating observables from scratch

Turn something into an observable

- Basic creation operators
 - `create()` – create an observable manually
 - `from()` – turn an array, promise or iterable into an observable
 - `fromEvent()` – turn an event into an observable
 - `of()` – turn a variable into an observable, emit it's value('s) and complete
- <https://www.learnrxjs.io/learn-rxjs/operators/creation>

The screenshot shows the 'Learn RxJS' website interface. On the left, there is a sidebar with a navigation menu. The 'Operators' section is expanded, and 'Creation' is selected, indicated by a blue background. Other options in the 'Operators' menu include 'Combination' and 'Conditional'. Below 'Creation' are three sub-options: 'ajax', 'create', and 'defer'. The main content area has a title 'Creation' and a descriptive paragraph: 'These operators allow the creation of an observable from nearly anything. From generic to specific use-cases you are free, and encouraged, to turn [everything into a stream](#)'. Below this is a 'Contents' section with a list of operators: 'ajax ★', 'create', 'defer', 'empty', and 'from ★'.

Learn RxJS

Introduction

LEARN RXJS

Operators

Combination

Conditional

Creation

ajax

create

defer

Creation

These operators allow the creation of an observable from nearly anything. From generic to specific use-cases you are free, and encouraged, to turn [everything into a stream](#).

Contents

- ajax ★
- create
- defer
- empty
- from ★

Creating an observable from a textbox

```
<input class="form-control-lg" type="text" #text1  
      id="text1" placeholder="Text as a stream">  
<p>{{ textStream1 }}</p>
```

```
textStream1 = 'Type some text above...';  
@ViewChild('text1', {static: true}) text1; // two parameters for @ViewChild()  
  
// Suggestion: break the ngOnInit() up in smaller functions  
ngOnInit(): void {  
  this.onTextStream1();  
}  
  
onTextStream1() {  
  fromEvent(this.text1.nativeElement, 'keyup')  
    .subscribe((event: any) => this.textStream1 = event.target.value);  
}
```

Result

Basic stream: we subscribe to chars coming from the textbox:

Text as a stream

Type some text above...

Basic stream: we subscribe to chars coming from the textbox:

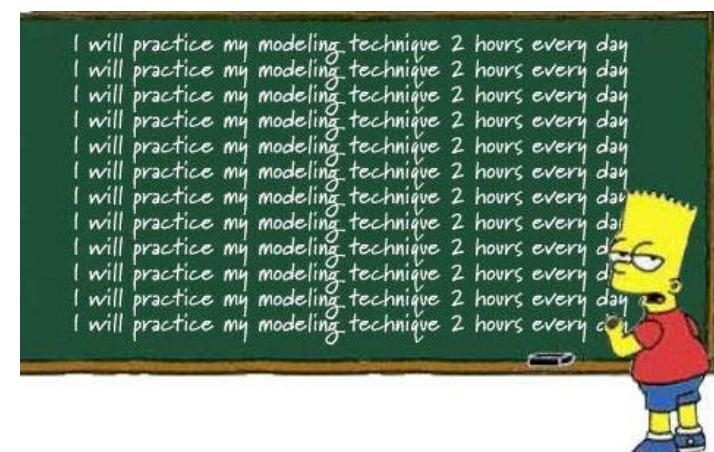
This is a stream...

This is a stream...



Workshop

- Create a textbox.
 - Add items that are typed in, to a Todo-list
 - Use an observable to subscribe to keyup-events.
- Capture key events, test if the key pressed is the Enter-key
 - If it's Enter, Add the current value of the textbox to a static array `todoItems[]`
- Example `.../basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component



Using the pipe()

- We can transform the stream by adding operators to the pipeline
- Inside the pipeline we calculate new values and bind them to the UI, using generic attribute binding [...]

```
<button #btnRight class="btn btn-secondary">Right</button>
Mario is moved by observable streams from the button click.
<div>
  
</div>
```

Inside our class

```
// mario
position: any;
@ViewChild('btnRight', {static: true}) btnRight;
```

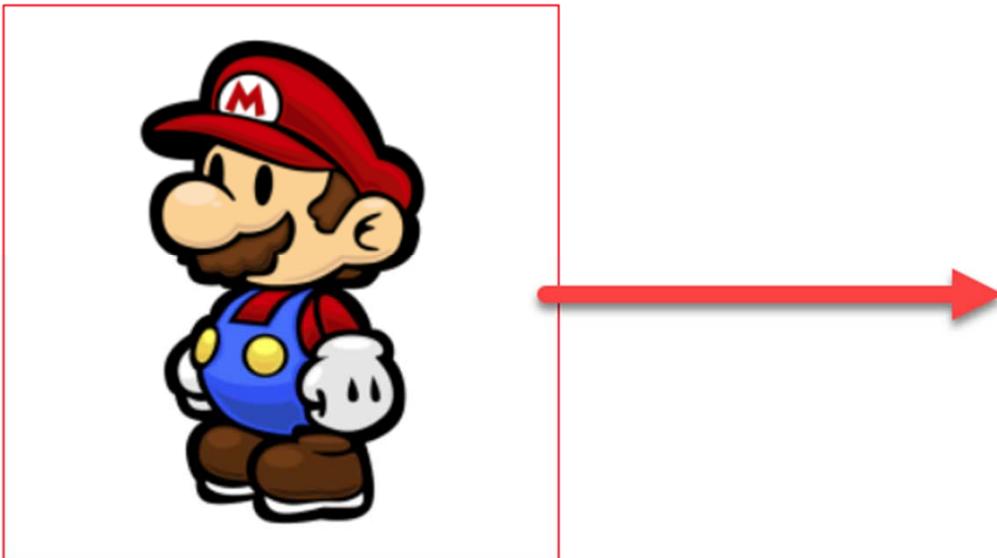
```
fromEvent(this.btnRight.nativeElement, 'click')
.pipe(
  map(event => 10), // 1. map the event to a useful value, in this case 10px
  startWith({x: 100, y: 100}), // 2. start with an object of { 100, 100}.
  scan((acc: any, current: number) => { // 3, use the scan operator as reducer function.
    return {
      x: acc.x + current,
      y: acc.y
    };
  })
)
.subscribe(result => {
  this.position = result;
});
```

The result of the previous operator is the input of the next operator in the pipeline

Result

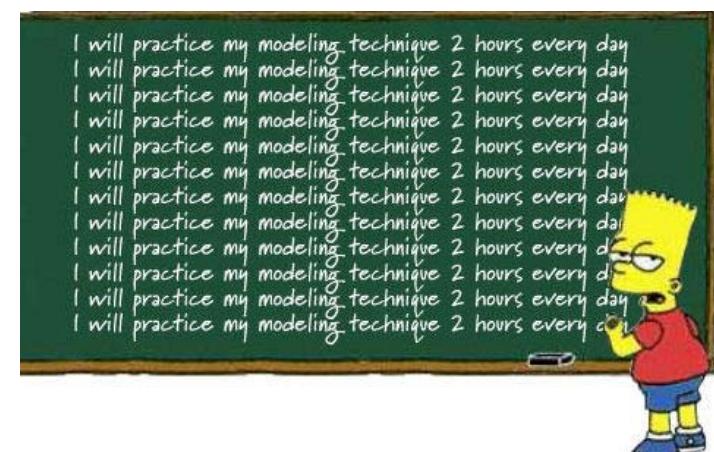
Right

Mario is moved by observable streams from the button click.



Workshop

- Create button that moves Mario to the left.
- Example `./basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component





Combining streams

There are **a lot** of options to combine multiple streams into one stream

Options for combining streams

- `combineLatest()` - When any observable emits a value, emit the last emitted value from each
- `concat()` - Subscribe to observables in order as previous completes
- `forkJoin()` - When all observables complete, emit the last emitted value from each
- `merge()` - Turn multiple observables into a single observable
- `startWith()` - Emit given value first
- <https://www.learnrxjs.io/learn-rxjs/operators/combination>

The screenshot shows the Learn RxJS website's navigation bar on the left and a detailed page on the right.

Navigation Bar:

- Learn RxJS logo
- Search icon
- Introduction
- LEARN RXJS
- Operators (dropdown menu)
- Combination (selected item in dropdown)
- combineAll
- combineLatest
- concat
- concatAll
- endWith
- forkJoin

Page Content:

Combination

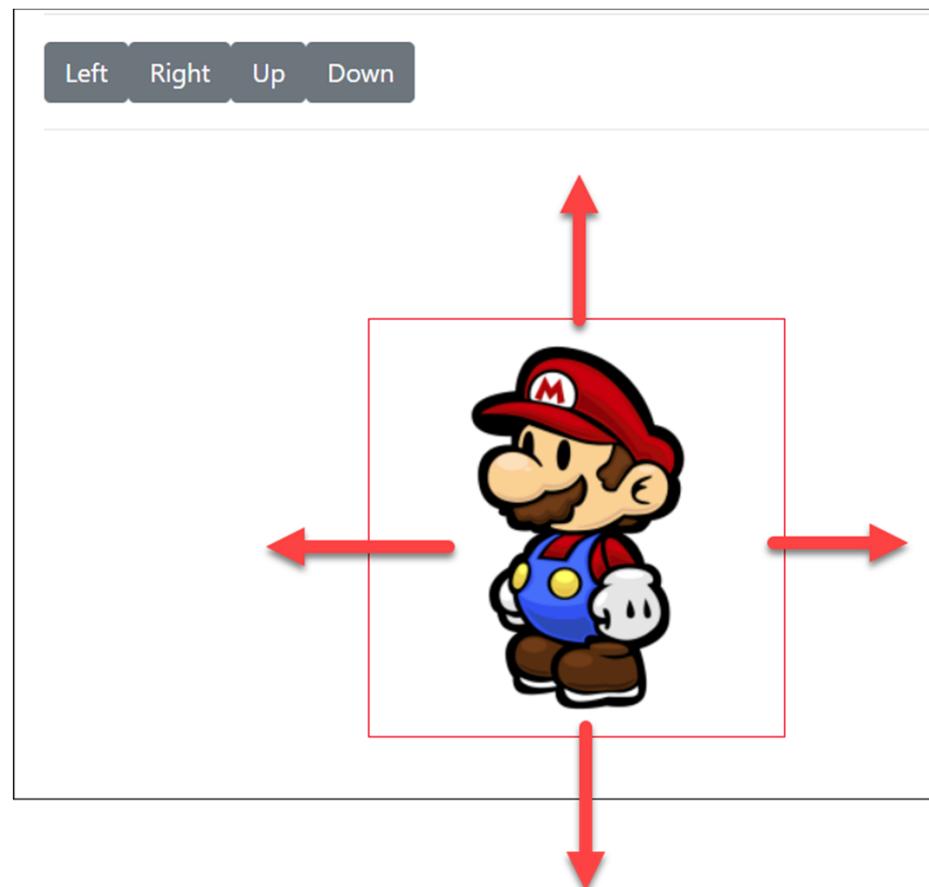
The combination operators allow the joining of information from multiple observables. Order, time, and structure of emitted values is the primary variation among these operators.

Contents

- [combineAll](#)
- [combineLatest ★](#)
- [concat ★](#)
- [concatAll](#)
- [endWith](#)
- [forkJoin](#)

Use case: move Mario

- Create four buttons to move Mario in different directions
 - all combined in one stream and one subscription



Template

```
<button #btnLeft> Left </button>
<button #btnRight> Right </button>
<button #btnUp> Up </button>
<button #btnDown> Down </button>
<hr>

<img id="mario" #mario
      [style.left] = "position.x + 'px'"
      [style.top] = "position.y + 'px'"
      src = "../assets/img/mario-1.png" alt = "Mario">
```

```

position: any;
@ViewChild('btnLeft', {static: true}) btnLeft;
@ViewChild('btnRight', {static: true}) btnRight;
@ViewChild('btnUp', {static: true}) btnUp;
@ViewChild('btnDown', {static: true}) btnDown;

ngOnInit(): void {
  // Create multiple streams
  const right$ = fromEvent(this.btnRight.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: 10};
      }) // 10 px to the right
    );
  const left$ = fromEvent(this.btnLeft.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: -10};
      }) // -10 px to the left
    );
  ...
  // combine our streams
  merge(right$, left$, up$, down$)
    .pipe(
      startWith({x: 200, y: 100}),
      scan((acc: any, current: any) => {
        return {
          x: acc.x + (current.direction === 'horizontal' ? current.value : 0),
          y: acc.y + (current.direction === 'vertical' ? current.value : 0)
        };
      })
    ).subscribe(result => {
      this.position = result;
    });
}

```

Create 4 different streams

Merge the streams

One subscriber



Sequencing streams

Use streams to start other streams so you can use the combined behavior

Use case: drag & drop

- We want to move Mario around with the mouse: **drag-n-drop**
- We compose different streams for that:
 - `down$`, when the mouse is pressed
 - `move$`, when the mouse is moved, return the current mouse position
 - `up$`, when the mouse is released
- Again, we have only one (1) subscription
 - **Subscribe** to the `down$`. This is the initiator
 - After the initial event, **switch** to the `move$`: using the `switchMap()` operator
 - But only **until** the `up$` event occurs!
 - Then complete the observable and release Mario
- This translates to the following code:

```

// correction factor for image and current page. Your mileage may vary!
const OFFSET_X = 180;
const OFFSET_Y = 280;

// 1. With Drag and drop, first you capture the mousedown event
const down$ = fromEvent(document, 'mousedown');

// 2. What to do when the mouse moves
const move$ = fromEvent(document, 'mousemove')
  .pipe(
    map((event: any) => {
      return {x: event.pageX - OFFSET_X, y: event.pageY - OFFSET_Y}; // OFFSET as correction factor
    })
  );

// 3. Capture the mouseup event
const up$ = fromEvent(document, 'mouseup');

// 4. extend the down$ stream and subscribe
down$
  .pipe(
    // IF the down$-event happens, we are no longer interested in it. Instead,
    // we switch the focus to the move$ event.
    // BUT: we are only interested in the move until the mouse is released again.
    // So we add *another* pipe and use the takeUntil() operator.
    switchMap(event => move$.pipe(
      takeUntil(up$)
    )),
    startWith(this.position)
  )
  .subscribe(result => this.position = result);
}

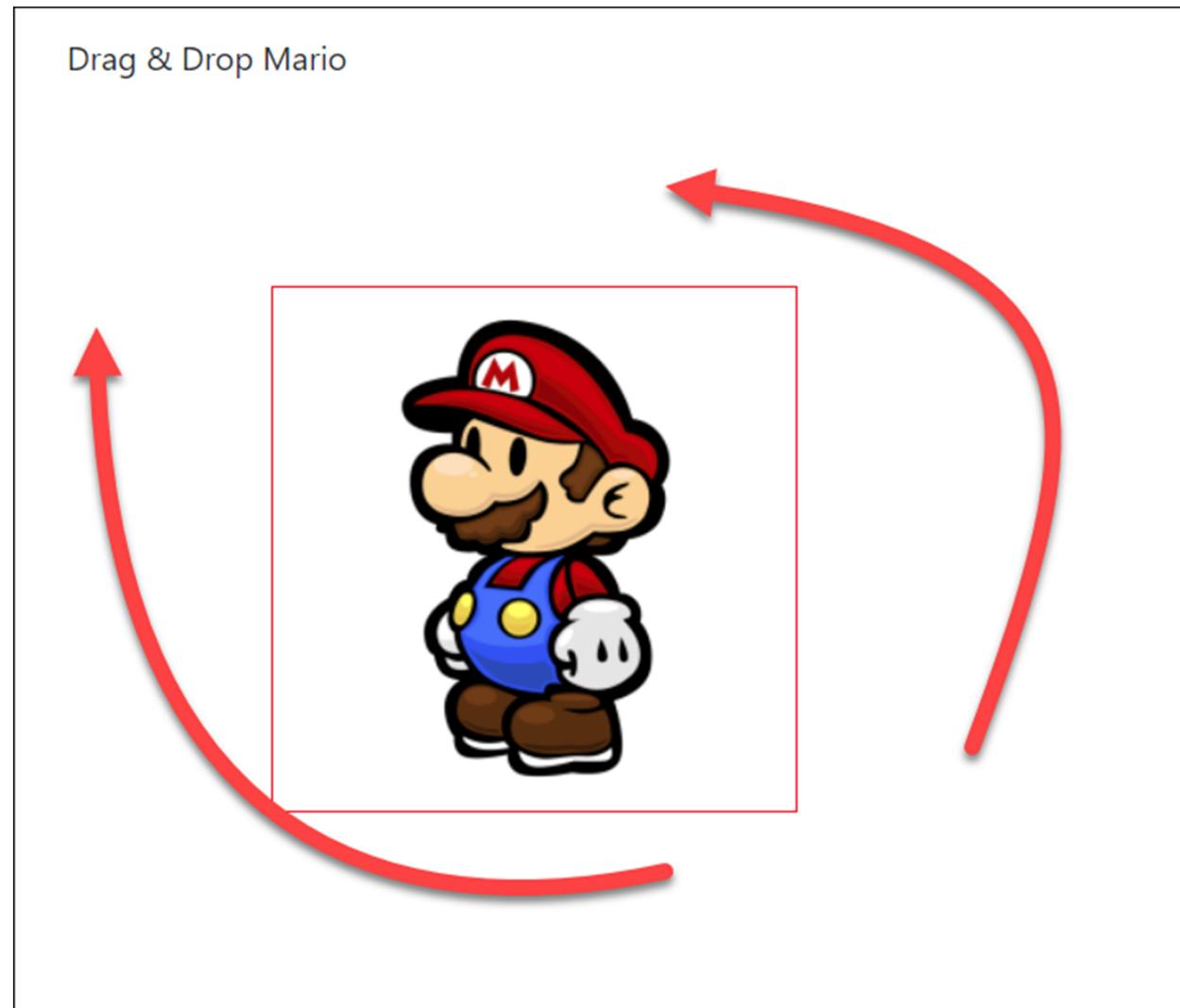
```

Compose streams

Switch execution to another observable

Subscribe and update position

Result



`../streams/drag-drop-stream.component.html|ts`

An autocomplete/typeahead demo

An autocomplete/typeahead demo

bel



Belarus
Minsk



Belgium
Brussels



Belize
Belmopan



Palau
Ngerulmud

Template

As per usual – little HTML

```
<h4>An autocomplete/typeahead demo</h4>
<!--Get the keyword, the class is subscribed to this inputbox-->
<input type="text" placeholder="Country name..." 
       #typeahead class="form-control-lg">
<hr/>
<!--Results-->
<ul class="list-group">
  <li class="list-group-item" *ngFor="let country of countries$ | async ">
    
    <p>
      <strong>{{ country.name }}</strong><br>
      {{ country.capital }}
    </p>
  </li>
</ul>
```

This time: using the
async pipe

Code

```
interface ICountry {  
  name: string;  
  capital: string;  
  flag: string;  
}  
  
const errorCountry: ICountry = {  
  name: 'Error',  
  capital: 'Not found',  
  flag: ''  
};
```

Creating interface and simple error message

```
@ViewChild('typeahead', {static: true}) typeahead;  
countries$: Observable<ICountry[]>;  
  
constructor(private http: HttpClient) {  
}
```

Inside the component class

```
ngOnInit(): void {  
  this.countries$ = fromEvent(this.typeahead.nativeElement, 'keyup')  
    .pipe(  
      filter((e: any) => e.target.value.length >= 2),  
      debounceTime(400),  
      map((e: any) => e.target.value),  
      distinctUntilChanged(),  
      switchMap(keyword => this.getCountries(keyword))  
    );  
}
```

Main method. See example code for comments

Fetching the countries

```
getCountries(keyword): Observable<ICountry[]> {
  // 7. Create the actual http-call.
  return this.http
    .get<ICountry[]>(`https://restcountries.eu/rest/v2/name/${keyword}?fields=name;capital;flag`)
    .pipe(
      // 8. catch http-errors and return a 'not found' country
      catchError(err => {
        console.log(err);
        return of([errorCountry]);
      })
    );
}
```

Documentation:

- filter: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/filter>
- debounceTime: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/debounceTime>
- distinctUntilChanged: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/distinctUntilChanged>
- switchMap: <https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap>
- catchError: https://www.learnrxjs.io/learn-rxjs/operators/error_handling/catch

Workshop

- Search movies in the Open Movie Database.
 - [https://www.omdbapi.com/?apikey=f1f56c8e&s=\[keyword\]](https://www.omdbapi.com/?apikey=f1f56c8e&s=[keyword])
 - Create an AutoComplete inputfield for movie titles.
 - For instance, if you type 'ava...' it should show movies like Avatar, and more
- Example [.../basic-stream/typeahead.component.ts](#)
 - Start from scratch or expand this component

OMDb API Usage Parameters Examples Change Log API Key

OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a one-time donation or become a patron.

Attention Users

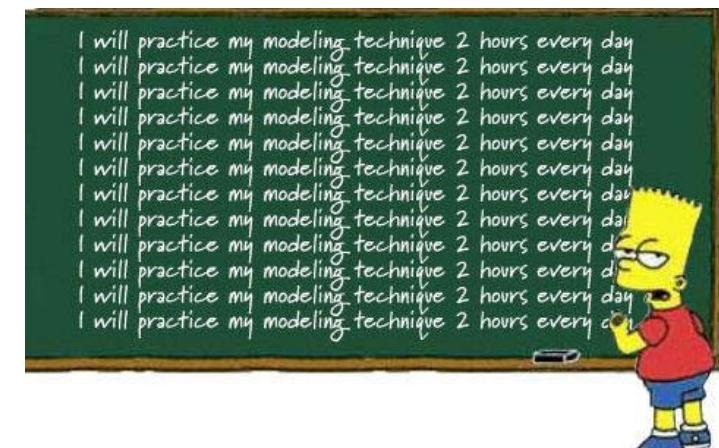
04/08/19 - Added support for eight digit IMDB IDs.

01/20/19 - Supressed adult content from search results.

01/20/19 - Added Swagger files ([YAML](#), [JSON](#)) to expose current API abilities and upcoming REST functions.

Poster API
The Poster API is only
Currently over 280.00
with resolutions up to







More operators

Getting familiar with some of the more used operators

map() and mapTo()

```
const source = from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

// .map() - apply a function every emitted output.
source.pipe(
  map((val: number) => val = val * 10)
)
  .subscribe(result => this.mapData.push(result));

// .mapTo() - map the emission to a constant value
source.pipe(
  mapTo('Hello World')
)

.subscribe(result => this.mapToData.push(result));
```

filter()

```
const source      = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
const sourceCities = Observable.from([
  'Haarlem',
  'Breda',
  'Amsterdam',
  'Groningen',
  'Hengelo',
  ...
]);
// filter() - only emit values that pass the provided condition.
// In this case: only even numbers are passed through.
source.pipe(
  filter(val => val % 2 === 0)
)
.subscribe(result => this.filterData.push(result));

// Emit only the cities that starts with an 'H'.
sourceCities.pipe(filter(city => city.startsWith('H')))
.subscribe(result => this.cityData.push(result));
```

scan()

```
const source      = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

```
// .scan() acts as the classic .reduce() function on array's. It takes an
// accumulator and the current value. The accumulator is persisted over time.
source
  .pipe(
    startWith(0),
    scan((acc, curr) => acc + curr)
  )
  .subscribe(result => this.scanData.push(result));
```

concat()

- Concat subscribes to observables *in order*.
- “Only when the first one completes, let me know.”
- Then move on to the next one.
- Use concat() when the order of your observables matters.
- Example code: simulated delay.
 - The second observable (which is finished first) only emits when the first observable completes.

app/shared/services/data.service.ts

```
// constants that are used as pointers to some json-data
const BOOKS: string    = 'assets/data/books.json';
const AUTHORS: string = 'assets/data/authors.json';

getConcatData(): Observable<any> {
  // First call. Simulate delay of 1 second
  const authors = this.http.get(AUTHORS)
    .pipe(
      delay(2000)
    );
  // Second call. Simulate delay of 2 seconds
  const books = this.http.get(BOOKS)
    .pipe(
      delay(1000)
    );
  // return the concatenated observable. It will always deliver
  // FIRST the results of the first call. No matter how long the delay.
  return concat(authors, books);
}
```

[https://rxjs-dev.firebaseio.com/api/operators\(concat](https://rxjs-dev.firebaseio.com/api/operators(concat)

merge()

- Merge combines multiple observables into one single observable.
- It emits *as soon as* it gets a result.
- Use `merge()` when order of observables is not important.

```
getMergeData(): Observable<any> {
  // First call. Simulate delay of 3 seconds, so this observable will emit last.
  const authors = this.http.get(AUTHORS)
    .pipe(
      delay(3000)
    );

  // Second call. Simulate delay of 1 seconds, so this observable will emit first.
  const books = this.http.get(BOOKS)
    .pipe(
      delay(1000)
    );

  // return the merged observable. BOOKS will be delivered first
  return merge(authors, books);
}
```

<https://rxjs-dev.firebaseio.com/api/index/function/merge>

mergeMap()

- Merges the value of an inner observable into an outer observable.
- Need only the *last* value emitted? Use `.switchMap()`.
- Use this for example to retrieve results in a second observable, based on the output of a first observable

.mergeMap() result

See [/app/shared/services/data.service.ts](#) for example code.

We are looking for books by authorName here. But we only have an authorID, not a name.

```
[  
  {  
    "title": "Web Development Library - TypeScript",  
    "author": "Peter Kassenaar",  
    "bookID": 1  
  },  
  {  
    "title": "Web Development Library - Angular",  
    "author": "Peter Kassenaar",  
    "bookID": 2  
  }  
]
```



```
getMergeMapData(authorID: number = 0): Observable<any> {
  // first http-call, outer observable.
  return this.http.get(AUTHORS)
    .pipe(
      tap(response => console.log(response)),
      map((authors: any[]) => {
        console.log(authors);
        // find the correct author, using the array .find() method
        return authors.find((author: any) => author.id === authorID);
      }),
      mergeMap((author: any) => {
        if (author) {
          // second http-call, inner observable
          return this.http.get(BOOKS)
            .pipe(map((books: any[]) => {
              // filter books, bases on authorname we found earlier.
              return books.filter((book: any) => book.author === author.name);
            }));
        } else {
          // nothing found. Return empty array
          return of([]);
        }
      })
    );
}
```

.forkJoin()

- Make multiple (http) requests and return one combined response *once all observables are completed.*
- .forkJoin() returns an array with the last emitted value from each observable.
- Compose results as per your needs

forkJoin() vs combineLatest()

In general, they are pretty similar:

forkJoin()

When all observables complete, emit the last emitted value from each.

combineLatest()

When any observable emits a value, emit the latest value from each.

*“Not only does `forkJoin` require **all input observables to be completed**, but it also returns an observable that produces a single value that is an array of the last values produced by the **input observables**. In other words, **it waits until the last input observable completes, and then produces a single value and completes.**”*

*...which means **you don't have to unsubscribe from a `forkjoin()`***

In contrast, `combineLatest` returns an Observable that produces a new value every time the input observables do, once all input observables have produced at least one value. This means it could have infinite values and may not complete. It also means that the input observables don't have to complete before producing a value”

...which means you have to unsubscribe from `combineLatest()`

<https://stackoverflow.com/questions/41797439/rxjs-observable-combinelatest-vs-observable-forkjoin>

```

getForkJoinData(): Observable<any> {
  return forkJoin(
    // first call
    this.http.get(AUTHORS)
      .pipe(
        delay(3000)
      ),
    // second call
    this.http.get(BOOKS)
  )
  .pipe(
    map((data: any[]) => {
      // data is now an array with 2 objects, b/c we did 2 http-calls.
      // First result, from the http-call to AUTHORS
      const author: any = data[0][0]; // Get just first author from file.
      // Second result, from the http-call to BOOKS
      const books: any[] = data[1];
      // Compose result, in this case adding the books to the extracted author.
      author.books = books.filter(book => book.author === author.name);
      return author;
    })
  );
}

```

Other interesting / often used Operators

- `from()`, `of()` - create observables from almost everything
- `fromEvent()` - create observable from a DOM-event.
- `debounce()`, `debounceTime()` - Discard emitted values that take less than the specified time between output
- `tap()` - utility operator - transform side-effects, such as logging
- ...and much more... See <http://www.learnrxjs.io>
- See also <https://rxjs-dev.firebaseio.com/>



Extra information

Some background info on RxJS and Operators

<https://dev.to/rxjs/observables-reactive-programming-and-regret-4jm6>

DEV Search... Write a post 216

262 84 259 ...



RxJS

Observables, Reactive Programming, and Regret

#rxjs #reactiveprogramming #webdev #javascript

Ben Lesh Jun 29 · 6 min read

As of this writing, I've been working on the RxJS project for almost 6 years, I think. When I started out, I really had **no idea** what I was getting into (And I wouldn't have been able to ship those first versions without Paul Taylor and others, for sure). I can remember looking at the number of weekly downloads

RxJS Follow More from RxJS RxJS debounce vs throttle vs audit vs sample — Difference You Should Know #rxjs #angular #webdev #javascript "RxJS Cheatsheet" VS Code extension #rxjs #vscode #cheatsheet #showdev New in RxJS v7: concatWith operator #rxjs #angular

More info on operators (good articles)

ARTICLES SPEAKING COURSES TRAINING GITHUB



My name is [Cory Rylan](#), [Google Developer Expert](#) and Front End Developer for [VMware Clarity](#). [Angular Boot Camp](#) instructor. I specialize in creating fast progressive web applications.

[Follow @coryrlan](#)

[Twitter](#) [Facebook](#) [LinkedIn](#) [Email](#)

Angular Multiple HTTP Requests with RxJS



Cory Rylan
Nov 15, 2016
Updated Feb 25, 2018 - 5 min read

[angular](#) [rxjs](#)

This article has been updated to the latest version of [Angular 7](#). Some content may still be applicable to Angular 2 or other previous versions.

This article has been updated to use the new [RxJS Pipeable Operators](#) which is the new default for RxJS 6.

ANGULAR BOOT CAMP

Sign up for Angular Boot Camp to get in person Angular training!



Web Component Essentials

Learn to write reusable UI components that work everywhere!

[GET E-BOOK NOW!](#)

<https://coryrlan.com/blog/angular-multiple-http-requests-with-rxjs>

Check out my [Angular article series with live demos](#)

Home [Angular](#) JavaScript Performance React NodeJS Svelte Aurelia Vue Q&A

Combining Multiple RxJs Streams In Angular



Torgeir "Tor" Helgevold

- JavaScript Developer and Blogger

Published: Sun Apr 17 2016

In RxJs we often deal with multiple streams, but the end consumer typically only subscribes to a single stream. In this article we will look at ways to combine multiple streams into a single stream.

There are many ways for RxJs streams to converge on a single stream, but in this article we will look at flatMap, forkJoin, merge and concat.

Concat

Concat will combine two observables into a combined sequence, but the second observable will not start emitting until the first one has completed.

In my sample I am concatenating two timer observables using concat.

<http://www.syntaxsuccess.com/viewarticle/combining-multiple-rxjs-streams-in-angular-2.0>

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

The screenshot shows the homepage of the Learn RxJS website. The left sidebar contains a search bar, a navigation menu with 'learn-rxjs' and 'LEARN RXJS' sections, and a detailed list of RxJS operators under 'Operators'. The main content area features a large title 'Learn RxJS' and a subtitle 'Clear examples, explanations, and resources for RxJS.' Below this is a section titled 'Introduction' with a paragraph about RxJS's popularity and utility. A 'But...' section follows, explaining the challenges of learning RxJS. The footer contains a 'Content' section.

Type to search

learn-rxjs

LEARN RXJS

Introduction

Operators

Combination

- combineAll
- combineLatest
- concat
- concatAll
- forkJoin
- merge
- mergeAll
- pairwise
- race
- startWith
- withLatestFrom
- zip

Conditional

Learn RxJS

Clear examples, explanations, and resources for RxJS.

Introduction

RxJS is one of the hottest libraries in web development today. Offering a powerful, functional approach for dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to utilize your knowledge across [nearly any language](#), having a solid grasp on reactive programming and what it can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is [hard](#). There's the multitude of concepts, large API surface, and fundamental shift in mindset from an [imperative to declarative style](#). This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the [official docs](#) and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and tackle the pain points. Learning Rx may be difficult but it is certainly worth the effort!

Content

Article - 6 Operators you must know

The screenshot shows a Medium article page. At the top right are 'Sign in / Sign up' and social sharing icons for LinkedIn and Twitter. Below that is the author's profile picture, name 'Netanel Basal', a 'Follow' button, and the date 'Jan 24 · 3 min read'. The main title 'RxJS—Six Operators That you Must Know' is centered above a code block. The code block contains TypeScript code for a 'TakeSubscriber' class that implements the Subscriber interface. It has a constructor taking a destination subscriber and a total count. The _next method increments a local count and checks if it reaches the total, returning true if so. At the bottom, there's a sidebar with the author's profile picture, a call to action 'Never miss a story from NetanelBasal, when you sign up for Medium. Learn more', and a 'GET UPDATES' button.

```
class TakeSubscriber<T> extends Subscriber<T> {  
  private count: number = 0;  
  
  constructor(destination: Subscriber<T>, private total: number) {  
    super(destination);  
  }  
  
  protected _next(value: T): void {  
    const total = this.total;  
    const count = ++this.count;  
    if (count <= total) {  
      return true;  
    }  
  }  
}
```

Never miss a story from **NetanelBasal**, when you sign up for Medium. [Learn more](#)

[GET UPDATES](#)

<https://netbasal.com/rxjs-six-operators-that-you-must-know-5ed3b6e238a0#.11of73aox>

Creating Observables from scratch

- André Staltz

André Staltz (@andrestaltz): You will learn RxJS at ng-europe 2016

```
function nextCallback(data) {
  console.log(data);
}

function errorCallback(err) {
}

function completeCallback() {
}

function giveMeSomeData(nextCB, errCB) {
  document.addEventListener('click', () => {
    nextCB();
    errCB();
  });
}

giveMeSomeData(
  nextCallback,
  errorCallback,
  completeCallback
);
```

The video frame shows André Staltz with a beard, wearing a grey hoodie, sitting at a desk with a laptop. The laptop screen displays the Angular logo and the text "ngeurope.org". The background is a blue wall with the "ng-europe" logo.

<https://www.youtube.com/watch?v=uQ1zhJHclvs>

GitHub Gist Search... All gists GitHub New gist

 staltz / introrx.md Last active an hour ago ★ Star 10,812 Fork 1203 ⚡

Code Revisions 259 Stars 10812 Forks 1203 Embed <script src="https://gist.github.com/staltz/868e7e9bc2a7b8c1f754.js"> Download ZIP

The introduction to Reactive Programming you've been missing

 introrx.md Raw

The introduction to Reactive Programming you've been missing

(by [@andrestaltz](#))

This tutorial as a series of videos

If you prefer to watch video tutorials with live-coding, then check out this series I recorded with the same contents as in this article: [Egghead.io - Introduction to Reactive Programming](#).

So you're curious in learning this new thing called Reactive Programming, particularly its variant comprising of Rx, Bacon.js, RAC, and others.

Learning it is hard, even harder by the lack of good material. When I started, I tried looking for tutorials. I found only a handful of practical guides, but they just scratched the surface and never tackled the challenge of building the whole architecture

<https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>

Also by Andre Stalz - RxMarbles

RxMarbles Interactive diagrams of Rx Observables

Fork me on GitHub

TRANSFORMING OPERATORS

- [delay](#)
- [delayWithSelector](#)
- [findIndex](#)
- [map](#)
- [scan](#)
- [debounce](#)
- [debounceWithSelector](#)

COMBINING OPERATORS

- [combineLatest](#)
- [concat](#)
- [merge](#)
- [sample](#)
- [startWith](#)
- [withLatestFrom](#)
- [zip](#)

FILTERING OPERATORS

- [distinct](#)
- [distinctUntilChanged](#)
- [elementAt](#)
- [filter](#)
- [find](#)
- [first](#)

merge

```
graph LR; S1[20] --- S2[40] --- S3[60] --- S4[80] --- S5[100]; S6[1] --- S7[1]; S1 --- S6; S2 --- S7; S3 --- S8[1]; S4 --- S9[80]; S5 --- S10[100]; S6 --- S11[1];
```

v1.4.1 built on RxJS v2.5.3 by @andrestalz

<http://rxmarbles.com/>

Fetching a collection of observables

<https://blog.angularindepth.com/practical-rxjs-in-the-wild-requests-with-concatmap-vs-mergemap-vs-forkjoin-11e5b2efe293>

The screenshot shows a blog post on the Angular In Depth website. The header features the 'Angular In Depth' logo with 'by AG-Grid'. Below the header is a navigation bar with links for HOME, ANGULAR, RXJS, NGRX, ABOUT, SUPPORT US, and AG-GRID: THE BEST ANGULAR GRID IN THE WORLD. A 'Follow' button is also present. The main content area displays a large title: 'Practical RxJS In The Wild' followed by a lion emoji. Below the title is a subtitle: 'Requests with concatMap() vs mergeMap() vs forkJoin()' followed by a boxing glove emoji. Underneath the title is a circular profile picture of a man, a 'Follow' button, and the author's name 'Tomas Trajan'. The post was published on 'Dec 19, 2017 · 7 min read'.

Practical RxJS In The Wild —
Requests with concatMap() vs
mergeMap() vs forkJoin()

Tomas Trajan [Follow](#)
Dec 19, 2017 · 7 min read

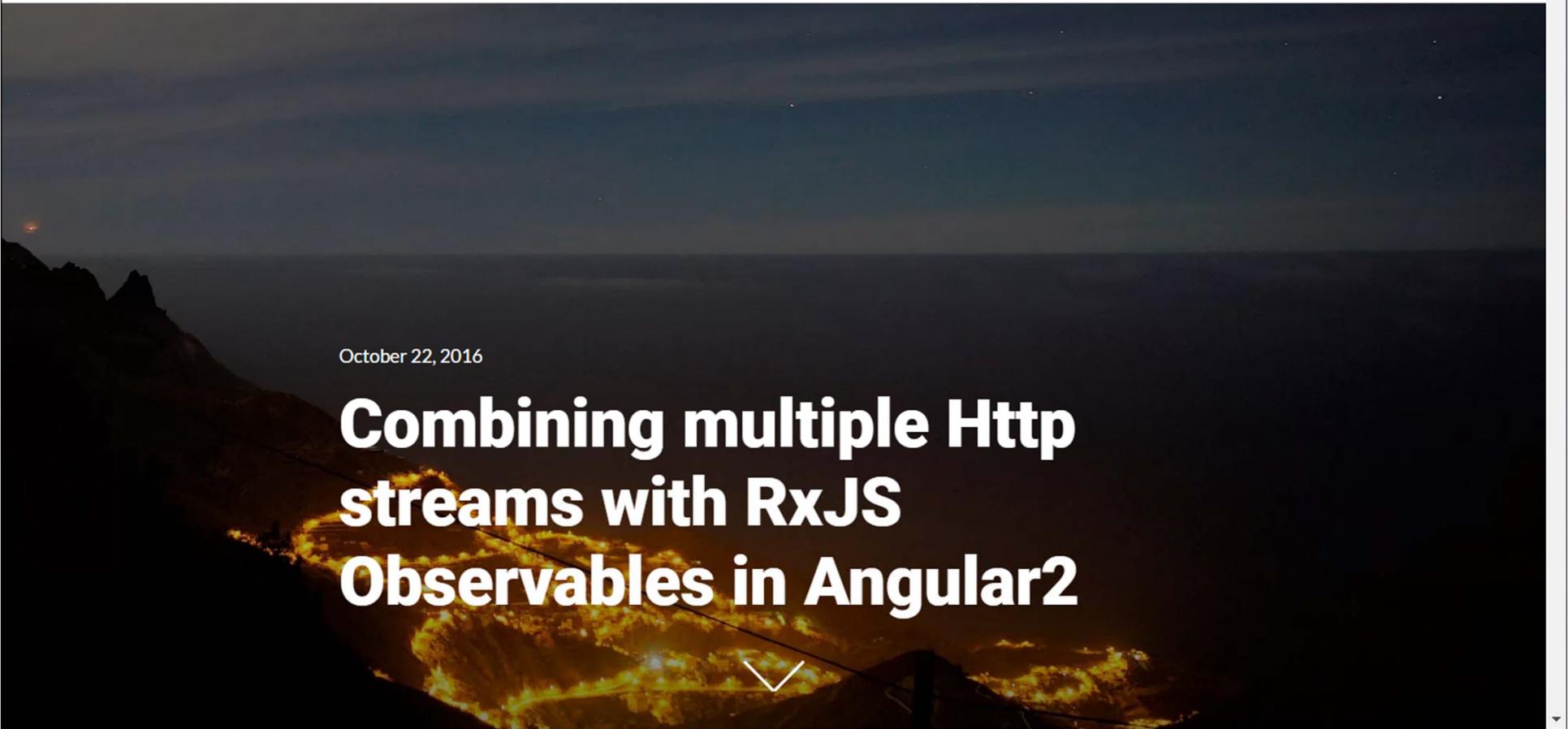
<https://github.com/tomastrajan/ngx-model-hacker-news-example>



Daniele Ghidoli

About me

...



October 22, 2016

Combining multiple Http streams with RxJS Observables in Angular2

<http://blog.danieleghidoli.it/2016/10/22/http-rxjs-observables-angular/>



RxJS

Reactive Extensions Library for JavaScript

[GET STARTED](#)

[API DOCS](#)

REACTIVE EXTENSIONS LIBRARY FOR JAVASCRIPT

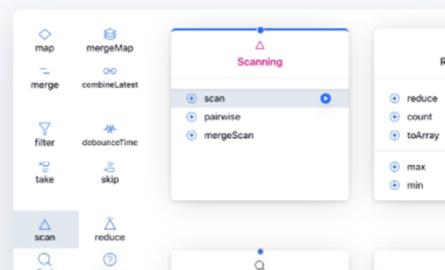
RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of Reactive-Extensions/RxJS with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface

<https://rxjs-dev.firebaseio.com/>

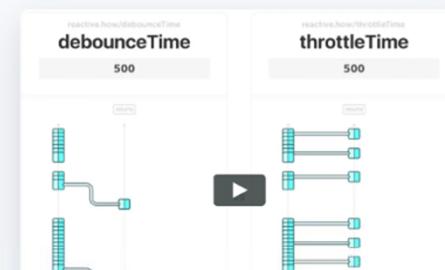
reactive.how 2.0-alpha.4   Launchpad for RxJS →

ESSENTIALS REDUCING TAKING SKIPPING COMBINING TIME CONCATENATING
map reduce take skip (soon) combineLatest delay startWith (soon)
filter max takeWhile skipWhile zip debounceTime endWith (soon)
merge count first takeLast throttleTime concat (soon)
scan last

Learn RxJS operators and Reactive Programming principles



Launchpad for RxJS



debounceTime vs throttleTime



Decks of cards for RxJS: learn, compare and memorize



scan reduce



map filter



zip combineLatest

<https://reactive.how/>

 SCOTCH

Courses Tutorials News App Hosting ...

FREE Webinar: Should I use React or Vue?

What are the differences?



+

- forkJoin
- zip
- combineLatest
- withLatestFrom

RxJS operators for dummies:
forkJoin, zip, combineLatest, withLatestFrom

RxJS Operators for Dummies: forkJoin, zip, combineLatest, withLatestFrom

 Jecelyn Yeen  @JecelynYeen September 10, 2018 30 Comments
86.934 Views 

<https://scotch.io/tutorials/rxjs-operators-for-dummies-forkjoin-zip-combinelatest-withlatestfrom>

Workshop

- Create a call to the Open Movie Database API, using a keyword to search for 10 movies
 - <http://www.omdbapi.com/?apikey=f1f56c8e&s=<keyword>>
- Create an additional call to get details for every movie returned. Use the `imdbID` property for that
 - <http://www.omdbapi.com/?apikey=f1f56c8e&i=<imdbID>>
- Display results in the UI, first a list of movies, then details per movie as soon as they come available.
- What is your best solution? Multiple ways of doing this!

