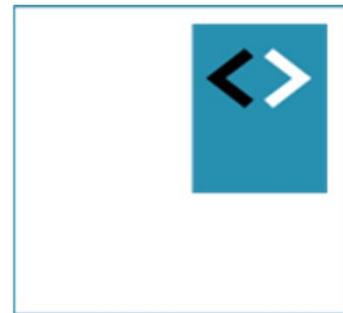




Gemeente
Haarlem

Angular - Maatwerk



Peter Kassenaar
info@kassenaar.com



Async services met RxJS/Observables

Reactive programming with asynchronous streams

Async Services

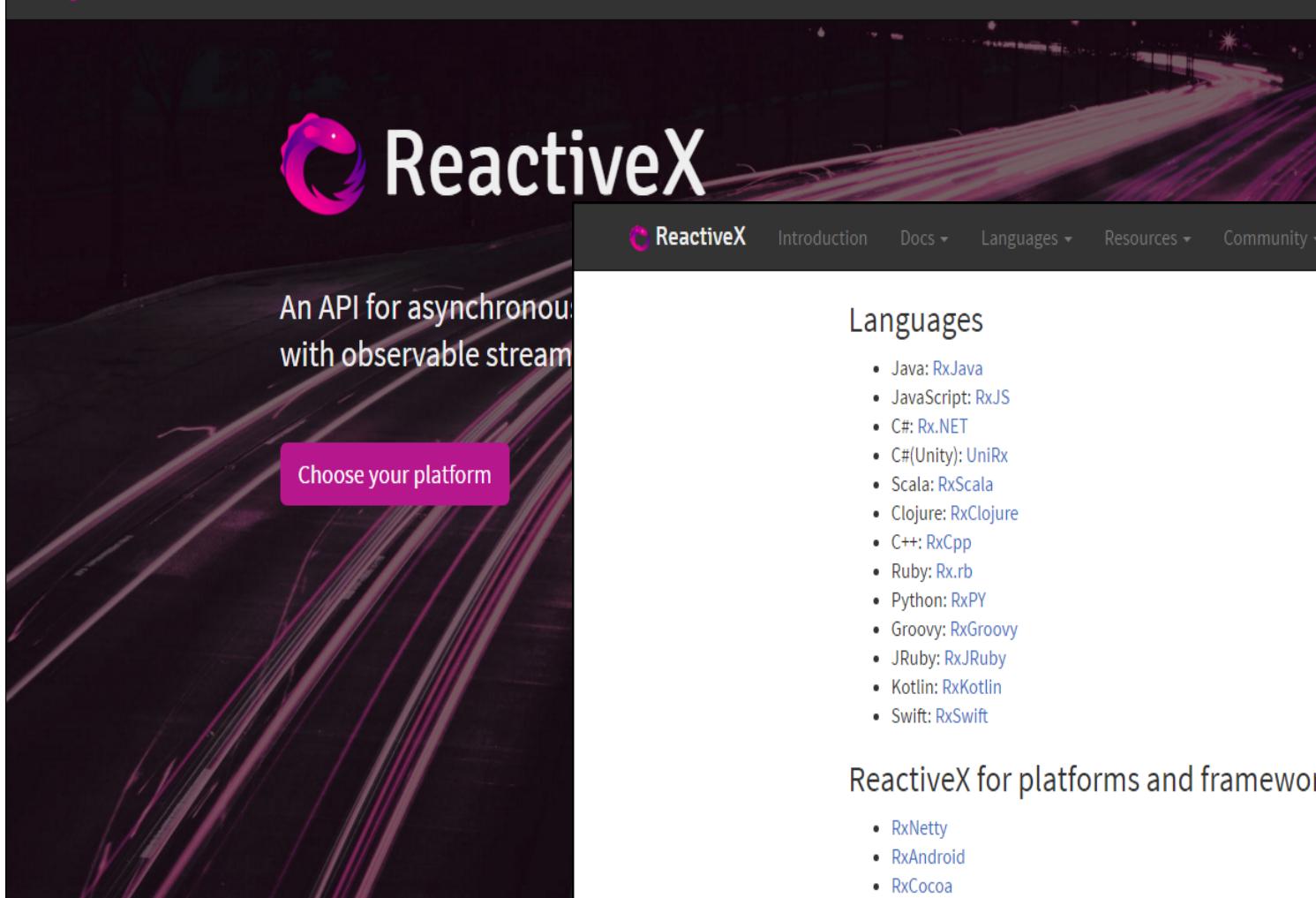


- Fetching static data: *synchronous* action
- Working via HttpClient: *asynchronous* action
- Angular 1 and others: Promises
- Angular : Observables

Also Angular : ReactiveX library RxJS



 **ReactiveX** Introduction Docs ▾ Languages ▾ Resources ▾ Community ▾



An API for asynchronous programming with observable streams

Choose your platform

<http://reactivex.io/>

ReactiveX Introduction Docs ▾ Languages ▾ Resources ▾ Community ▾

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

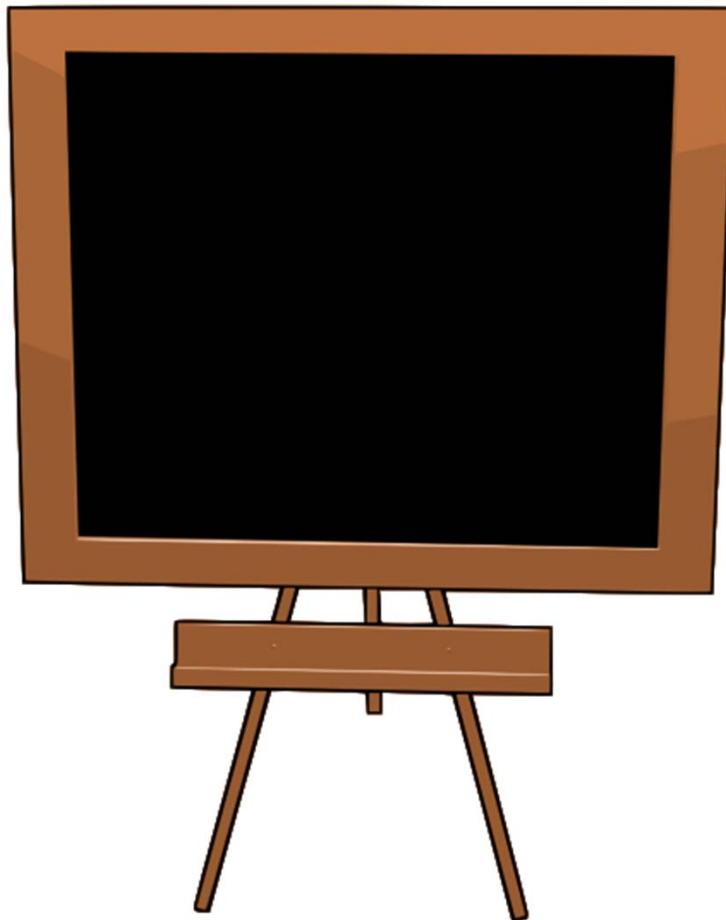
DOCUMENTATION	LANGUAGES	RESOURCES	COMMUNITY
Observable	RxJava[↗]	Tutorials	GitHub[↗]
Operators	RxJS[↗]		Twitter[↗]
Single	Rx.NET[↗]		Others
Connectable	Rx.Cocoa		



The observable design pattern

*“Understanding the
observable stream”*

Explanation





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 and RxJs



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



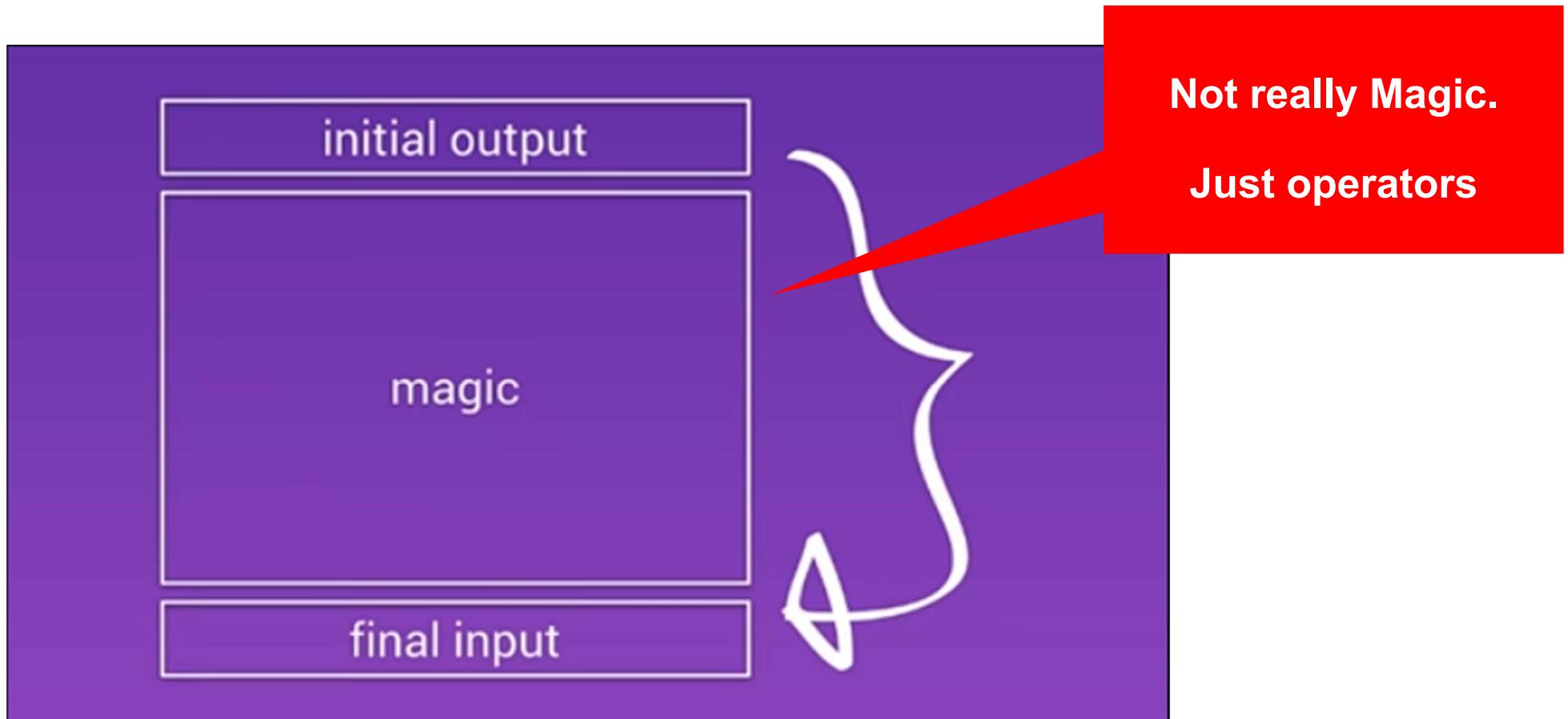
Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

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

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



"The observable sandwich"



Most common usage of observables: http



Modern applications are moduleless
provide the imported httpClient()

```
// app.config.ts
import ...;
import {provideHttpClient} from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [
    provideHttpClient(),
    ...
  ]
};
```



In classical applications

When (still) using ngModules,

import HttpClientModule()

```
// Angular Modules
import ...
import {HttpClientModule} from "@angular/common/http";

// Module declaration
@NgModule({
  imports      : [BrowserModule, HttpClientModule],
  declarations: [AppComponent],
  bootstrap    : [AppComponent],
})
export class AppModule {
```



In even newer applications (v19.2+)

- Angular httpResource available.
- <https://blog.angular.dev/seamless-data-fetching-with-httpresource->

The screenshot shows a blog post on the Angular Blog. The title is "Seamless data fetching with httpResource". The author is Matthieu Riegler, published on Mar 7, 2025. The post has 535 likes and 7 comments. Below the title is a large photo of a brown dog holding a stick in its mouth, standing in a forest.

Angular Blog

Seamless data fetching with httpResource

Matthieu Riegler · [Follow](#)
Published in Angular Blog · 4 min read · Mar 7, 2025

535 7



Classical: inject http in Component / service



- Create an http variable, of type HttpClient
- Use Constructor Injection, or inject() function

```
export class AppComponent {  
  ...  
  constructor(private http: HttpClient) {}  
}
```

```
export class AppComponent {  
  // using inject()  
  private http = inject (HttpClient);  
}
```

<https://angular.dev/guide/http>

Making an http call, for instance

Initial Output

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

Optioneel:
operator(s)

Final Input



Using the `async` pipe

Automagically `.subscribe()` and `.unsubscribe()`



Async Pipe

- On `.subscribe()`, you actually need to`.unsubscribe()` to avoid memory leaks
 - Best practice
 - Not *really* necessary on HTTP-requests, but you do in other subscriptions.
- No more manually `.subscribe()` and`.unsubscribe()`:
 - **Use Angular async pipe**

Where to change/update your code:



Component:

```
1. cities$: Observable<City[ ]>; // Now: Observable to Type
```

...

```
2. ngOnInit() {  
    // Call service, returns an Observable  
  
    this.cities$ = this.cityService.getCities()  
}
```

View:

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

OR: using the modern @for syntax with async



For instance, retrieving users instead of cities:

```
users$: Observable<any[]> | undefined;  
ngOnInit(){  
  ...  
  this.users$ = this.http.get<any[]>  
    ('https://jsonplaceholder.typicode.com/users');  
}
```



```
@for (user of users$ | async; track user.id) {  
  <div>  
    {{ user.name }}  
  </div>  
}
```



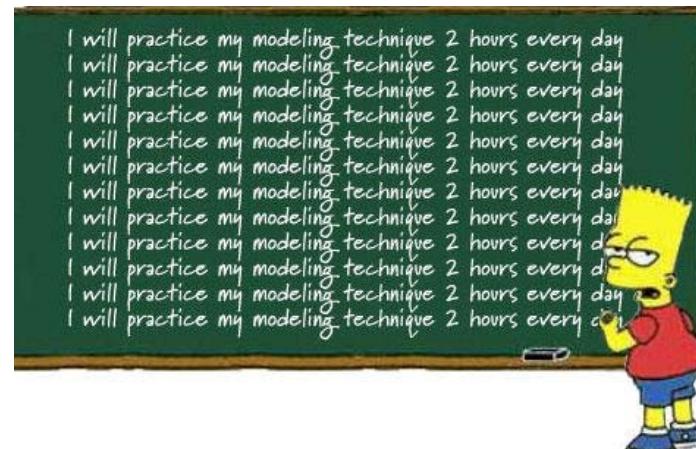
Example API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weather forecast)
- <https://jsonplaceholder.typicode.com/> random users, posts, photos
- <http://ergast.com/mrd/> - Ergast Motor (F1) API
- <http://www.omdbapi.com/> - Open Movie Database
- <http://swapi.dev/> - Star Wars API
- See also `JavaScript APIs.txt` with other API's.

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
 - ...





More on operators

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

The Problem with RxJS Operators...



- There are so many of them...

LEARN RXJS

Introduction

Operators

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

Conditional

EDIT THIS PAGE A

Star 457 Watch 37

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 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 apply your knowledge across [nearly any language](#), having a solid grasp on reactive programming and what RxJS 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 pre-existing learning material while offering a new, fresh perspective to clear any hurdles and take advantage of the power of RxJS.

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

[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 RxJS 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](#). 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 perspective 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 / **ngx-rxjs-operators**

Type to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

ngx-rxjs-operators Public

Pin Unwatch 1

master 1 Branch 0 Tags

Go to file Add file Code

File	Commit Message	Time
.vscode	initial commit	25 minutes ago
src	feat: initial commit of (old) components	3 minutes ago
.editorconfig	initial commit	25 minutes ago
.gitignore	initial commit	25 minutes ago
README.md	feat: initial commit of (old) components	3 minutes ago

github.com/PeterKassenaar/ngx-rxjs-operators

This part of the 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



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 with the URL <https://www.learnrxjs.io/learn-rxjs/operators/creation>. The left sidebar has a navigation menu with sections like 'Introduction', 'LEARN RXJS' (which is currently selected), 'Operators', 'Combination', 'Conditional', and 'Creation'. Under 'Creation', there are links for 'ajax', 'create', 'defer', 'empty', and 'from'. The main content area is titled 'Creation' and contains the text: '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 ★'.

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...

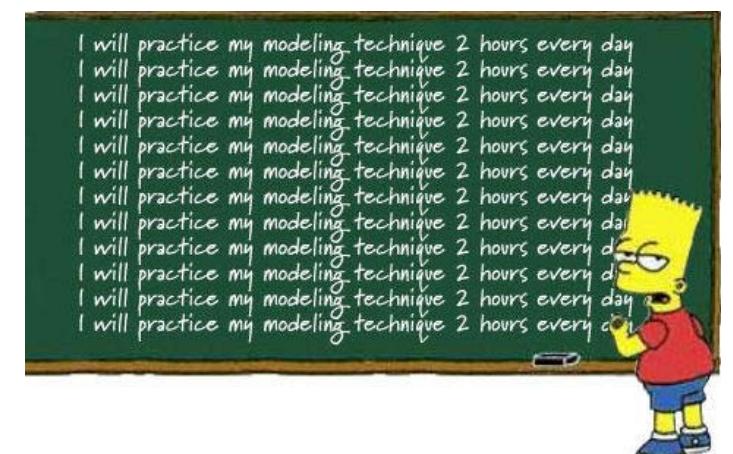
This is a stream...



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;
```

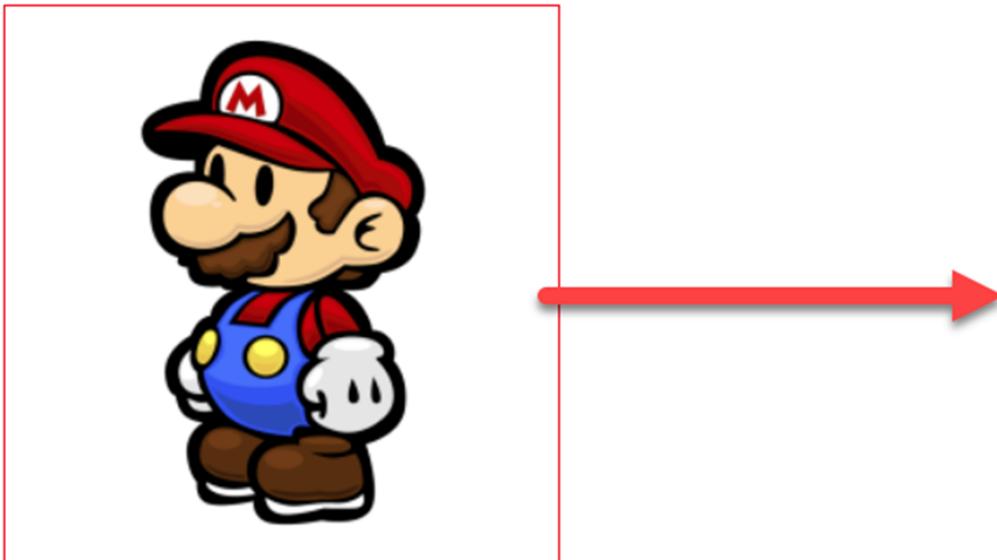
```
fromEvent(this.btnRight.nativeElement, 'click')
.pipe(
  map(event => 10), // 1. map the event to a useful value, in this case 10px
  startWith({x: 100, y: 100}), // 2. start with an object of { 100, 100}.
  scan((acc: any, current: number) => { // 3, use the scan operator as reducer function.
 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

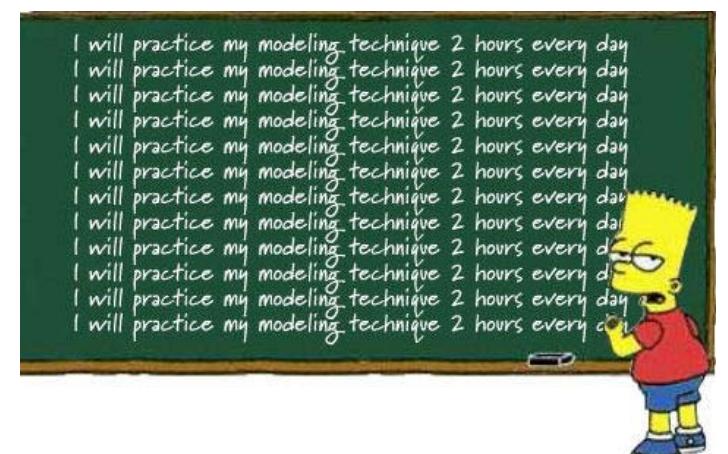
Mario is moved by observable streams from the button click.



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 sidebar on the left and the main content area on the right.

Navigation Sidebar:

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

Main Content Area:

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.

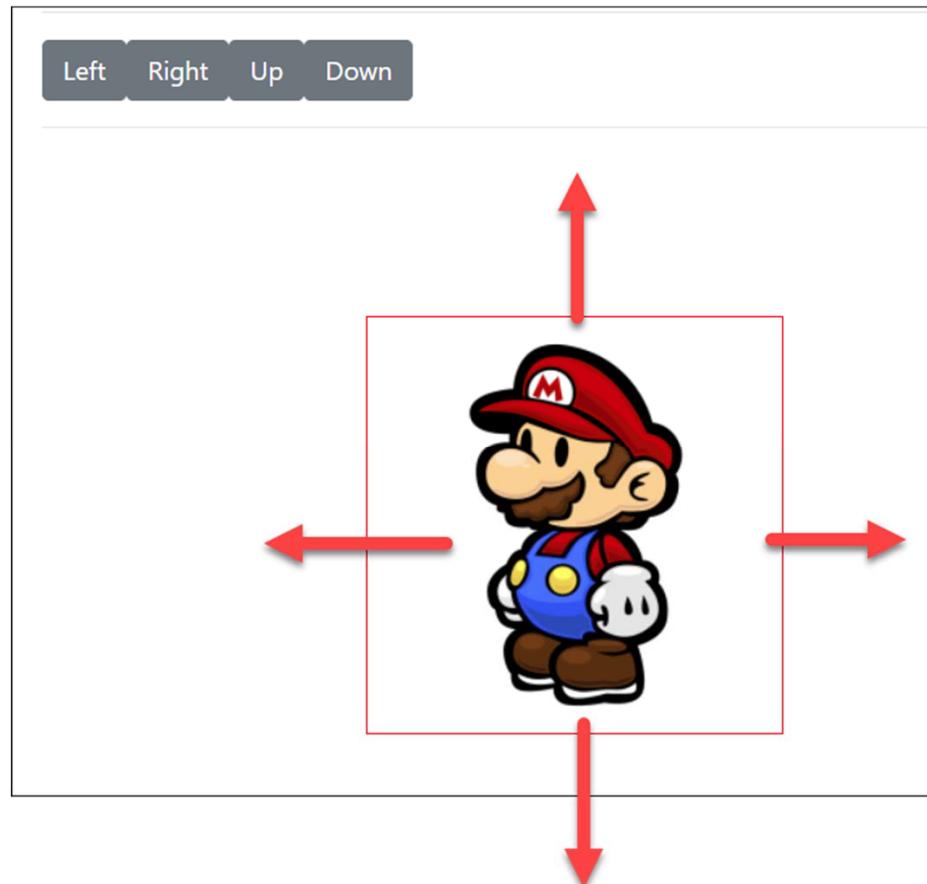
Contents

- [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);
}

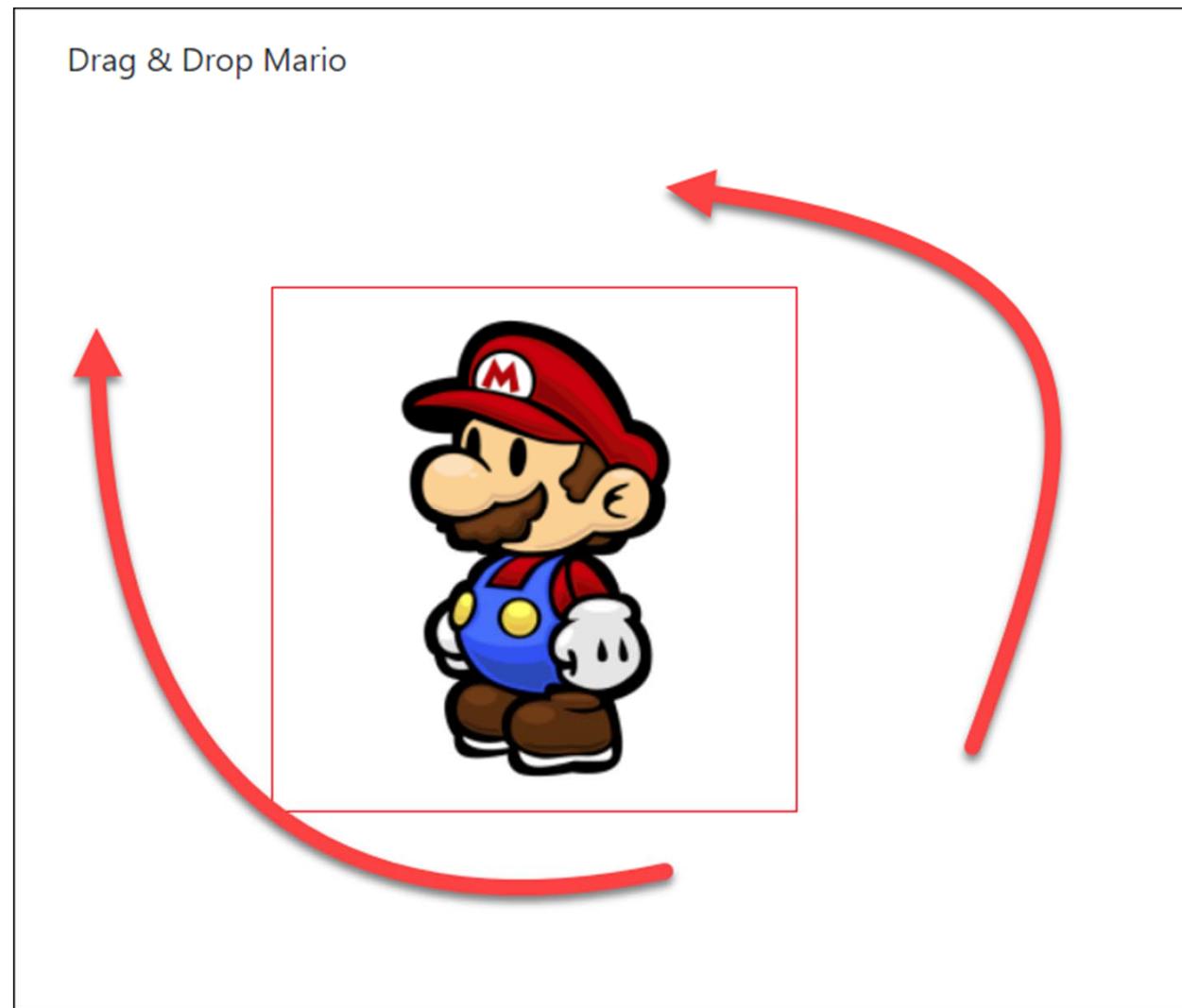
```

Compose streams

Switch execution to another observable

Subscribe and update position

Result



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.

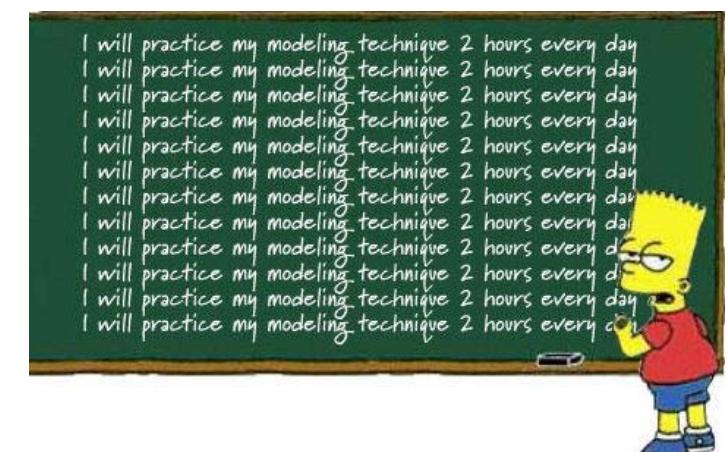
Attention Users

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



Summary on some operators : .tap()



- Side effects without modifying the stream. Great for logging, debugging. NOT for production!

```
// 1. The Tap operator is used for debugging purposes.  
of(1, 2, 3).pipe(  
  tap(x => console.log('before map:', x)),  
  map(x => x * 2)  
) subscribe();
```

The screenshot shows the 'Console' tab of a browser's developer tools. The log output is as follows:

```
before map: 1  
before map: 2  
before map: 3  
Angular is running in development mode.  
Attempting initialization Thu Apr 24 2025 16:14:24 GMT+0200 (Midden
```

<https://rxjs.dev/api/operators/tap>



.map()

- Transforms values in the stream (1:1 mapping).
- *"Map takes a value and transforms it to another value. The transformed value is returned"*

```
// 2. The Map operator is used to transform the data.  
of(1, 2, 3).pipe(  
  map(x => x * 10)  
) subscribe(console.log); // 10, 20, 30
```

```
10  
20  
30  
Angular is running in development mode.  
Attempting initialization Thu Apr 24 2025 16:16:59 GMT+0200  
(Midden-Furonese zomertiid)
```

[cont](#)

<https://rxjs.dev/api/operators/map>

.debounce() (and debounceTime())



- Waits for silence; emits only after a pause
- Note: since we're using an <input> element here, we also need to wait for it to initialize in the `ngAfterViewInit()` lifecycle hook!

```
ngAfterViewInit() {
  if(this.inputRef) {
    // 3. The DebounceTime operator is used to delay the emission of an item.
    fromEvent(this.inputRef?.nativeElement, 'input').pipe(
      debounceTime(300),
      map((e: any) => e.target.value)
    ).subscribe(console.log);
  }
}
```

```
Angular is running in development mode.

Attempting initialization Thu Apr 24 2025 16:27:21 GMT+0200 (Midden-Europese zomertijd)

te
tes
test
```

✖ Unchecked runtime lastError: A listener indicated an asynchronous response.

<https://rxjs.dev/api/operators/debounceTime>



.switchMap()

- Cancels previous inner observable on new emission. Ideal for typeahead or HTTP calls.
- Dummy Code!:

```
fromEvent(this.inputRef.nativeElement, 'input').pipe(  
  map((e: any) => e.target.value),  
  switchMap(query => this.http.get(`/search?q=${query}`))  
) subscribe(console.log);
```

- Here, the outer observable cancels the previous `http.get()` call when a new value is received, so 'old' results from the API are not shown in the console.

<https://rxjs.dev/api/operators/switchMap>



.forkJoin()

- Waits for all observables to complete, then **emits once** with all last values.
- NOTE: Dummy code!

```
forkJoin([
  this.http.get('/user'),
  this.http.get('/settings')
]).subscribe(([user, settings]) => {
  console.log(user, settings);
});
```

<https://rxjs.dev/api/index/function/forkJoin>



mergeMap()

- Flattens inner observables concurrently.
- Use when order/cancellation doesn't matter.
- NOTE: Dummy code:

```
from([1, 2, 3]).pipe(  
  mergeMap(id => this.http.get(`/item/${id}`))  
).subscribe(console.log);
```

- This would show `1 + result http-call1`, `2 + result http-call2` etc. in the console.

<https://rxjs.dev/api/operators/mergeMap>