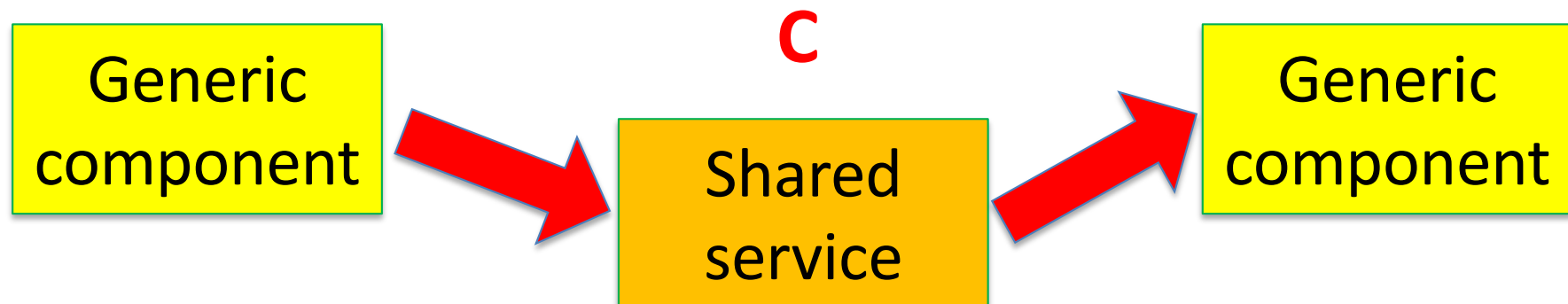
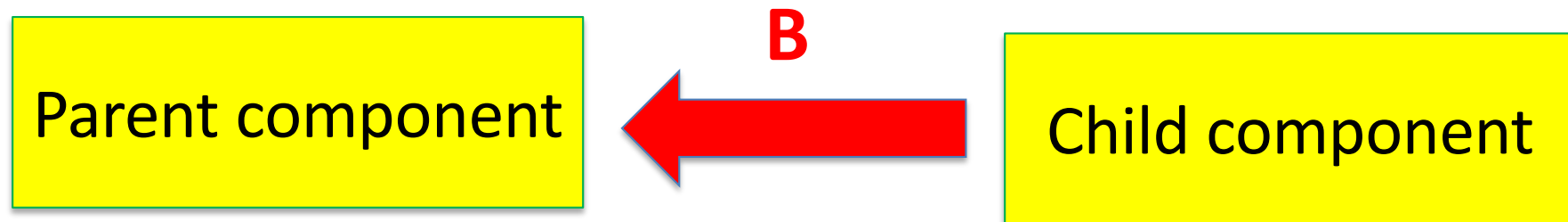
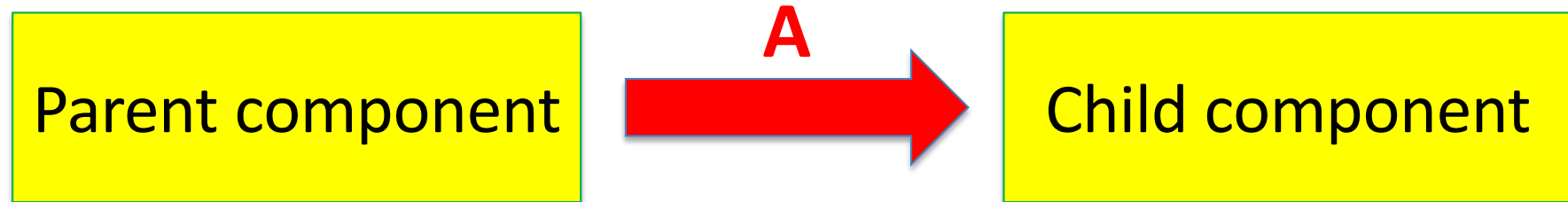


# Q

**How can components communicate?**

# 3 scenarios:



# A - Pass data to a child component - 1

In the Child Component:

1. Import the @Input module from @angular/Core Library
2. Mark those property, which you need data from parent as input property using @Input decorator.

```
import { Component, Input } from '@angular/core';
```

```
@Component({  
  selector: 'child-component',  
  template: `

## Child Component</h2> current count is {{ count }}

`  
})  
export class ChildComponent {  
  @Input() count: number;  
}
```



in-line template

# A -Pass data to a child component - 2

```
import { Component }  
  from '@angular/core';
```

In the Parent Component:  
1. Bind to Child Property

```
@Component({  
  selector: 'app-root',  
  template: `  
    <h1>Welcome to {{title}}!</h1>  
    <button (click)="increment()">Increment</button>  
    <button (click)="decrement()">decrement</button>  
    <child-component [count]=Counter></child-component>  
  ` , styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'Component Interaction';  
  Counter = 5;  
  increment() { this.Counter++; }  
  decrement() { this.Counter--; }  
}
```

in-line template

# B - Pass data to a parent component - 1

There are three ways to let parent get info from child

1. Parent Listens to Child Event
2. Parent uses Local Variable to access the child
3. Parent uses a @ViewChild to get reference to the child component

For options 1 and 3, see:

<https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/>

## B - Pass data to a parent component - 2

### Child:

```
import { Component } from '@angular/core';
@Component({
  selector: 'child-component',
  template: `

## Child Component</h2> current count is {{ count }}

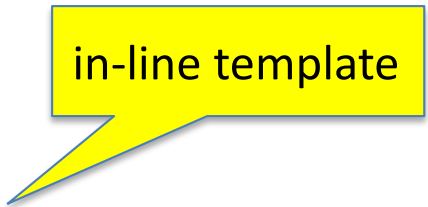
`
})
export class ChildComponent {
  count = 0;
  increment() {this.count++;}
  decrement() {this.count--;}
}
```

in-line template

## B - Pass data to a parent component - 3

### Parent:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}!</h1>
    <p> current count is {{child.count}} </p>
    <button (click)="child.increment()">Increment</button>
    <button (click)="child.decrement()">decrement</button>
    <child-component #child></child-component>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Parent interacts with child via local variable';
}
```



# C - Using components vs services from other modules

The most common way to get a hold of shared services is through Angular **dependency injection** rather than through the module system (importing a module will result in a new service instance, which is not a typical usage).



# Q

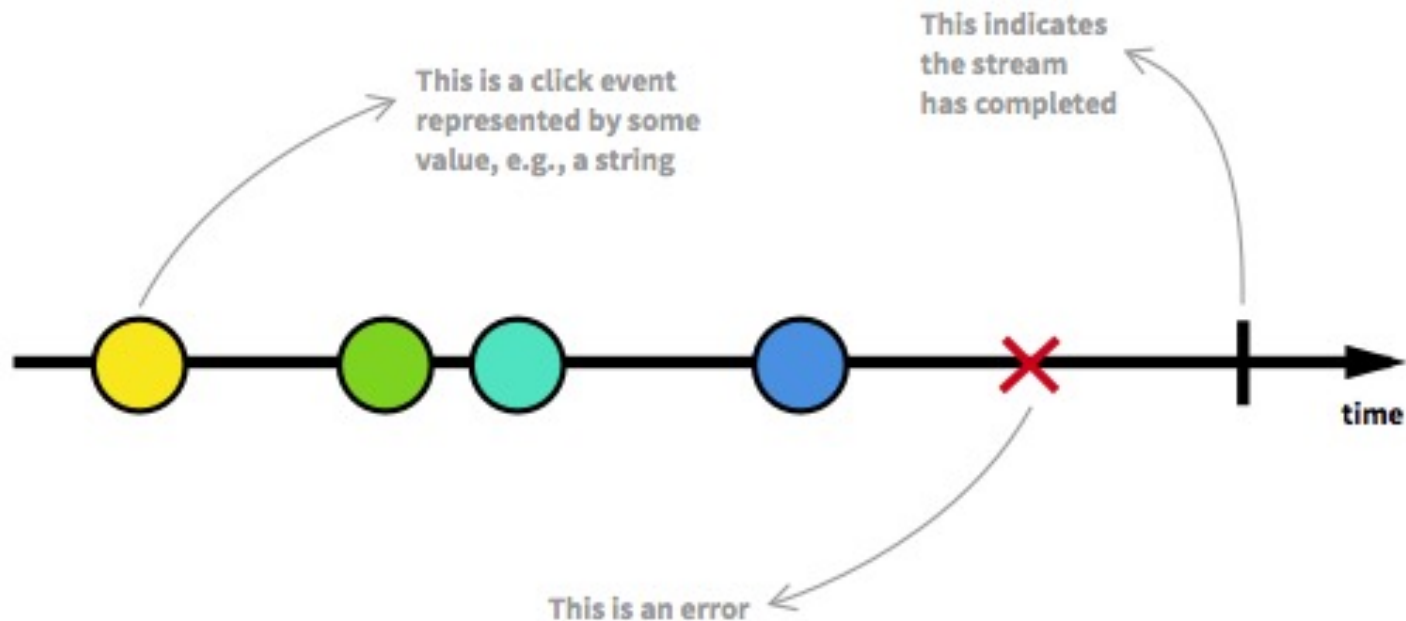
Can we generate events ?

What is reactive programming ?

# Reactive programming

Reactive programming is programming with asynchronous data streams.

A stream is a sequence of ongoing events ordered in time. It can emit three different things: a value (of some type), an error, or a "completed" signal.



# Reactive programming

We capture these emitted events **asynchronously**, by defining functions that will execute :

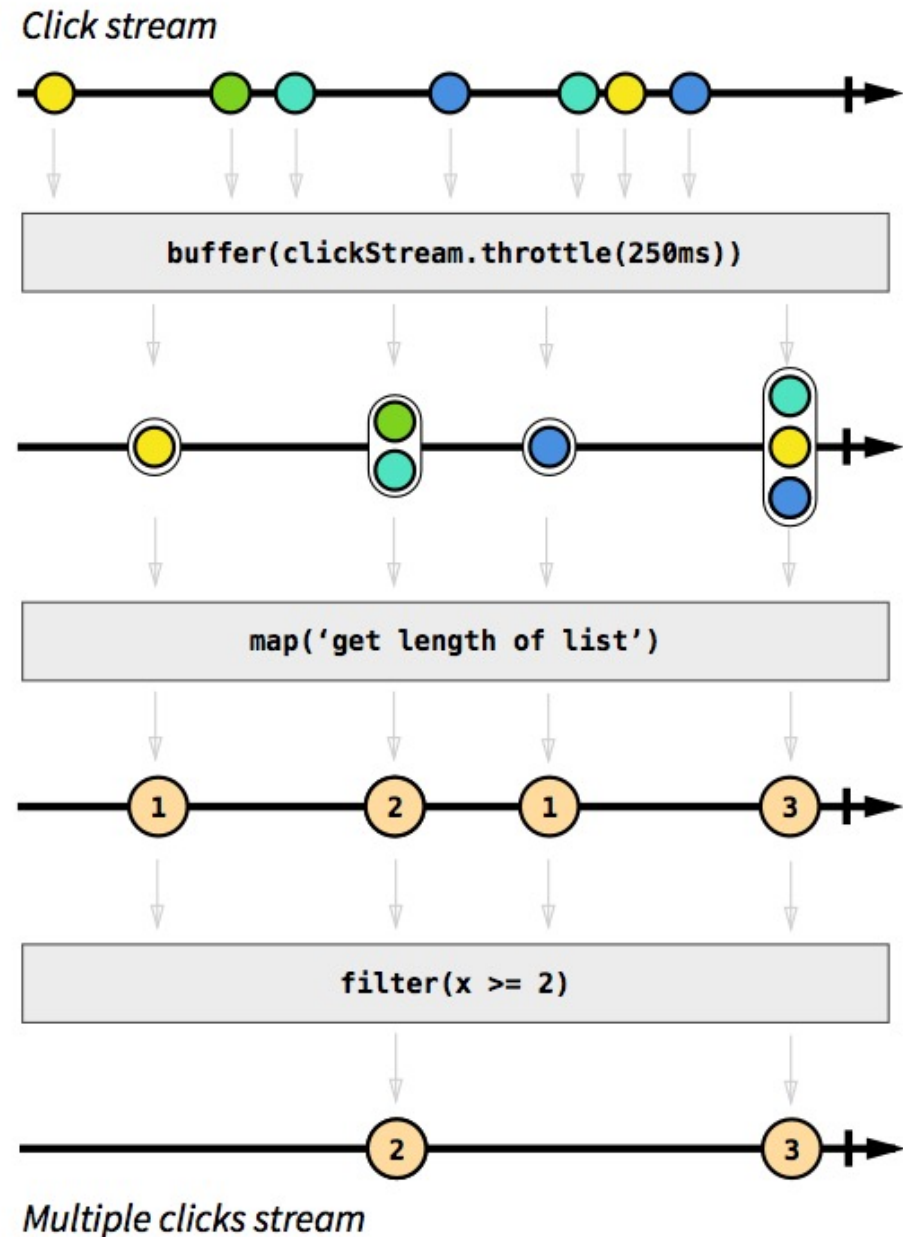
- when a value is emitted,
- when an error is emitted,
- when 'completed' is emitted.

According to the Observer Design Pattern:

- The stream "observable" being observed.
- The "listening" to the stream is called subscribing.
- The functions we are defining are "observers".

# Reactive programming

In a reactive programming environment, you are generally given a toolbox of functions to combine, create and filter any of those streams.



# RxJS

RxJS is a library for composing asynchronous and event-based programs by using observable sequences.

It provides:

- one core type, the Observable,
- satellite types (Observer, Schedulers, Subjects)
- operators (map, filter, reduce, every, etc) to allow handling asynchronous events as collections.

see <https://rxjs-dev.firebaseapp.com/guide/overview>

# RxJS - operators

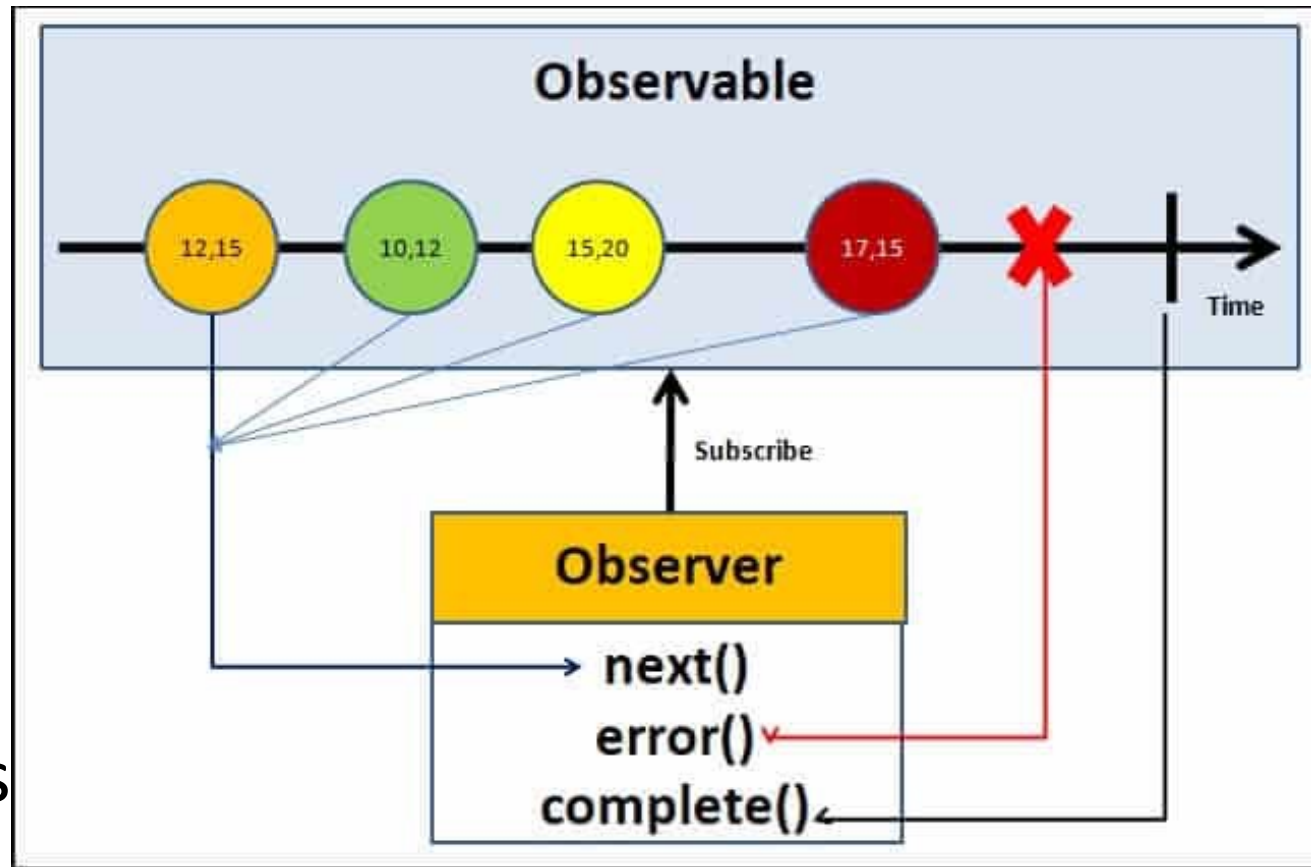
AREA	OPERATORS
Creation	from, fromEvent, of
Combination	combineLatest, concat, merge, startWith, withLatestFrom, zip
Filtering	debounceTime, distinctUntilChanged, filter, take, takeUntil
Transformation	bufferTime, concatMap, map, mergeMap, scan, switchMap
Utility	tap
Multicasting	share

see <https://angular.io/guide/rx-library#operators>

# RxJS

The observable:

- invokes the next() callback whenever a value arrives in the stream. It passes the value as the argument to the next callback.
- If the error occurs, then the error() callback is invoked.
- It invokes the complete() callback when the stream completes.



# RxJS

Observers subscribe to Observables.

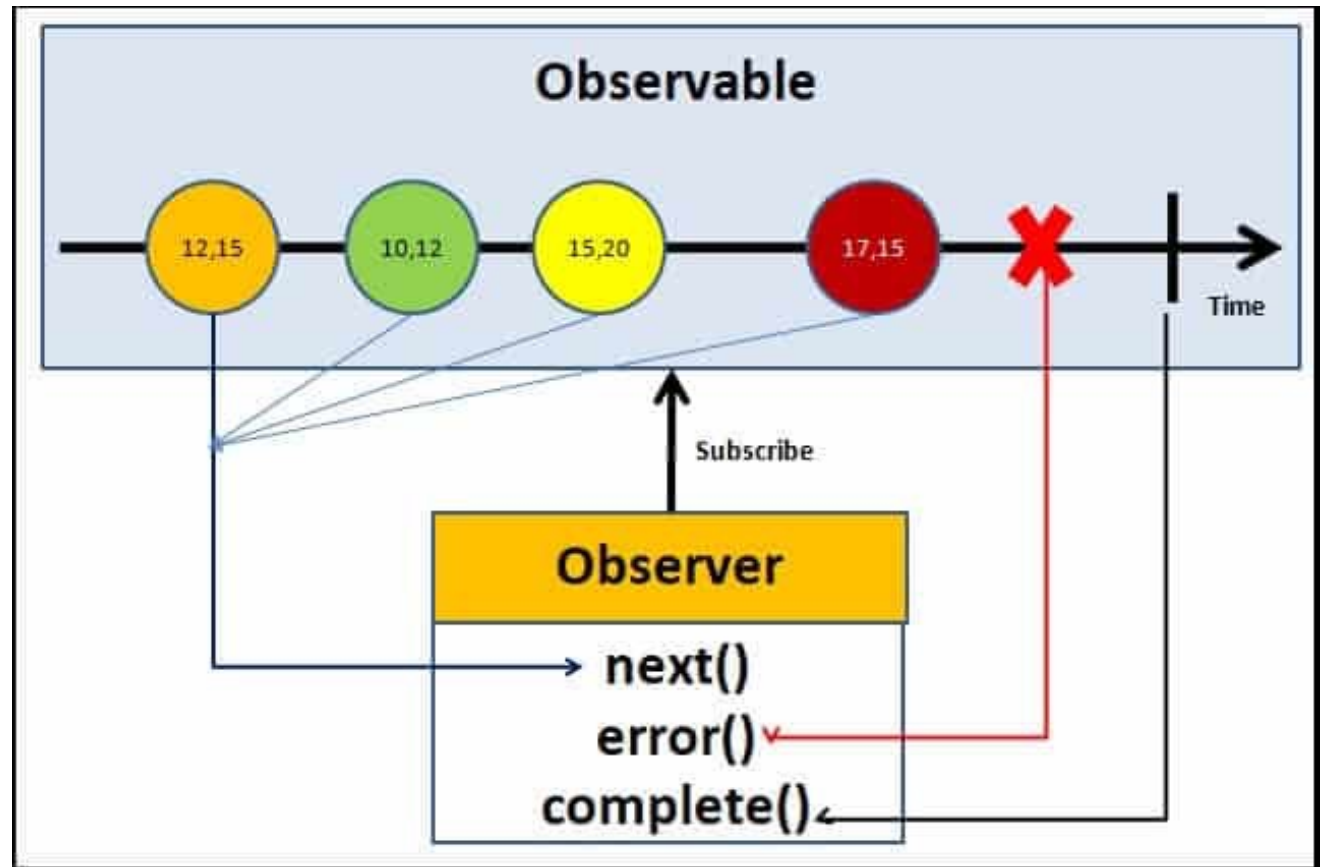
Observer registers three callbacks with the observable at the time of subscribing:

- `next()`,
- `error()`
- `complete()`

**error and complete callbacks are optional.**

The observer receives the data from the observable via the `next()` callback

They also receive the errors and completion events via the `error()` & `complete()` callbacks





# Example – app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Angular Observable using RxJs - Getting
Started';
  obs = new Observable((observer) => {
    console.log("Observable starts");
    setTimeout(() => { observer.next("1") }, 1000);
    setTimeout(() => { observer.next("2") }, 2000);
    setTimeout(() => { observer.next("3") }, 3000);
    setTimeout(() => { observer.error("error emitted")
}, 3500);
    //this code is never called
    setTimeout(() => { observer.next("5") }, 5000);  });
```

# Example – app.component.ts

## ngOnInit

A callback method that is invoked immediately after the default change detector has checked the directive's data-bound properties for the first time, and before any of the view or content children have been checked. It is invoked only once when the directive is instantiated.

```
data=[];  
  
ngOnInit() {  
  
    this.obs.subscribe(  
        next: (val) => console.log(val) ,  
        error: (error) => console.log("error") ,  
        complete: () => console.log("Completed")  
    )  
}  
}
```

For a full example see

<https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/>

# Example – app.component.ts

```
data=[];
```

```
ngOnInit() {
```

Subscribe with positional arguments

```
    this.obs.subscribe(  
        val => { console.log(val) },  
        error => { console.log("error") },  
        () => { console.log("Completed") }  
    )
```

```
}
```

```
}
```

For a full example see

<https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/>

# Observable typing

In the end, an **Observable** is a (dynamic, asynchronous collection of "things": like Java collections it can be typed.

So an **Observable** is, by default, an ensemble of anything:

**Observable<Any>**

but it could be restricted

**Observable<SomeType>**

Most often **SomeType** is an interface

# Observable typing

Although the Angular framework does not enforce a naming convention for observables, you will often see observables named with a trailing “\$” sign.

example:

```
export class StopwatchComponent {  
  stopwatchValue: number;  
  stopwatchValue$: Observable<number>;  
  start() {  
    this.stopwatchValue$.subscribe(  
      num => this.stopwatchValue = num  
    );  
  }  
}
```

# Emitting events and custom events

Components can also raise events with **EventEmitter**.

Using EventEmitter you can create a property and raise it using the `EventEmitter.emit(payload)`.

The Parent component can listen to these events using the event binding and read the payload using the `$event` argument.

A full, well elaborated example of custom events is here:

<https://www.concretepage.com/angular-2/angular-2-custom-event-binding-eventemitter-example>

# Q

How can a component communicate with the web server?

# HttpClient Observable

- This particular type of Observables are single-value streams: If the HTTP request is successful, these observables will emit only one value and then complete
- These observables will emit an error if the HTTP request fails



# Get syntax

```
get(url: string,  
    options: {  
        headers?: HttpHeaders | { [header: string]:  
            string | string[]; };  
        params?: HttpParams | { [param: string]:  
            string | string[]; };  
        observe?: "body|events|response|";  
        responseType: "arraybuffer|json|blob|text";  
        reportProgress?: boolean;  
        withCredentials?: boolean;  
    }): Observable<>
```

# Options

- **responseType**: Json is the default .The other allowed options are **arraybuffer**, **blob**, and **text**.
- **params** Allows to add URL or Get parameters
- **headers** allows you to add HTTP headers to the outgoing requests.
- **withCredentials** (boolean) When true HttpClient.get will request data with credentials (cookies)
- **observe**: by default HttpClient.get returns the **body** of the response. If you may need to read the entire response (including headers and status codes) . You can set the observe property to **response**.

# Post syntax

```
post(url: string,  
     body: any,  
     options: {  
         headers?: HttpHeaders | { [header: string]:  
             string | string[]; };  
         observe?: "body|events|response|";  
         params?: HttpParams | { [param: string]:  
             string | string[]; };  
         reportProgress?: boolean;  
         responseType: "arraybuffer|json|blob|text";  
         withCredentials?: boolean;  
     })  
): Observable<>
```

NOTE: similar syntax for the other HTTP commands

# Get example – blind type

## *get-request-component.ts*


```
import { Component, OnInit } from "@angular/core";
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'get-request',
  templateUrl: 'get-request.component.html' })
export class GetRequestComponent implements OnInit {
  totalAngularPackages;

  constructor(private http: HttpClient) { }

  ngOnInit() {
    this.http.get<any>('https://api.npms.io/v2/search?q=scope:
      angular').subscribe(data => {
        this.totalAngularPackages = data.total;
      })
  }
}
```

# Response to our HTTP request:

← → ↺  api.npmjs.io/v2/search?q=scope:angular

```
{
  "total": 40,
  "results": [
    {
      "package": {
        "name": "@angular/animations",
        "scope": "angular",
        "version": "12.2.12",
        "27T18:24:59.832Z",
        "links": {
          "npm": "https://www.npmjs.com/package/%40angular%2Fanimations",
          "homepage": "https://github.com/angular/angular/issues",
          "author": {
            "name": "angular",
            "publisher": {
              "username": "google-wombot",
              "email": "devops+npm@angular.io",
              "quality": 0.7530534723933453,
              "popularity": 0.739208532627518,
              "maintenance": 0.6666666666666666
            }
          },
          "search": "layout",
          "scope": "angular",
          "version": "12.0.0-beta.35",
          "description": "Angular Flex-Layout",
          "keywords": [
            "15T18:57:24.964Z",
            "links": {
              "npm": "https://www.npmjs.com/package/%40angular%2Fflex-layout",
              "homepage": "layout#readme",
              "repository": "https://github.com/angular/flex-layout",
              "bugs": "https://github.com/angular",
              "username": "angular",
              "email": "devops+npm@angular.io",
              "maintainers": [
                {
                  "username": "angular",
                  "email": "c",
                  "quality": 0.8640511656052347,
                  "popularity": 0.43824918491594533,
                  "maintenance": 0.9585954475022794
                }
              ],
              "search": {
                "name": "@angular/fire",
                "scope": "angular",
                "version": "7.1.1",
                "description": "The official Angular library",
                "angular",
                "firebase",
                "rxjs",
                "angularfire",
                "angularfire2",
                "date": "2021-10-18T15:41:48.795Z",
                "links": {
                  "npm": "https://www.npmjs.com/package/%40angular%2Ffire",
                  "homepage": "https://github.com/angular/angularfire",
                  "author": {
                    "name": "angular",
                    "publisher": {
                      "username": "angular",
                      "email": "devops+npm@angular.io",
                      "quality": 0.743833888048106,
                      "popularity": 0.38003563313838884,
                      "maintenance": 0.994461974532992
                    }
                  },
                  "search": "server",
                  "scope": "angular",
                  "version": "12.2.2",
                  "description": "LSP server for Angular Language Service",
                  "npm": "https://www.npmjs.com/package/%40angular%2Flanguage-server",
                  "homepage": "https://github.com/angular/vscode-ng-language-service",
                  "bugs": "https://github.com/angular/vscode-ng-language-service/issues",
                  "author": {
                    "name": "angular",
                    "publisher": {
                      "username": "angular",
                      "email": "devops+npm@angular.io",
                      "quality": 0.6188819000868024,
                      "detail": {
                        "quality": 0.7990352222503,
                        "popularity": 0.14909516781479393,
                        "maintainer": {
                          "name": "@angular/material-luxon-adapter",
                          "scope": "angular",
                          "version": "12.2.12",
                          "description": "Angular",
                          "npm": "https://www.npmjs.com/package/%40angular%2Fmaterial-luxon-
```

# Get example

*get-request-component.html*

```
<div class="card text-center m-3">
<h5 class="card-header">Simple GET Request with response type
<any></h5>
<div class="card-body">
Total @angular packages: {{totalAngularPackages}}
</div>
</div>
```

examples adapted from

<https://stackblitz.com/edit/angular-http-get-examples?file=app%2Fcomponents%2Fget-request-typed.component.html>

# Get example with typing

*get-request-typed-component.ts*

```
interface SearchResults {  
  total: number;  
  results: Array<object>;  
}
```

```
import { Component, OnInit } from "@angular/core";  
import { HttpClient } from '@angular/common/http';  
  
@Component({  
  selector: 'get-request-typed',  
  templateUrl: 'get-request-typed.component.html' })  
export class GetRequestTypedComponent implements OnInit {  
  totalAngularPackages;  
  
  constructor(private http: HttpClient) { }  
  
  ngOnInit() {  
    this.http.get<SearchResults>('https://api.npms.io/v2/search?q=scope: angular').subscribe(data => {  
      this.totalAngularPackages = data.total;  
    })  
  }  
}
```

# Get example

*get-request-typed-component.html*

```
<div class="card text-center m-3">
<h5 class="card-header">Simple GET Request with strongly
typed response </h5>
<div class="card-body">
Total @angular packages: {{totalAngularPackages}}
</div>
</div>
```



# Including html fragments

```
<div class="card text-center m-3">
<h5 class="card-header">Included external html</h5>
<div class="card-body">
  {{htmlString}}
  <div [innerHTML]="htmlString"></div>
</div>
</div>
```

```
export class GetRequestComponent implements OnInit {
  htmlString: string;
  ngOnInit() {
    this.htmlString="<h2>hello</h2>";
  }
}
```

Included external html

<h2>hello</h2>

**hello**

# Including html fragments from web

← → ↻ [latemar.science.unitn.it/demo/fragment.html](https://latemar.science.unitn.it/demo/fragment.html)

**hello from latemar**

```
export class GetRequestComponent implements OnInit {
  htmlString : string;
  ngOnInit() {
    const headers = new HttpHeaders({
      Accept: "text/html",
      Origin: "https://latemar.science.unitn.it"
    });
    const request = this.http.get<string>(
      'https://latemar.science.unitn.it/demo/fragment.html', {
        headers: headers,
        responseType: 'text' as 'json'
      }).subscribe(res => this.htmlString = res);
  }
}
```

# Cross Origin error

```
✖ ERROR
▼HttpErrorResponse {headers: {...}, status: 0,
  statusText: "Unknown Error",
  url: "https://latemar.science.unitn.it
  /demo/fragment.html"...}
  ▶error: ProgressEvent
  ▶headers: HttpHeaders
    message: "Http failure response for
    https://latemar.science.unitn.it
    /demo/fragment.html: 0 Unknown Error"
    name: "HttpErrorResponse"
    ok: false
    status: 0
    statusText: "Unknown Error"
    url: "https://latemar.science.unitn.it
    /demo/fragment.html"
    __proto__: HttpErrorResponse
```

# CORS - Cross-Origin Resource Sharing

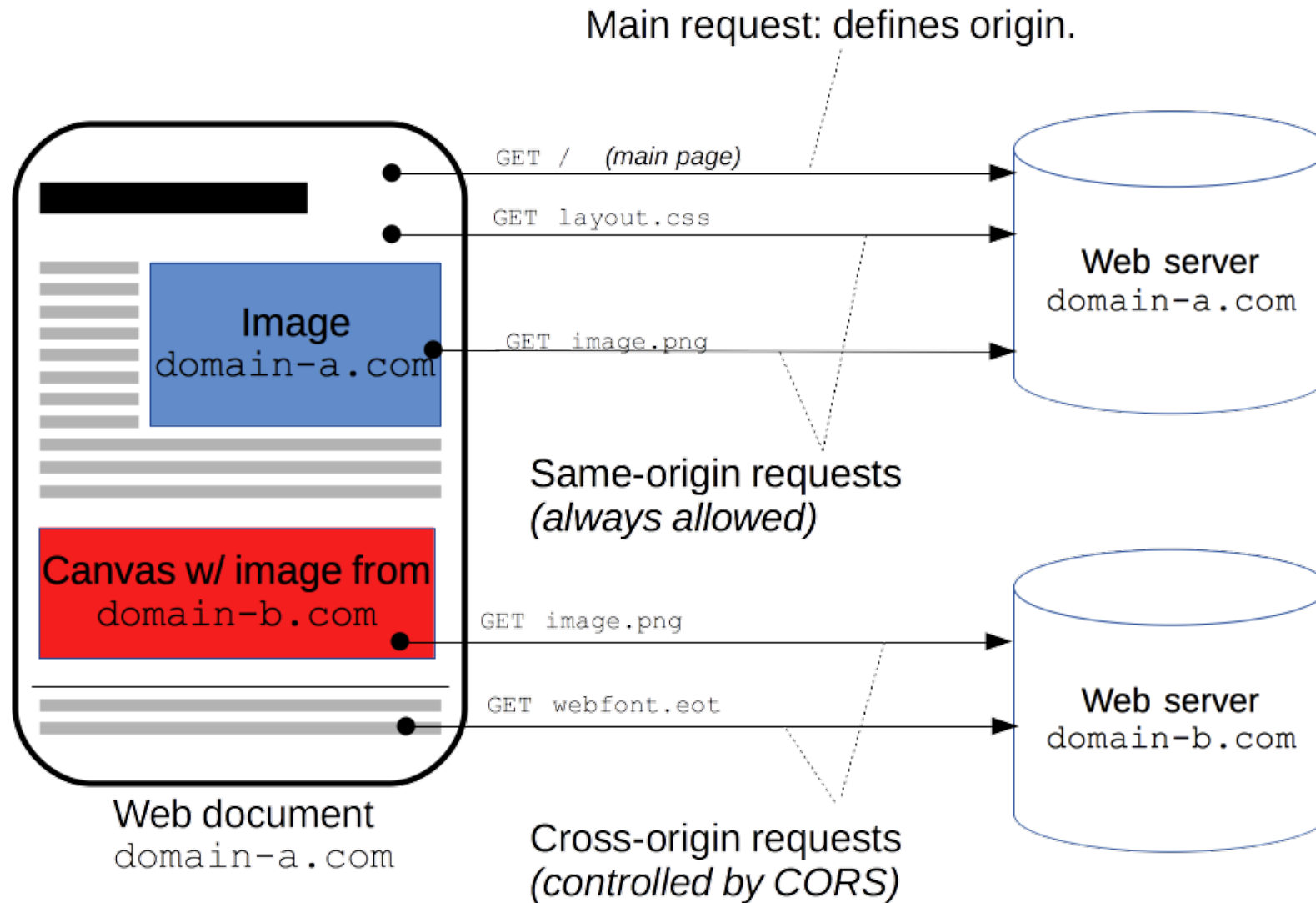


immagine da <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

# CORS

- **Cross-site XMLHttpRequest**
- Modern browsers support cross-site requests by implementing the **Cross-Origin Resource Sharing** (CORS) standard. As long as the server is configured to allow requests from your web application's origin, XMLHttpRequest will work. Otherwise, an `INVALID_ACCESS_ERR` exception is thrown.
- CORS failures result in errors, but for security reasons, specifics about the error *are not available to JavaScript*. All the code knows is that an error occurred.

```
const xhr = new XMLHttpRequest();  
const url = 'https://bar.other/resources/public-data/';  
xhr.open('GET', url);  
xhr.onreadystatechange = someHandler;  
xhr.send();
```

What the browser will send to the server:

GET /resources/public-data/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS  
X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0

Accept: text/html,application/xhtml+xml,  
application/xml;

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Connection: keep-alive

Origin: https://foo.example

# How the server will respond:

## CORS

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml
```

[...XML Data...]

Alternatives:

**Access-Control-Allow-Origin: <https://foo.example>**

For more detail, see <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

# Adding CORS to Apache

- To add the CORS authorization to the header using Apache, simply add the following line inside either the <Directory>, <Location>, <Files> or <VirtualHost> sections of your server config (usually located in a \*.conf file, such as httpd.conf or apache.conf), or within a .htaccess file:

**Header set Access-Control-Allow-Origin "\*"**

- [https://enable-cors.org/server\\_apache.html](https://enable-cors.org/server_apache.html)
- <https://poanchen.github.io/blog/2016/11/20/how-to-enable-cross-origin-resource-sharing-on-an-apache-server>



# Cross origin

This API enables cross-origin requests to anywhere:


<https://cors-anywhere.herokuapp.com/>

Usage example:

```
https://cors-  
anywhere.herokuapp.com/https://latemar.science.unitn.it/demo/  
fragment.html
```

Note: cookies are stripped!

# https://cors-anywhere.herokuapp.com

← → ↻  cors-anywhere.herokuapp.com/corsdemo

This demo of CORS Anywhere should only be used for development purposes, see <https://github.com/Rob--W/cors-anywhere/issues/301>.

To temporarily unlock access to the demo, click on the following button:

**You currently have temporary access to the demo server.**

← → ↻  cors-anywhere.herokuapp.com/https://latemar.science.unitn.it/demo/fragment.html

Missing required request header. Must specify one of: origin,x-requested-with

# Including html fragments from web

```
export class GetRequestComponent implements OnInit {
  htmlString : string;
  ngOnInit() {
    const headers = new HttpHeaders({
      Accept: "text/html",
      Origin: "https://latemar.science.unitn.it"
    });
    const request = this.http.get<string>
      ('https://cors-
anywhere.herokuapp.com/https://latemar.science.unitn.it/demo/f
ragment.html', {
      headers: headers,
      responseType: 'text' as 'json'
    }).subscribe(res => this.htmlString = res);
  }
}
```

see <https://stackblitz.com/edit/angular-http-get-with-include-uwhl6>

# Sanitization

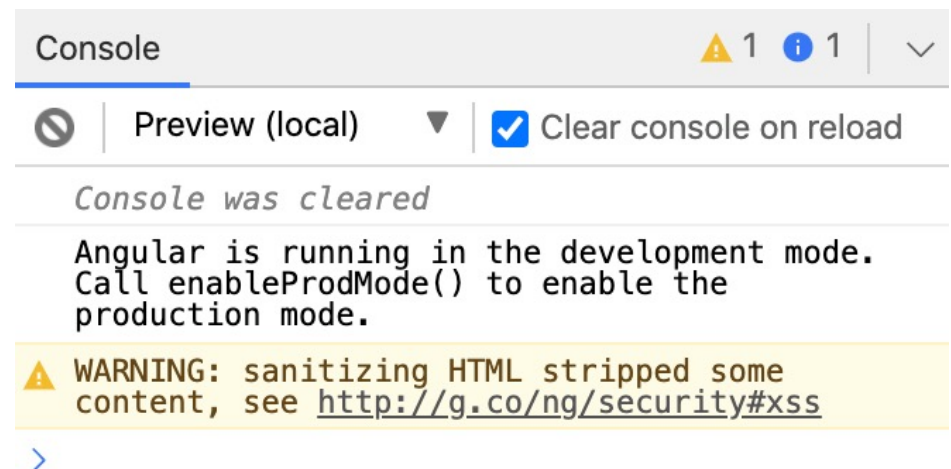
*Sanitization* is the inspection of an untrusted value, turning it into a value that's safe to insert into the DOM.

Angular sanitizes untrusted values for HTML, styles, and URLs; sanitizing resource URLs isn't possible because they contain arbitrary code.

In development mode, Angular prints a console warning when it has to change a value during sanitization.

see

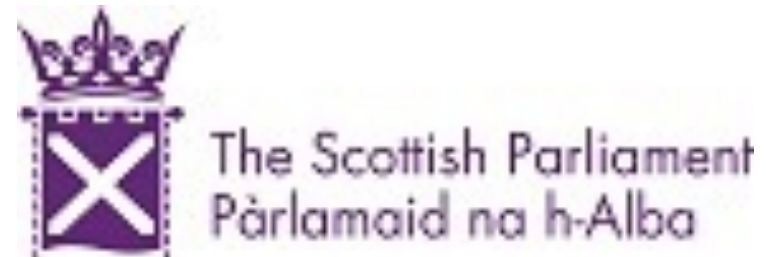
<https://angular.io/guide/security#sanitization-and-security-contexts>



# Get example

see <https://stackblitz.com/edit/angular-http-get-examples?file=app%2Fcomponents%2Fget-request-error-handling.component.html>

play with a json datasource, like



<https://data.parliament.scot/api/members>

<https://data.parliament.scot/api/members/1848>

<https://data.parliament.scot/#/api-list>