**Observables**

Observable – o sursa de date, un obiect care se importa, de obicei pentru task-uri async pe care o importam in RxJS

Poate fi constituit din:

* Events: User input
* Http requests
* Trigger-uri din cod, etc.

**Observer** **Observable**

**Observable:**

Exista 3 modalitati de a procesa un “data package” (3 etape in care poti rula cod, 3 hooks):

1. Handle data – pentru requesturi ASYNC
2. Handle Error – in caz ca nu-i convine ceva sau crapa
3. Handle Completion (un Observable nu trebuie neaparat sa fie finalizat)

Diagram, timeline

Description automatically generated

***Observables*** *are constructs to which you subscribe to be informed about changes in data.*

**Observables** nu se opresc, by default, din a emite event-uri la care esti **subscribed.** ( cu cateva exceptii, ex: route.params )

Pentru a opri subscribtions -> unsubscribe:

Subscribe() returneaza un obj type: Subscription, care se poate stoca intr-o var.

Pe OnDestroy se poate folosi subscription.unsubscribe() -> cand componentul este distrus -> unsubscribe.

Text

Description automatically generated

**Custom Observables**

Text

Description automatically generated**Observer.next()** – emite o valoare noua

**Observer.error()** – pentru a arunca o eroare

Text

Description automatically generated

Dupa ce o eroare a fost emisa, Observable se opreste.

In subscribe() –poate primi eroarea ca un al 2-lea arg, sub forma de functie care tine eroarea ca si arg.

Graphical user interface, text

Description automatically generated with medium confidence

**Observer.complete()** – instinteaza observerul cu privire la finalizarea task-ului.

Opreste task-ul fara a arunca vreun mesaj

Text

Description automatically generatedIn subscribe() –poate primi eroarea ca un al 3-lea arg, sub forma de functie fara arg.

Dupa ce observer.error() arunca mesajul, angular **nu ruleaza** observer.complete() ! O eroare anuleaza Observable, nu il finalizeaza.

A picture containing timeline

Description automatically generated**Operators**

Se acceseaza prin **observable.pipe()** (built-in in RxJS)

**Observable.pipe()** primeste o functie ca si argument. Functia callback primeste ca si argument parametrul primei functii din .subscribe()

Text

Description automatically generated

In momentul de fata subscribe inca returneaza linia 42. Ca sa filtram **data**, pasam fn cu .pipe() ca si prim argument in .subscribe(). Cu alte cuvinte, ne abonam la **data** filtrata/transformata prin pipe(), in loc de data primita direct de la observable (raw).

Text

Description automatically generated

Folositor mai ales atunci cand se primesc date de la un server sau API si se doreste filtrarea sau transformarea datelor primate inainte, date la care ulterior se face subscription.

**Subjects**

Un fel de Observable, mai special: primeste metoda .next(), ca si un observer normal, doar ca aceasta **poate fi chemata si din afara componentului unde a fost definit observerul.**

Ca si functionalitate se aseamana cu metoda EventEmitter – service.

Chart, diagram, funnel chart

Description automatically generated

Syntax:

Text

Description automatically generated

Pe componenta de unde se emite:

Text

Description automatically generated

Text

Description automatically generatedPe componenta in care se primeste:

Subjects necesita **unsubscribe()** pe ngOnDestroy().

Se recomanda folosirea de **Subjects** in schimbul la **EventEmitter**, din motive de performanta si de compatibilitate. In plus, folosind **Subjects**, datele trimise prin event pot fi filtrate / transformate cu ajutorul **Operatorilor** (pipe).

Pentru comunicare pe directia copil -> parinte, atunci cand se foloseste **Output(), trebuie folosit EventEmitter**, **cu Subject nu functioneaza.**