Date: / /

RxJS Operators

1. Pipe → Function that takes an observable as its Input and return another Observable

2. Filter → Filter source observable before subscribe

3. Tap → perform other side-effects / different actions without modify the data stream.

Eg →
```
obs = new Observable ((observer) => {
  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.complete() }).pipe(
  tap( data => console.log('tap' + data))
  filter( data => data > 2 ),
  tap(data => console.log('filter' + dat
  ,map((val => return val as no*2 })
)
```

Date: / /

We use pipe to chain the tap operator which just logs value of source observable to console.

We use filter, to filter out observable according to our requirement.

4. Take → emits only the first count values by the source observable mentioned in take operator.

5. Take Until → use for unsubscribe observable until not getting response. Can use for complete and destroy observable after subscribe

6. Take Last → emits only last count values.

7. Take While → emit value if condition is true.

Date: __/__/__

Eg → obs = of (1, 2, 3, 4, 5, 6) . pipe (

```
    takeWhile (val => val < 3, true),
    takeLast (3),
    take (2),
    takeUntil (Subject)
);
```

Output → 1, 2 (takeWhile)
4, 5, 6 (takeLast)
1, 2 (take)

8. retryWhen ⇒ retry with condition.
If you want to retry after few seconds, not instantly then use retryWhen

Foreg – If getting 404 error, then you should retry

9. FromEvent – handling DOM events using target Element and event type

Date: / /

10. debounce Time → delays the value emitted by source.

11. distinct Until Changed → emits all items that are distinct from previous items

12. Subject - Subject does not return the current value of subscription.

It triggers only on .next(value) call and return output.

Eg → var Subject = new Subject();
Subject.next(1);
// will not output any value

Subject.subscribe({
next: (v) => conse.log(v);
});
Subjec.next(1);

// Logs 1

13. behaviour Subject → It will return current value or initial value on subscription.

14. asyncSubject → emits last value on completion.

15. Concat ⇒ join multiple Observable and emits one by one

Eg → Concat (
of (1,2,3), of (4,5,6),)
. subscribe (console.log);

// 1,2,3,4,5,6

16. Share Reply ⇒ use with async, avoid duplicate request

17. Zip ⇒ pass 2 observables using zip. when change will be detected in 2 observables then only subscription will emit value

Date: / /

Use Case – When you have to make several http requests at same time and need to complete all before emitting value.

Eg → const sourceOne = of('Deepa');
const sourceTwo = of('Jyoti');

```
const eg = zip(
    sourceOne,
    sourceTwo.pipe(delay(1000)),
);
const subscribe = eg.subscribe(
    val => console.log(val));
```

18. ForkJoin → Wait for all observables to complete and then emit an array of last emitted values.

Join multiple Observable to a single one.

If any observable came through error, forkJoin will not return any value

Flattening Operators

While using multiple observables into nested Subscription

1) Merge Map → Simultaneously executes observables.

* Sequese of execution not guaranteed
* If any observable throw error, it will emit other values.

2) Concat Map → does not subscribe to next observable until previous completes

* Execute one by one
* sequest of execution guaranteed.

3) exhaust Map → map to inner observable and ignore other until observable complete

4) Switch Map → try to resubscribe observable by cancelling previous