RXJS

1. **Pipe** - function that takes an Observable as its input and returns another Observable
2. **Filter** - filter source observable before subscribe
3. **Tap** - Perform other side-effects / different actions without modify the data stream. When fetching data, then if need to perform any action.
4. **Take** - emits only the first count values by the source observable mentioned in take operator. Like take(5) return only first 5 values after subscription.
5. **TakeLast** - emits only the last count values by the source observable mentioned in take operator. Like takeLast(5) return only last 5 values after subscription.
6. **TakeUntil** - use for unsubscribe observable until not getting response. Can use for complete and destroy observables after subscribe.
7. **retryWhen** - retry with condition. If you want to retry after few seconds not instantly then use retryWhen. If getting 404 error then should not retry, because that's mean server url mismatch.
8. **fromEvent** - handling DOM events using targetElement and event type
9. **debounceTime** - delays the value emitted by the source
10. **distinctUntilChanged** - emits all items by the source observable that are distinct from the previous items
11. **Subject** - subscriber will only get published value that were emitted after the subscription
12. **behaviorSubject** - subscribe will only get last emitting value. Cache previous value
13. Subject & behaviorSubject are multicast observable, instead of sending information to one subscriber they can send data to multiple subscribers.
14. **asyncSubject** - emits last value on completion. When observable will be complete then only it will emit last value
15. **Concat** - join multiple observables and emit one by one
16. **shareReply** - avoiding duplicate httprequests. Use with async operator
17. **Zip** - pass two observables using zip. When change will be detected in two observables then only subscription will emit value. Needed when have to make severals http requests at the same time and need to be completed all before emitting the values.
18. **forkJoin** - wait for all observables to complete and then emits an array of last emitted values. Join multiple observables to a single one. If any of the observables through error, then forkJoin will not return any value
19. **Merge** - merge multiple observables and emit values as per order
20. Flattening operators - comes to rescue when we have nested subscription. 1. MergeMap 2. concatMap 3. exhaustMap 4. switchMap

21. **exhaustMap** - map to inner observable and ignores other until the observable complete. Suppose you are saving data, after saving data button will be disabled. Then you can use this operator to ignore other actions until not getting any response from server.
22. **switchMap** - try to resubscribe the observable by cancelling the previous one. Mainly used with search box. Complete previous observable and create new one

23. **mergeMap** - simultaneously executes observables. if any of the observables throw error, mergeMap will emits other observable values.
24. Executes all observables parallelly, that means sequence of execution is not guaranteed.

25. **concatMap** - does not subscribe to the next observable until the previous completes. waits for the previous Observable to complete before creating the next one.
26. Execute observables one by one. One execution will finish then only another execution will start. That means sequence of execution is guaranteed.
27. Example is mobile notification.

mergeMap flattering

```
import { Component } from '@angular/core';
import {
  mergeMap,
  flatMap,
  concatMap,
  switchMap,
  exhaustMap,
  delay,
  tap,
} from 'rxjs/operators';
import { from, of, forkJoin } from 'rxjs';
import { HttpClient } from '@angular/common/http';

interface User {
  id: number;
}

interface Post {
  id: number;
}

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
```

```
})
export class AppComponent {
  USERS = 'https://jsonplaceholder.typicode.com/users/';
  POSTS = 'https://jsonplaceholder.typicode.com/posts/';
  data: [User[], Post[]];

  constructor(private http: HttpClient) {
    this.withoutMergMap();
    this.withMergMap();
  }

  withoutMergMap() {
    const users = this.http.get<User[]>(this.USERS);
    const posts = this.http.get<Post[]>(this.POSTS);

    forkJoin([users, posts]).subscribe((res) => {
      console.log('withoutMergMap ------------------------');
      console.log(res);
    });
  }

  withMergMap() {
    const users = this.http.get<User[]>(this.USERS);
    const posts = this.http.get<Post[]>(this.POSTS);

    forkJoin([users, posts])
      .pipe(mergeMap((res) => res))
      .subscribe((res) => {
        console.log('withMergMap ------------------------');
        console.log(res);
      });
  }
}
```