


Technical Debt and Research

This page is reserved for discussion and proposing technical debts found in the project. It is not only technical lead's responsibility to contribute here, but also team's responsibility.

Shipment Service Tech Debt

Technical Debt epic for Q3 to be tracked at:

 [AFN-4336](#) - [Q3 2022] Technical Debt & Research IN PROGRESS

key	summary	created	updated	due	assignee	reporter	priority	status	resolution
-----	---------	---------	---------	-----	----------	----------	----------	--------	------------

 Jira project doesn't exist or you don't have permission to view it.

[View these issues in Jira](#)


Pudo Aggregator Tech Debt

key	summary	created	updated	due	assignee	reporter	priority	status	resolution
-----	---------	---------	---------	-----	----------	----------	----------	--------	------------

 Jira project doesn't exist or you don't have permission to view it.

[View these issues in Jira](#)

Reduce creation of Mono and Flux objects

 [AFN-4344](#) - Remove manual creation of Mono and Flux objects OPEN

In multiple places and especially in converters there are creation of Mono and Flux for absolutely no reason other than filter them out with the flatMap just after. The webflux already provides Mono and Flux object on the: webclient, repository, request and response objects. Use those instead of creating new ones to flatMap them just after.

The code will look easier to understand, smaller and better performant when less objects are created.

Example :

```
return Mono.defer(() -> siteIdReadRepository.findByName(siteId)
    .flatMap(siteIdResponse -> redisOperations.opsForValue()
        .set(cacheKey, objectMapper.toJsonString(siteIdResponse), Duration.ofSeconds(cacheExpiryTime))
        .then(Mono.just(siteIdResponse))));
```

```
return siteIdReadRepository.findByName(siteId)
    .doOnSuccess(siteIdResponse ->
        redisOperations.opsForValue()
        .set(cacheKey, objectMapper.toJsonString(siteIdResponse), Duration.ofSeconds(cacheExpiryTime))
    );
```

The 2 blocks of code do the same, but one has less code and less objects created.

In general ***Mono.just, Mono.defer, Flux.just, Flux.fromIterable*** must be used only on some special cases when really there is a need to do it, otherwise provides almost 0 value and decreases readability.

1. Difference between map and flatMap

Yes there is a difference between a flatmap and map.

flatMap should be used for non-blocking operations, or in short anything which returns back Mono, Flux.

map should be used when you want to do the transformation of an object /data in fixed time. The operations which are done synchronously.

eg: In the below example, we have used **flatMap** for saving person in database, because it's an async operation for which time is never deterministic, while for converting the person into EnhancedPerson object, we have used map, because it's a blocking and synchronous operation for which time taken is always going to be deterministic.

```
return Mono.just(Person("name", "age:12"))
    .map { person ->
        EnhancedPerson(person, "id-set", "savedInDb")
    }.flatMap { person ->
        reactiveMongoDb.save(person)
    }
```

2. Using nested flatMap/map

Most of the programmers with imperative programming mindset often end up writing a lot of code in [map/flatMap](#). If you ever see a flatMap/map operation nested inside another flatMap/map or a map operation it's a code smell. For example in the below code we see 2 nested flatmap operations which makes it very difficult to read.

```
fun makePersonASalariedEmployee(personId: String): Mono<Person> {

    return personsRepository.findPerson(personId)
        .flatMap { person ->
            employeeService.toEmployee(person)
                .flatMap { employee ->

salariedEmployeeService.toSalariedEmployee(employee)
                }
        }
}
```

The same code can be transformed by following our age old time tested Unix rule:

Do One Thing and Do It Well

Inside every flatMap/map restrict yourself to transform/process data only once. And if you ever need more transformations, then that's **not and never** going to be in the jurisdiction of the current flatMap/map, rather let the data flow and attach a separate flatmap/map transformers in the chain. It's good to treat every flatMap/map with [Single responsibility principle](#).

```
fun makePersonASalariedEmployee(personId: String): Mono<Person> {

    return personsRepository.findPerson(personId)
        .flatMap { person ->
            employeeService.toEmployee(person)
        }
        .flatMap { employee ->
            salariedEmployeeService.toSalariedEmployee(employee)
        }
}
```