Visual Atlas of Software Development

Yeray Darias

Index

Authors

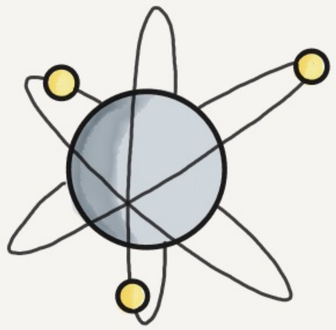
Introduction

General Concepts

ACID properties

from Wikipedia

Atomicity



Each transaction will be atomic

↓

"All or nothing"

Durability

Once a transaction has been comitted, it will remain, "no matter what"

Consistency

Each transaction will bring the database from one valid state to another valid state

Isolation

The concurrent execution of transactions result in a system that would be obtained if transactions were executed sequentially

CAP theorem

In a distributed system you can only provide two options between...

CONSISTENCY

every read receives the most recent write...

every request receives a response...

AVAILABILITY

PARTITION TOLERANCE

In a network problem the system continues to operate

C

A

P

Postgre SQL

Spanner

?

MongoDB

Couch DB

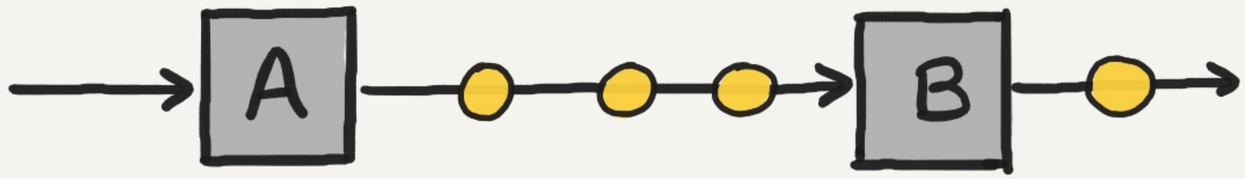
... in the presence of a network partition you can only provide A or C ...

So, nowadays it is A or C

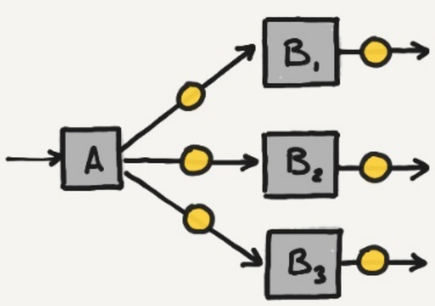
Backpressure

A system A produces requests at a pace of 3 req/second ...

... but system B processes the requests at a pace of 1 req/second.

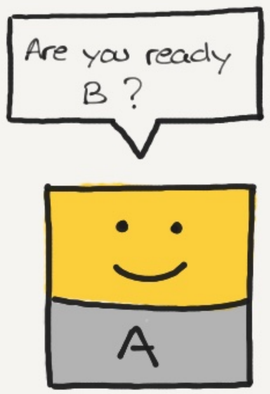


Scale up

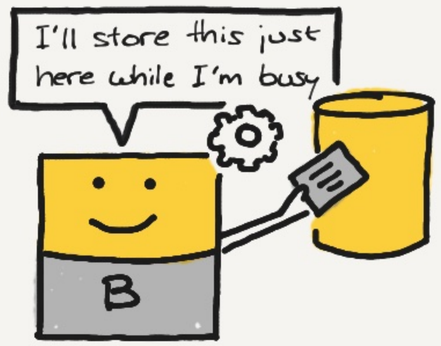


It could be costly but sometimes this is the way to proceed if you can not lose requests, and you have "real time" requirements.

Control producer speed to allow the consumer ending before receiving another request.



Buffer the requests to process them later.



A queue also works well !

Drop the overload !

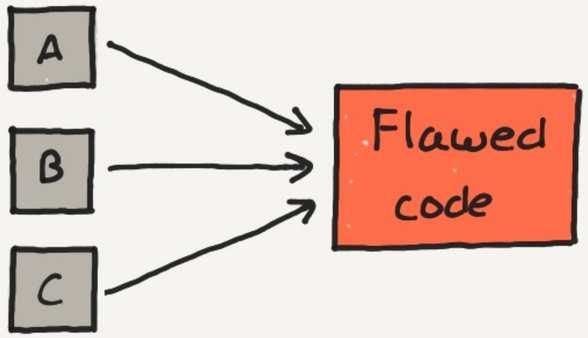
There are quite a lot of systems where back pressure can just be ignored.

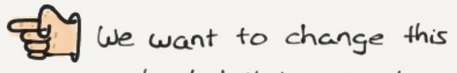
Patterns

Gang of four

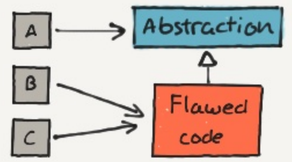
Refactoring

Branch By Abstraction as explained by Martin Fowler



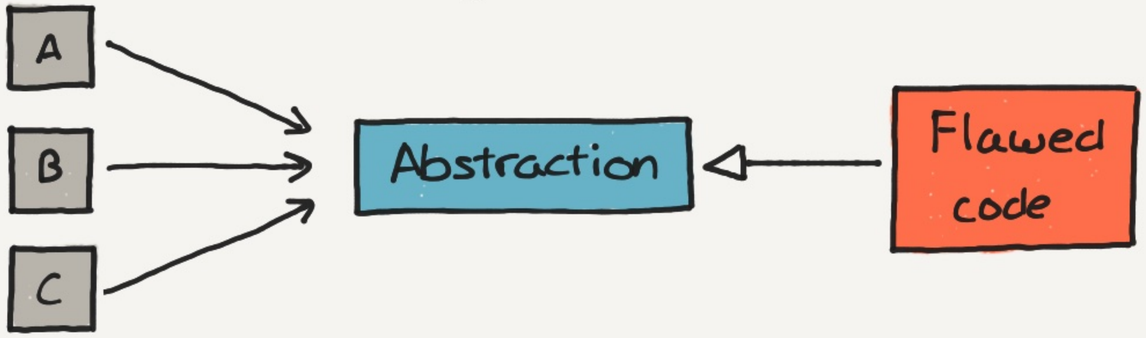


We want to change this code but it is going to take some time and we want to keep pushing the code to production meanwhile.

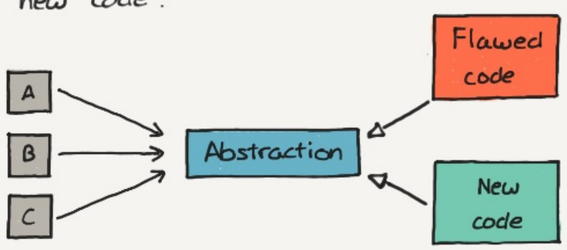


It can be done in separate steps while keeping everything working.

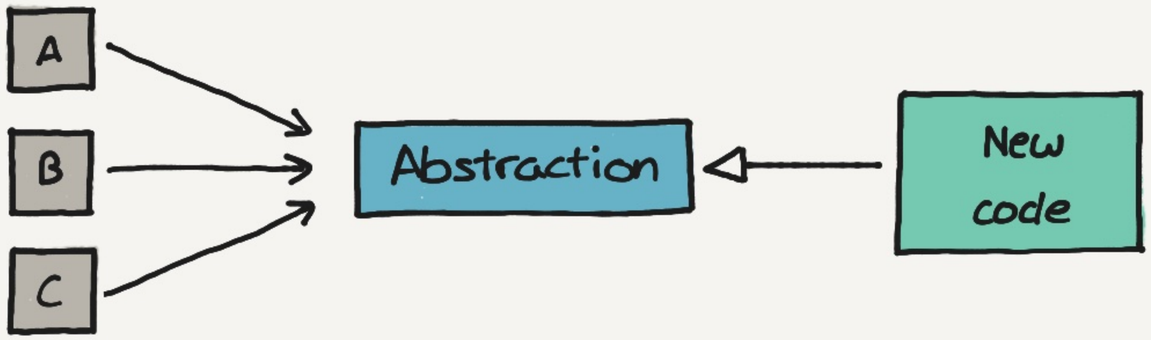
Create a common abstraction

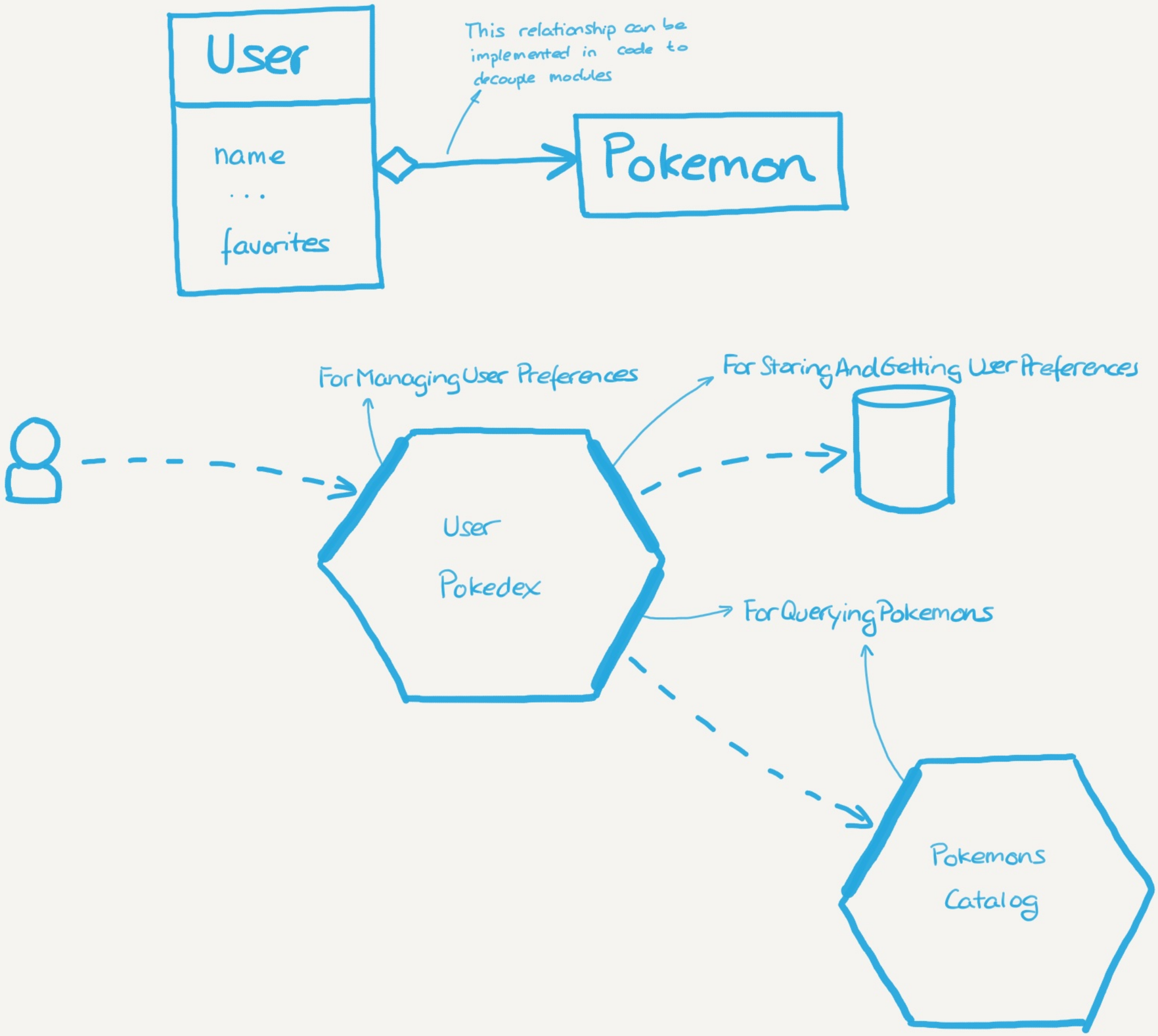


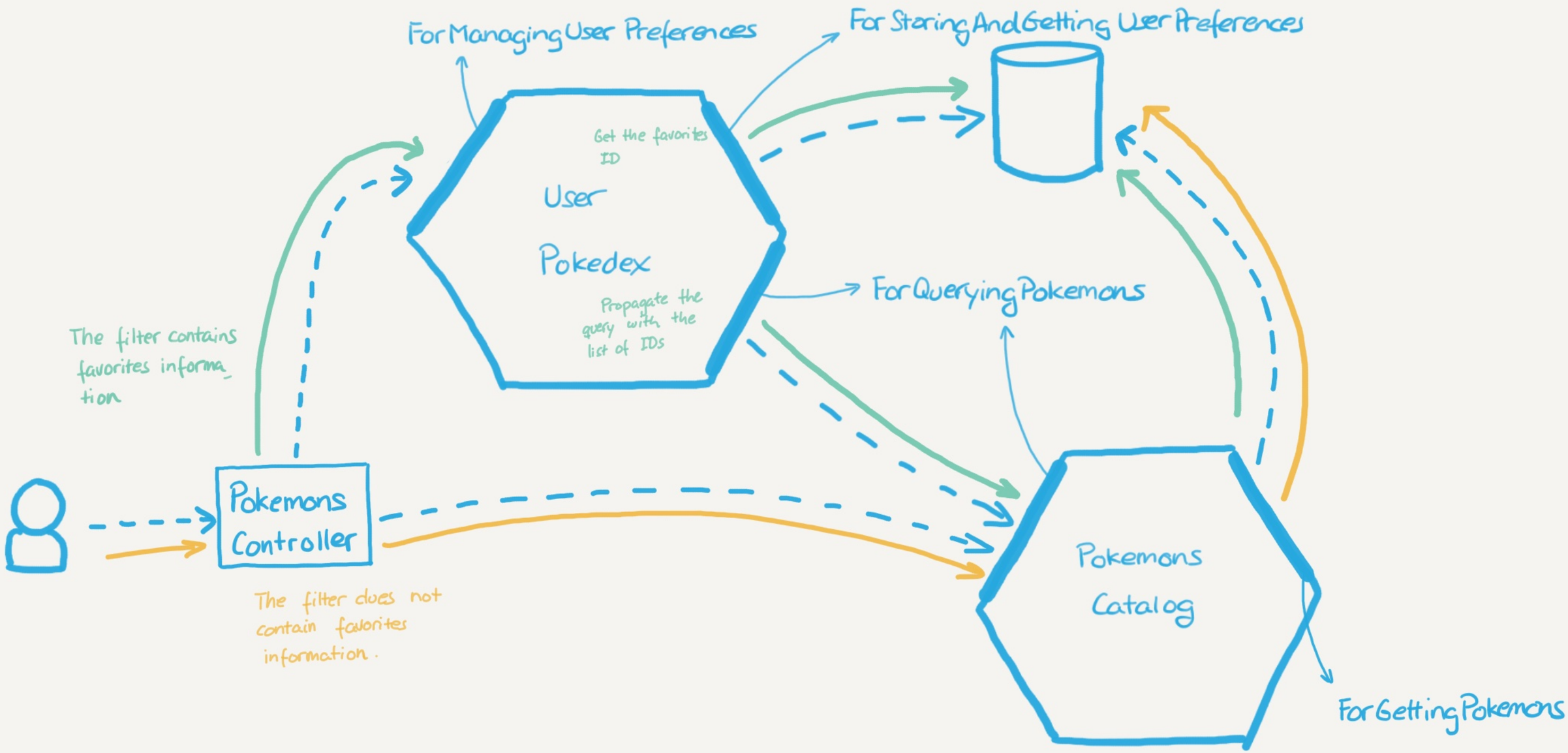
We can implement a small slice each time. Using dep. injection we can decide to use the new code.

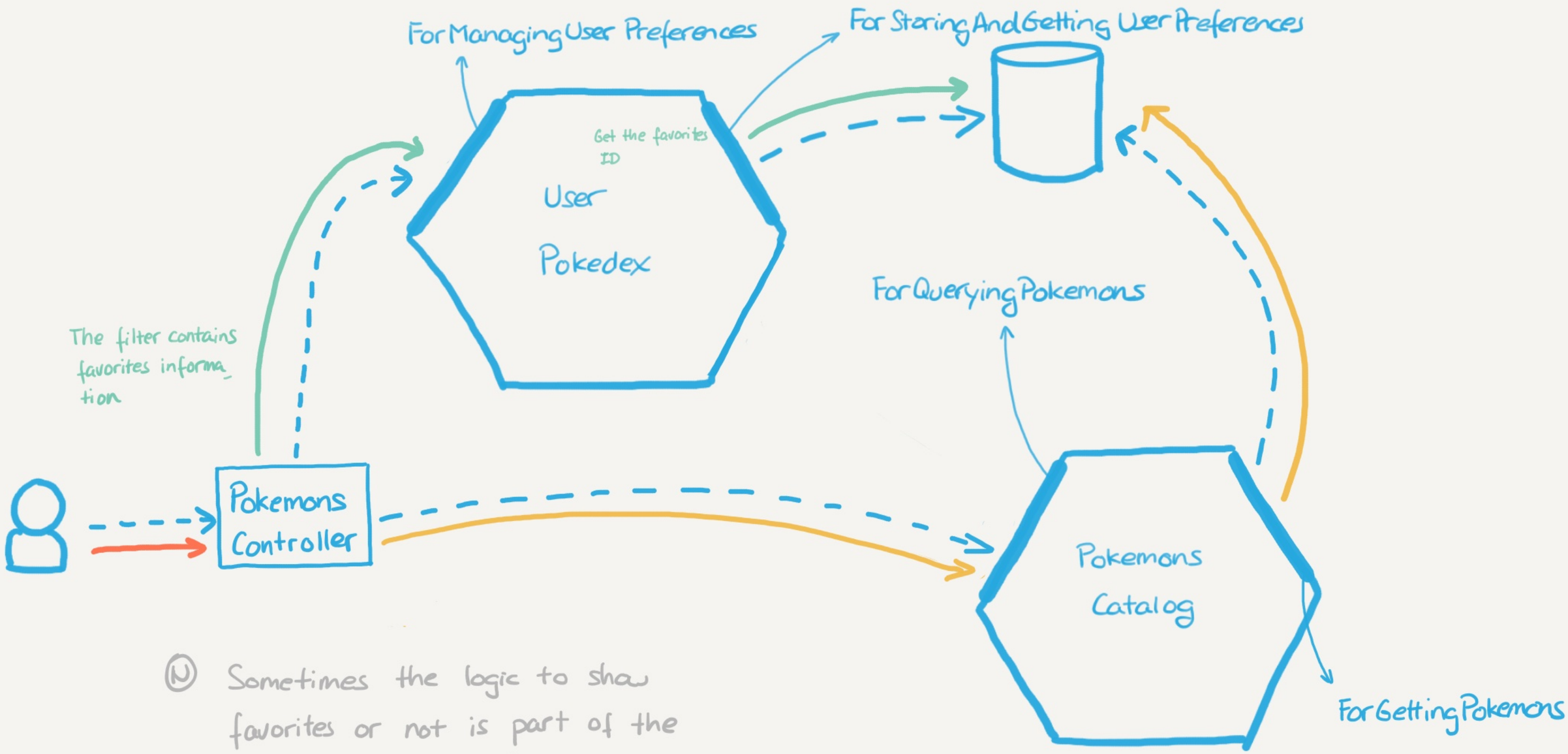


Implement new code until you can get rid of the old one




User
name
...
favorites
This relationship can be implemented in code to decouple modules
Pokemon
For Managing User Preferences
For Storing And Getting User Preferences
User
Pokedex
For Querying Pokemons
Pokemons
Catalog


For Managing User Preferences
For Storing And Getting User Preferences
Get the favorites ID
User
Pokedex
Propagate the query with the list of IDs
For Querying Pokemons
The filter contains favorites information
Pokemons Controller
The filter does not contain favorites information.
Pokemons Catalog
For Getting Pokemons

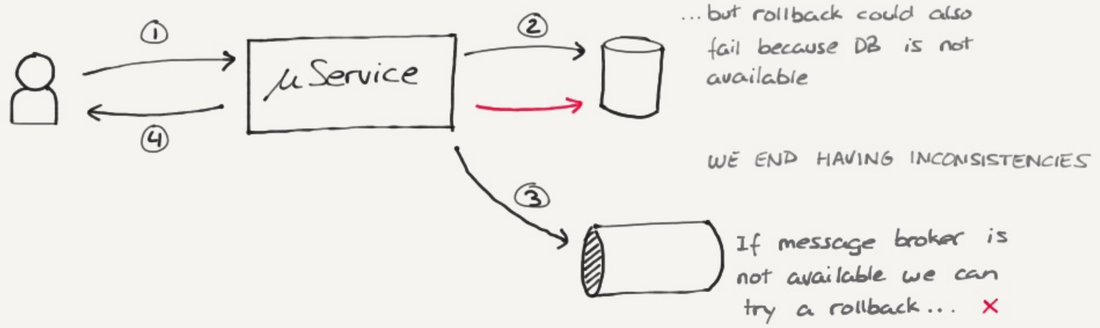


N Sometimes the logic to show favorites or not is part of the UI not business. So, leaving some conditionals to controller is not so bad.

Microservices

Outbox pattern

solves



Ensures Guaranteed Message Delivery

If 2. fails there is no problem because the request ends in error, no data is stored.

Send request

1

HTTP API

2 (a) update entity
(b) save message to be published

outbox
data
state

The payload of the event

3

request attended

Check for messages

4

Undelivered

Delivered

Discarded

After X retries it should be discarded.

6 Update message state

Dispatcher

If 4. or 5. fail it keeps retrying.
If 6. fails it will retry leading to the same event sent twice.

¯_(ツ)_/¯

Better than the alternative.

message broker

Ideally, consumers will be idempotent

5

Push message

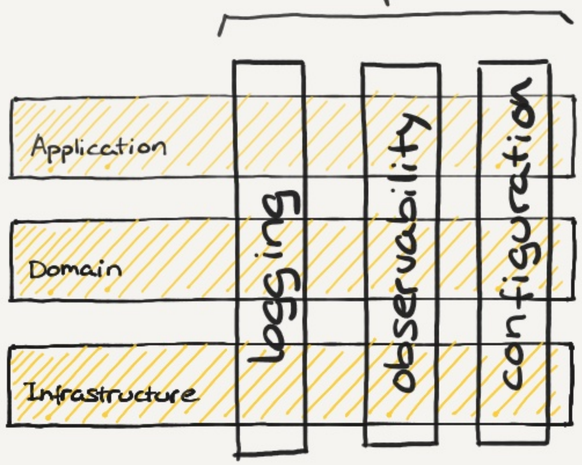
Based on @pkritiotis works

@ydarias

Sidecar Pattern

by @mstrYoda_

Cross-cutting concerns are requirements needed in different layers of an application and different applications

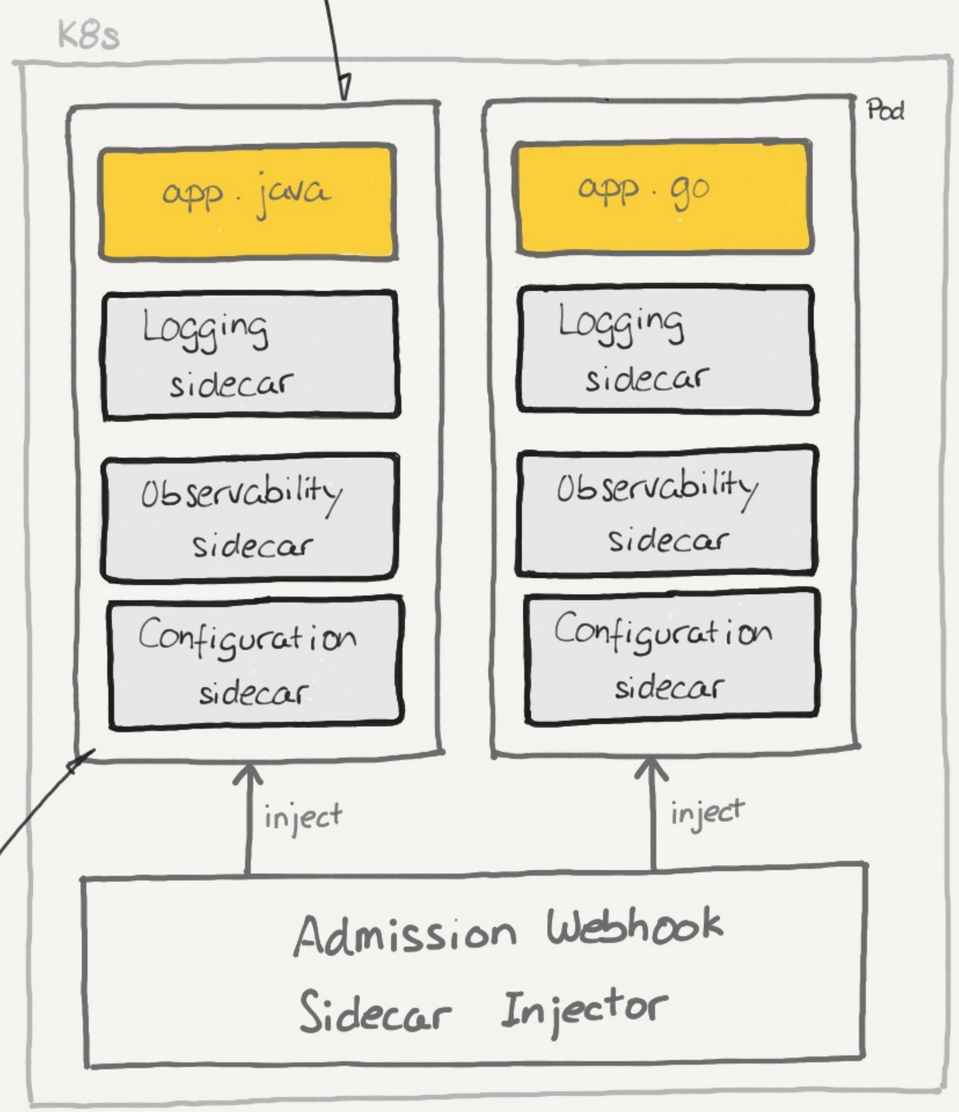


In microservices architectures is usual to have more than one language. You don't want to implement a library for each one. And an API could not scale nicely, creating too much latency.

Istio is the clear example of how to use this pattern to provide common requirements.

Each μservice is deployed with its own set of sidecars independently of the language.

Ⓝ The interaction could be through disk or an API accessed with localhost. Depends on the necessity.



The main container and the sidecars share the same network (and other resources)

Based on @mstrYoda_ works

@ydarias

Miscellaneous

LEAKY BUCKET

Imagine a bucket with a hole at the bottom.

If the input flow is larger...

... than the output flow, the bucket will overflow, eventually.

It is used at computer networks algorithms to decide on traffic policing.

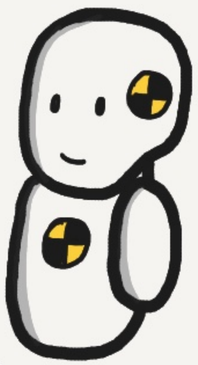
Add diagram of an implementation

It can also be used in other cases, like a decision maker on retry policies (depending on the error type)

Testing

TEST DOUBLES

Martin Fowler Ed.



THE DUMMY

is passed but never actually used

PAPPLE

THE FAKE

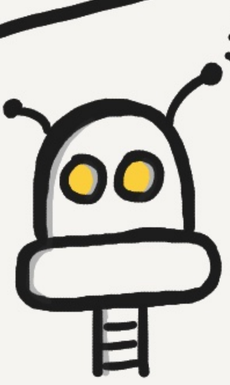
is a "working" implementation, with shortcuts (not production code)

THE MOCK

can be pre-programmed with expectations and can be checked during verification

THE SPY

is a stub that also records some information based on how they were called



THE STUB

provides canned answers to calls during the test just what is programmed at the test