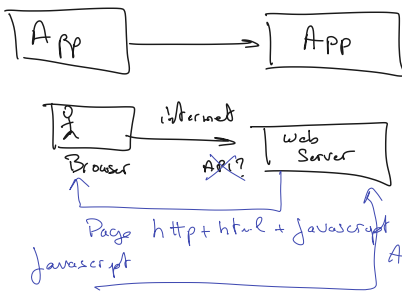
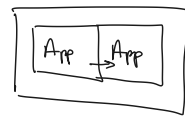
Hello Every Body

API is Application Programming Interface

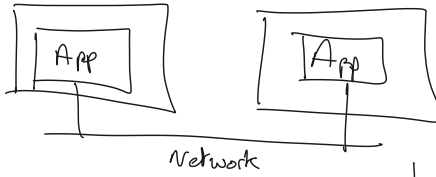


LPC
Local Procedure Call
OS



RPC

Remote Procedure Call



Remote → Base, Tech Protocol
→ Format of Data

RPC
- CORBA → 80s
- DCOM → 90s (Ms)
- 2000 th → SOAP (Open)
- 2005 → Rest

2000, SOAP Web Services

SOAP Protocol → XML Based

Schema / Specifications using

WSDL flow

Web Service Description Language

WS_ Security

- —
- --
- ...

→ ALL Standard

→ Server to Server

→ Computer to Server

No | ~~IoT~~ Telephone → To Server

Biggest Problem is XML Processing

~~Issue~~ is Energy Consumption

We Needed Something Lighter

~~SOAP~~ → outdated

→ Rest API

SOAP → XML + http

REST → JSON (xml, csv, PDF, ...) + http

SOAP → Many Specifications

REST → Guidelines (open)

↳ Too open

(No) Few Specifications

Do whatever you want (No Tech Rules)

⇒

SOAP → operation based

↳ Verb, method, procedure

→ Parameters

→ 1 Result

Class (dev point of view) → expose as Web Service

REST → Entity based

Entity → like a Table / Database

↳ can be enumerated

ex: Product, Category, Customer, Invoice, Order

C reate
R ead
U pdate
D elete

execute a workflow

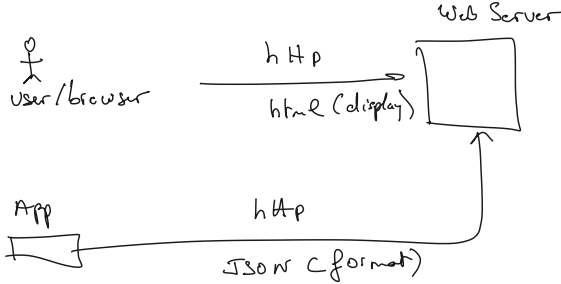
↓ Transaction

"Create" a workflow Entity

HTTP

Hyper Text Transfer Protocol

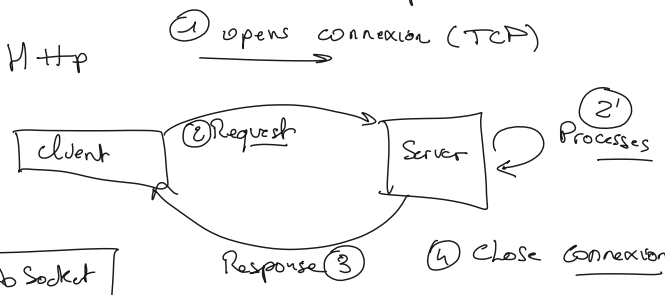
↳ Web

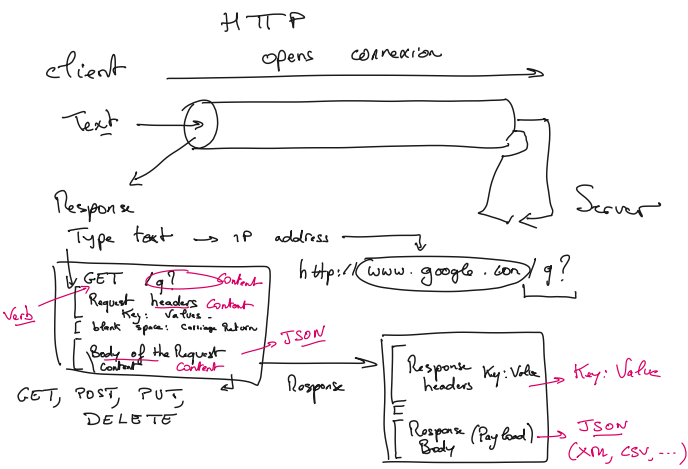


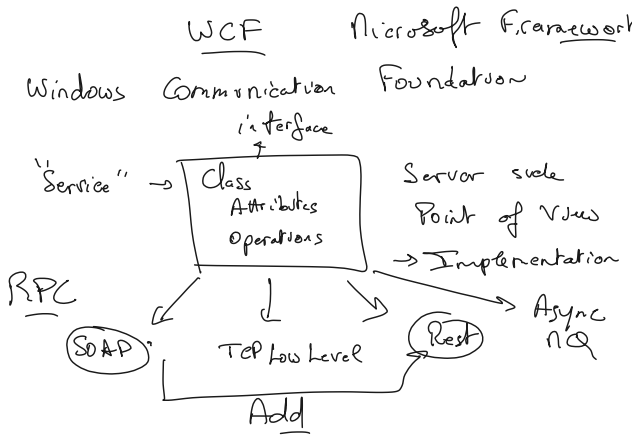
OAS (Swagger v3) → WADL(v2)

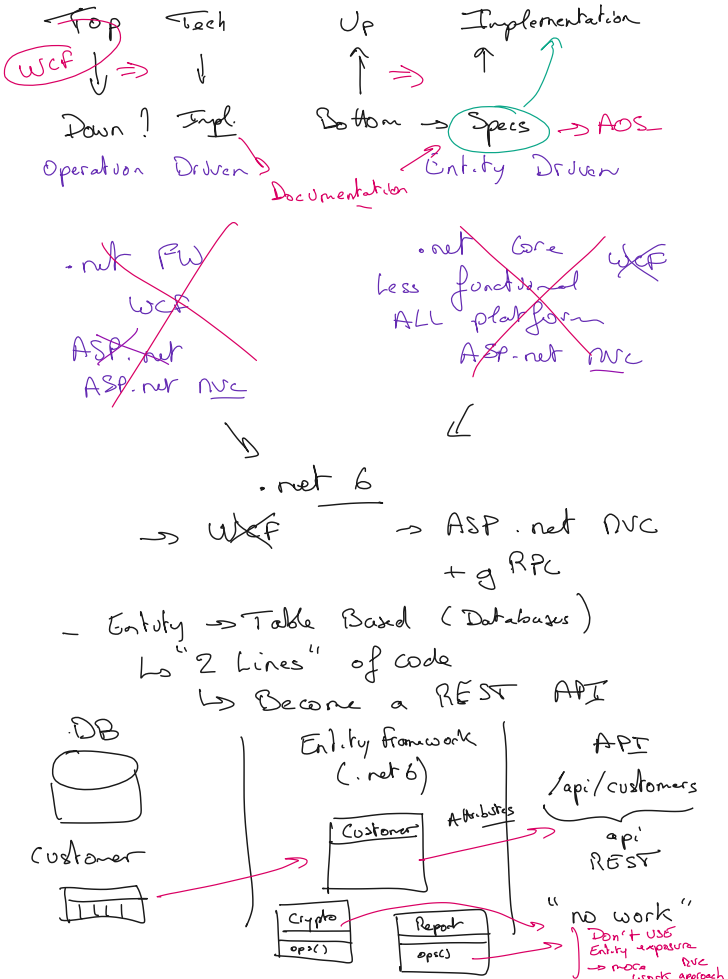
WADL(v1) → WADL(v2)

REST = hHp + JSON




HTTP
client
opens
connection
Text
Response
Server
Type text → IP address
GET
q?
Content
Request
headers
Content
Key: Values
blank space: Carriage Return
Body of the Request
Content
Content
Verb
JSON
GET, POST, PUT, DELETE
http://www.google.com/q?
Response
Response headers
Key: Value
Key: Value
Response (Payload) Body
JSON
(xml, csv, ...)


WCF
Microsoft Framework
Windows Communication Foundation
interface
"Service"
Class
Attributes
Operations
Server side
Point of View
→ Implementation
RPC
SOAP
TCP Low Level
Rest
Async
nQ
Add


Top
Tech
Up
Implementation
WCF
Down ?
Impl.
Bottom
Specs
AOS
Operation Driven
Documentation
Entity Driven
.net FW
WCF
ASP.net
ASP.net MVC
.net Core
less functional
ALL platform
WCF
ASP.net MVC
.net 6
WCF
ASP.net MVC
+ gRPC
Entity → Table Based (Databases)
"2 Lines" of code
Become a REST API
DB
Customer
Entity framework
(.net 6)
Customer
Attributes
API
/api/customers
api
REST
Crypto
op()
Report
op()
"no work"
Don't use
Entity
→ more
work
approach

Server Side

NS C#	ASP.net MVC (.net6)
php	Symfony
Java	Spring Boot
Python	Jengo + flask (?)

Entity Exposure as Rest APIs are common approaches

Asynchronous Operations

- Operation should be 2 seconds long maximum

Synchronous → await the result

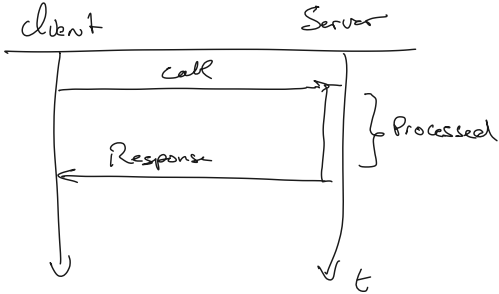
↓

Asynchronous → launch an operation and (be inform when done (or not))

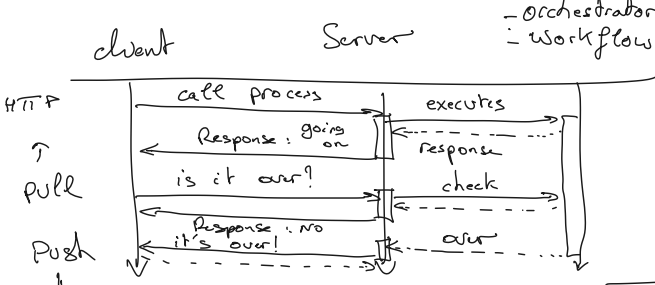
Synchronous → RPC (Place a phone call)

Asynchronous → Message Queuing (Send a Text)

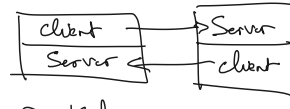
Sync Service:



Async call via Service



→ both are client and servers



→ More technical = Web Sockets

Polling → check every n - seconds / - minutes } NOT GOOD

Practices → good Practice → Industrial Solution

Cooking

Mix ingredients → Recipes → Industrial Products

To have a plate → follow Rules → Spend Time

IT

I develop the Way I feel/think it language → implementation

Recognized Recipes → DESIGN PATTERNS

- GoF
- Architecture ....
  - MVC (also MVP, MVVM ...)
  - ORM
  - AOP
  - IOC

95% of used Patterns

DESIGN PATTERNS → FRAMEWORK

→ an implementation of a Design Pattern(s)

→ Need To be learned

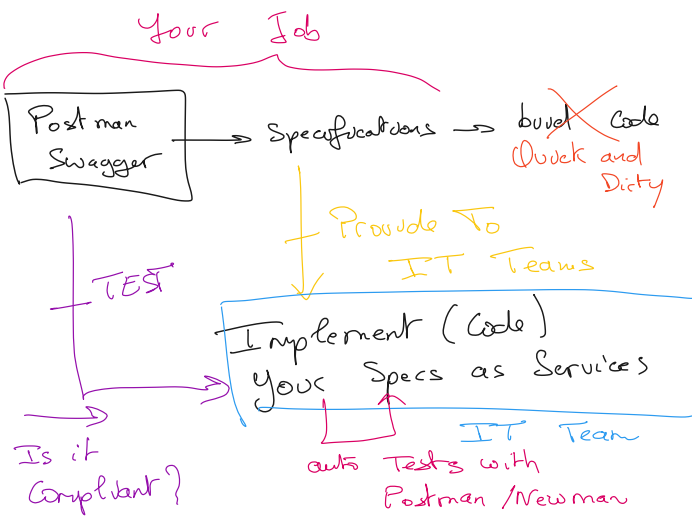
→ efficient

ex: .net (C#)

Postman / Swagger } Build | client Side | Server Side Skeleton

Server Side Skeleton: javascript, C#, Java, Python, Php

No Framework

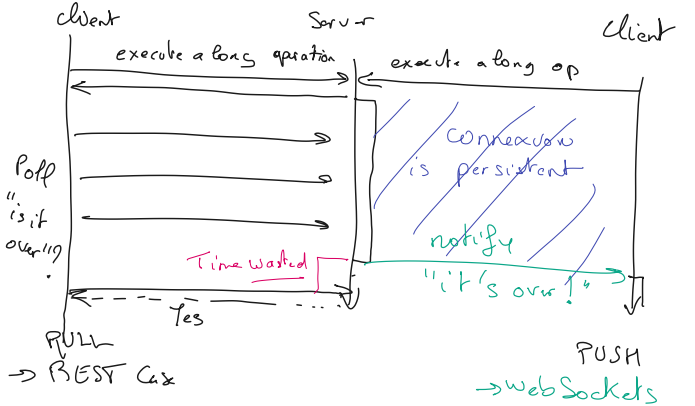


- Versionning
- Git

Pull → POP
Push → JMAP
} has ever existed

Rest API client → Server is PULL
↳ pattern, good Practise, point of view
is Standard / Common use Case

WebSocket (Not Rest API) Provides PUSH
↳ Tech point of view System



Versions

a Version ⟶ a Code

Semantic Versioning

<Major Version>.<Minor Version>[.revision.build number]

0.9 → an alpha / beta prerelease of an app

1.0 → first "acceptable" version

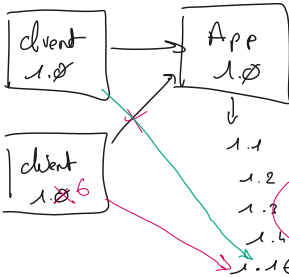
1.1 → increase of functionalities from version 1.1

Major

Minor → New functions, Respect ascendancy compliance

Revision → - correction of bug
- optimization
- no new functionalities

1.0.0 → 1.0.1



No breakdown with old clients for the same major

SAME MAJOR = Need to guarantee Compatibility!

PREFERED as much as possible

ex.

① method (int , int)
method (float , float)

② m (int , int) → m (int, int, int = 0)

v1 Person
id
Name

Create (Person)

v ?

v 2.0

incompatibility

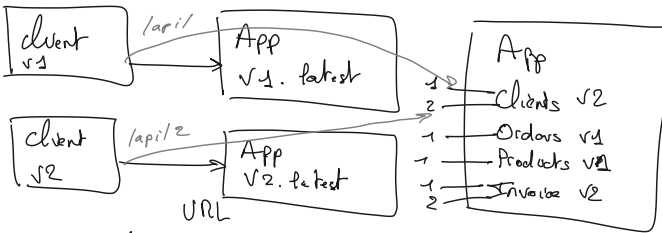
Person 2
id
first name
last name

v 1.1 Create (Person)
Create (Person 2)

µSvc

Version 1 allowed Major at a time

allowed to own v1 and v2 at the same time

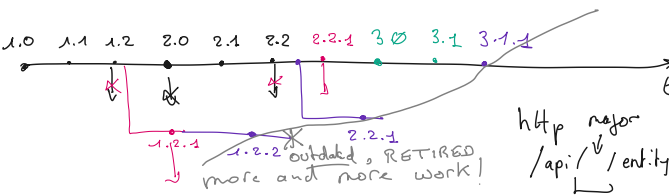


/api/customers (v1) → Legacy

/api/v2/customers (v2 endpoint)

More Majors = more work

Branches



Yearly Versioning

yy.nn

Ubuntu 14 16 18 20 — 18.04, 20.10

Oracle 9, 10, 11, 18

Postgres 9, 10, 18, 19, ...

month

ex /api/orders

/api/2/orders

/api/3/orders

Semantic ⟷ Yearly ⟷ Codename

(Internally at least)

Android
- Lollipop
- Jellybean

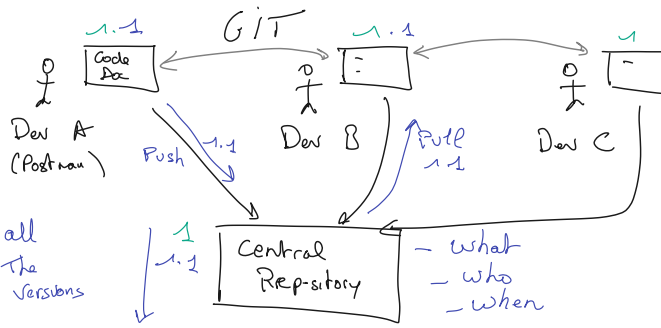
Ubuntu
- xenial trusty

OpenStack

/api/orders

/api/xenial/orders

/api/2011-11/orders



Past: ~~CVS, SubVersion, Microsoft TFS~~ => GIT

μSvc Nuvra Services

μSvc → Architecture Pattern
↳ 2010's

- extension
- Based on REST

SOA
EIA
Service Bus

} Legacy Pattern now
↳ 2000's

μSvc Comes when

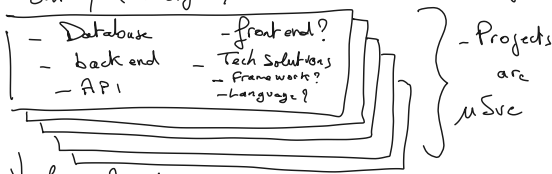
a Solution (a whole Implementation)

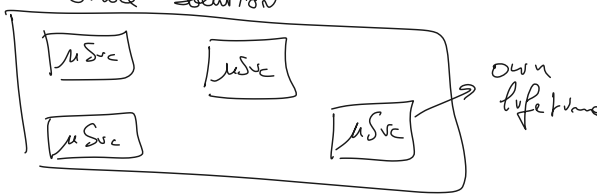
(ex: Amazon, WhatsApp, AirBnb)

Is Too Complicated To be seen as

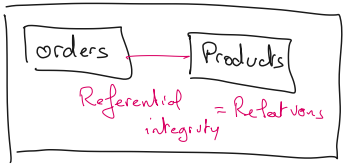
a Project / a bunch of Projects

- A Complex Solution has to be exploded in many simple solutions
- Each Entity (RestfulAPI oriented) is a Project





A Solution (eg CRM) = anti-pattern



Database

→ Pattern in 80, 90, 2000's

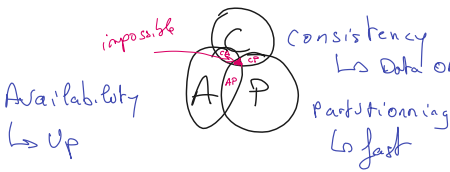
Referential Integrity was the pattern

→ Consistency

2000 → internet Wide

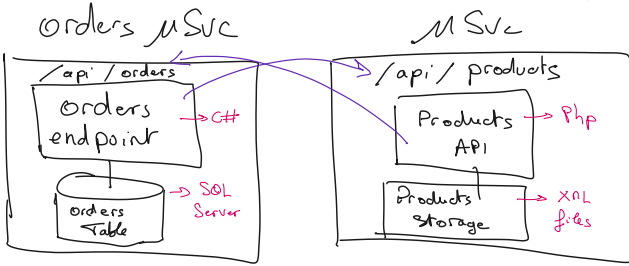
↳ Load is much Higher

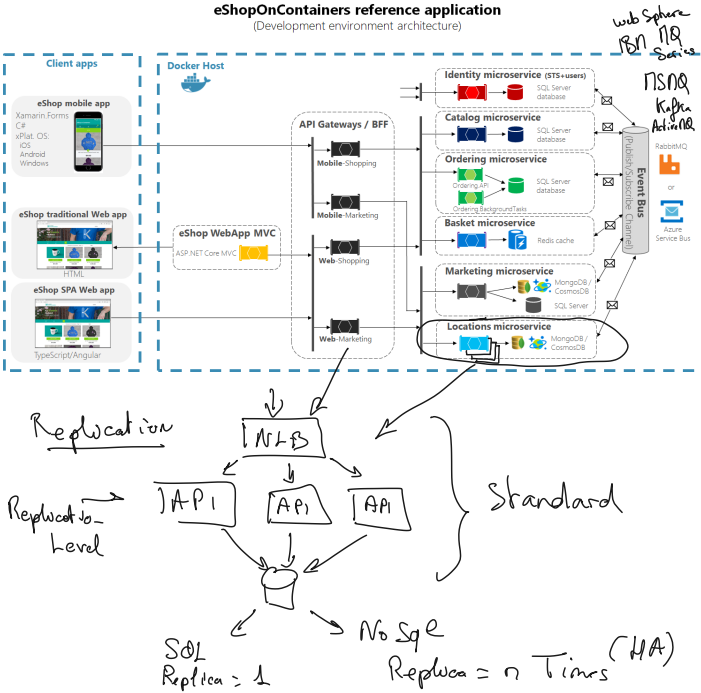
→ High Availability



CAP → impossible to have C, A, P Same Time

μSvc → There is no Central Repo or dependancy

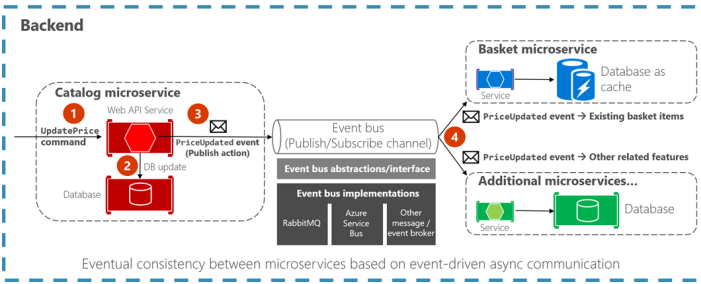



eShopOnContainers reference application
(Development environment architecture)
Web Sphere
IBM MQ
Series
Client apps
Docker Host
eShop mobile app
Xamarin.Forms
C#
xPlat. OS:
iOS
Android
Windows
eShop traditional Web app
HTML
eShop SPA Web app
TypeScript/Angular
eShop WebApp MVC
ASP.NET Core MVC
API Gateways / BFF
Mobile-Shopping
Mobile-Marketing
Web-Shopping
Web-Marketing
Identity microservice (STS+users)
SQL Server database
Catalog microservice
SQL Server database
Ordering microservice
Ordering API
SQL Server database
Ordering BackgroundTasks
Basket microservice
Redis cache
Marketing microservice
MongoDB / CosmosDB
SQL Server
Locations microservice
MongoDB / CosmosDB
Event Bus
(Publish/Subscribe Channel)
NSQ
Kafka
ActiveMQ
RabbitMQ
or
Azure Service Bus

Replcation
NLB
API
API
API
Replcation Level
Standard
SQL
Replica = 1
No Sql
Replica = n Times
(HA)

Consistency Management in MSsc

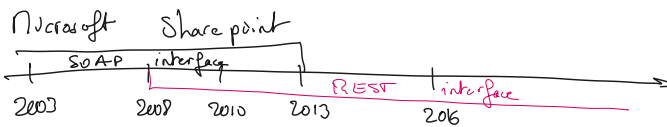
Implementing asynchronous event-driven communication with an event bus



API Recommandations

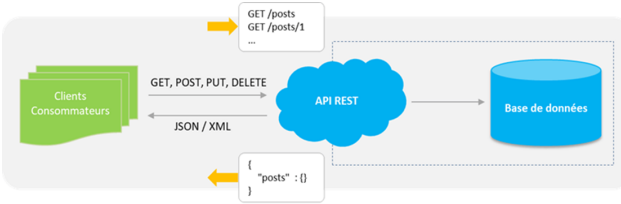
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Accepted in 2000



REST has to be:

- UNIFORM interfaces
- Stateless
- Codable
- Cacheable (client Server side) - Proxy
- Client / Server
- Layers



option

/api/ products [/ id]

/api/products/1

/{id}→Param

Rules :

1) URI Ressource identification ex BAD /api/getproducts?id=1

- Create
  Read → 1 particular res
- Update
- Delete
  Query → many -

2) HTTP Verb is the operation id

GET
PUT
POST
DELETE
PATCH
OPTION
- ...

3) HTTP Response is the ressource representation
   → JSON most of the time

4) Link maturité relations between ressources —
   → Seldonly follown

/api → { "products" : "http://.../api/products", "orders" : "etc..." }

/api/products (Q) → including pagination (pages)

```
{ "1": "http://.../api/products/1",
  "chang": "...../2",
  "syrup": "...../3", ...}
```

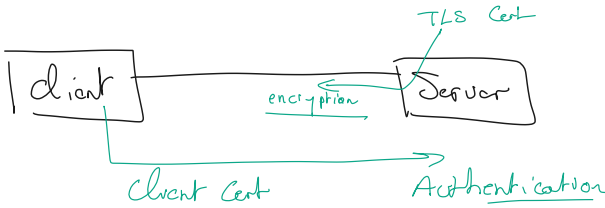
/api/product/1: (R)

Product → Categorie

```
{ "id": 1,
  "name": "chai"
  "category": { "id": "1",
                "name": "Beverage",
                ...},
  "prev": "http://... null",
  "next": "http://.../api/products/2"
  "Products": "http://.../api/products" }
```

5) Use Tokens for auth

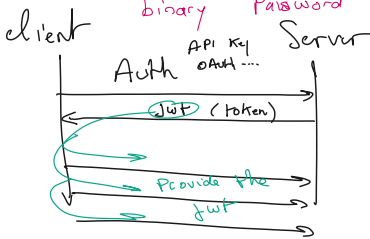
- Web Browsing = Auth = ~~Basic~~ ~~Digest~~ ~~NTLM~~ ~~Forms~~ — Please not with APIs
- API over TLS (encrypting)
  - → OAuth2 → SAMLv → OpenId
  - → JWT (JSON Web Token) → etc.... → API Keys etc...



- Password
- Key → machine
- Secret
- Auth certificate

→ human decoded
build – Key
atf "the same"
→ Password built by a machine (text – copy/paste)

binary "Password" → in a file



IDEMPOTENCY

→ idempotent is when 4 Time the same operation is same stak in the end

GET → idempotent, safe
HEAD → " "
POST → Create, ~~idemp.~~, ~~safe~~
→ without id = create each time
with id = create or update (Merge)

PUT → UPDATE by id (all props)

idemp , ~~safe~~

PATCH → UPDATE part of by id

idemp , ~~safe~~

DELETE → idemp , ~~safe~~

OPTIONS → discover the VERBS

Http Response Codes

1xx → Informations (no content)

2xx → OK

3xx → Actions client side

4xx → Client side errors

5xx → Server side errors

- Exceptions
- Too many requests
- etc...

---

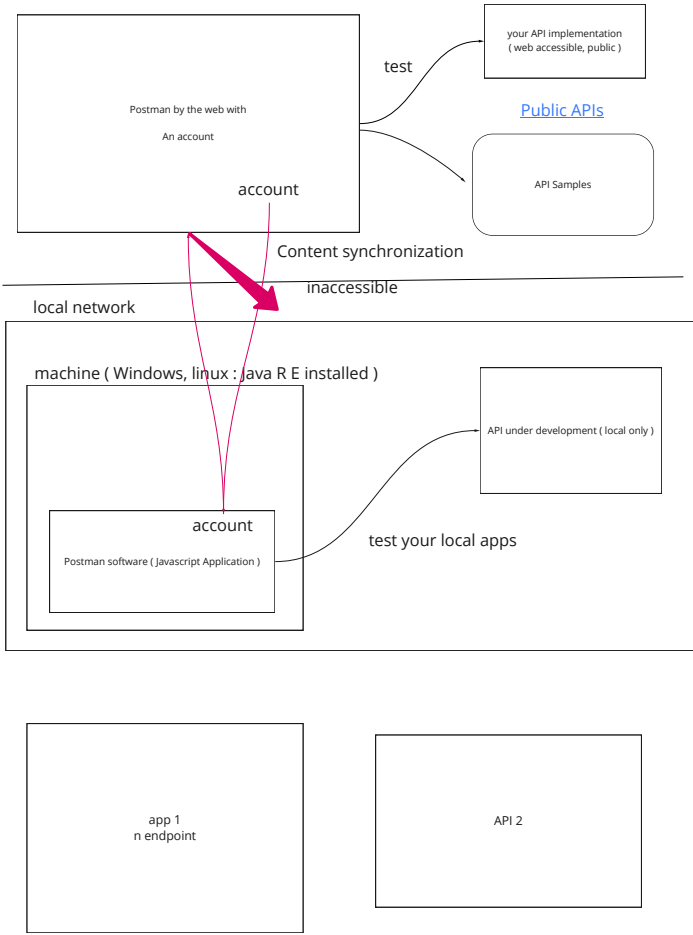
200 → OK

404 → Not Found

500 → Server Error

https://developer.mozilla.org/fr/docs/Web/HTTP/Status

- GET: 200 OK
- POST: 201 Created
- PUT: 200 OK
- PATCH: 200 OK
- DELETE: 204 No Content


Postman by the web with
An account
test
your API implementation
(web accessible, public)
Public APIs
API Samples
account
Content synchronization
inaccessible
local network
machine (Windows, linux : Java R E installed)
API under development (local only)
account
Postman software (Javascript Application)
test your local apps
app 1
n endpoint
API 2

Endpoint => Entity Class
Verb => Operation Method
GET POST PUT custom EXECUTE, LAUNCH, STOP, ...
Body : Parameters, or : Headers, or : Query parameters.
Query parameters : Call customizations (paginations, sorting, filters, ...)
simple parameters (string, int, date) : customer headers (maybe)
Advised : Whole parameters as a JSon payload in the Body

managing mails : close items, operations
map to Rest Verbs
cds :
single endpoint; many operations

/api/mails : GET, POST, DELETE
/api/cds : http verbs : CREATE, EXECUTE , ...
/api/cds GET, DELETE, ...

Complex operations : GraphQL

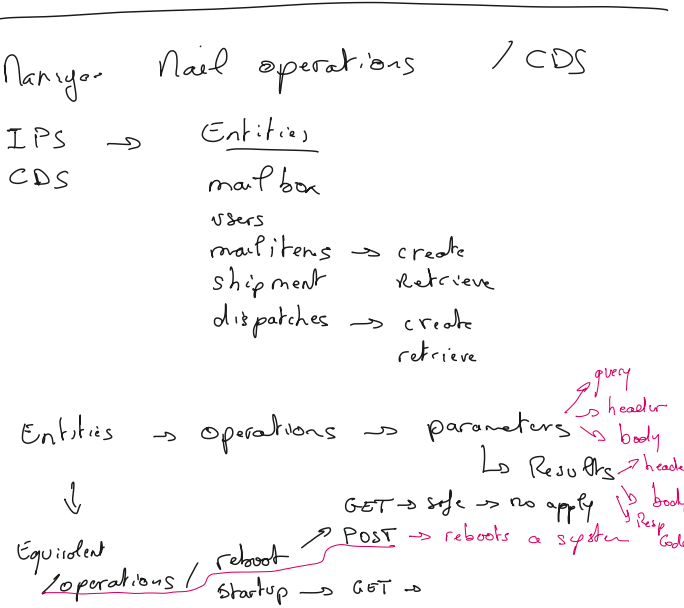
https://swapi.dev/api/
Very good to start

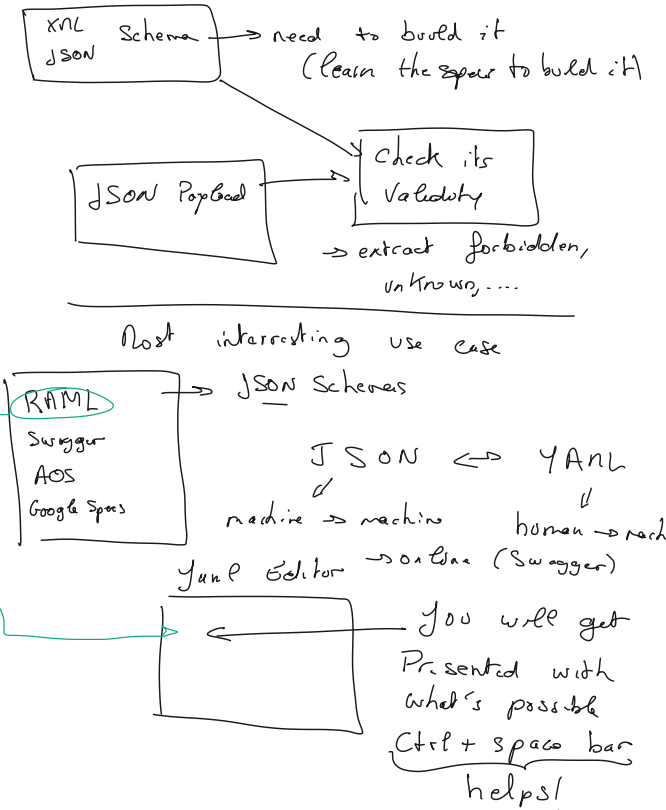
Markdown Cheat Sheet :
https://www.markdownguide.org/cheat-sheet/

The book Workspace :
https://github.com/PacktPublishing/API-Testing-and-Development-with-Postman

Postman public Workspaces :
https://www.postman.com/postman/workspace/published-postman-templates/overview

---





https://editor.swagger.io/
Is better than Postman to write down Specifications

Specifications of AOSv3 :
https://swagger.io/specification/

Tests use the Chai.JS Javascript Library for Assertions :
https://www.chaijs.com/api/