Architecture Logicielle

ARNAUD

LEMETTRE

Programme de la journée

09h – 10h30 : Evolution des architectures

10h30 – 10h45 : PAUSE

10h45 – 12H00 : Concept de développement

12h00 – 14h00 : PAUSE

14h00 – 15h30 : API

15h30 - 15h45 : PAUSE

15h45 – 17h00 : API

Pourquoi architecturer ?

L’architecture

Couplage

Maintenance

Testabilité !

- Limiter le nombre de lignes de code
- Simplification du debug
- Moins de bugs en prod (50% du cout supplémentaire par rapport à un bug en dev)
- Travail collaboratif plus simple

Les évolutions des architectures

L'application standalone

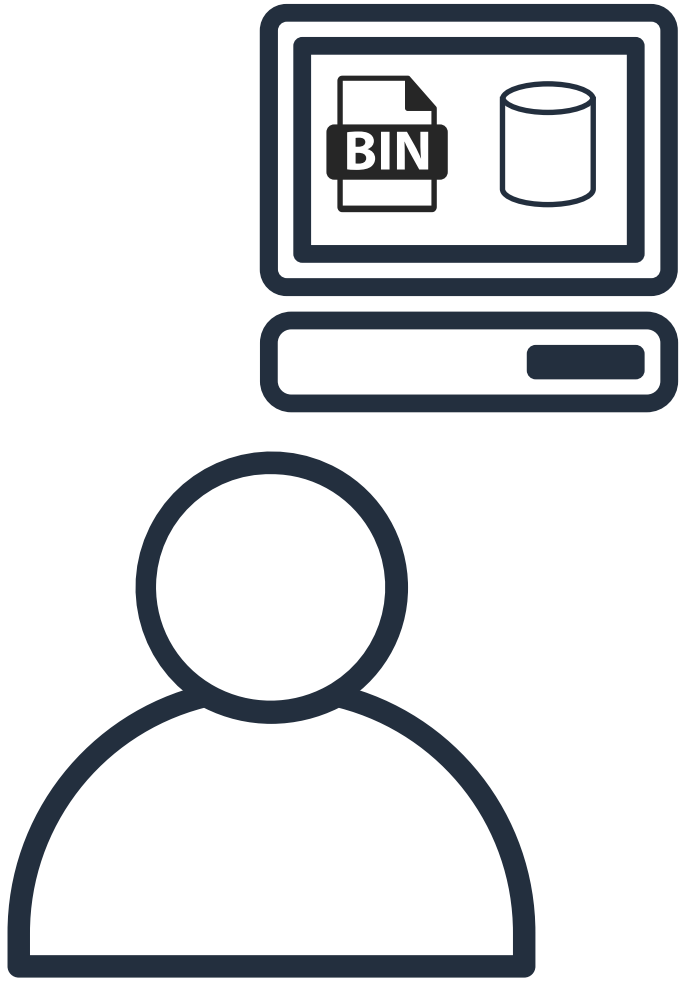
Juste du calcul, pas de stockage

C'est facile avec 1 user sur son poste !

Problème !

Pas de sauvegarde des données

Besoin de stocker des informations

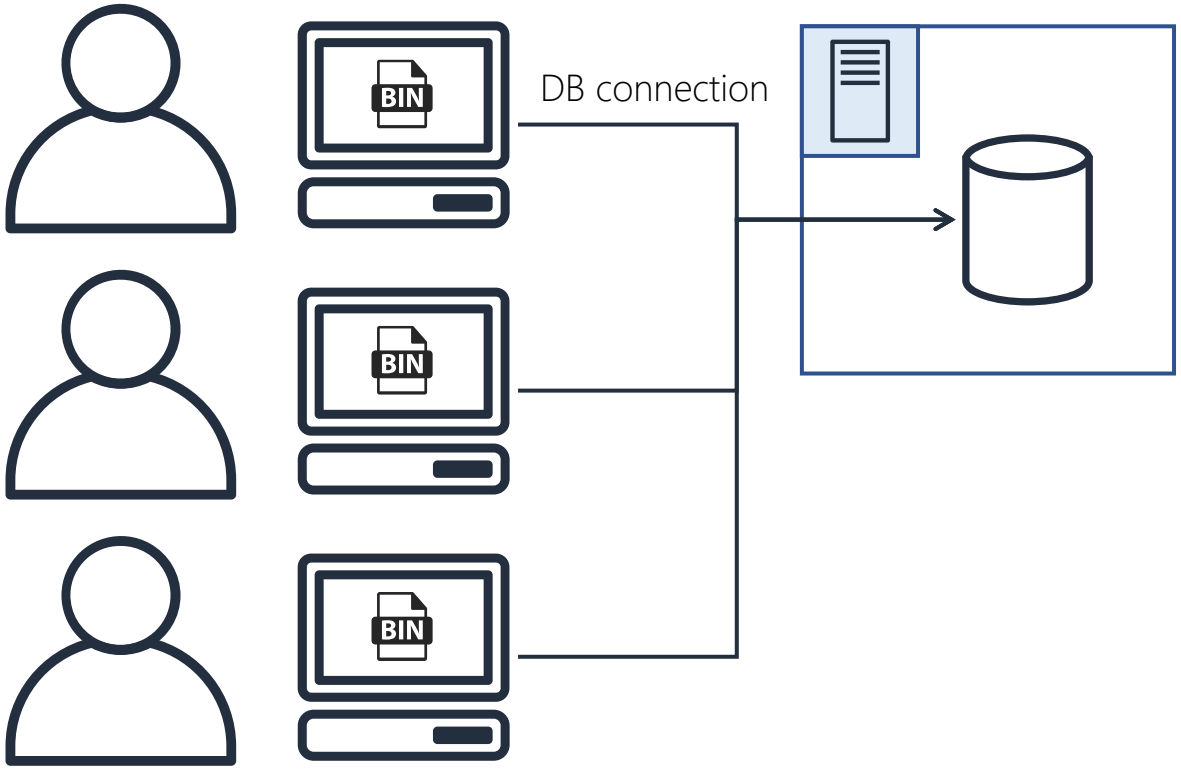


BDD pour stocker les données

Problème !

Données pas mutualisées entre les users
Pas de stats, pas de sauvegarde globale
Données liées à poste

Mutualisation des données

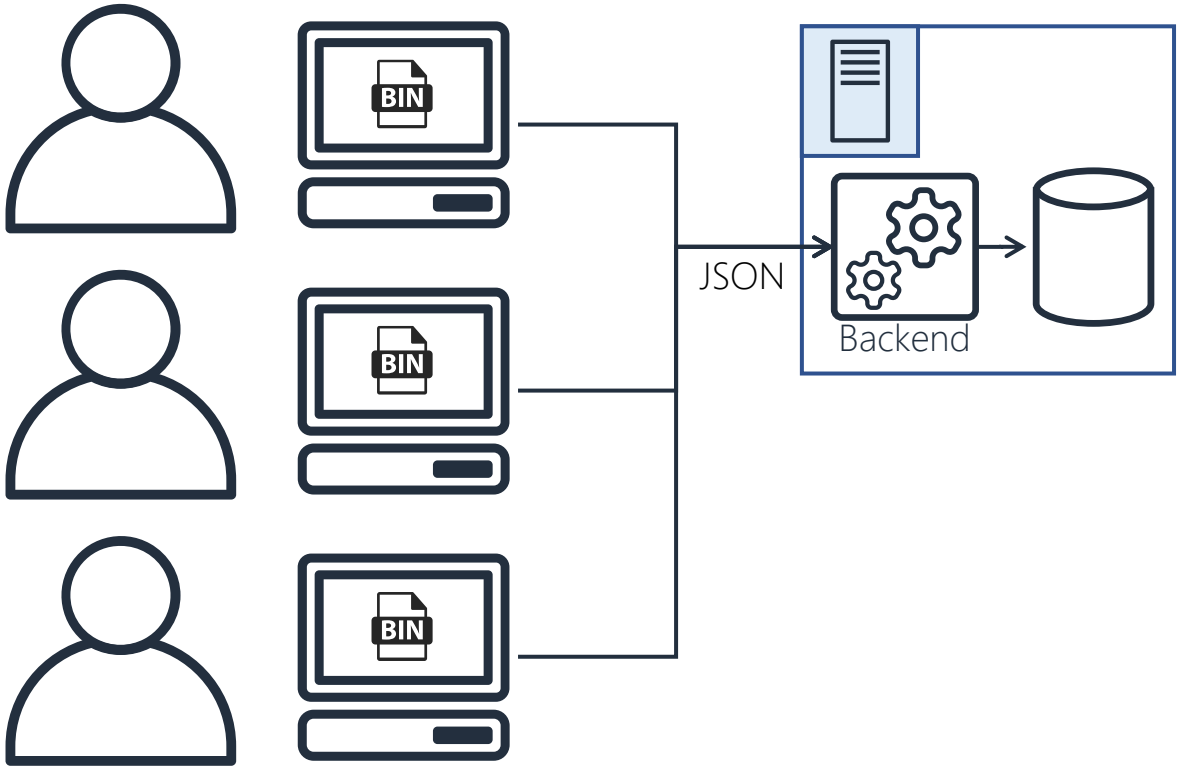


Serveur de données ajouté !

Problème !

DB exposée sans contrôle des actions

Abstraction de la data access

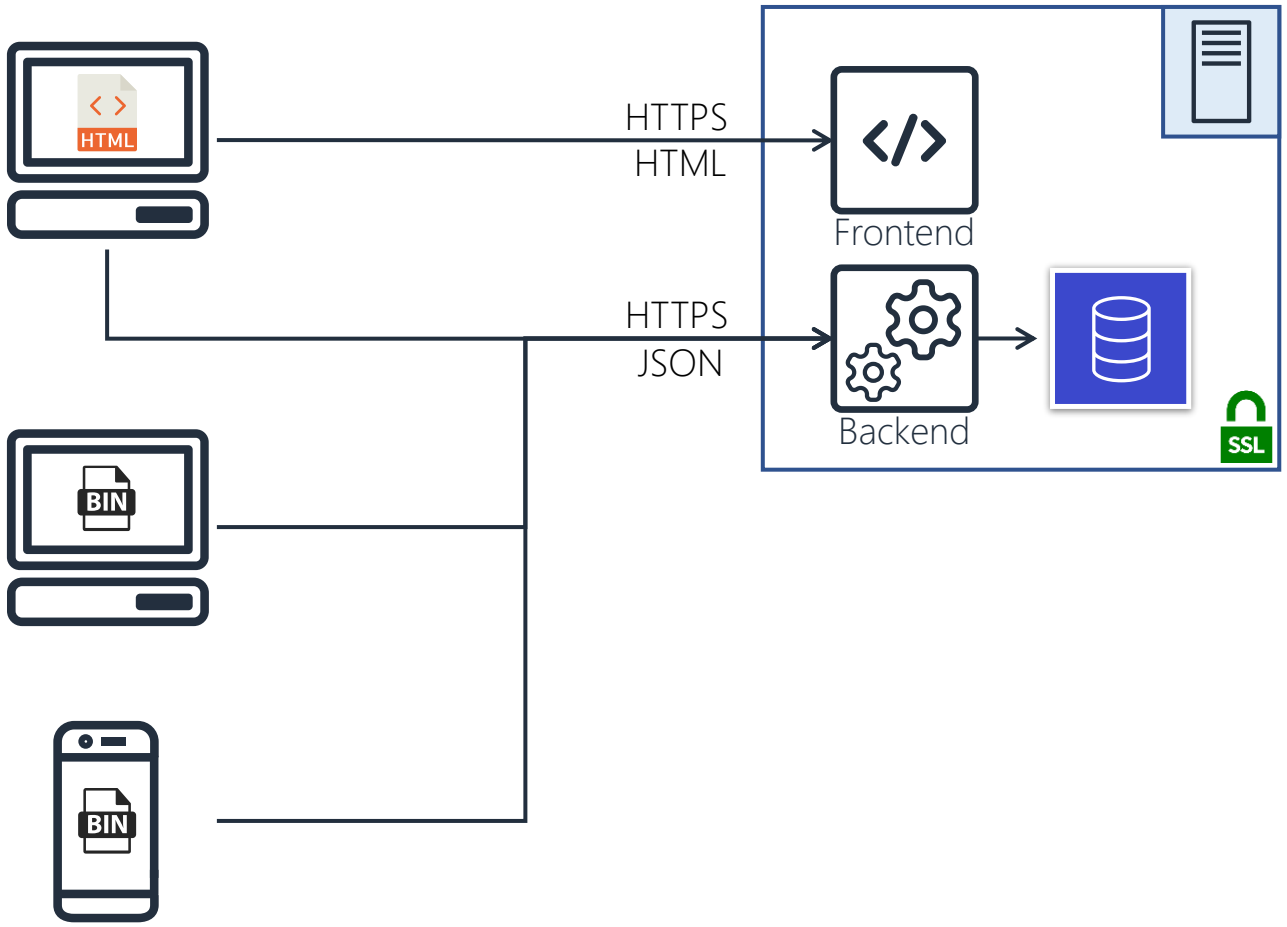


Notion de client - serveur
« Separation of concerns »
Contrôle de l'accès au données

Problème !

Pas de possibilité de corriger sans redéployer

Plusieurs clients

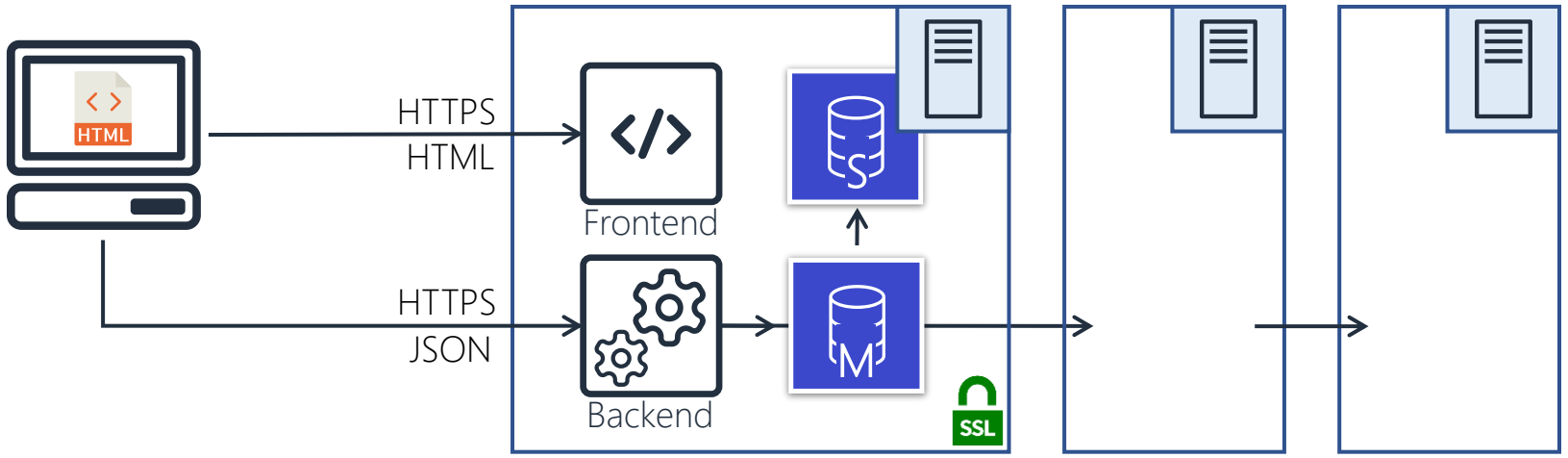


Architecture WEB classique

Problème !

Pas de backup des données

Redondance de la donnée

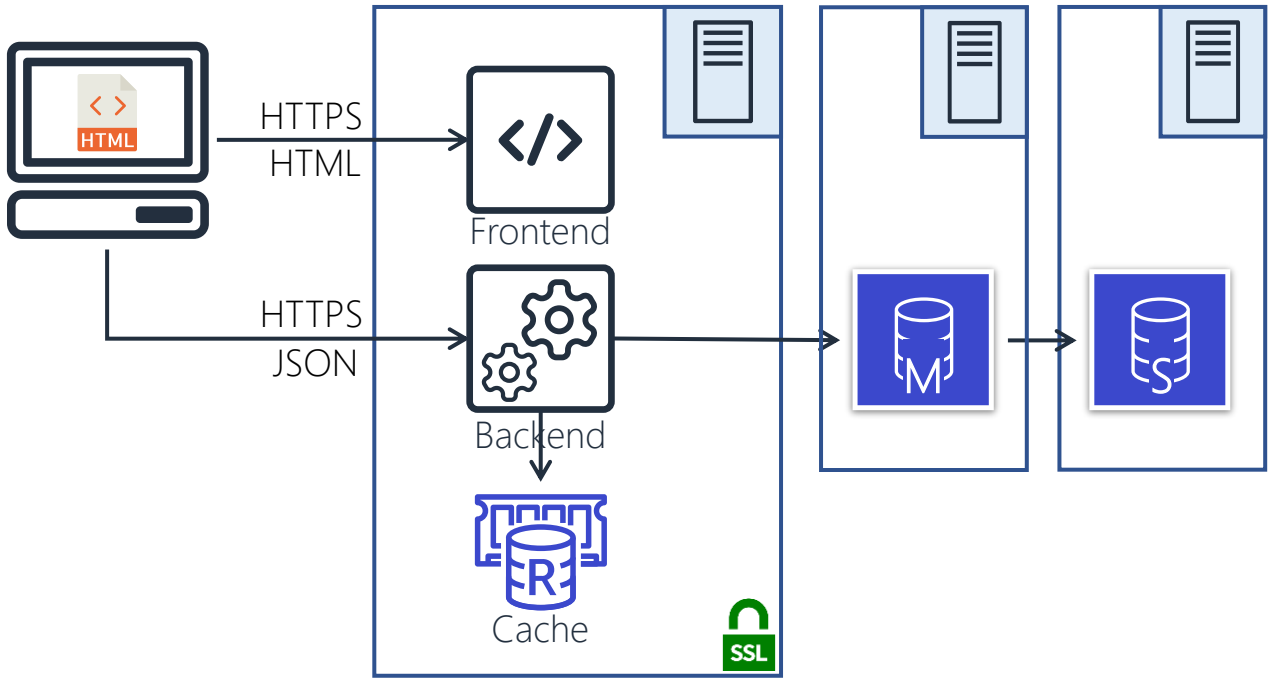


Réplication dans la DB esclave

Problème !

Comment améliorer les perfs ?

Accélération des requêtes



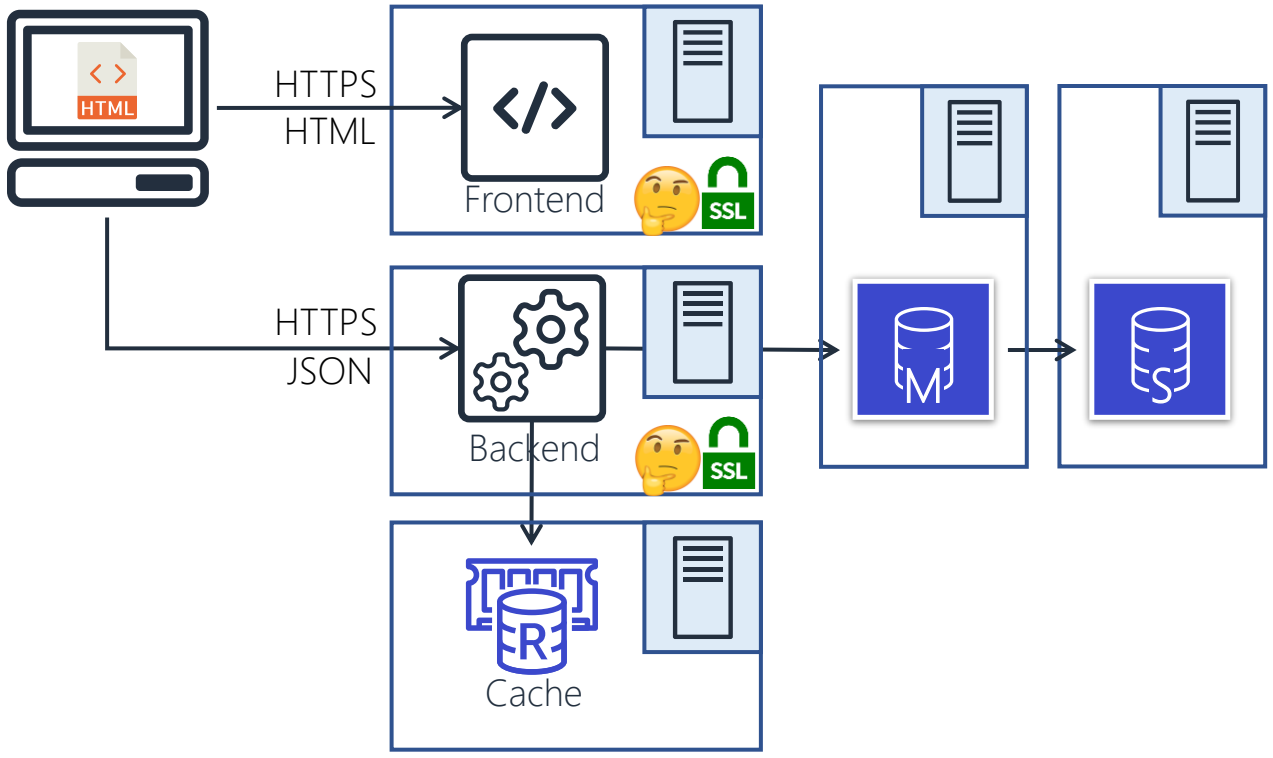
Mise en cache des réponses

Economie de BDD & CPU

Problème !

Montée en puissance pas flexible

Accélération des requêtes



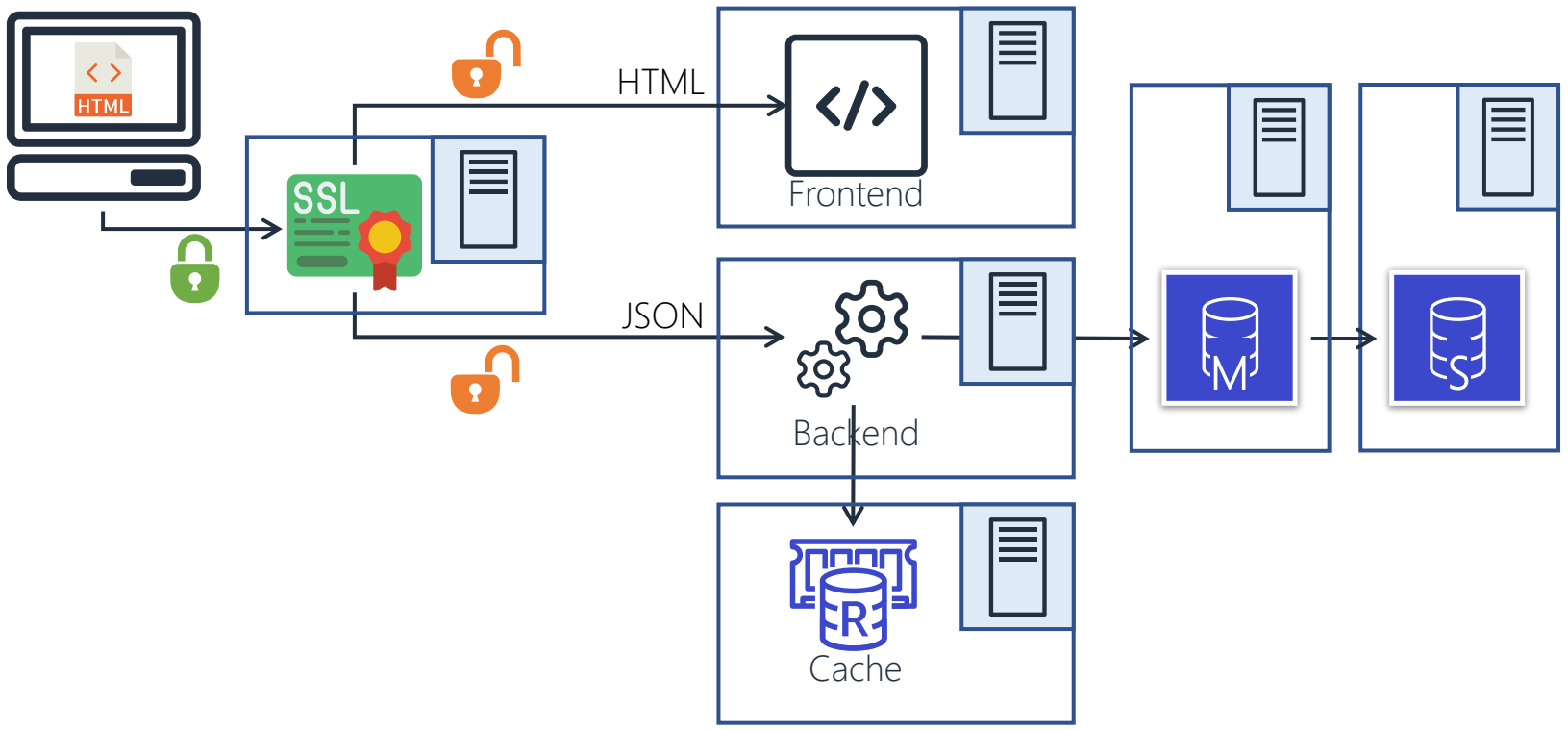
1 serveur par composant

Meilleure flexibilité

Problème !

En info, on n'aime pas faire 2x fois

Externalisation de la vérif SSL

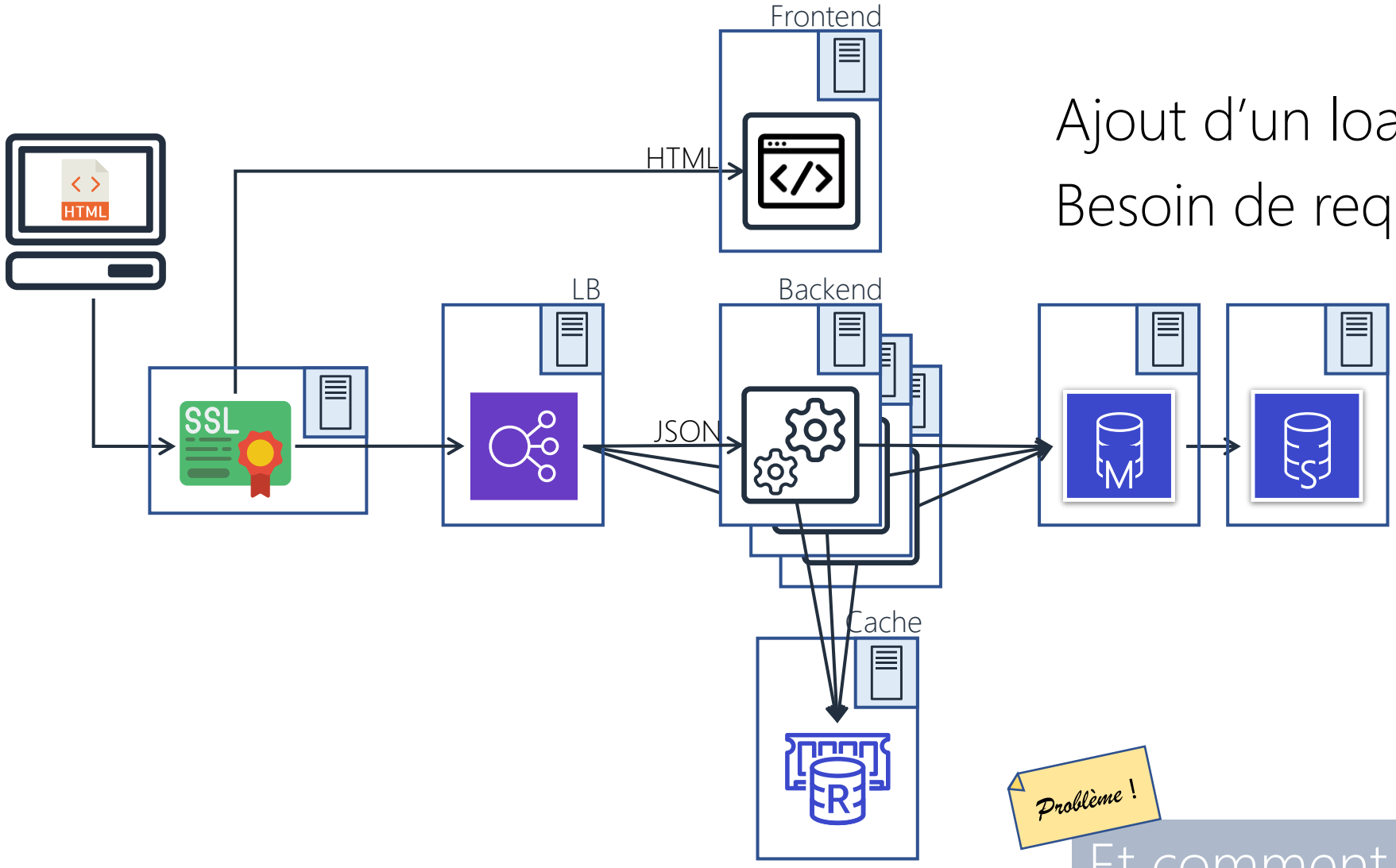


SSL Termination

Problème !

Et si le cache ne suffit plus ?

Répartition des taches



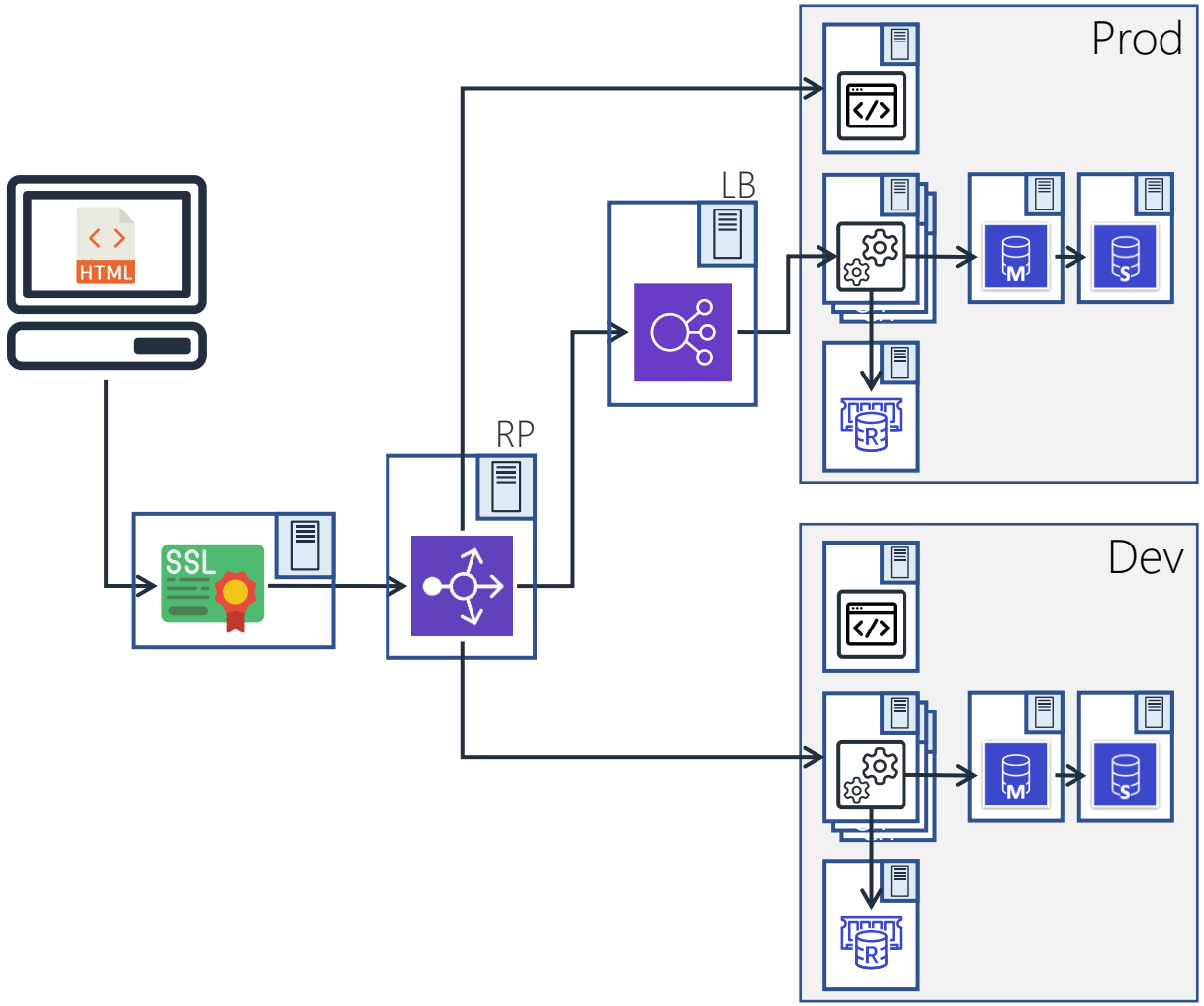
Ajout d'un load balancer

Besoin de requêtes stateless

Problème !

Et comment on passe en prod ?

Plusieurs environnements

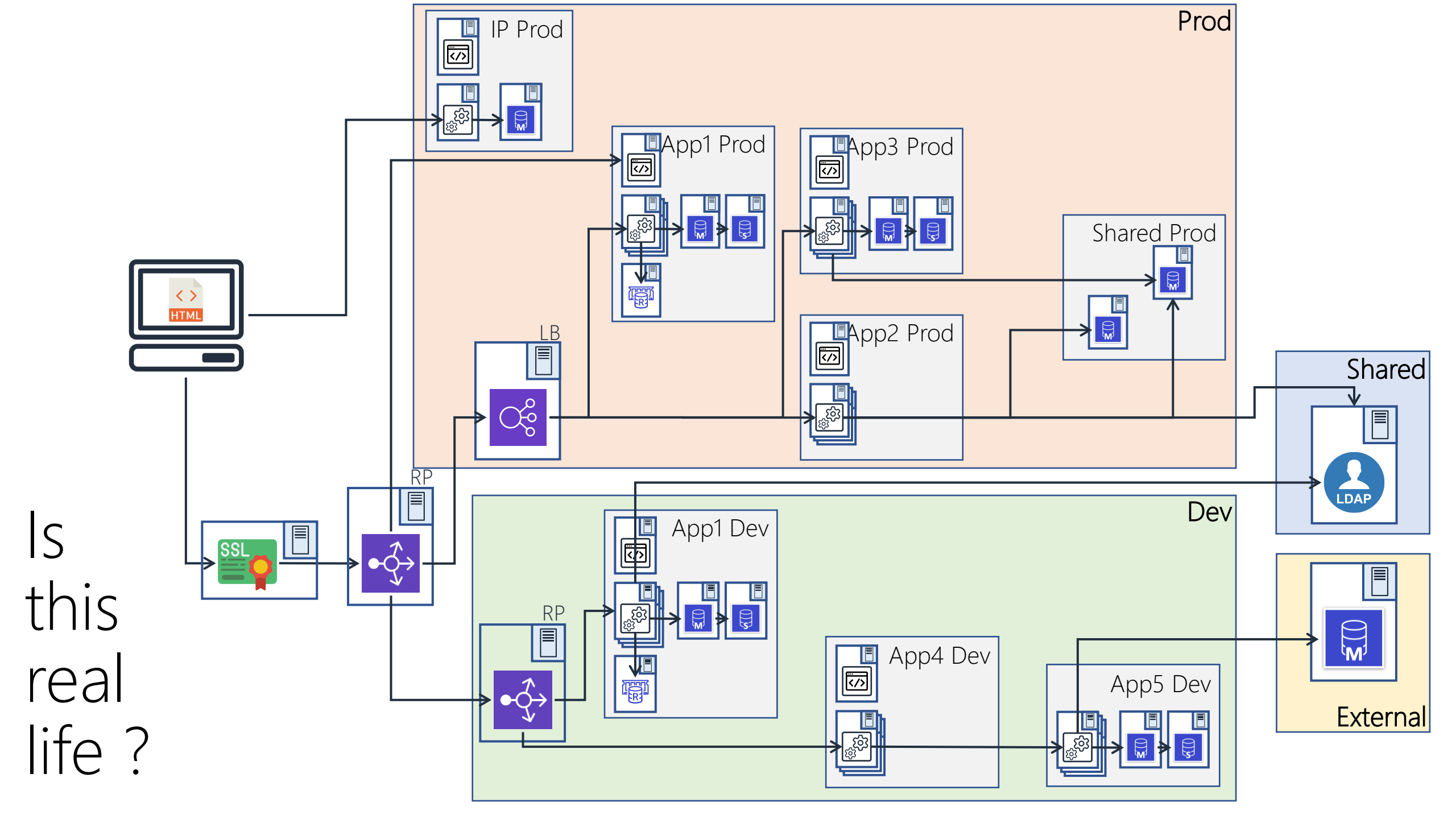


Le Reverse Proxy aiguille

Le RP modifie les IP sources

Problème !

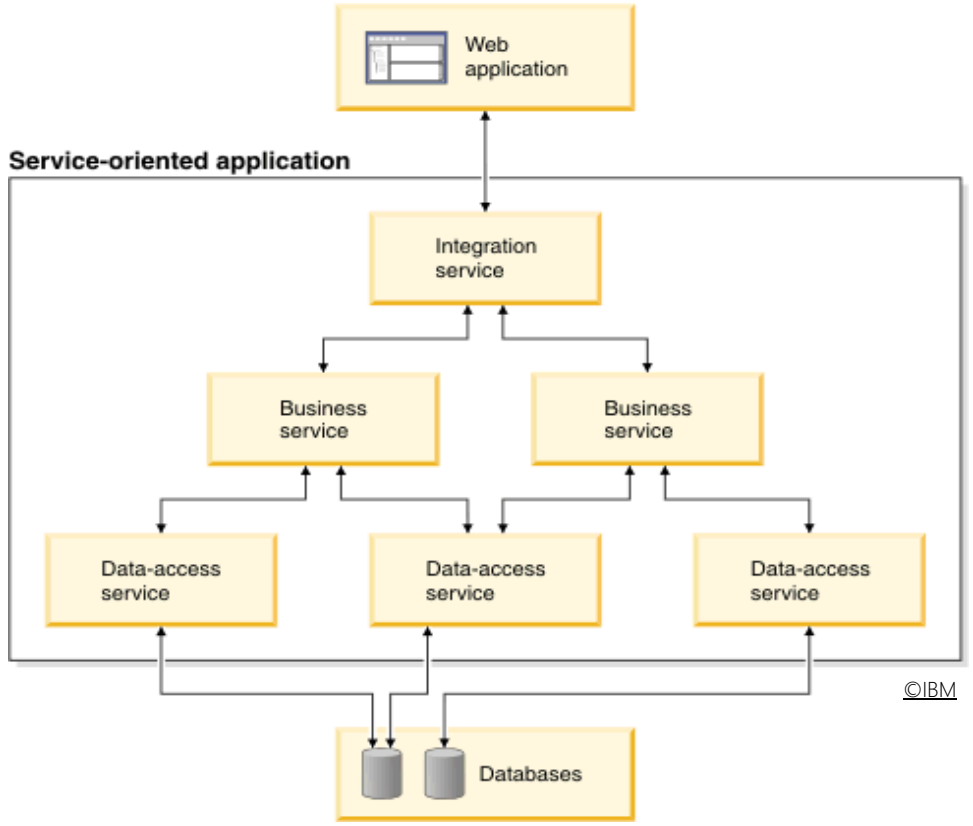
C'est pas drôle en vrai


Prod
IP Prod
App1 Prod
App3 Prod
Shared Prod
App2 Prod
LB
HTML
SSL
RP
Shared
LDAP
Dev
App1 Dev
RP
App4 Dev
App5 Dev
External
Is this real life ?

Le SOA - Service Oriented Architecture

Les services sont fournis par des composants applicatifs

Les services sont accessibles via un réseau



SOA – les avantages

Indépendances des services

Possibilité de varier les stacks

Compositions des services

Scalabilité

Les µ-services

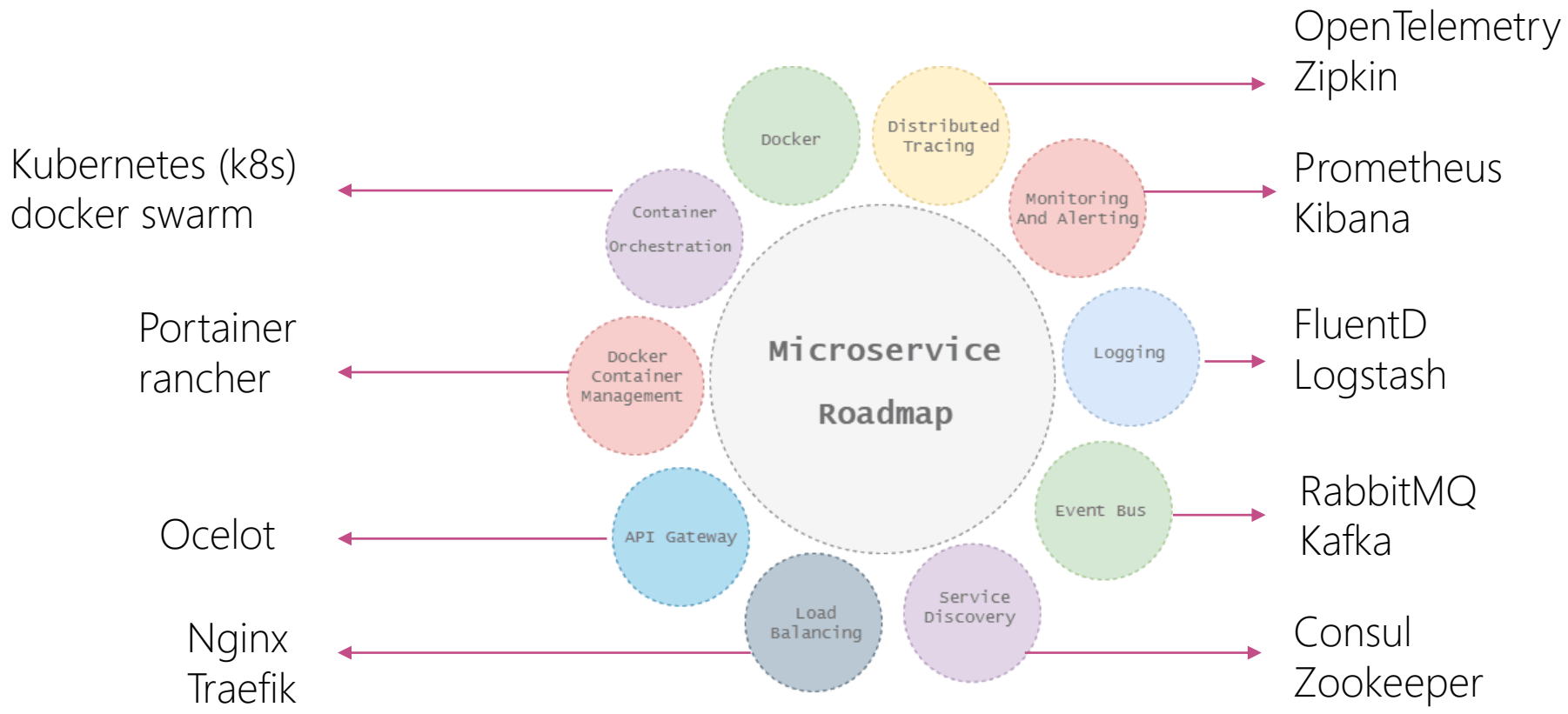
Evolution du SOA

Les services sont couplés de manière lâche

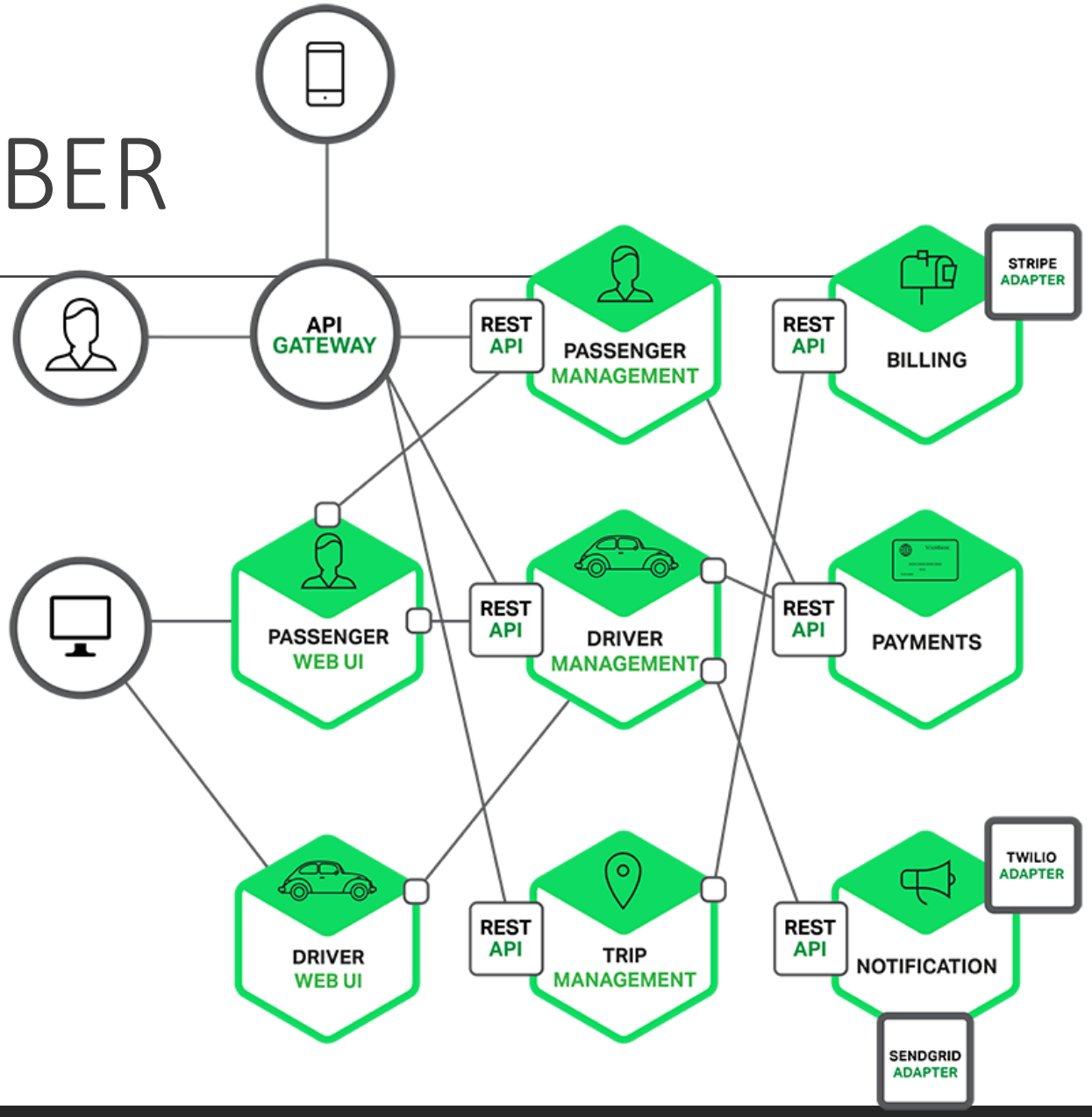
Communication inter-services over HTTP/2

Cette évolution ne doit pas être que technique mais organisationnelle

Les µ-services



µ-services : Chez UBER

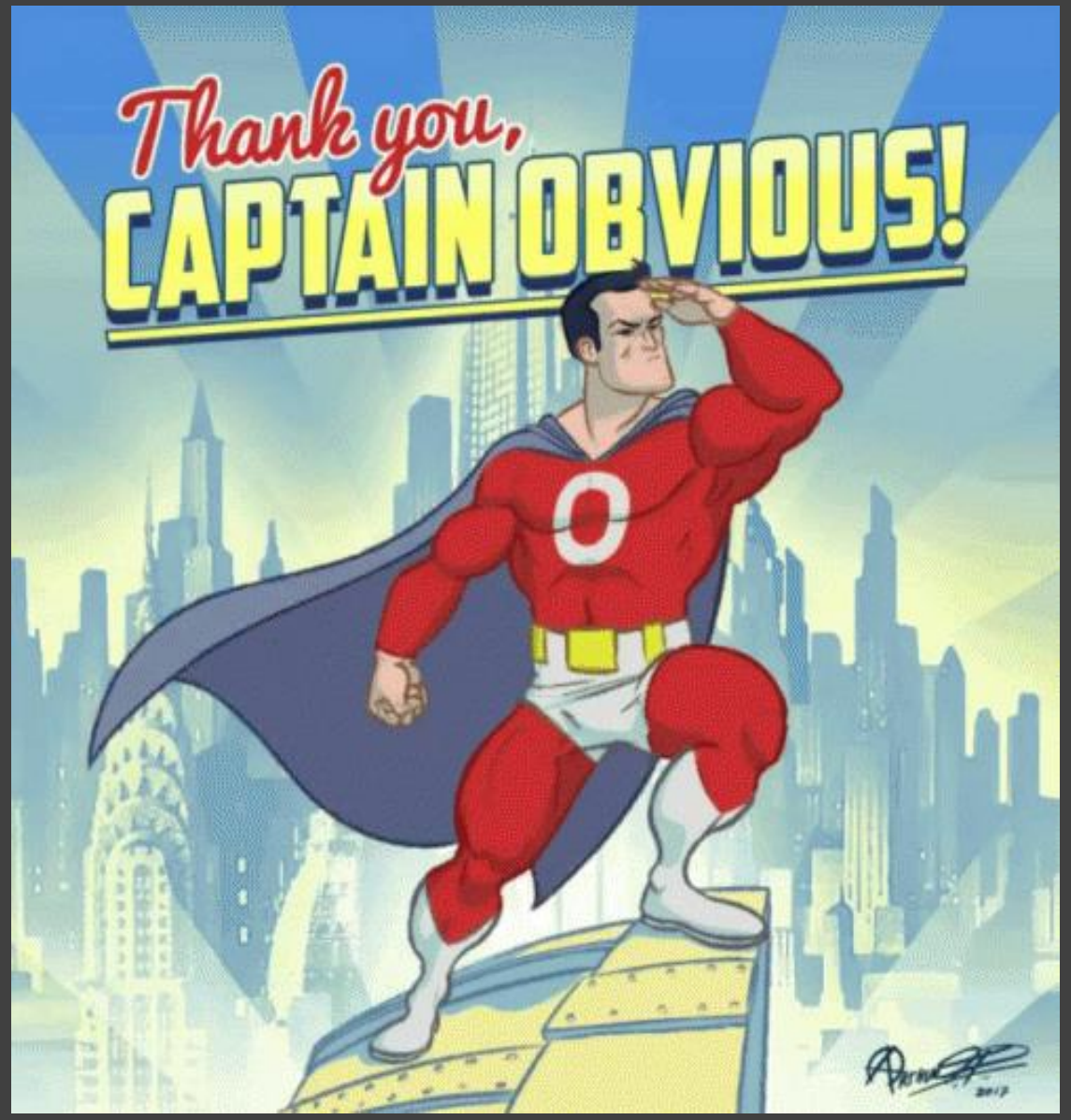


©sayantan khan

Le serverless

Le serverless, c'est Server Less

- Opérationnel géré par le fournisseur
- Propagation de fonction en fonction
- Logique évènementielle possible
- Scaling infini
- Paiement à la consommation



Le serverless : des inconvénients aussi

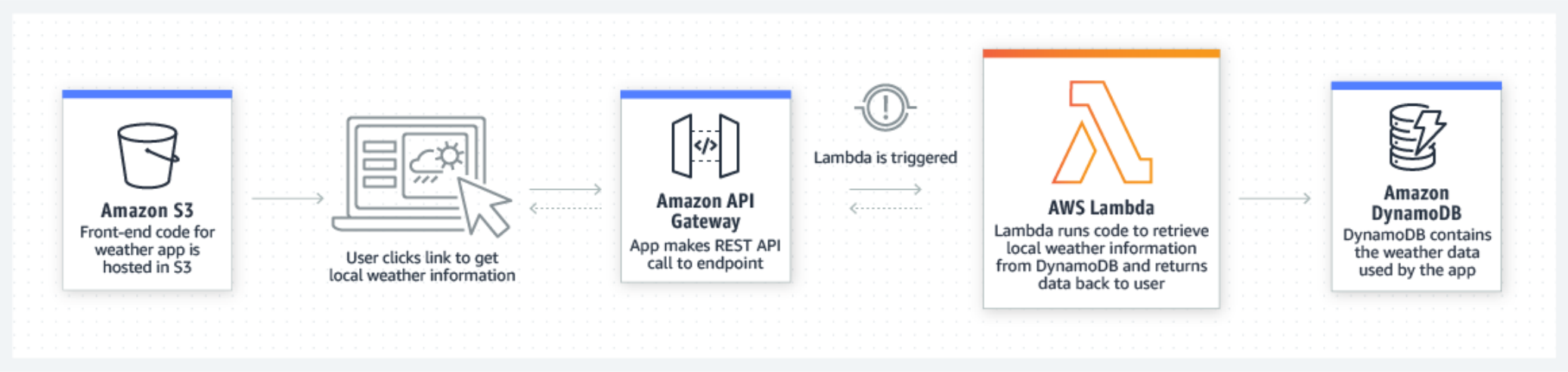
Peut être plus cher

Attention au warmup

Nouvelle manière d'architecturer

AWS Lambda: exemple zéro code

AWS Lambda : example site web



Concept de développement

Rappel

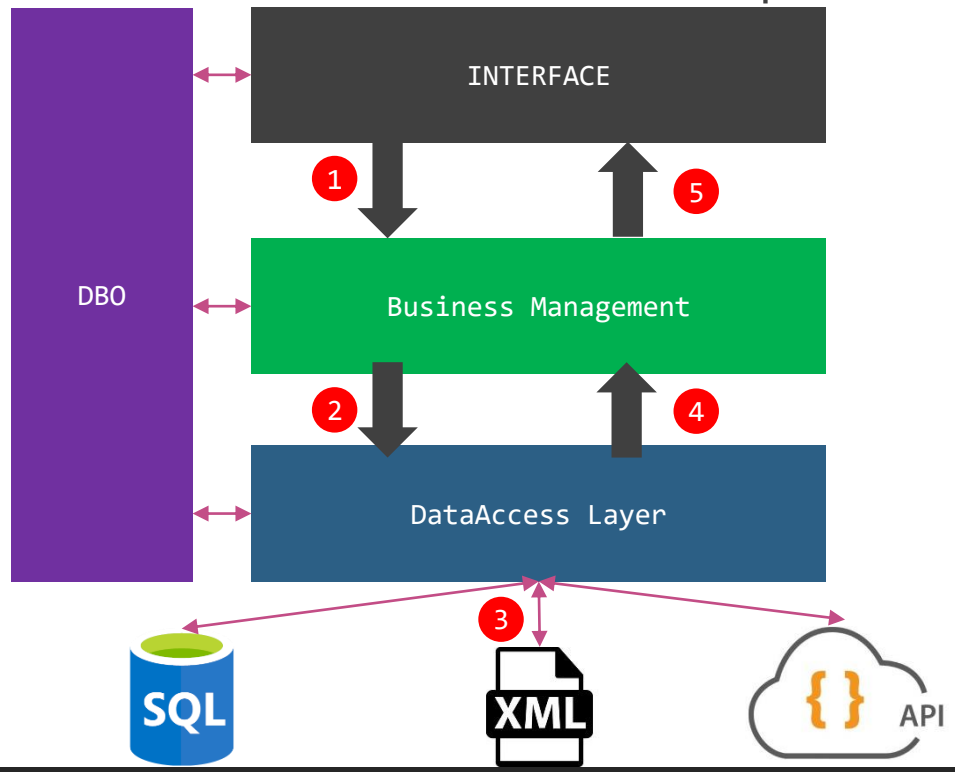
Un bon nommage peut éviter des bugs !

Il faut penser générique :

```
suspectRegexString.replace(/[.*+?^${}()|[\]\\]/g, '\\$&')

regexCleaner.escape(suspectRegexString)


count + " bird" + (count == 1 ? "" : "s")

pluralizer.match(count, “bird”)


"http://foo.com/bar?id="+id+"&term="+termValue

urlBuilder.build( "http://foo.com/bar", { id: id, term: termValue } )
```

Architecture N-TIER

Architecturer son code, c'est aussi savoir le découper. Pour nous aider nous pouvons nous appuyer sur une architecture en couche. C'est surtout une vue de l'esprit et une convention pour développer à plusieurs.



Où mettre son code ?

Le but de faire une architecture en couche est de rendre le couplage le plus souple possible.

La question à se poser :

« Si je dois changer mon implémentation dans le futur est ce que c'est la seule couche qui sera impactée ? »

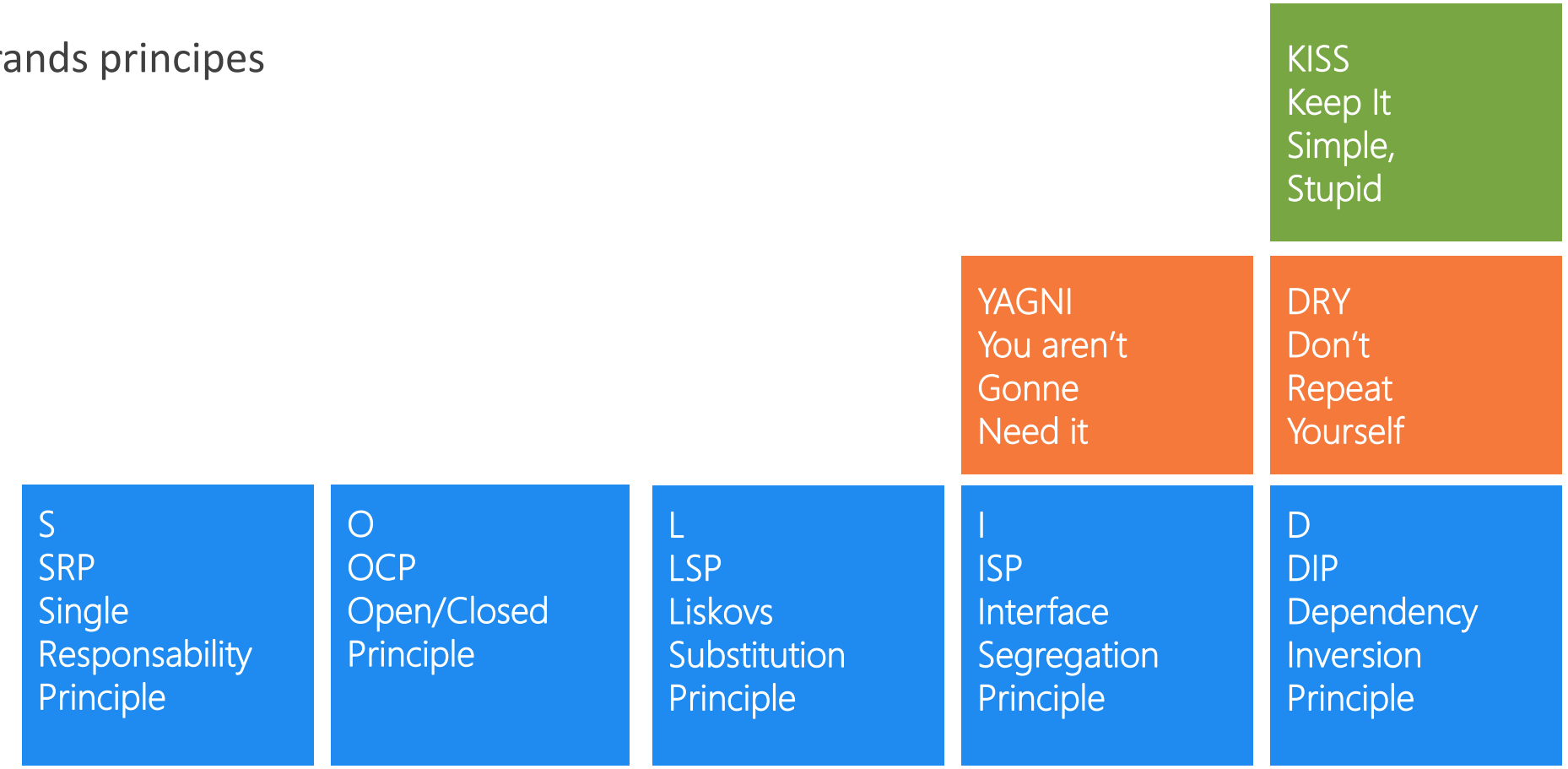
Si la réponse est non alors ce n'est pas à la bonne place

Exemples

- Je veux ajouter mon user uniquement si firstname est égal à Arnaud.
- Je veux ajouter le numéro de téléphone à mon user.
- Je veux stocker mon User en base de données où j'effectue cette sauvegarde ?

Principes de développement

Les grands principes



KISS

Keep It Simple Stupid

> *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*
> —Martin Fowler

> *Measuring programming progress by lines of code is like measuring aircraft building progress by weight.*
> *—Bill Gates*

Qu’est ce que cela veut dire ?

- Réduire le nombre de ligne de codes (C’est pas parce qu’on écrit beaucoup de ligne de codes qu’on est un bon développeur au contraire)
- Factoriser le code (trop de lignes de codes augmente la probabilité du nombre de bugs)
- Avoir un code simple, une bonne sémantique permet une meilleure maintenance
- Ne pas réinventer la roue (bien connaitre son framework permet d’utiliser la bonne méthode)

Exemple

```java
public static boolean tryParse(String value) {
    try {
        Integer.parseInt(value);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}
```


```java
boolean isNumeric = someString.chars().
                                    allMatch( Character::isDigit );
```

YAGNI

You aren't gonna need it

> *Always implement things when you actually need them, never when you just foresee that you need them.*
> *—Ron Jeffries, XP co-founder*

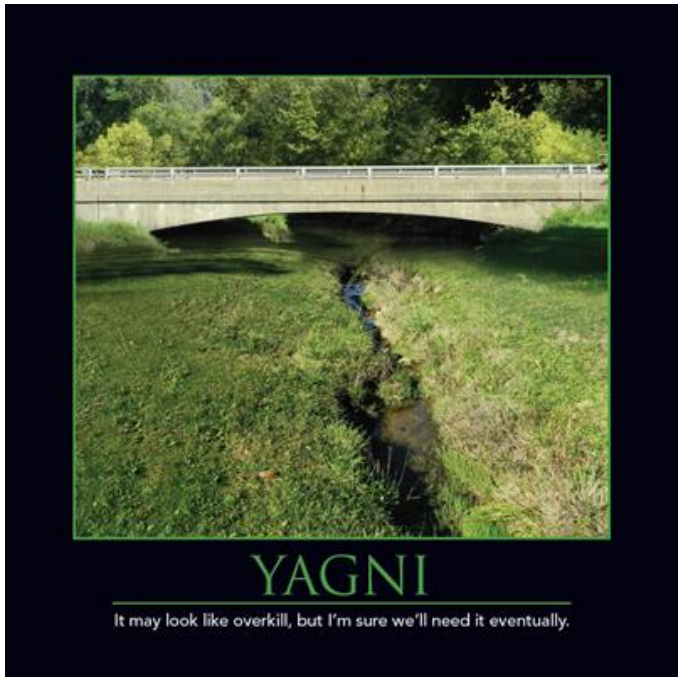
Qu’est ce que ca veut dire ?

Trop d’ingénierie :

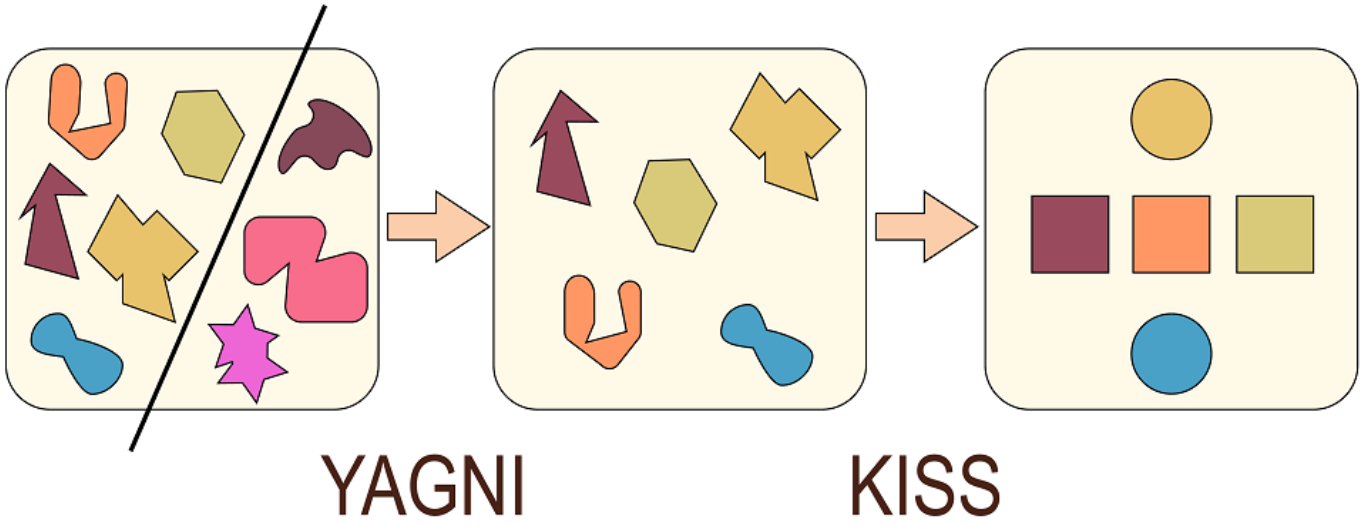
- Temps de développement, des tests
- Temps pour le documenter
- Ca Bloque pour ajouter de nouvelles fonctionnalités
- Complexifie et augmente la taille du soft
- Fonctionnalité inconnue pour le client

Exemple


YOU AIN'T GONNA NEED IT
Don't waste resources on what you *might* need.


YAGNI
It may look like overkill, but I'm sure we'll need it eventually.

Avec les deux principes



DRY

Don’t Repeat Yourself

> *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*
> *—Andy Hunt & Dave Thomas*

Détecter qu’on n’est pas DRY !

- Duplication de code (copié / collé)
- Duplication de la logique à plusieurs endroits
- Chaine / nombre magique
- Répétition de Branche if / else
- Conditionnel plutôt que polymorphisme

Exemple

```
class Bird
{
//…
    public double GetSpeed(Integer type) throws Exception
    {
        switch (type)
        {
            case 1:
                return GetBaseSpeed();
            case 2:
                return GetBaseSpeed() - GetLoadFactor() * numberOfCoconuts;
            case 3:
                return isNailed ? 0 : GetBaseSpeed(voltage);
            default:
                throw new Exception("Should be unreachable");
        }
    }
}
```

```
abstract class Bird
{
    private Integer milesPerhour = 20;
    public Integer GetBaseSpeed(){
        return milesPerhour;
    }

    public abstract double GetSpeed();
}

class European extends  Bird
{
    public  double GetSpeed()
    {
        return GetBaseSpeed();
    }
}
class African extends  Bird
{
    public  double GetSpeed()
    {
        return GetBaseSpeed() -
            GetLoadFactor() *
            numberOfCoconuts;
    }
}
class NorwegianBlue extends  Bird
{
    public  double GetSpeed()
    {
        return isNailed ? 0 : GetBaseSpeed(voltage);
    }
}
```

SOLID

Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion

> *there should never be more than one reason for a class to change*
> *—Robert C. Martin*

Qu’est ce que ca veut dire ?

Single responsibility
- Une classe ne doit avoir qu’une seule responsabilité (pas gérer une écriture dans un fichier de logs et faire une requête en BDD)

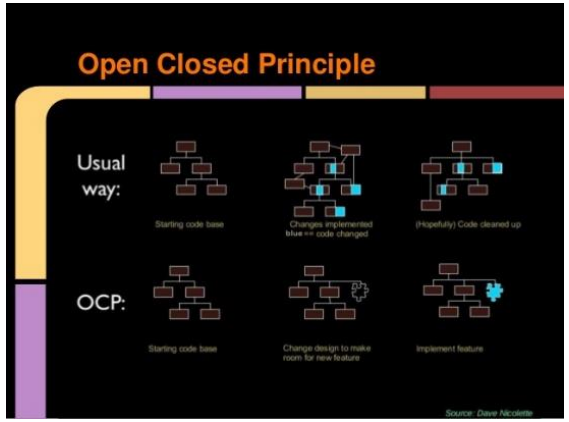
Open / closed
- Une classe doit pouvoir être étendue mais pas modifiée (l’ajout de fonctionnalités doit se faire en ajoutant du code et non en éditant du code existant)

Liskov substitution
- Utilisation d’interface pour éviter de mettre des ifs dans le code. (Bien que carré et rectangle aient des largeurs / longueurs, il faut rajouter une contrainte sur la largeur / longueur du carré)

Interface Segregation
- On modifie pas une interface mais on rajoute une nouvelle interface qui hérite de la première. (Permet le refacto en gardant le code legacy)

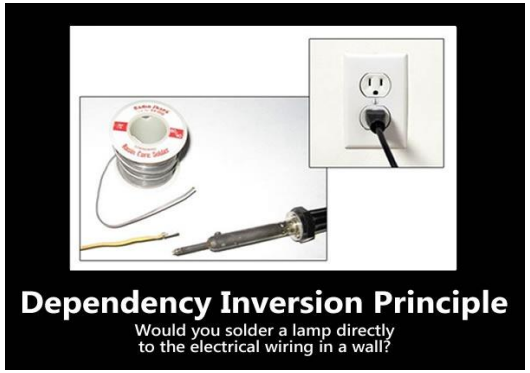
Dependency Injection
- Permet de garder une flexibilité sur le code

Exemple


Open Closed Principle
Usual way:
OCP:


LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?


Dependency Inversion Principle
Would you solder a lamp directly to the electrical wiring in a wall?

Quelques patterns - Repository

```
class BaseRepository<T>{
    public T Create(T data) { }
    public T Read(int id){ }
    public T Update(T data) {  }
    public boolean Delete(int id) { }
}
```

```
class ClientRepository extends BaseRepository<Client>{

}
```

```
class AddressRepository extends BaseRepository<Address>{
    public List<Address> find(){   }
}
```

Quelques patterns - Repository

```
interface IBaseRepository<T>{
    T Create(T data);
    T Read(int id);
    T Update(T data);
    boolean Delete(int id);
}

class BaseRepository<T> implements IBaseRepository<T>{
    public T Create(T data) { }
    public T Read(int id){ }
    public T Update(T data) { }
    public boolean Delete(int id) { }
}
```

```
interface IAddressRepository extends IBaseRepository<Address>{
    List<Address> find();
}

class AddressRepository extends BaseRepository<Address>
implements IAddressRepository{
    public List<Address> find(){ }
}
```

Quelques patterns – Fluent API

```
class StreetMap
{
  public String Country;
  public boolean City;
  public boolean River;
  public int Zoom;
  public String ApiKey;
}

interface IStreetMapBuilder
{
  IStreetMapBuilder IncludeCity (Supplier<Boolean> predicate);
  IStreetMapBuilder IncludeRiver ();
  IStreetMapBuilder WithZoom (int zoomLevel);
  StreetMap Build ();
}
```

```
class GoogleStreetMapBuilder implements IStreetMapBuilder
{
  private StreetMap _streetMap;
  public GoogleStreetMapBuilder (String country)
  {
    _streetMap = new StreetMap ();
    // un changement par exemple
    _streetMap.ApiKey = "GoogleStreetMap key api";
    _streetMap.Country = country;
  }
  public IStreetMapBuilder IncludeCity (Supplier<Boolean> predicate)
  {
    _streetMap.City = predicate.get();
    return this;
  }
  public IStreetMapBuilder IncludeRiver ()
  {
    _streetMap.River = true;
    return this;
  }
  public IStreetMapBuilder WithZoom (int zoomLevel)
  {
    _streetMap.Zoom = zoomLevel;
    return this;
  }
  public StreetMap Build ()
  {
    return _streetMap;
  }
}
```

Utilisation

```
int zoomLevel = 4;

StreetMap streetMap = new StreetMap();
streetMap.ApiKey = "GoogleStreetMap key api";
streetMap.Country = "France";
streetMap.City = false;
streetMap.River = true;
streetMap.Zoom = zoomLevel;

StreetMap map2 = new GoogleStreetMapBuilder("France")
                    .WithZoom(zoomLevel)
                    .IncludeCity(()-> zoomLevel<4)
                    .Build();
```

Quelques patterns – Factory

```
enum EnumMap
{
    GoogleMap,
    OpenstreetMap
}
class MapFactory
{
    public static IStreetMapBuilder getFactory(EnumMap mapType, String country)
    {
        switch (mapType)
        {
            case GoogleMap:
                return new GoogleStreetMapBuilder(country);
            case OpenstreetMap:
                return new OpenStreetMapBuilder(country);
            default:
                return null;
        }
    }
}
```

Utilisation

```
StreetMap map3 = MapFactory.getFactory(EnumMap.GoogleMap, "France")
                    .WithZoom(zoomLevel)
                    .IncludeCity(()-> zoomLevel<4)
                    .Build();
```

Quelques patterns – Decorator

```
class Report
{
    public void Export(String fileName) throws Exception
    {
        if (fileName == null || fileName.trim().length() == 0)
        {
            String msg = "Le nom du fichier ne peut être null";
            Logger.Log(msg);
            throw new Exception(msg);
        }
        File file = new File(fileName);

        if (file.exists())
        {
            String msg = "Le fichier existe déjà !";
            Logger.Log(msg);
            throw new Exception(msg);
        }

        // le code intéressant commence ici...
        // ...
    }
}
```

```
class Logger{
    static void Log(String message){}
}
```

Quelques patterns – Decorator

```
class Report
{
    public void Export(String fileName) throws Exception
    {
        if (fileName == null || fileName.trim().length() == 0)
        {
            String msg = "Le nom du fichier ne peut être null";
            Logger.Log(msg);
            throw new Exception(msg);
        }
        File file = new File(fileName);

        if (file.exists())
        {
            String msg = "Le fichier existe déjà !";
            Logger.Log(msg);
            throw new Exception(msg);
        }

        // le code intéressant commence ici...
        // ...
    }
}
```

```
class Logger{
    static void Log(String message){}
}
```

Quelques patterns – Decorator

La solution :

```
interface IReport
{
    void Export(String fileName) throws Exception;
}

class DefaultReport implements IReport
{
    public void Export(String fileName)
    {
        // Code utile uniquement !
    }
}
```

Quelques patterns – Decorator

L'implémentation des décorateurs :

```
class NoNullReport implements IReport
{
    private IReport origin;

    public NoNullReport(IReport report)
    {
        origin = report;
    }

    public void Export(String fileName) throws Exception
    {
        if (fileName == null || fileName.trim().length() == 0)
        {
            String msg = "Le nom du fichier ne peut être null";
            Logger.Log(msg);
            throw new Exception(msg);
        }
        origin.Export(fileName);
    }
}
```

```
class NoWriteOverReport implements IReport
{
    private IReport origin;

    public NoWriteOverReport(IReport report)
    {
        origin = report;
    }

    public void Export(String fileName) throws Exception
    {
        File file = new File(fileName);

        if (file.exists())
        {
            String msg = "Le fichier existe déjà !";
            Logger.Log(msg);
            throw new Exception(msg);
        }
        origin.Export(fileName);
    }
}
```

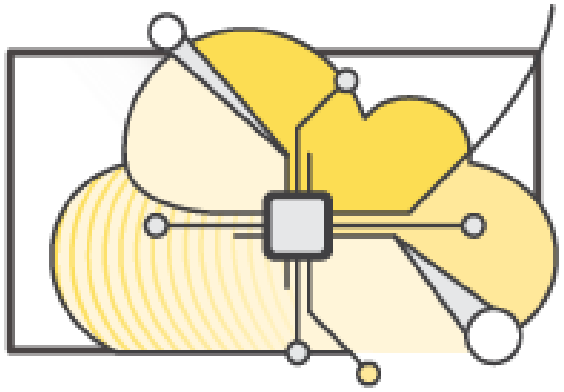
Quelques patterns – Decorator

L'utilisation :

```
IReport myReport;
myReport = new DefaultReport();
myReport.Export("toto.dat");

myReport = new NoNullReport(new DefaultReport());
myReport.Export("toto.dat");

myReport = new NoWriteOverReport(new DefaultReport());
myReport.Export("toto.dat");

myReport = new NoNullReport(new NoWriteOverReport(new DefaultReport()));
myReport.Export("toto.dat");

myReport = new NoWriteOverReport(new NoNullReport(new DefaultReport()));
myReport.Export("toto.dat");
```

Déploiement

Les différents déploiements



All-In Cloud

Hybrid

Private Cloud

(On-premises)

Zoom sur le cloud

IaaS

Infrastructure
as a Service

PaaS

Platform
as a Service

SaaS

Software
as a Service

API

Qu’est ce qu’une API ?

Application Programming Interface

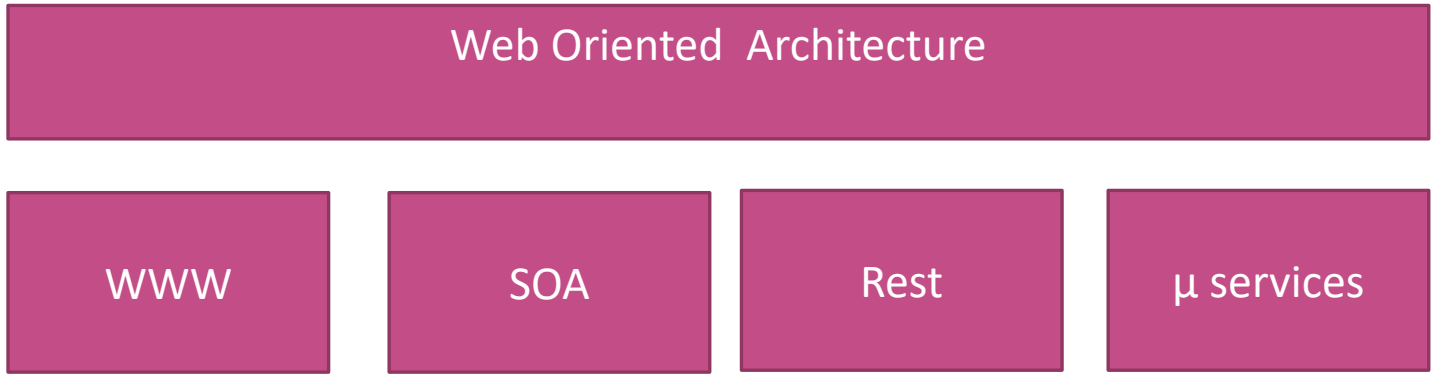
- offre une interface standard pour d’autre logiciel (on parle de WEB API)

on expose des ressources à d’autres systèmes (public ou en interne)

Permet le découplage et la composition de services pour différentes applications

Généralement on va parler d’API Rest mais ce n’est pas que ca

WOA - Web Oriented Architecture



SOAP

Schéma WSDL

Création d’un proxy, pour cacher la complexité d’appel

SOAP

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" ...>
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      ...
      <s:element name="GetWordListResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWordListResult" type="tns:ArrayOfString" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetWordListSoapIn">
    <wsdl:part name="parameters" element="tns:GetWordList" />
  </wsdl:message>
  <wsdl:message name="GetWordListSoapOut">
    <wsdl:part name="parameters" element="tns:GetWordListResponse" />
  </wsdl:message>
  <wsdl:portType name="AutoWebServiceSoap">
    <wsdl:operation name="GetWordList">
      <wsdl:input message="tns:GetWordListSoapIn" />
      <wsdl:output message="tns:GetWordListSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="AutoWebServiceSoap" type="tns:AutoWebServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWordList">
      <soap:operation soapAction="http://tempuri.org/GetWordList" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AutoWebService">
    <wsdl:port name="AutoWebServiceSoap" binding="tns:AutoWebServiceSoap">
      <soap:address location="http://www.vietnamofficeexpress.com/AutoWebService.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

```
public class DynamicWebServiceClientTest {

    private DynamicWebServiceClient webServiceClient = new DynamicWebServiceClient();

    @Test
    public void invoke_mathService_isPrimeNumber_happyPath() {

        ServiceDetail wsdlInfor = setMathServiceDetail();

        try {
            SOAPMessage response = webServiceClient.invokeOperation(wsdlInfor, "isPrimeNumber
                    Arrays.asList("5".split(",")));
            assertNotNull(response);
            String ret = SoapMessageUtil.outputSoapMessage(response);
            assertTrue(ret.contains("true"));

        } catch (Exception e) {
            System.out.println("Error " + e.getMessage());
        }
    }

    @Test
    public void invoke_mathService_sum_happyPath() {
```

SOAP – Exemple de requête

Requête :

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ns0:additionner
xmlns:ns0="http://axis.test.com"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<valeur1 xsi:type="xsd:int">40</valeur1>
<valeur2 xsi:type="xsd:int">2</valeur2>
</ns0:additionner>
</soapenv:Body>
</soapenv:Envelope>
```

Réponse :

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ns1:additionnerResponse
xmlns:ns1="http://axis.test.com"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<additionnerReturn href="#id0" />
</ns1:additionnerResponse>
<multiRef xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:long">42</multiRef>
</soapenv:Body>
</soapenv:Envelope>
```

REST

Modélisation orienté ressource et non opération

Basé sur les principes du HTTP (headers, verbes, …)

Pour favoriser l'émergence de ces API, il faut que cela soit une stratégie de l'entreprise

- changement de paradigme (on passe d'une structure fermée à un contexte beaucoup plus ouvert) On pouvait le faire avant avec SOAP mais c'etait quelque chose de lourd (VPN ou WS-Security)

- emergence des outils (API Gateway, …) initié par les géants du WEB

Exemple de requête :

```
GET https://api.{fakecompany}.com/sum?a=40&b=2
```

Stratégie d’API

Open API

- On laisse libre accès à nos API (avec un systéme de facturation) et on regarde ce qu’il se passe.

Ex : GoogleMaps, Twitter, Salesforce

API First

On commence par développer notre API que nous consommons également en interne, de cette manière on peut proposer des API utiles pour notre communauté.

Les régles d’or, d’une bonne API

respecte les standards HTTP (Header, Verbes)

s’inspire des bonnes pratiques des Géants du Web

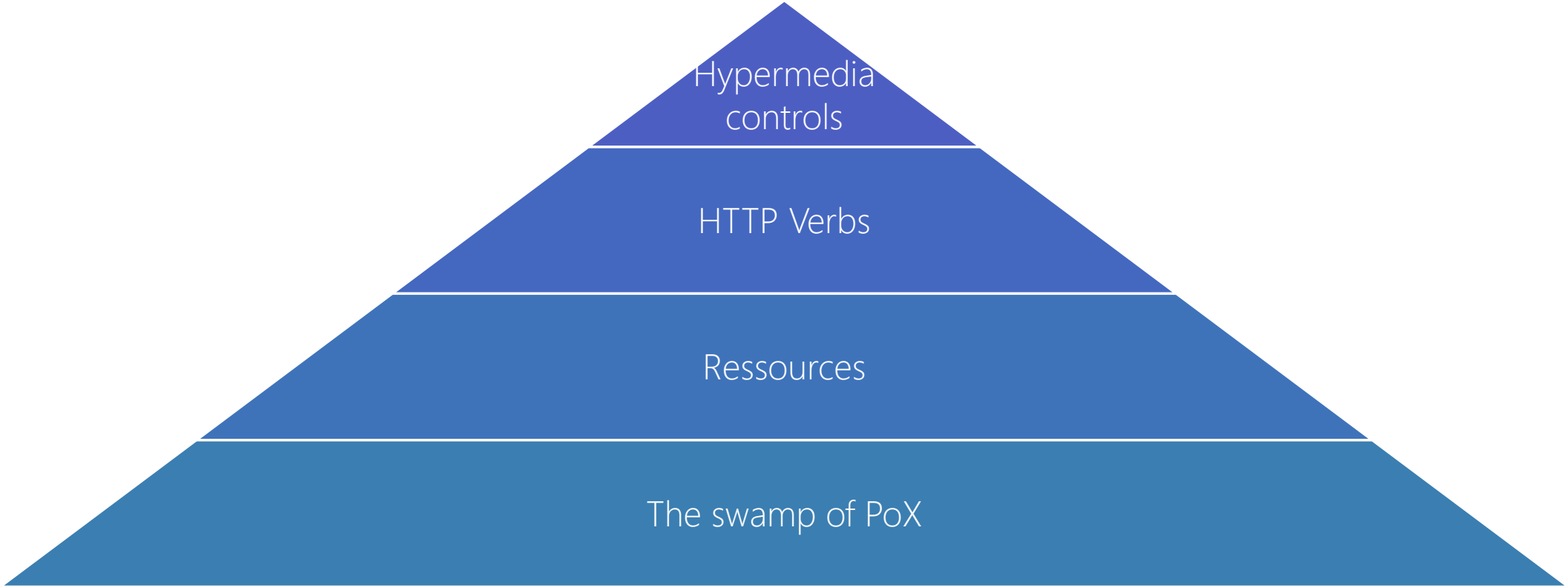
Est bien documenté ou autoporteuse de sens

est simple d’usage

est agnostique des consommateurs (il ne faut pas dépendre d’une plateforme, ou faire des cas particuliers pour telle ou telle équipe)

Stateless !

Restfull or Not Restfull



Stateless ?

On ne doit garder aucun contexte entre deux appels (session, authentification, …)

Avantages :

- User Agnostique, pas besoin de renvoyer une réponse particulière en fonction de qui m’appelle

- Rend cachable les requêtes, si on ne dépend pas d’un contexte on peut sauvegarder les résultats d’une requête

- On peut scaller de façon simple car nous n’avons pas de sticky session (loadbalancer)

SOAP vs REST

SOAP/RPC	REST
Modélisation par opération	Modélisation par ressources
Cherche à unifier le modèle de programmation distribué et local par l'intermédiaire d'un proxy	Modèle de programmation distribué explicite, et basé sur le WWW
Repose sur un toolkit pour être interprété	Mise sur une bonne expérience développeurs tous niveaux

Pour alimenter le débat …

Le stateless c’est bien (cf les avantages), mais que se passe t’il dans un contexte où votre application doit débiter un compte pour en créditer un autre ?

Trouver la bonne granularité

La théorie une « ressource = une URL »

Si on est trop radical on arrive à ca :

$> wget https://api.fakes.com/users/1243
StatusCode : 200
Content : {"id":"1234", "name":”Arnaud Lemettre",
"address":"https://api.fakes.com/addresses/4567"}

$> wget https://api.fakes.com/addresses/4567
StatusCode : 200
Content : {"id":"4567", "street":“rue voltaire", "country": "http://api.fakes.com/countries/98"}

Nommage – approche REST vs SOAP

L’approche SOAP :

getClient(1)

getAccount(1)

createTransaction(1, 2)

deleteAlert(123)

L’approche REST :

GET /clients/1

POST /transaction + {Payload}

DELETE /alert/123

Nommage – la casse

CamelCase

snake_case

spinal-case

Les deux derniers seront les plus inter-opérables entre les différents systèmes (windows/linux).

Nommage – les verbes

La liste des verbes :

GET

POST

DELETE

HEAD

OPTION

PUT

PATCH

Les codes retours

Verbes	Code
GET	200 (OK)
POST	201 (CREATED)
PUT	200 ou 201
PATCH	200
DELETE	200

Retour	Code
Unauthorize	403
Bad request	400
ServerError	500
NotFound	404
Partial Content (pagination)	206

Les codes retours – erreur

Attention lors d'une erreur serveur on peut être tenté de retourner de l'information (c'est bien) mais penser à pas changer les codes retours.

Exemple :

$> wget https://api.fake.com/v1/users/1

StatusCode : 200 OK

Content: « NotFound »

Les headers

Accept

Content-type

Versionning

Une API évolue avec le temps, et vous aurez des systèmes en production

La monté de version par défaut est dangereux (vous ne pouvez pas garantir de tout mettre à jour dans les temps)

Plusieurs solutions :

- URL : /v1/users

- Header : Accept: application/fake.v3+json

- sous-domaine : api.v1.fake.com

- Parametre : /users?v=1

Pagination

Sur de grands volumes de données, il y aura des problèmes de performance (latence réseau, charge serveur, client n'arrivant pas à traiter la donnée)

Il faut donc renvoyer page par page les données, et prévoir également cette mécanique lorsque celle-ci n'est pas nécessaire au début. => uniformité des APIs

Bonne pratique :

- prendre en compte dans la query string (permet une mise en cache, /!\ dangereux sur une liste)

- Retour dans le headers (content-range [offset – limit | count], Accept-Range)

- si la réponse n'est pas complete mettre code 206

Pagination - exemple

$> wget https://api.fake.com/v1/order?range=0-5 → Param dans le query string

StatusCode: 206 Partial Content → Réponse indiquant qu'il reste encore des datas

ContentRange: 0-5/6 → Index de début – fin / nbr éléments

AcceptRange : order 12 → Ressource, nombre d'éléments dans la collection

HATEOAS – le graal de l'API

Hypermedia As The Engine of Application State

Très lourd à mettre en place, mais nous arrivons à une API quasi auto porteuse de sens

Chaques resources sont liées entre elles par un lien

Pour cela il faut utiliser la balise link

Exemple Odata

ODATA

Open Data Protocol

GET https://services.odata.org/TripPinRESTierService/People Params

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Accept	application/json;odata.metadata=full	
New key	Value	Description

Body Cookies Headers (13) Test Results

Pretty Raw Preview JSON

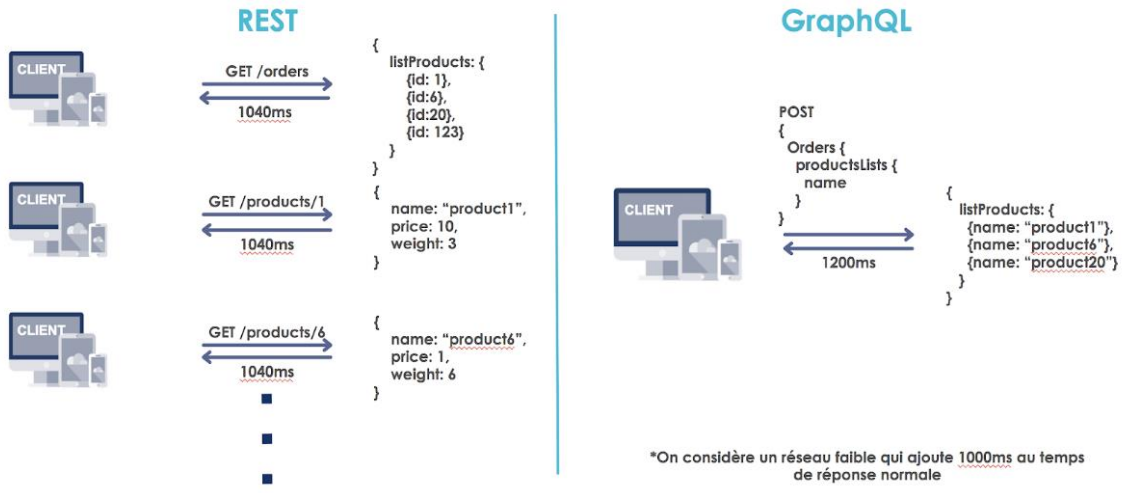

```
{
    "@odata.context": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/$metadata#People",
    "value": [
        {
            "@odata.type": "#Microsoft.OData.Service.Sample.TrippinInMemory.Models.Person",
            "@odata.id": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')",
            "@odata.editLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')",
            "UserName": "russellwhyte",
            "FirstName": "Russell",
            "LastName": "Whyte",
            "MiddleName": null,
            "Gender@odata.type": "#Microsoft.OData.Service.Sample.TrippinInMemory.Models.PersonGender",
            "Gender": "Male",
            "Age": null,
            "Emails@odata.type": "#Collection(String)",
            "Emails": [
                "Russell@example.com",
                "Russell@contoso.com"
            ],
            "FavoriteFeature@odata.type": "#Microsoft.OData.Service.Sample.TrippinInMemory.Models.Feature",
            "FavoriteFeature": "Feature1",
            "Features@odata.type": "#Collection(Microsoft.OData.Service.Sample.TrippinInMemory.Models.Feature)",
            "Features": [],
            "AddressInfo": [],
            "HomeAddress": null,
            "Friends@odata.associationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Friends/$ref",
            "Friends@odata.navigationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Friends",
            "BestFriend@odata.associationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/BestFriend/$ref",
            "BestFriend@odata.navigationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/BestFriend",
            "Trips@odata.associationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Trips/$ref",
            "Trips@odata.navigationLink": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Trips",
            "#Microsoft.OData.Service.Sample.TrippinInMemory.Models.ShareTrip": {
                "title": "Microsoft.OData.Service.Sample.TrippinInMemory.Models.ShareTrip",
                "target": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Microsoft.OData.Service.Sample.TrippinInMemory.Models.ShareTrip"
            },
            "#Microsoft.OData.Service.Sample.TrippinInMemory.Models.GetFavoriteAirline": {
                "title": "Microsoft.OData.Service.Sample.TrippinInMemory.Models.GetFavoriteAirline",
                "target": "https://services.odata.org/TripPinRESTierService/(S(fqu0qbnz5cgmjchbjj3zq211))/People('russellwhyte')/Microsoft.OData.Service.Sample.TrippinInMemory.Models.GetFavoriteAirline"
            },
```

Le petit nouveau - GraphQL

Pourquoi introduire une nouvelle façon de faire les API ?

- Optimisation charge réseau (data)

- Optimisation de la latence (nombre d'appels pour un résultat)



GraphQL – les concepts clefs

Mais qu'est ce que le GraphQL ?

- le langage de requêtes (Schema Definition Langage)
- le schéma (Query, Mutation, Subscription, Object)
- Les resolvers (liaison entre le couche de routage API, et la couche Business)

GraphQL

Pour en savoir plus :

Coté Serveur

https://www.graphql-java.com/tutorials/getting-started-with-spring-boot/

Coté client

APOLLO

https://www.apollographql.com/docs/angular/

Et Grpc dans tout ca …

gRPC est une infrastructure moderne pour la création de services en réseau et d'applications distribuées.

gRPC s'appuie sur HTTP/2 et le protocole d'encodage de message Protobuf pour fournir une communication haute performance et à faible bande passante entre les applications et les services.

gRPC utilise le langage IDL (Interface Definition Language) à partir des mémoires tampons de protocole. Vous devez coder manuellement le fichier *.proto* pour décrire les services ainsi que les entrées / sorties.

A partir de ce fichier, nous pouvons générer un code spécifique à chaque plateforme.

Pourquoi utiliser Grpc

gRPC offre quelques avantages :

- Performance, utilise le format binaire avec le protocole HTTP/2
- Interopérabilité, grâce au fichier .proto, nous pouvons targetter plusieurs langages (c++, c#, java ...)
- Diffusion, permet également la communication bi-directionnelle
- Délais d'attente et annulation,
- Sécurité, sécurisé de base par HTTPS

Exemple de fichier .proto

```
syntax = "proto3";
option java_package = "TraderSys";
import "google/protobuf/timestamp.proto";

message ConnectionClosedResponse { }

message Thing{
            string Description = 1;
            google.protobuf.Timestamp Time = 2;
}

service ThingLog {
  rpc OpenConnection(stream Thing) returns (ConnectionClosedResponse);
}
```

Pour aller plus loin …

Lien vers la documentation

Lien vers les exemples

Sécurité

OAuth2 / OpenId connect

=> repose sur la couche de transport TLS

En complément on peut utiliser JWT (JSON Web Token) permet de stocker des informations sur l'utilisateur (id) et ses claims (droit sur l'application). Normalisé avec OpenIdConnect

Sécurité – OAuth2

Un peu de vocabulaire :

- Resource Owner : généralement l'utilisateur
- Resource Server : le serveur possédant les ressources à utiliser
- Client Application : l'application demandant l'accès
- Authorization Server : le serveur donnant l'accés

Les types de token :

- access_token, permet l'accès à la ressource
- refresh_token, permet de récupérer un nouveau token en cas d'expiration de l'access_token

Sécurité – OAuth2

Différents types d'authorisation :

- Authorization Code Grant (forme la plus sécurisé, mais le token est stocké en session coté serveur)
- Implicit Grant (utilisé plus dans le cas d'une SPA, mais ne permet pas d'avoir de token de renouvellement)
- Resource Owner Password Credentials Grant (on doit saisir son mot de passe, il faut que le client et le serveur soit développé par la même équipe)
- Client Credentials Grant (utilisé pour authentifier l'application auprès du serveur, utile dans le cadre de communication API to API)

Sécurité - JWT

Un token JWT =

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. → Header

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ. → Payload

SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c → Signature

Sécurité - JWT

A récupérer lors de votre login

C’est au navigateur de le garder

Ce token est à rajouter à chaque requête

Header: Authorization = Bearer xxxxx.xxxxxx.xxxxx


Questions ?