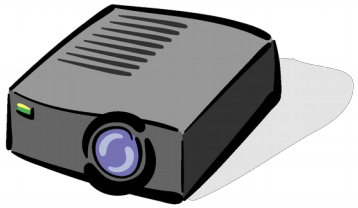
Université IBM i – 2017

18 Mai 2017 – API REST, retour d'expérience

VOLUBIS
DEVENEZ UN EXPERT



Volubis.fr

Conseil et formation sur OS/400, I5/OS puis IBM ***i*** depuis 1994 !

Dans nos locaux, vos locaux ou par Internet

Christian Massé - cmasse@volubis.fr

IBM i : API REST, retour d'expérience

- **Session 2: API REST, retour d'expérience**

Avec U GIE IRIS

Patrick TARNAUD

Jeremy GOIZIL

IBM i : API REST, retour d'expérience

- **U GIE IRIS**

  Présentation

  de la société

  du projet

API REST & Architecture

Forum IBMi - 17/05/2017

Patrick TARNAUD
Système U / U GIE IRIS
Architecte Logiciel

Sommaire

- Système U // U GIE IRIS
- Système d'information : Existant & Cible
- API Management IRIS
- API Prototype
- API Designer
- Projets
- Demain

- 4ème groupe de distribution alimentaire français
- 1566 points de vente
- 65 000 collaborateurs
- 4 Centrales Régionales et 1 Centrale Nationale

Discount

HYPER U SUPER U

831 magasins

16,99 mds € de CA TTC (hors carburants)

89,8% de parts sur CA Groupe

2 230 535 m² de surface totale de vente

HYPER U SUPER U MARCHE U U express Utile

Proximité

MARCHE U U express Utile

735 magasins

1,930 mds € de CA TTC (hors carburants)

10,2% de parts sur CA Groupe

354 125 m² de surface totale de vente

Parts de marché :

U GIE IRIS

Le prestataire informatique interne au Groupement U (400 collaborateurs)

Définir et mettre en œuvre la stratégie informatique

Garantir l'efficacité et la disponibilité des systèmes d'informations pour l'ensemble des utilisateurs

Assurer la gestion, le développement, la maintenance et l'exploitation des solutions informatiques des magasins et des centrales

22 000
postes de travail

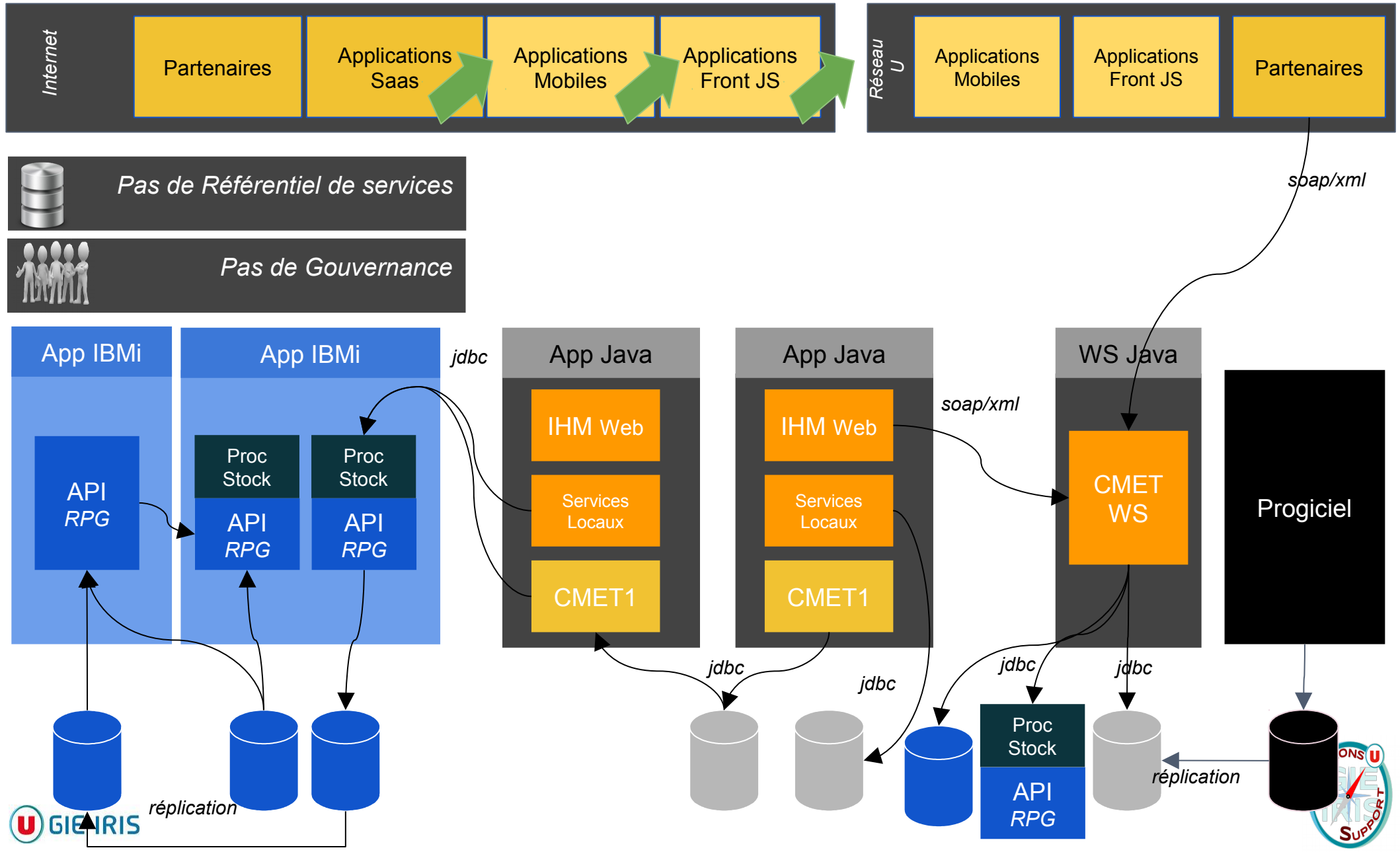
600
To de données

2 500
serveurs en magasins

1 000
serveurs en centrales

Système d'information Existant & Cible

SI : Existant


Internet
Partenaires
Applications Saas
Applications Mobiles
Applications Front JS
Réseau U
Applications Mobiles
Applications Front JS
Partenaires
soap/xml
Pas de Référentiel de services
Pas de Gouvernance
App IBMi
App IBMi
jdbc
App Java
App Java
WS Java
IHM Web
IHM Web
soap/xml
CMET WS
Progiciel
API RPG
Proc Stock
API RPG
Proc Stock
API RPG
Services Locaux
Services Locaux
CMET1
CMET1
jdbc
jdbc
jdbc
jdbc
Proc Stock
API RPG
réplication
réplication
GIE IRIS

SI : Objectifs Cibles

SI ouvert et interopérable

>> mise en oeuvre d'APIs <<
>> utilisation de protocoles d'échanges standards <<
>> mise en oeuvre de solutions légères <<

SI agile, réactif, modulaire

>> APIs de faible granularité <<
>> valoriser l'existant <<
>> innover/tester des nouvelles technos <<
>> SI scalable <<

SI gouverné

>> référentiel des APIs <<
>> urbaniser le SI <<

SI sécurisé/supervisé

>> système centralisé d'APIs <<

Faible Granularité

Logique Métier

Interface d'accès aux Ressources

Moyen de persistance

Référencé en un point unique

Technologiquement agnostique

Dépendances

Responsabilité d'un Secteur du POS (et donc d'une équipe)

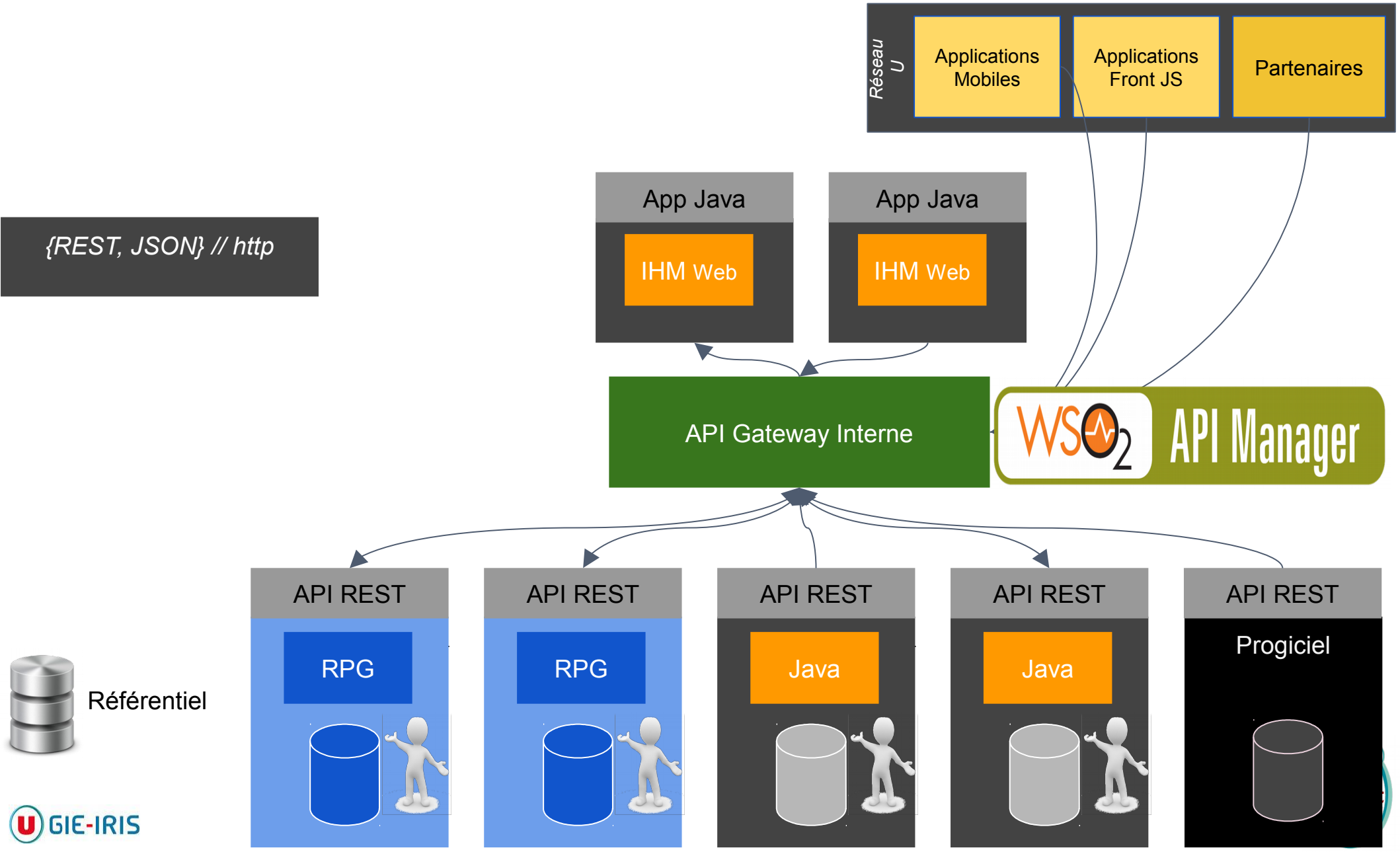
SOAP = RPC + protocole applicatif + des frameworks
REST = utilisation de ce qui fonctionne déjà : accès à des Ressources sur le Web

Http comme protocole de transport
Les verbes http pour les actions (GET, POST, PUT, DELETE)
Des URIs pour identifier les Ressources
Les codes statut http comme code de retour (404)

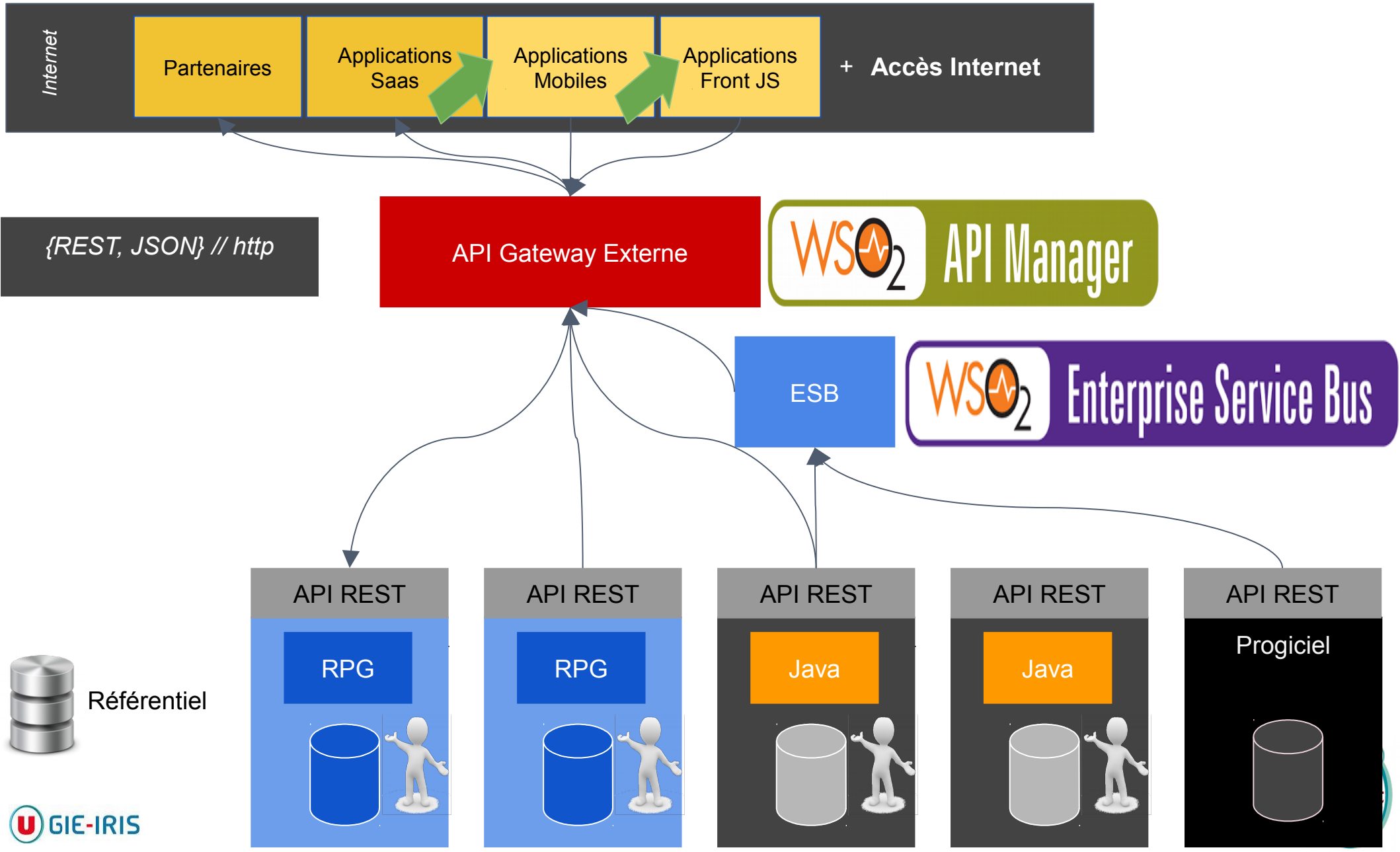
Des services accessibles depuis un simple navigateur, ou du JS

Des données lisibles au format JSON, à plus faible empreinte réseau

Scalable, spécifié (RAML), répandu (GAFA) …..


Réseau
Applications Mobiles
Applications Front JS
Partenaires
{REST, JSON} // http
App Java
IHM Web
App Java
IHM Web
API Gateway Interne
WSO2 API Manager
API REST
RPG
API REST
RPG
API REST
Java
API REST
Java
API REST
Progiciel
Référentiel

Cible : Externe



API Management IRIS

Architecture Applicative commune JEE/IBMi

Architecture REST

Référentiel d'API commun (API Designer)

Processus de build / Usine logicielle

Prototype gestion commande

Frameworks exposition, consommation, trace, erreurs

Middleware **Liberty**

IBM

Prototype gestion commande

Frameworks exposition, consommation, trace, erreurs

Middleware

Java

Accompagnement

Plateforme d'Echanges Centralisée SOFIA
API Gateway Externe/Interne, ESB

API Management **IRIS**

Processus

Référentiel

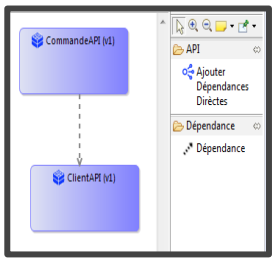
Génération

Spécification interface RAML

Génération

Code

Déploiement



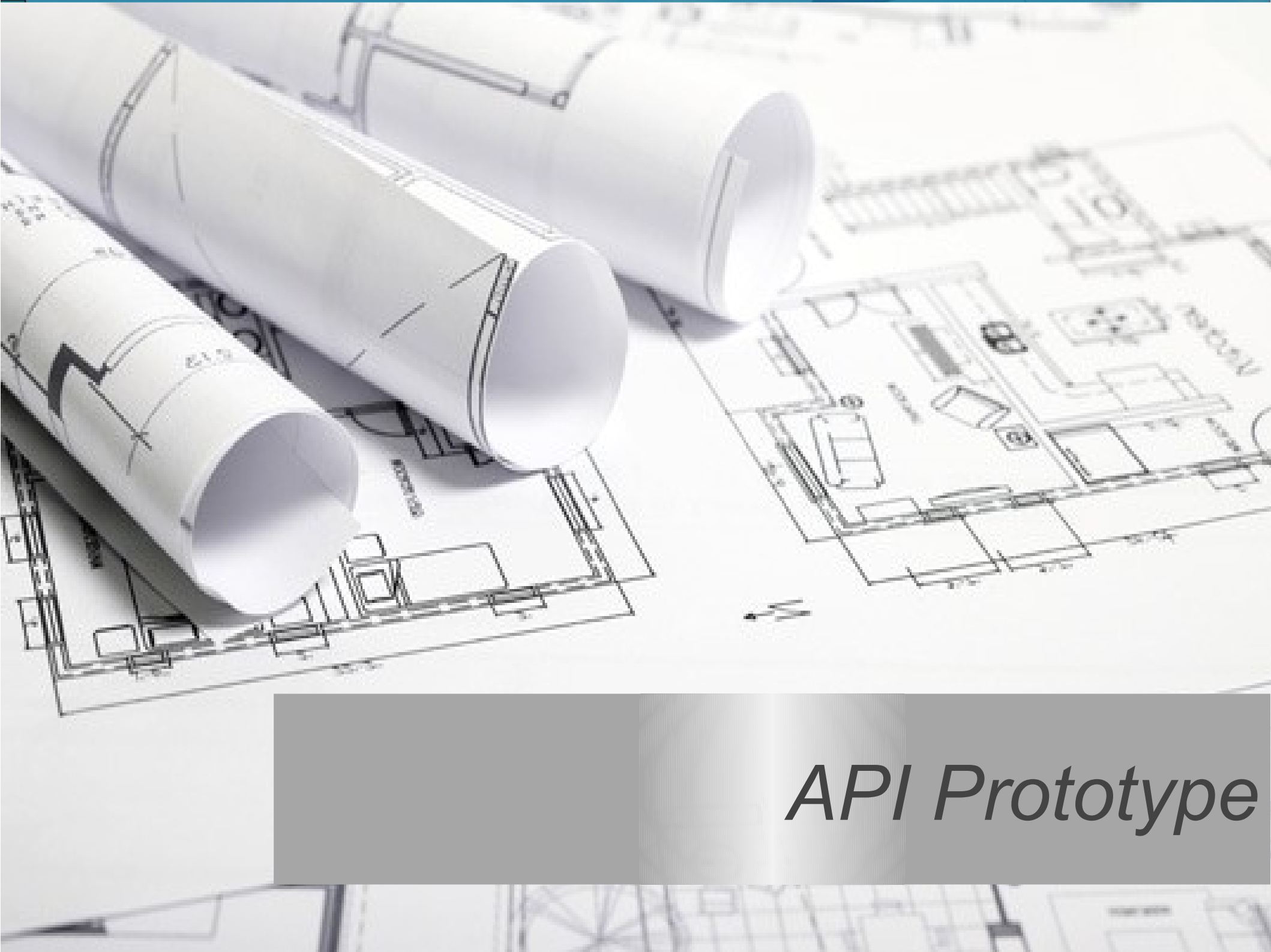
API Designer

```
{
  "Time"      : "2014-02-12 14:20:05",
  "Latitude"  : 37.33233141,
  "Longitude" : -122.0312186,
  "Count"     : 101,
  "Comments"  : "Bad Data. SNOW DAY!!",
  "Luma"      : 0,
  "Habitat"   : "Back yard, grass",
  "Types"     : [
                  "5",
                  "6"
                ],
  "Address"   : {
                  "Street"  : "2522 West Georgia Road",
                  "City"    : "Piednont",
                  "State"   : "South Carolina",
                  "Country" : "United States",
                }
}
```

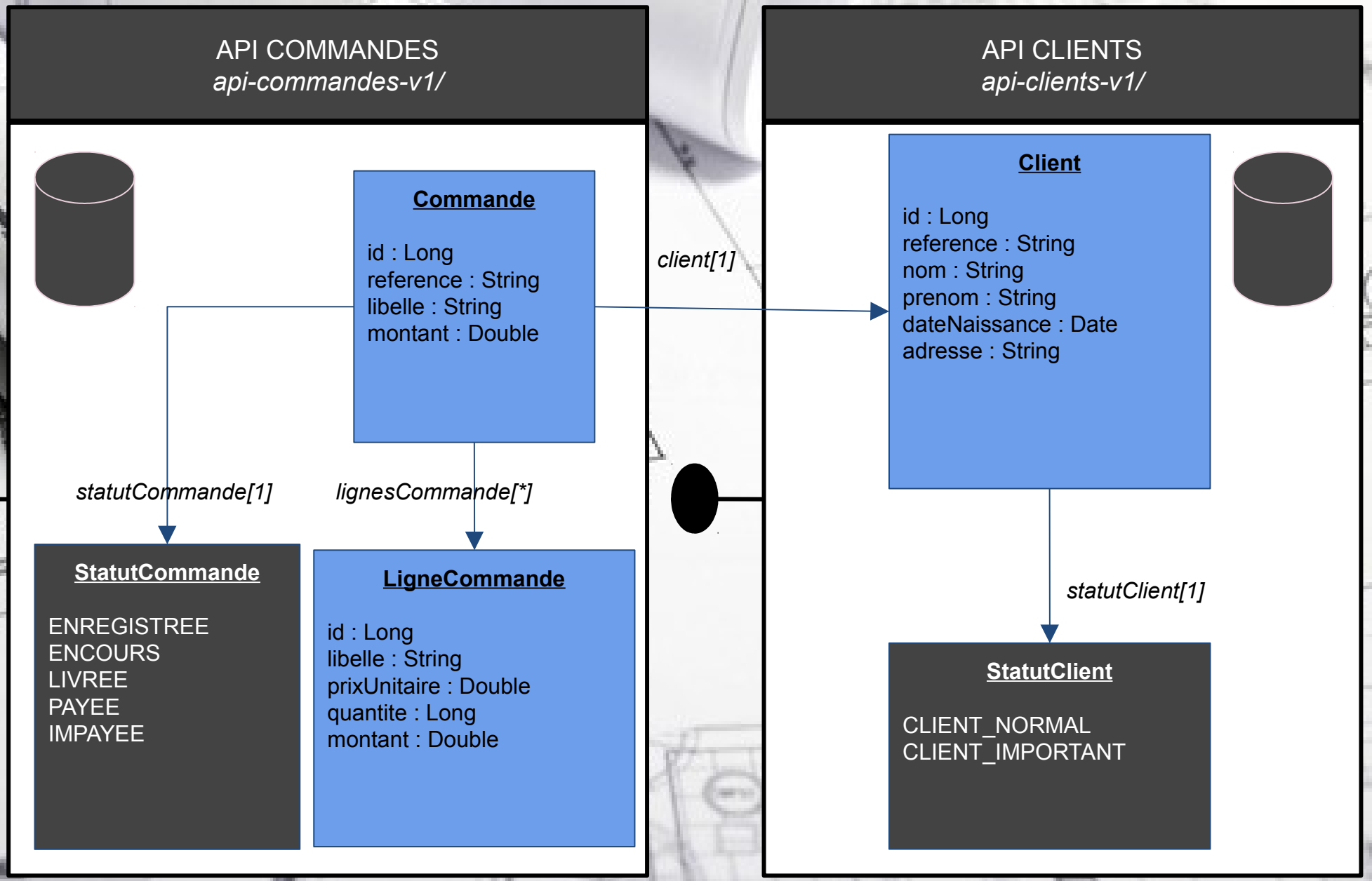
Swagger

RDI

IBM Liberty



API Prototype


API COMMANDES
api-commandes-v1/
Commande
id : Long
reference : String
libelle : String
montant : Double
client[1]
statutCommande[1]
lignesCommande[*]
StatutCommande
ENREGISTREE
ENCOURS
LIVREE
PAYEE
IMPAYEE
LigneCommande
id : Long
libelle : String
prixUnitaire : Double
quantite : Long
montant : Double
API CLIENTS
api-clients-v1/
Client
id : Long
reference : String
nom : String
prenom : String
dateNaissance : Date
adresse : String
statutClient[1]
StatutClient
CLIENT_NORMAL
CLIENT_IMPORTANT

clients

Show/Hide | List Operations | Expand Operations

DELETE	/clients	Suppression de tous les clients
GET	/clients	Recherche des clients
POST	/clients	Création d'un client
GET	/clients/ageMoyen	Age moyen des clients
POST	/clients/lireParIdentifiants	Recherche des clients par une liste d'identifiants
DELETE	/clients/{id}	Suppression d'un client
GET	/clients/{id}	Lecture d'un client
PUT	/clients/{id}	Mise à jour d'un client
POST	/clients/{id}/modifierStatutClient	Modification du statut d'un client

swagger

api-commandes-v1

GET	/commandes	Recherche des commandes
POST	/commandes	Création d'une commande
DELETE	/commandes/{id}	Suppression d'une commande
GET	/commandes/{id}	Lecture d'une commande
PUT	/commandes/{id}	Mise à jour d'une commande
POST	/commandes/{id}/changerStatutCommande	Mise à jour du statut d'une commande
GET	/commandes/{id}/lignesCommande	Lecture des lignes de commande d'une commande
POST	/commandes/{id}/lignesCommande	Ajout d'une ligne de commande à une commande
PUT	/commandes/{id}/lignesCommande	Mise à jour de la liste des lignes de commande d'une commande
GET	/clients/debiteurs	Récupère la liste des clients débiteurs
GET	/clients/{id}/commandes	Récupère la liste des commandes d'un client

swagger

api-commandes-v1

Request URL

```
http://u3antu148:8001/TutorialCommande/commandes?idClient=1
```

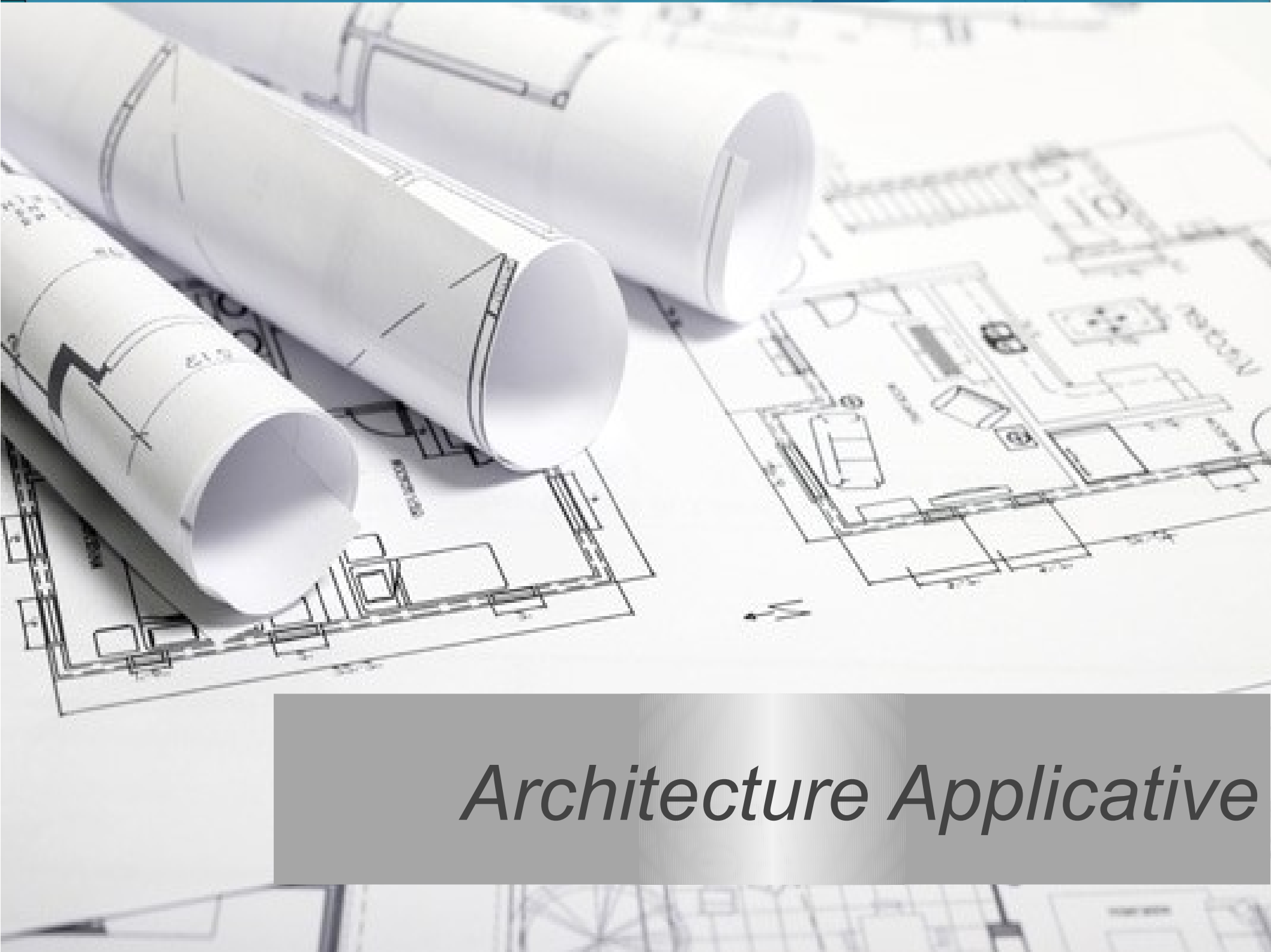
Response Body


```
[
  "id": 1,
  "reference": "CMD001",
  "libelle": "Commande CMD001",
  "montant": 1000,
  "statutCommande": "ENREGISTREE",
  "client": {
    "id": 1
  },
  "lignesCommande": [
    {
      "id": 1,
      "libelle": "Ligne commande 001",
      "prixUnitaire": 500,
      "quantite": 1,
      "montant": 500
    },
    {
      "id": 2,
      "libelle": "Ligne commande 002",
      "prixUnitaire": 500,
      "quantite": 1,
```


Response Code

```
200
```

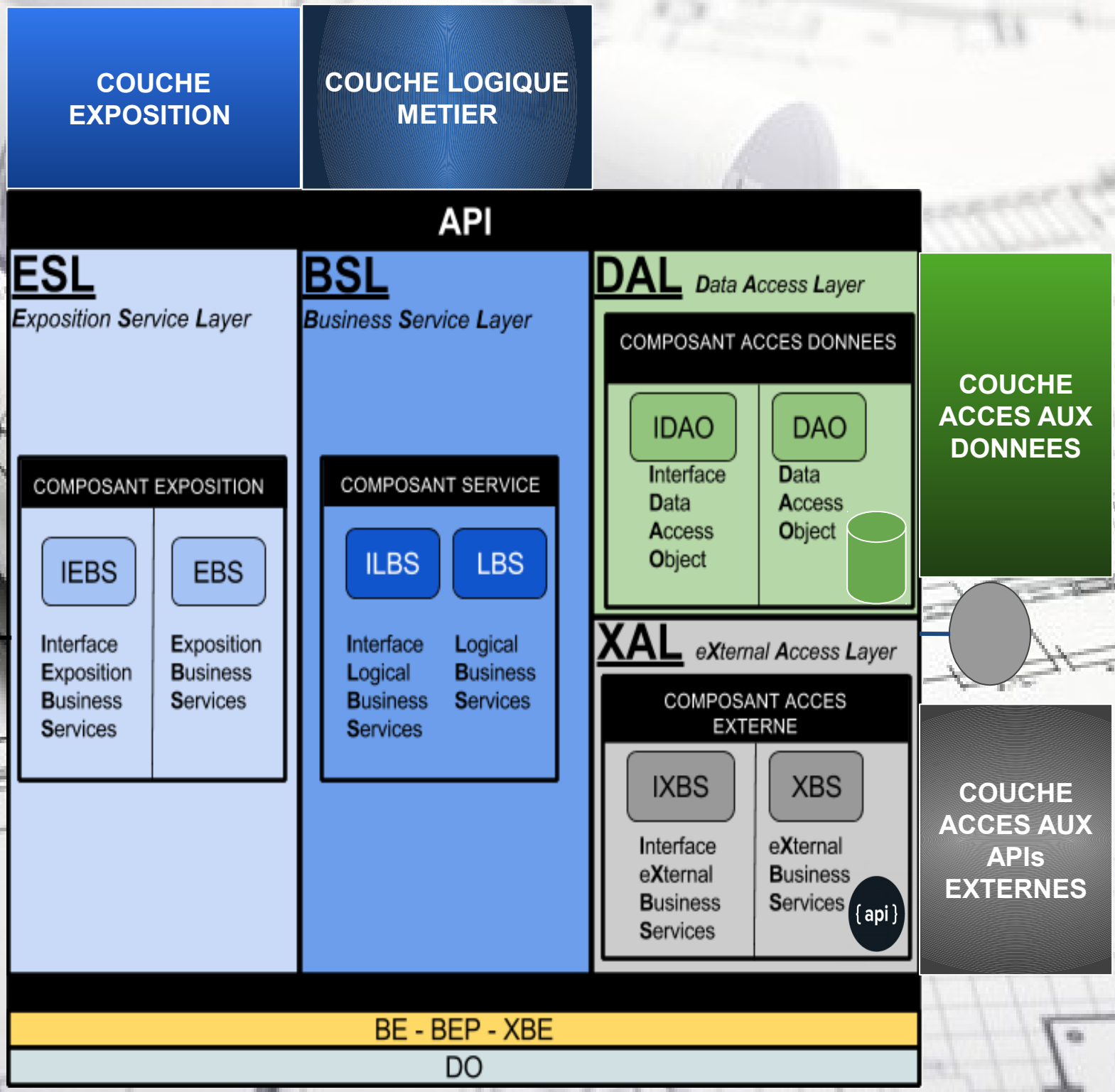
Response Headers

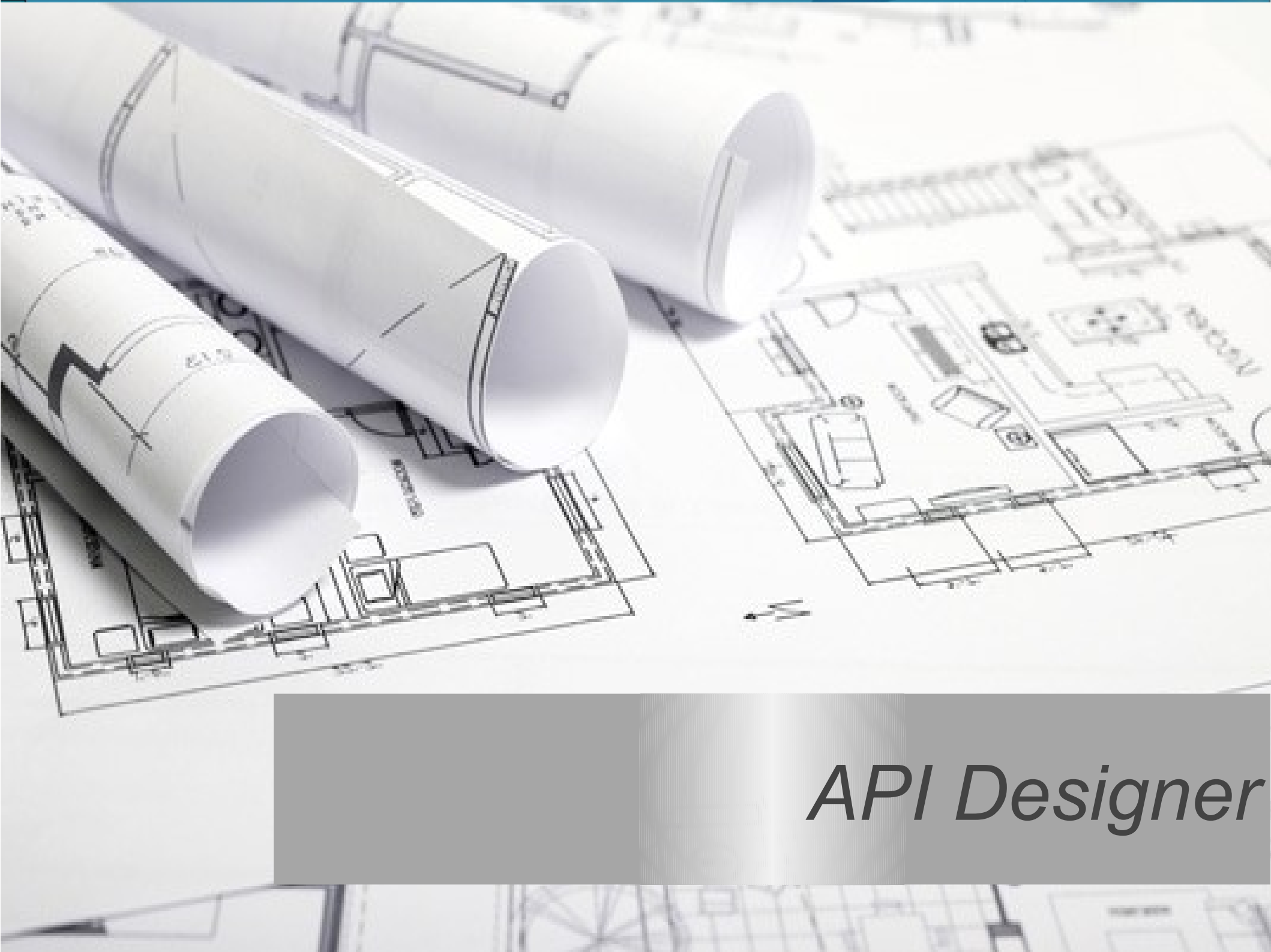


Architecture Applicative

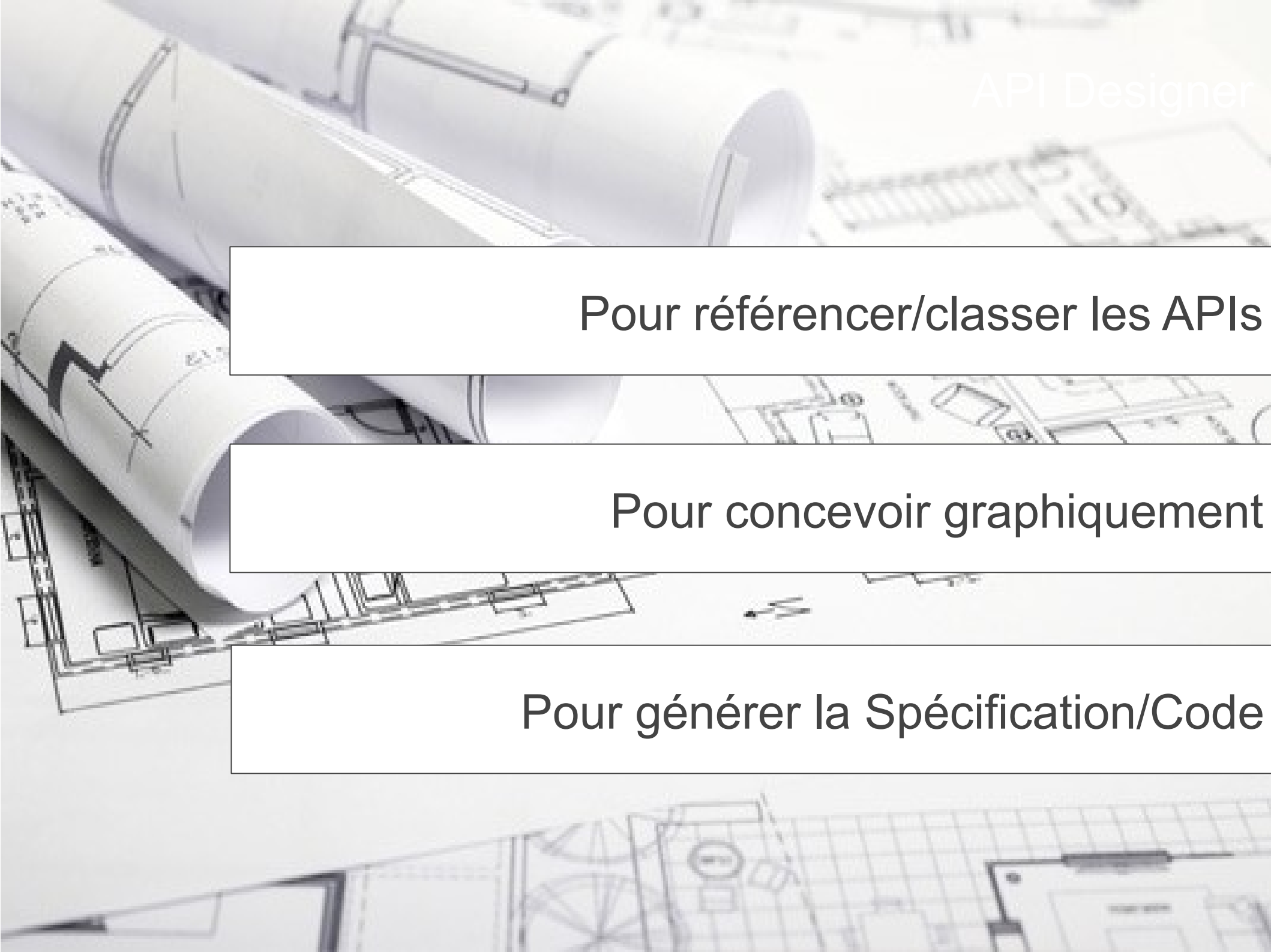
Architecture Applicative

- Supporter les APIs
- Disposer d’un vocabulaire/concepts communs
- Structurer l’implémentation par composant et responsabilité
- Faciliter la maintenance et l’évolutivité


COUCHE EXPOSITION
COUCHE LOGIQUE METIER
API
ESL
Exposition Service Layer
BSL
Business Service Layer
DAL Data Access Layer
COMPOSANT ACCES DONNEES
IDAO
Interface Data Access Object
DAO
Data Access Object
COUCHE ACCES AUX DONNEES
COMPOSANT EXPOSITION
IEBS
Interface Exposition Business Services
EBS
Exposition Business Services
COMPOSANT SERVICE
ILBS
Interface Logical Business Services
LBS
Logical Business Services
XAL eXternal Access Layer
COMPOSANT ACCES EXTERNE
IXBS
Interface eXternal Business Services
XBS
eXternal Business Services
{api}
COUCHE ACCES AUX APIs EXTERNES
BE - BEP - XBE
DO



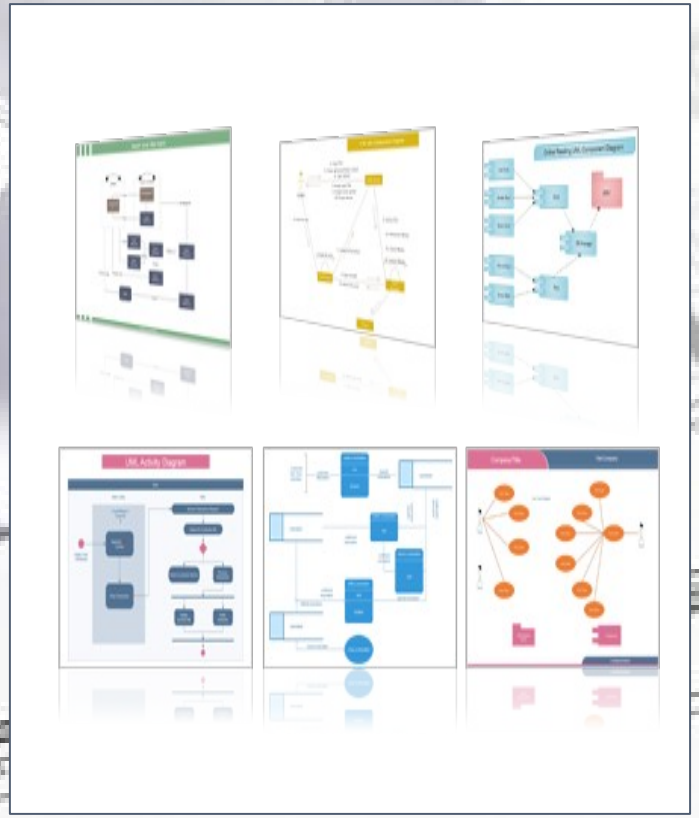
API Designer


API Designer
Pour référencer/classer les APIs
Pour concevoir graphiquement
Pour générer la Spécification/Code

API Designer

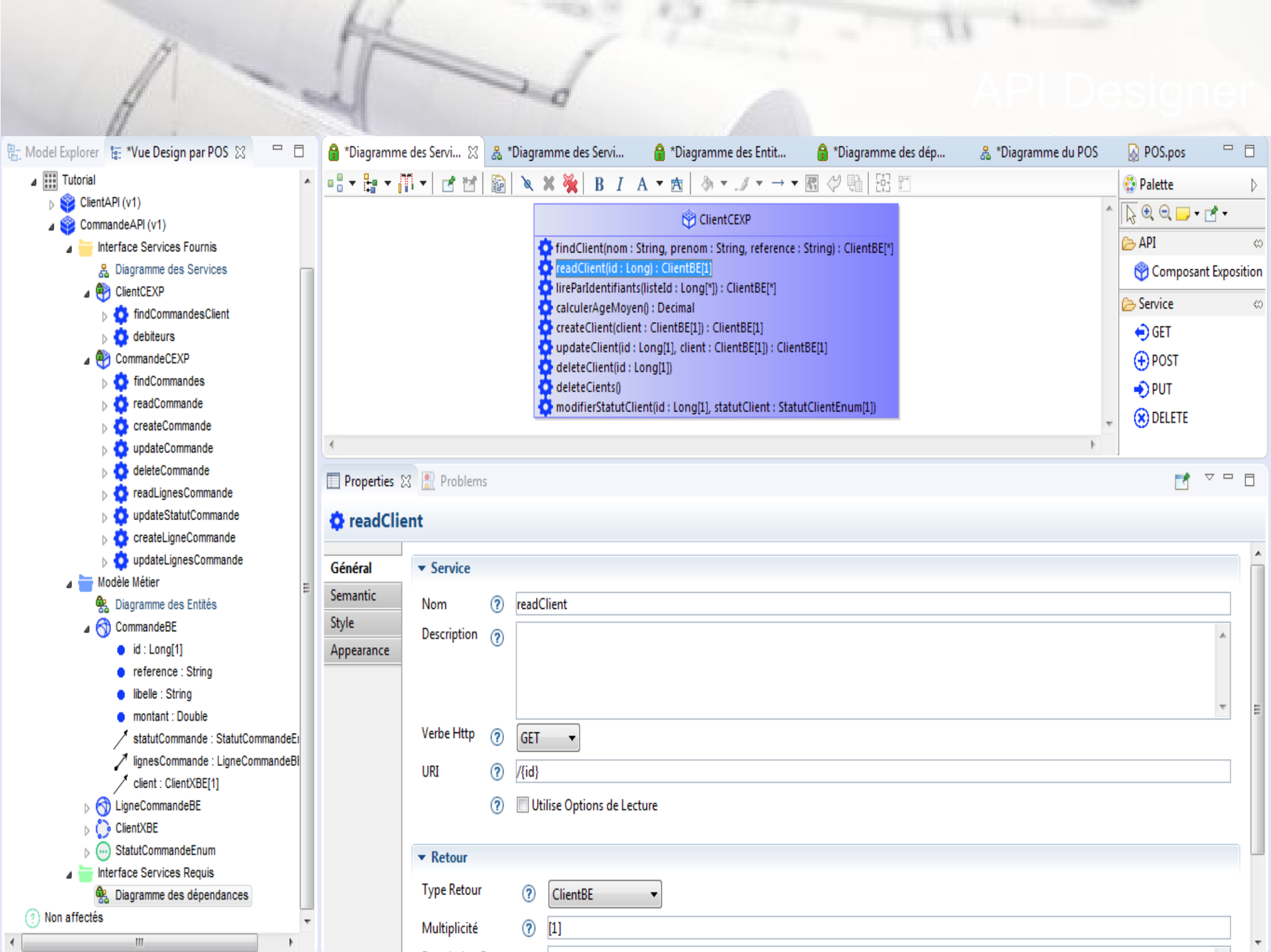
OBEO Designer

eclipse

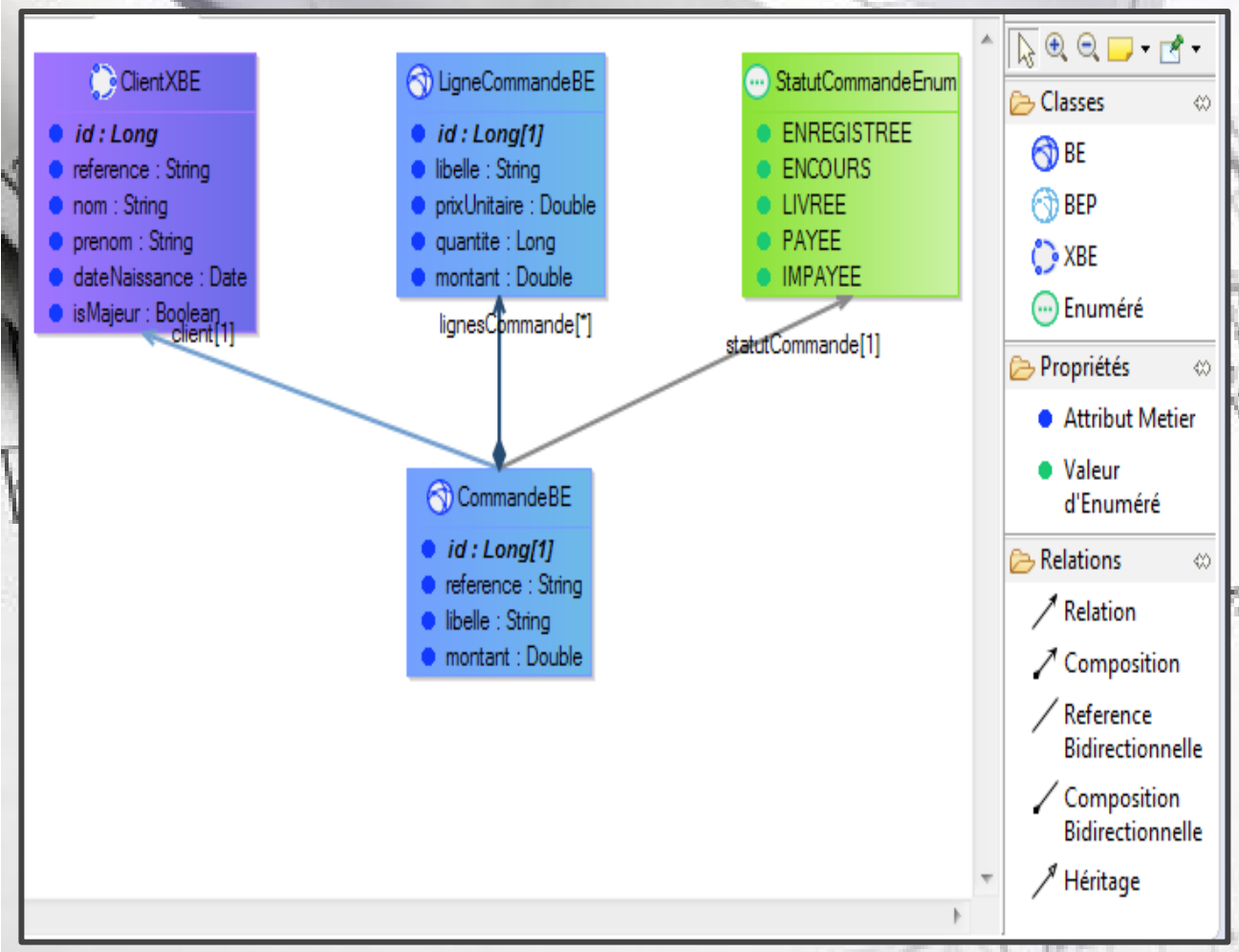


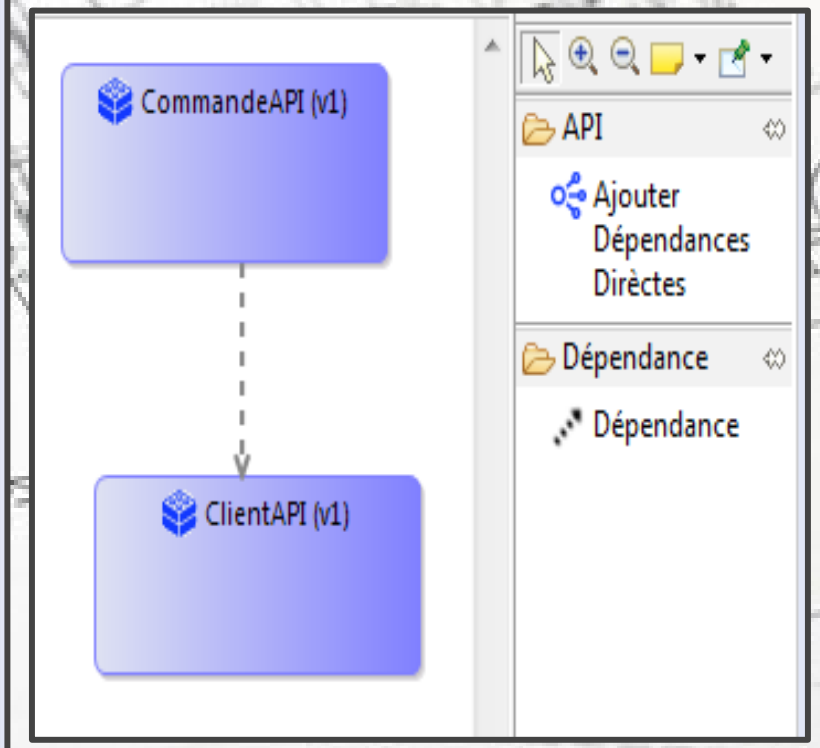
API → **Services Ressources** → **Dépendances**

Plan d'Occupation des Sols (POS)


API Designer
Model Explorer
*Vue Design par POS
Tutorial
ClientAPI (v1)
CommandeAPI (v1)
Interface Services Fournis
Diagramme des Services
ClientCEXP
findCommandesClient
debiteurs
CommandeCEXP
findCommandes
readCommande
createCommande
updateCommande
deleteCommande
readLignesCommande
updateStatutCommande
createLigneCommande
updateLignesCommande
Modèle Métier
Diagramme des Entités
CommandeBE
id : Long[1]
reference : String
libelle : String
montant : Double
statutCommande : StatutCommandeEn
lignesCommande : LigneCommandeBE
client : ClientXBE[1]
LigneCommandeBE
ClientXBE
StatutCommandeEnum
Interface Services Requis
Diagramme des dépendances
Non affectés
*Diagramme des Servi...
*Diagramme des Servi...
*Diagramme des Entit...
*Diagramme des dép...
*Diagramme du POS
POS.pos
ClientCEXP
findClient(nom : String, prenom : String, reference : String) : ClientBE[*]
readClient(id : Long) : ClientBE[1]
lireParIdentifiants(listeId : Long[*]) : ClientBE[*]
calculerAgeMoyen() : Decimal
createClient(client : ClientBE[1]) : ClientBE[1]
updateClient(id : Long[1], client : ClientBE[1]) : ClientBE[1]
deleteClient(id : Long[1])
deleteCients()
modifierStatutClient(id : Long[1], statutClient : StatutClientEnum[1])
Palette
API
Composant Exposition
Service
GET
POST
PUT
DELETE
Properties
Problems
readClient
Général
Semantic
Style
Appearance
Service
Nom
readClient
Description
Verbe Http
GET
URI
/{id}
Utilise Options de Lecture
Retour
Type Retour
ClientBE
Multiplicité
[1]

API Designer



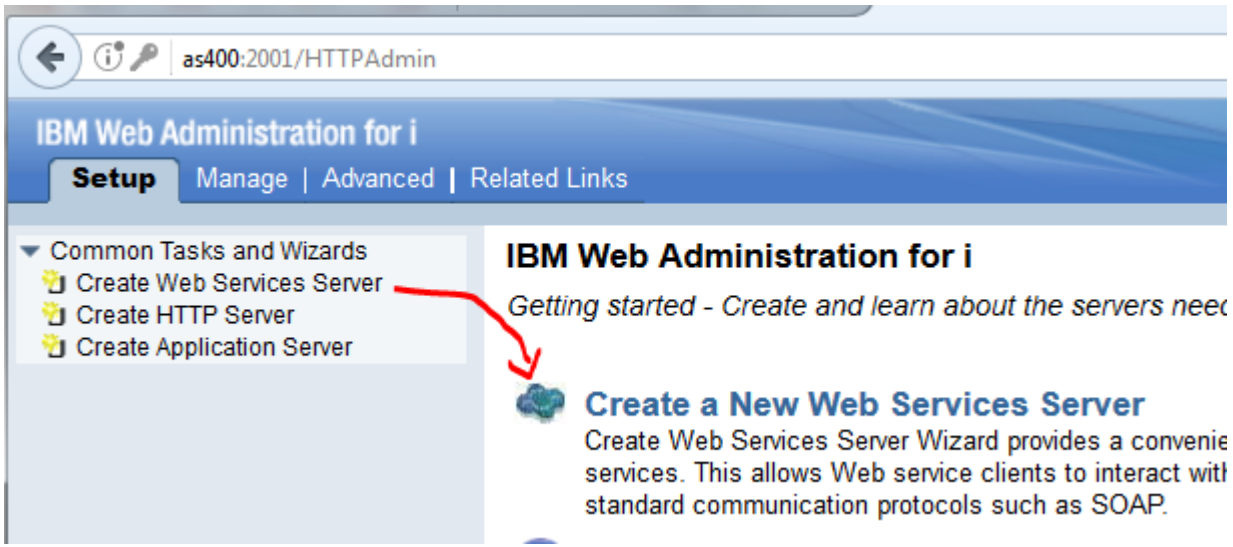


IBM i : API REST, retour d'expérience

- **Aspects techniques**

Nous avons utilisé, bien évidemment, le serveur intégré (Liberty)

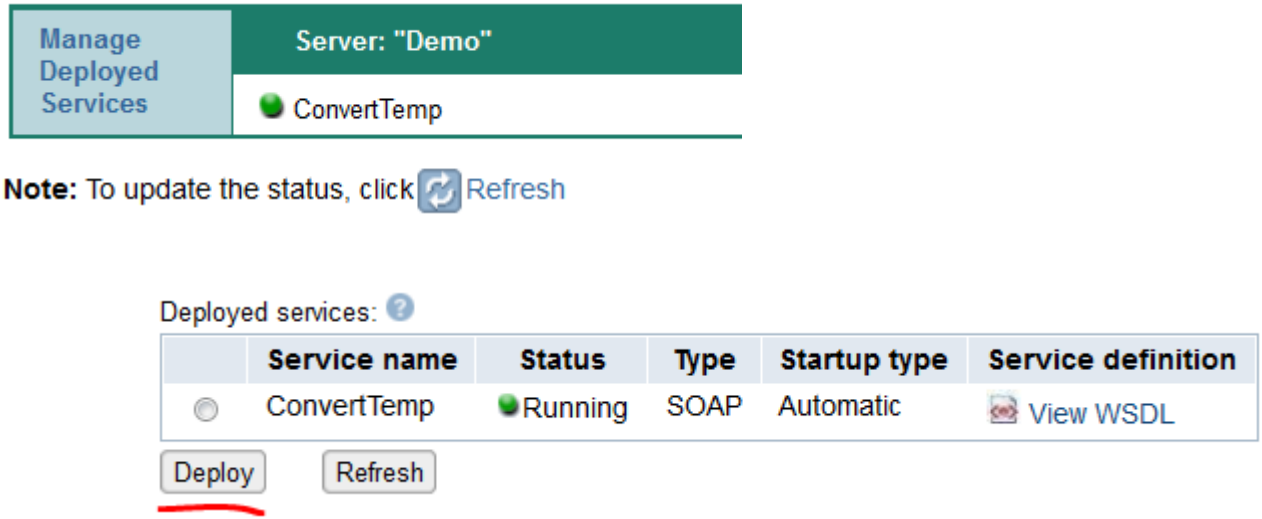
```
QHTTPSVR      QSYS         SBS      0,0                    DEQW
  ADMIN       QTMHHTTP     BCH      0,0   PGM-QZHBMAIN     SIGW
  ADMIN       QTMHHTTP     BCI      0,0   PGM-QZSRLOG      SIGW
                                                    A suivre...
```



IBM i : API REST, retour d'expérience

- **Aspects techniques**

  Ce dernier permet ensuite de déployer des programmes RPG/Cobol en tant que web services



IBM i : API REST, retour d'expérience

- **Aspects techniques**

  Les choix architecturaux préconisaient REST

Specify Web service type:

(•) SOAP

A SOAP-based Web service is a self-contained software component with a well-defined interface that describes a set of operations that are accessible over the Internet and exchange XML messages that are based on the SOAP protocol.

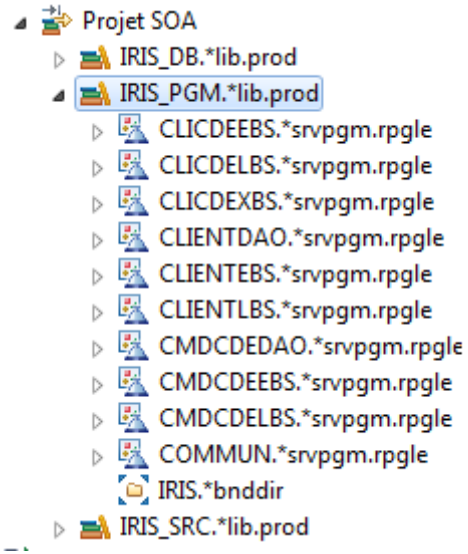
( ) REST

A REST-based Web service exposes resources, where client requests are handled by resource methods and the format of messages that are exchanged is defined by the resource itself.

IBM i : API REST, retour d'expérience

- **Aspects techniques**

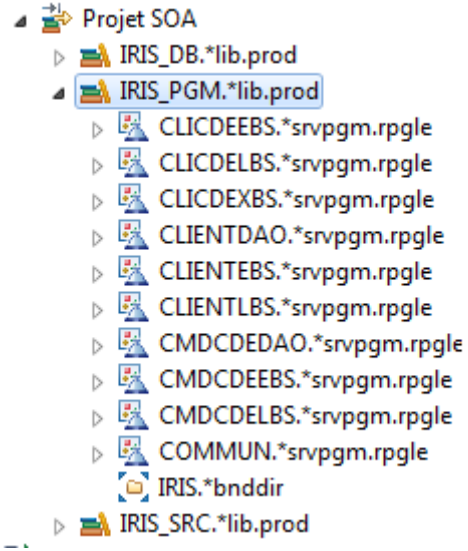
UN même service devant répondre à plusieurs méthodes (GET pour lire, POST pour créer, PUT pour mettre à jour, DELETE pour détruire) le choix des programmes de service, pouvant contenir plusieurs procédures (autant que de méthodes) , s'est vite imposé !



IBM i : API REST, retour d'expérience

- **Aspects techniques**

Ont été déployés, trois services sous forme de *SRVPGM, donc



CLICDEBS **-> les commandes PAR CLIENT**

CLIENTEBS **-> les clients**

CMDCDEEBS **-> les commandes**

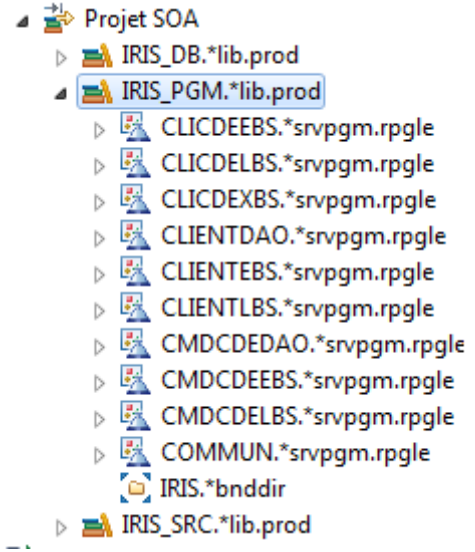
IBM i : API REST, retour d'expérience

- **Aspects techniques**

  les composants EBS gèrent l'exposition

  les composants LBS contiennent la logique métier

  les composants DAO masquent l'accès à la base de donnée (DB2)



Clients et Commandes n'étant pas dans le même quartier (dans le cadre d'une urbanisation du code), le composant CLICDEXBS accède aux informations client en consommant le web service adéquat.

IBM i : API REST, retour d'expérience

- **Aspects techniques**

A l'origine, une structure de type DTO était exposée (1)
Une structure de type BE (Business Objet) était manipulée ensuite(2)

```
dcl-proc lireclient export;
  dcl-pi *n ;
    id int(10);
    client likeds(clientDTO);  1
    codeHttp like(JSENDcodeHttp);
    erreurs_LENGTH like(JSENDerreurs_length);
    erreurs likeds(JSENDerreurs) DIM(20);
    httpStatus int(10);
  END-PI;
  dcl-ds unCLientBE likeds(clientBE);  2
  dcl-s idref char(10);
  clear erreurs;
  monitor;
    // au début était prévu un accès par id OU reference
    idref = %char(id);
    unCLientBE = lireclientLBS(idRef);
    eval-corr client = unCLientBE;
    httpstatus = HTTP_OK;
    codeHttp = HTTP_OK;
  on-error;
```

Merci EVAL-CORR

IBM i : API REST, retour d'expérience

- **Aspects techniques**

A l'origine, une structure de type DTO était exposée
Une structure de type BE (Business Objet) était manipulée ensuite

Ce découpage a été supprimé ensuite, car dans l'approche REST on expose des ressources et non des services (SOAP).

Une ressource a toujours la même interface, les services ont chacun leurs propres interfaces (DTO).

De plus techniquement cela évite du code de mapping (EVAL-CORR) et simplifie le développement.

IBM i : API REST, retour d'expérience

- **Aspects techniques**

L'accès à la base de données est réalisé par SQL

```
dcl-proc  lireClientDAO Export;
  dcl-pi *n likeds(clientBE);
    idRef char(10);
  END-Pi;
  dcl-ds unCLient likeds(clientBE);
  // si numérique (pas de lettres) , une ID
  if isNumeric(idref);
    EXEC SQL
      select id, reference, nom, prenom, date_naissance, adresse,
             statut into :unClient.id, :unClient.reference, :unCLient.nom,
                         :unClient.prenom, :unClient.datenaissance,
                         :unClient.adresse, :unClient.Statut
             from client
             Where id = :idref;
  else;
    // aplha ==> une référence
    EXEC SQL
      select id, reference, nom, prenom, date_naissance, adresse,
             statut into :unClient.id, :unClient.reference, :unCLient.nom,
                         :unClient.prenom, :unClient.datenaissance,
                         :unClient.adresse, :unClient.Statut
             from client
             Where UCASE(reference) = UCASE(:idref);
  endif;
```

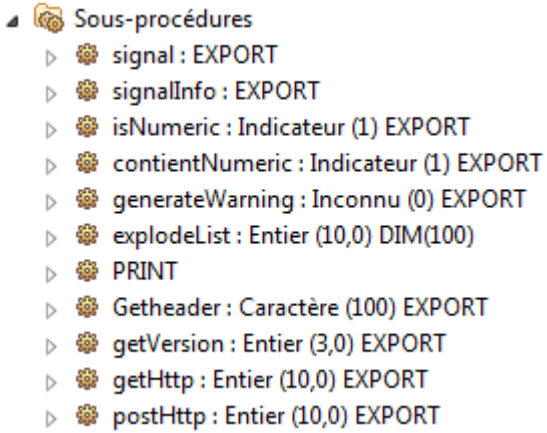
IBM i : API REST, retour d'expérience

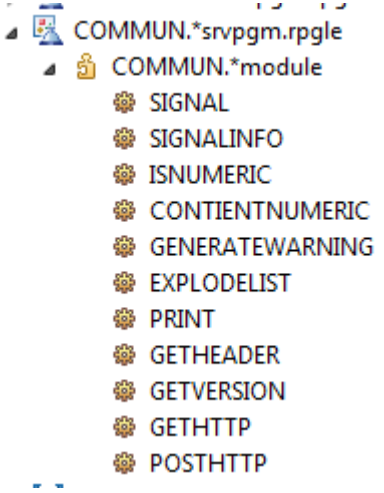
- **Aspects techniques**

ILE et les fonctions RPG sont des outils précieux

```
if isNumeric(idref);
```

Dans le source





l'objet *SRVPGM

IBM i : API REST, retour d'expérience

- **Aspects techniques**

ET rendent le code lisibles par un "*non gapiste*"

```
dcl-proc signal export;
  dcl-pi *n;
    message char(80) CONST;
  END-PI;
  dcl-pr QMHSNDPM EXTPGM;
    msgid char(7)   CONST;
    msgf  char(20)  CONST;
    msgdta char(80) CONST;
    msgdtal int(10) CONST;
    msgtyp char(10) CONST;
    callstack char(10) CONST;
    callcount INT(10) CONST;
    msgkey    CHAR(4);
    erreur    likeds(erreurDS);
  END-PR;
  dcl-s cle char(4);
  erreurds.errlg = 16;
  // envoi un message d'erreur deux crans au dessus (paramètre "callcount")
  QMHSNDPM('CPF9898' : 'QCPFMSG   QSYS': message : %len(%trimr(message)) :
           '*ESCAPE': '*' : 2 : cle : erreurDS);
END-PROC;
```

```
  // problème ?
  if %subst(SQLSTATE : 1 : 2) > '01';
    signal('Client non trouvé');
  else;
    return unCLient;
  ENDIF;
end-proc;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Cela permet une logique "objet"

DAO

```
dcl-proc  lireClientDAO Export;
  dcl-pi *n likeds(clientBE);
    idRef char(10);
  END-Pi;
  dcl-ds unCLient likeds(clientBE);
  // si numérique (pas de lettres) , une ID
  if isNumeric(idref);
    EXEC SQL
      select id, reference, nom, prenom, dat
----- 15 lignes retirées. -----
  // problème ?
  if %subst(SQLSTATE : 1 : 2) > '01';
    signal('Client non trouvé');
  else;
    return unCLient;
  ENDIF;
end-proc;
```

Ici, l'accès à la base

LBS

```
dcl-proc lireCLientLBS export;
  dcl-pi *n likeds(clientBE);
    idREf char(10);
  END-PI;
  dcl-ds unCLient likeds(clientBE);
  monitor;
    unCLient = lireCLientDAO(idRef);
    return unCLient;
  on-error;
    signal('erreur lecture');
  endmon;
END-PROC;
```

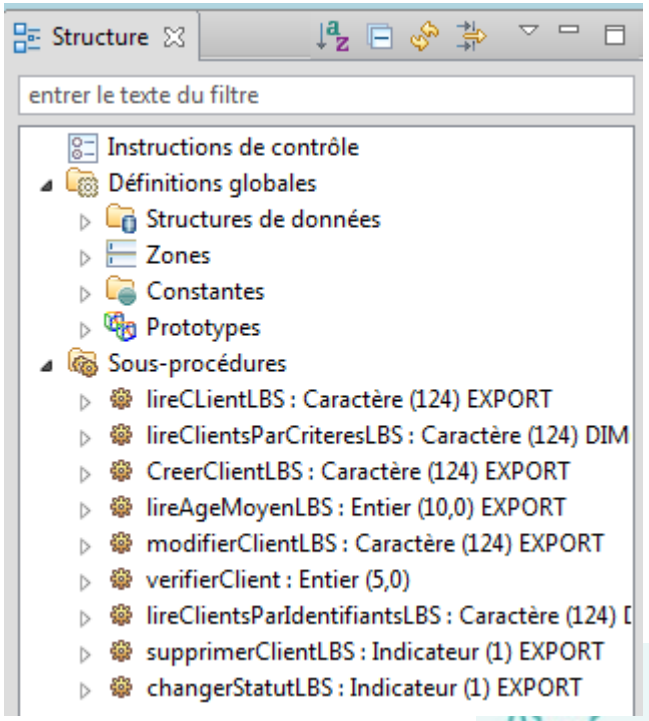
Ici des calculs (règles métier)

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Par exemple, en cas de création, c'est ici que sont fait les contrôles

```
dcl-proc CreerClientLBS export;
  dcl-pi *n likeds(CLientBE);
    unClient likeds(ClientBE);
    desErreurs like(listErreur) DIM(10);
  END-PI;
  dcl-s nbErreur int(5);
  if not isReferenceUniqueDAO(unClient.reference) ;
    desErreurs(1) = generateWarning('id':'CLIENT_REF_004' :
     'La référence existe déjà.');
  else;
    nbErreur = verifierClient(unCLient:desErreurs);
    if nbErreur > 0;
      signal('des erreurs lors de la vérification du client');
    else;
      unCLient.statut = 'ClientNormal';
      Monitor;
        unCLient = creerCLientDAO(unCLient);
      on-error *ALL;
        signal('erreur lors de la création client');
      ENDMON;
    endif;
  endif;
  return unCLient;
END-PROC;
```



IBM i : API REST, retour d'expérience

- **Aspects techniques**

Par exemple, en cas de création, c'est ici que sont fait les contrôles

```
dcl-proc CreerClientLBS export;
  dcl-pi *n likeds(CLientBE);
    unClient likeds(ClientBE);
    desErreurs like(listErreur) DIM(10);
  END-PI;
  dcl-s nbErreur int(5);
  if not isReferenceUniqueDAO(unClient.reference) ;
    desErreurs(1) = generateWarning('id':'CLIENT_REF_004'
     'La référence existe déjà.');
  else;
    nbErreur = verifierClient(unCLient:desErreurs);
    if nbErreur > 0;
      signal('des erreurs lors de la vérification du clier
    else;
      unCLient.statut = 'ClientNormal';
      Monitor;
        unCLient = creerCLientDAO(unCLient);
      on-error *ALL;
        signal('erreur lors de la création client');
      ENDMON;
    endif;
  endif;
  return unCLient;
END-PROC;
```

```
dcl-proc  isReferenceUniqueDAO export;
  dcl-pi *n ind;
    ref char(10);
  END-Pi;
  dcl-s id int(10);
  EXEC SQL
    select id into :id
           from client
           Where reference = :ref;
  // a-t-on lu ?
  if %subst(SQLSTATE : 1 : 2) = '02';
    return *ON;
  else;
    return *OFF;
  ENDIF;
end-proc;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Par exemple, en cas de création, c'est ici que sont fait les contrôles

```
// verification client (procédure interne)
dcl-proc verifierClient;
  dcl-pi *n int(5);
    unClient likeds(ClientBE);
    desErreurs like(listErreur) Dim(10);
  END-PI;
  dcl-s ErreurIDX int(5);
  // contrôles de la référence
  if unClient.reference = *Blanks;
    erreurIDX += 1;
    desErreurs(erreurIDX) = generateWarning('id':'CLIENT_REF_001' :
      'La référence du client est obligatoire');
  else;
    if %len(%trim(unClient.reference)) > 10;
      erreurIDX += 1;
      desErreurs(erreurIDX) = generateWarning('id':'CLIENT_REF_002' :
        'La référence du client ne doit etre > a 10 c');
    endif;
    if %subst(unClient.reference:1:3) <> 'CLT';
      erreurIDX += 1;
      desErreurs(erreurIDX) = generateWarning('id':'CLIENT_REF_003' :
        'La référence du client doit commencer par CLT.');
    endif;
  endif;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Critère multiples

On prévoit de faire une recherche client par nom et/ou prénom

☑	▼ LIRECLIENTSPARCRITERES		
	nom	input	char
	prenom	input	char
	clients_LENGTH	output	int
	clients	output	struct
	httpStatus	output	int

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Critère multiples transmis en paramètres *?nom=xxx&prenom=yyy*

Procedure name: LIRECLIENTSPARCRITERES
URI path template for resource: /
HTTP request method: GET
URI path template for method: *NONE or...
HTTP response code output parameter: httpStatus
HTTP header array output parameter: *NONE
Allowed input media types: *ALL or...
Returned output media types: *JSON or...
Whether to wrap input parameters:
- Wrap input parameters
- Do not wrap input parameters

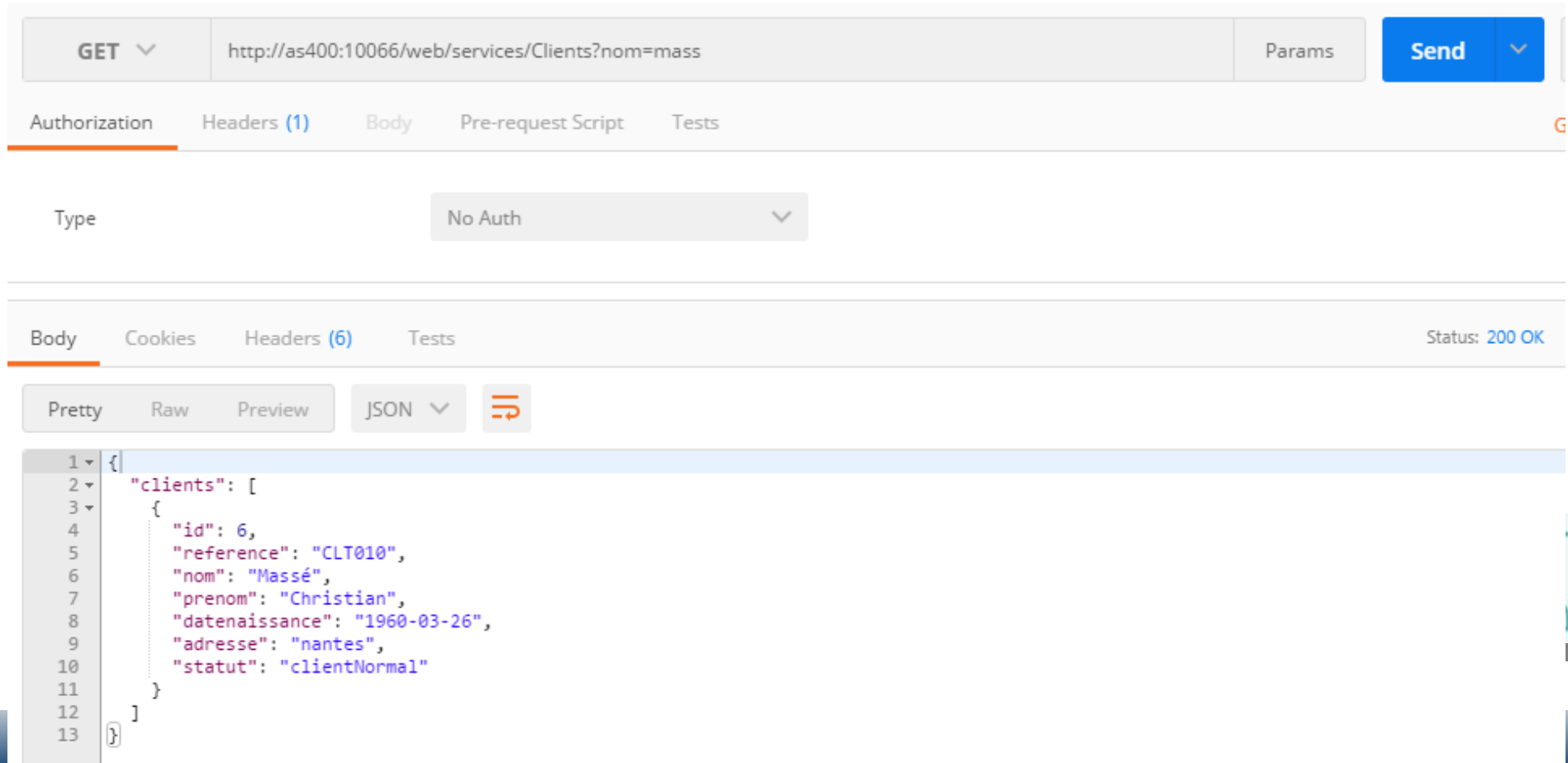
Input parameter mappings:

Parameter name	Data type	Input source	Identifier	Default Value
nom	char	*QUERY_PARAM	nom	*NONE or...
prenom	char	*QUERY_PARAM	prenom	*NONE or...

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Critère multiples transmis en paramètres *?nom=xxx&prenom=yyy*



IBM i : API REST, retour d'expérience

- **Aspects techniques**

Nous avons utilisé SQL dynamique

```
dcl-proc  lireClientsParCriteresDAO Export;
  dcl-pi *n likeds(clientBE) dim(100);
    nom char(15);
    prenom char(15);
  END-Pi;

    requete =
        'select id, reference, nom, prenom, date_naissance, adresse, statut from client';
    if nom <> vide;
      requete += ' Where ucase(nom) like UCASE(''%' + %trim(nom) + '%'')';
      where = *on;
    ENDIF;
    if prenom <> vide;
      if not where;
        requete += ' Where';
      else;
        requete += ' and';
      endif;
      requete += ' ucase(prenom) like UCASE(''%' + prenom + '%'')';
    ENDIF;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Nous avons utilisé SQL dynamique

```
EXEC SQL PREPARE P1 from :requete;
EXEC SQL DECLARE C1 CURSOR FOR P1;
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 into :wid, :wreference, :wnom, :wprenom,
                       :wdatenaissance, :wadresse, :wstatut;
// problème ?
if %subst(SQLSTATE : 1 : 2) > '02';
  signal('Impossible de lire clients(' + SQLSTATE + ')');
endif;
// OK
```

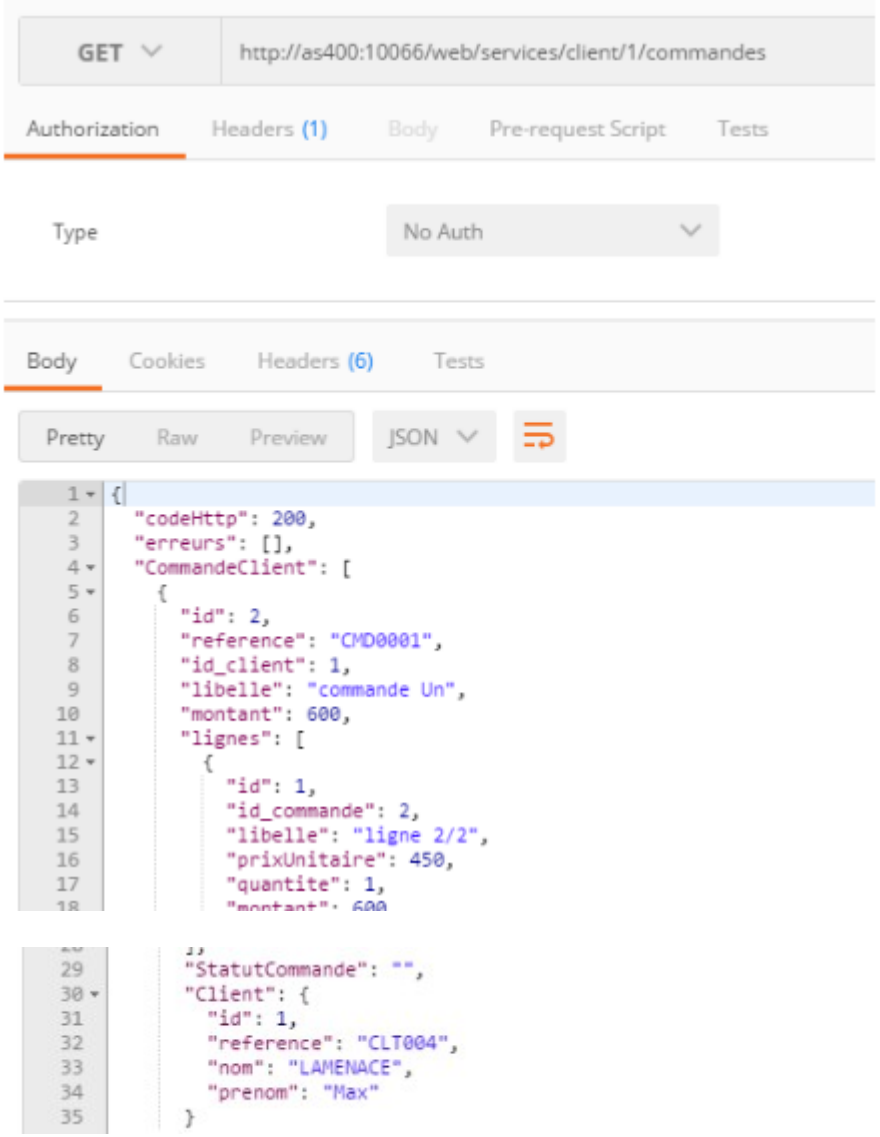
```
dow %subst(SQLSTATE:1:2) < '02';
  i +=1;
  if i> %elem(desCLients);
    leave;
  ENDIF;
  desCLients(i).id = wid;
  desCLients(i).reference = wreference;
  desCLients(i).nom = wnom;
  desCLients(i).prenom = wprenom;
  desCLients(i).datenaissance = wdatenaissance;
  desCLients(i).adresse = wadresse;
  desCLients(i).statut = wstatut;
  EXEC SQL FETCH C1 into :wid, :wreference, :wnom, :wprenom,
                         :wdatenaissance, :wadresse, :wstatut;
ENDDO;
EXEC SQL CLOSE C1;
return desCLients;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Dans commande, nous devons accéder aux commandes ET aux informations client

(ici les commandes du client 1)



suivit des infos client

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Dans "l'objet" commande, nous devons accéder aux informations client puis commandes (5 maxi)

```
clear CommandeClient;
//lire le client (XBS)
unClient = lireClientcdeLBS(id);
//puis, lire la liste des commandes d'un client
desCommandeBE = lireCommandesParCriteresLBS(vide : vide : vide : %char(id));
eval-corr CommandeCLient = descommandeBE;
// récupération nbr de commandes renseignées
CommandeClient_LENGTH = %lookup(0 : CommandeClient(*).id) -1;
for i = 1 to CommandeClient_LENGTH;
// manque les informations CLient
  eval-corr CommandeClient(i).Client = unCLient;
  // et le nbr de lignes de cette commande
  L =  %lookup(0 : CommandeClient(i).lignes(*).id);
  // toutes les occurences sont remplies => 5 lignes
  if L = 0;
    CommandeClient(i).lignes_LENGTH = 5;
  else;
    CommandeClient(i).lignes_LENGTH = L-1;
  ENDIF;
ENDFOR;
```

!

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Dans "l'objet" commande, nous devons accéder aux informations client en consommant un service.

```
dcl-proc lireclientcdeLBS export;
  dcl-pi *n likeds(ClientXBE);
    id int(10);
  end-pi;
 dcl-ds unCLient likeds(clientXBE);
 dcl-s nbrAnnees int(3);
 Monitor;
   unCLient = lireclientcdeXBS(id);
   nbrAnnees = %diff(%date() : unCLient.datenaissance : *YEARS);
   if nbrAnnees >= 18;
     unCLient.isMajeur = *on;
   ENDIF;
 on-error;
   signal('erreur lecture service client');
 ENDMON;
 return unCLient;
end-proc;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Dans "l'objet" commande, nous devons accéder aux informations client (fonction getHttp)

```
dcl-proc lireclientcdeXBS export;
  dcl-pi *n likeds(CLientXBE);
    id int(10);
  end-pi;
  dcl-ds unCLient likeds(clientXBE);
----- 7 lignes retirées. -----
  //
  clear unClient;
 // lecture du flux JSON
 url = serveurweb + 'clients/' + %char(id);
 status = getHttp(url:data);

 if status < 0
   or
   status > 400;
     signal('erreur lecture service client');
 ENDIF;
```

!

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Nous avons utilisé directement les API AXIS

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/dW%20IBM%20Integrated%20Web%20Services%20for%20i/page/Send%20user-defined%20payloads%20using%20new%20client%20APIs

```
dcl-proc getHttp export;
  dcl-pi *n int(10);
    url     char(200) CONST;
    data varchar(32767) ;
  END-PI;

  /COPY /QIBM/ProdData/OS/WebServices/V1/client/include/Axis.rpgleinc

----- 13 lignes retirées. -----

  // Create HTTP transport handle.
  tHandle = axiscTransportCreate(url:AXISC_PROTOCOL_HTTP11);
  if (tHandle = *NULL);
    return -1;
  endif;
  // méthode GET
  propBuf = 'GET' + X'00';
  axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD: %addr(propBuf));

  // Exécution, lecture du flux.
  clear response;
  clear header;

  rc = axiscTransportFlush(tHandle);
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Puis parsé le JSON à l'aide du projet YAJL

```
// traitement du flux JSON  (il est VARCHAR et YAJL attend du char )
 docnode = yajl_buf_load_tree(%addr(data)+2:%len(%trimr(data)):errmsg);
 if docnode <> *NULL;
   node =  YAJL_OBJECT_FIND(docNode: 'client');
   if node <> *NULL;
    val =   YAJL_OBJECT_FIND(node: 'id');
   endif;
   unCLient.id =   YAJL_GET_NUMBER(val);
   if node <> *NULL;
    val =   YAJL_OBJECT_FIND(node: 'reference');
   endif;
   unCLient.reference=  YAJL_GET_STRING(val);
   if node <> *NULL;
    val =   YAJL_OBJECT_FIND(node: 'nom');
   endif;
   unCLient.nom =  YAJL_GET_STRING(val);
   if node <> *NULL;
    val =   YAJL_OBJECT_FIND(node: 'prenom');
   endif;
   unCLient.prenom =  YAJL_GET_STRING(val);
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

Aujourd'hui, nous reverrions probablement notre copie :

- **HTTPGETCLOB pour aller chercher le flux**
  utilise une JVM (lourd en mémoire)
  mais, en java, on peut indiquer un proxy pour sortir

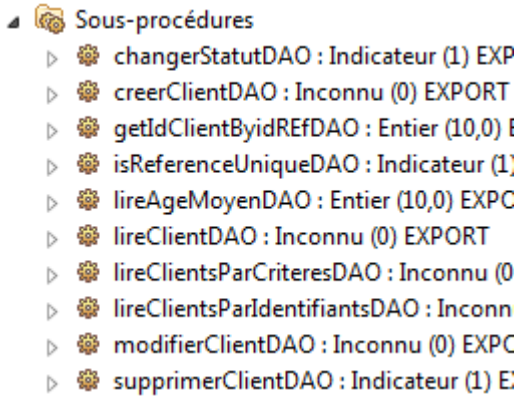
  - **JSON_TABLE pour parser (c'est livré depuis TR1/TR5 seulement)**

  - **A voir … suivant les temps de réponse constatés.**

IBM i : API REST, retour d'expérience

- **Aspects techniques**

A titre personnel, je ne sais pas comment j'aurais fait sans RDI



- **Fenêtre structure**

- **Complétion de code**

  quand une zone se nomme

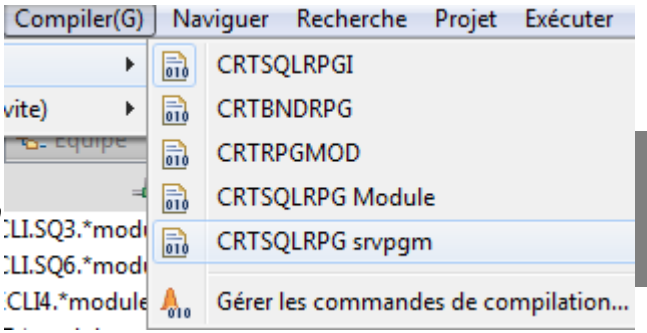
  CommandeClient(i).Client

  quand une procédure se nomme

  lireCommandesParCriteresLBS

  - ***les commandes de compil. personn***

  etc, etc...

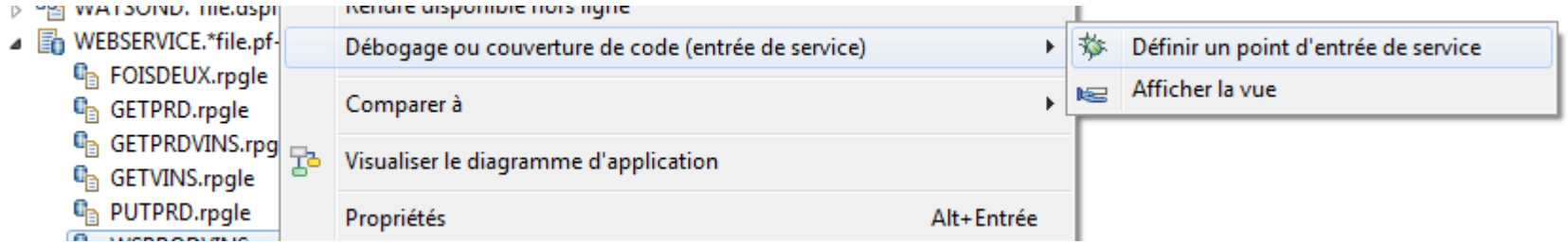


IBM i : API REST, retour d'expérience

- **Aspects techniques**

A titre personnel, je ne sais pas comment j'aurais fait sans RDI

- **Particulièrement pour debugger**



!

IBM i : API REST, retour d'expérience

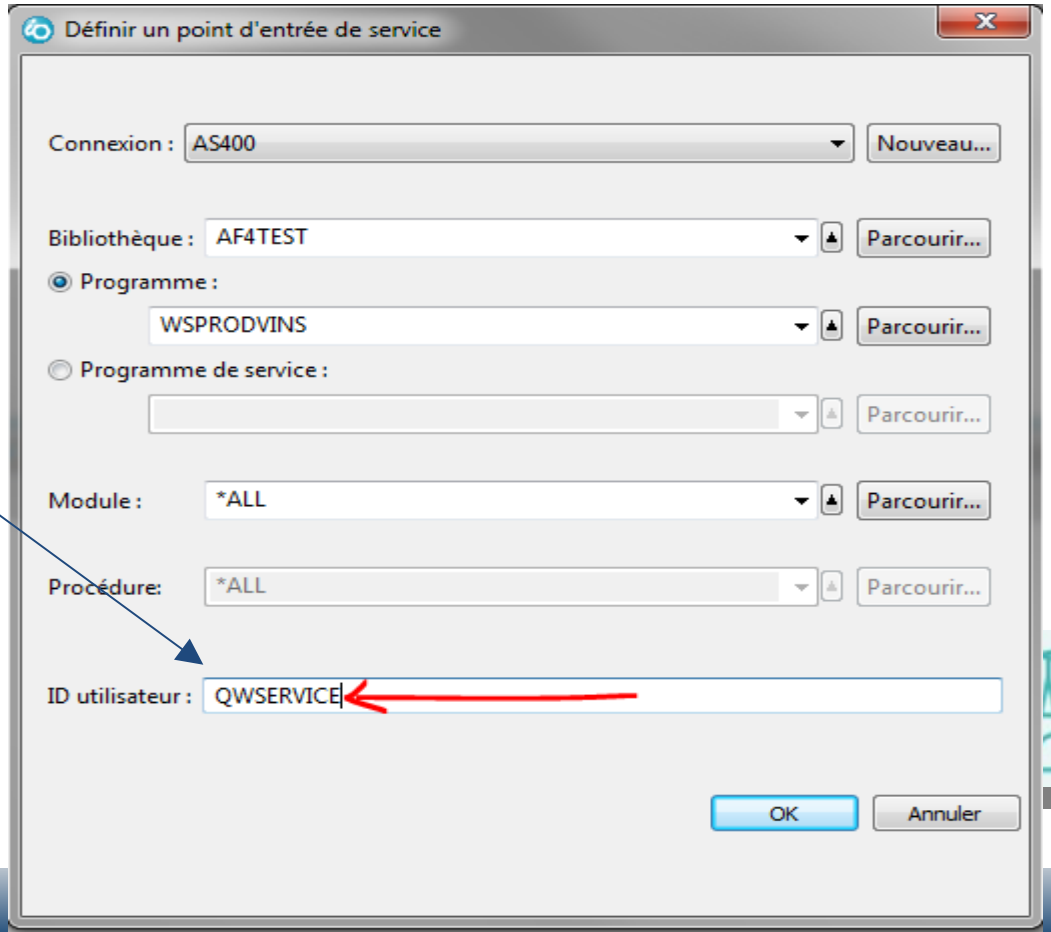
- **Aspects techniques**

A titre personnel, je ne sais pas comment j'aurais fait sans RDI

- **Particulièrement pour debugger**

Indiquer ici le profil indiqué pour CE service

le debug se lancera automatiquement

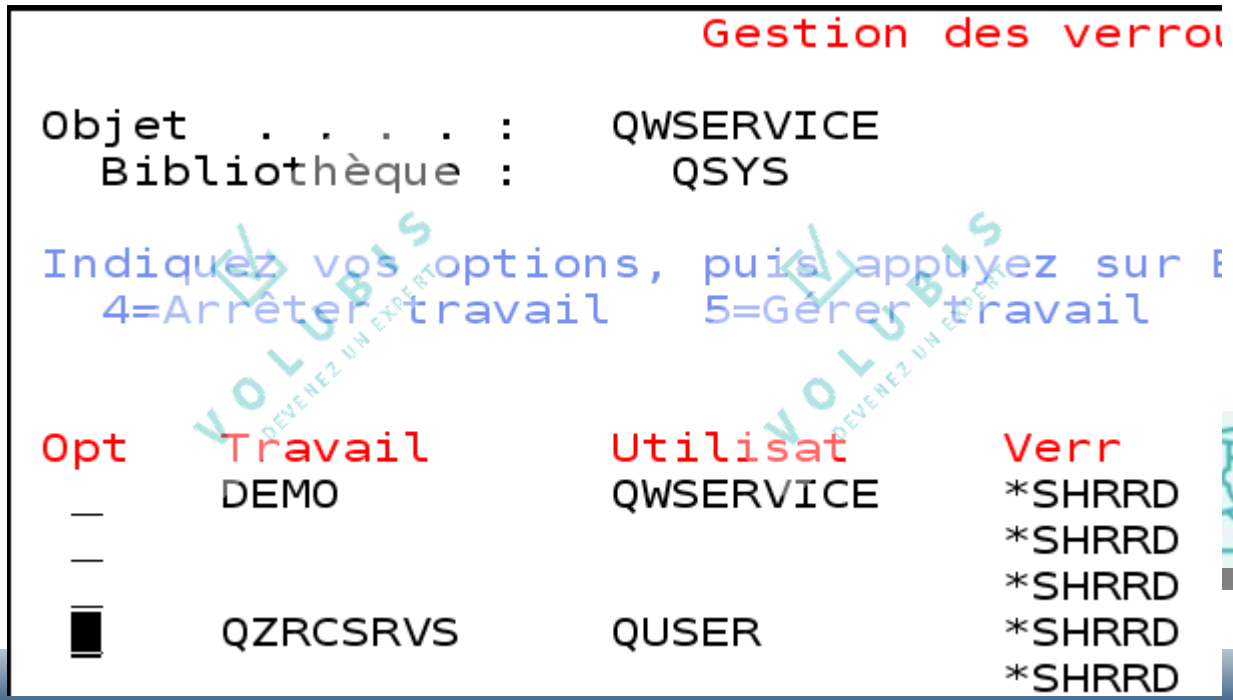


IBM i : API REST, retour d'expérience

- **Aspects techniques**

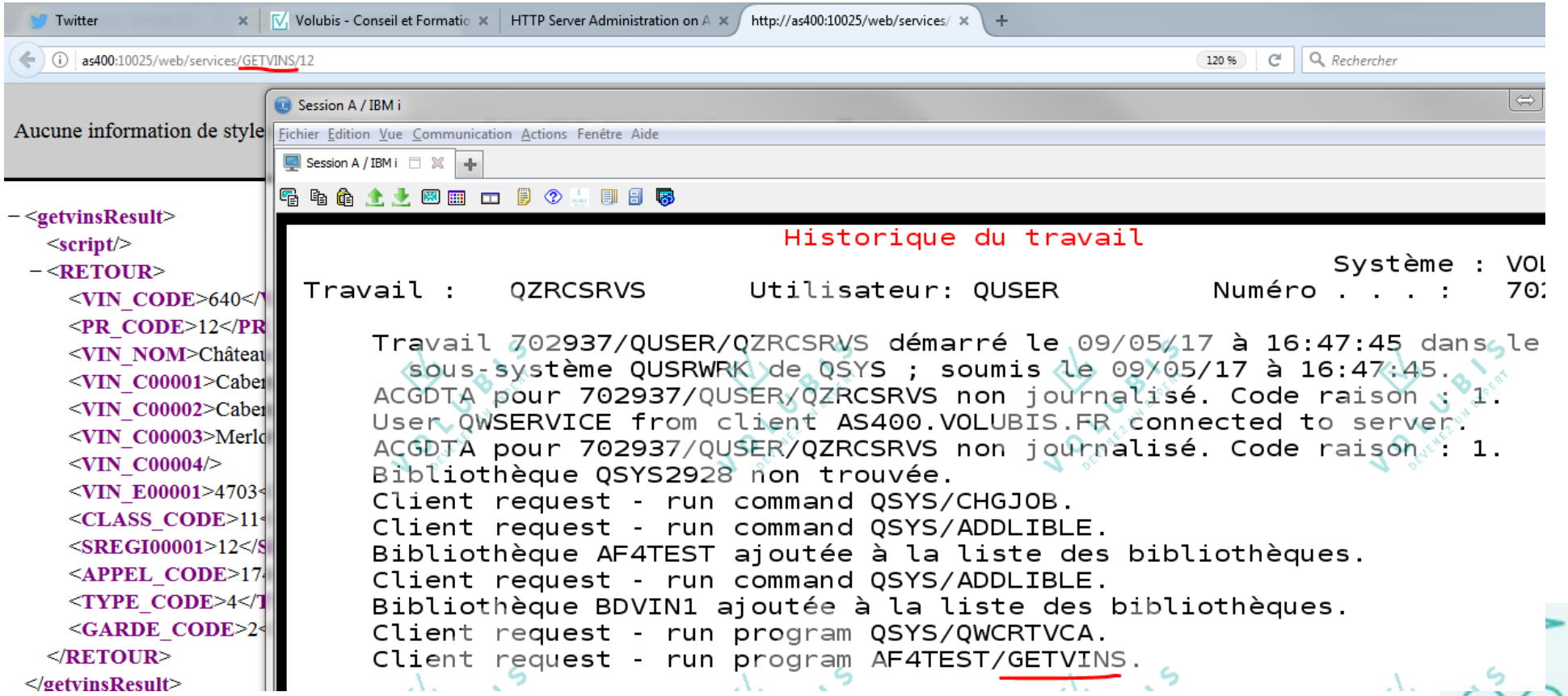
Sauf si le programme ne se lance pas
(bibliothèque manquante, *SRVPGM dépendant non trouvé, ...)

il faut alors faire un WRKOBJLCK du profil et aller voir la JOBLOG du job QZRCSRVS correspondant



IBM i : API REST, retour d'expérience

- **Aspects techniques**



IBM i : API REST, retour d'expérience

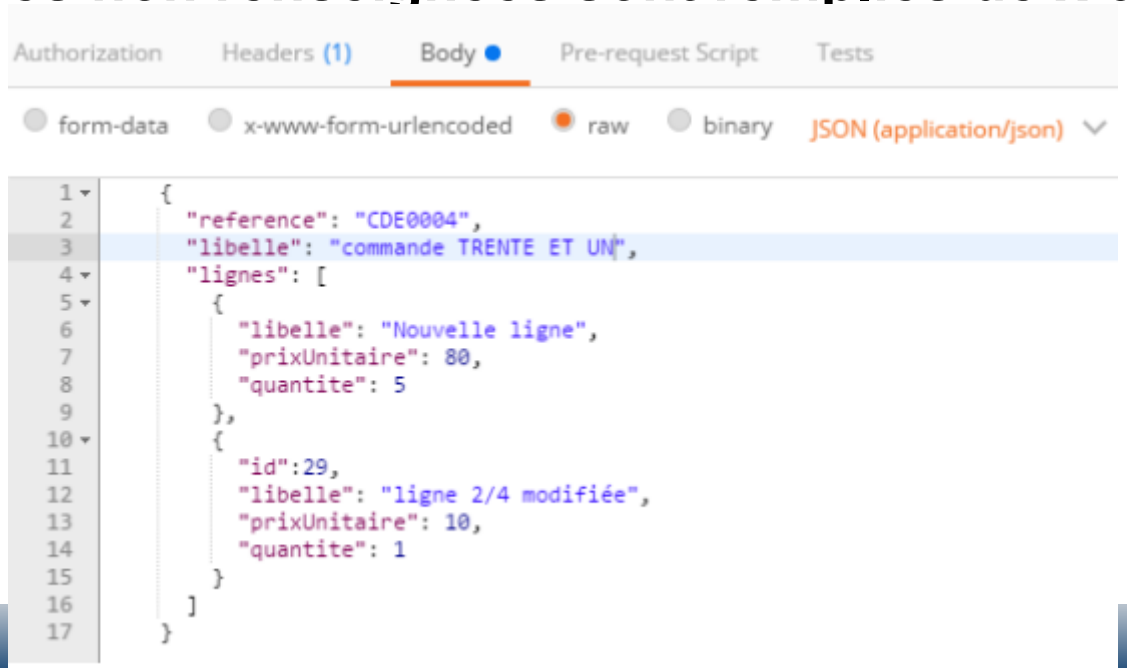
- **Aspects techniques**

les problèmes

- **Pour les structures à occurrence, en entrée (dans le body)**

Les occurrences non renseignées sont remplies de x'00' !

sous PostMan



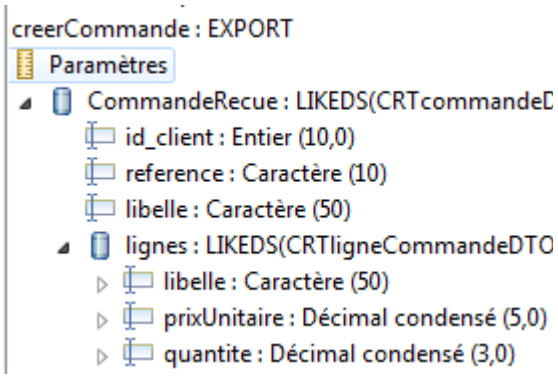
IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **Pour les structures à occurrence, en entrée (dans le body)**

IL a fallu "nettoyer" les data !




```
// les élements non fournis sont rempli à x'00'
// cela fait planter eval-corr
for i = 1 to %elem(CommandeRecue.lignes);
   if  CommandeRecue.lignes(i).libelle  = *allx'00';
       // mise au propre
       CommandeRecue.lignes(i).prixUnitaire = 0;
       CommandeRecue.lignes(i).quantite = 0;
   ENDIF;
ENDFOR;
eval-corr uneCommandeBE = commandeRecue;
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **Nous pouvons retourner un status et des entêtes**
  - **les entêtes fonctionnent (ici l'URL vers le client créé par "location:")**

attention
un tableau
CHAR(200)
DIM(x)

```
dcl-proc creerClient export;
  dcl-pi *n;
    ClientRecu likeds(crtCLientDTO);
    client likeds(clientDTO);
    codeHttp like(JSENDcodeHttp);
    erreurs_LENGTH like(JSENDerreurs_length);
    erreurs likeds(JSENDerreurs) DIM(20);
    HTTPstatus int(10);
    Httpheaders like(httpheader) dim(10);
  END-PI;
----- 6 lignes retirées. -----
    unCLientBE = creerclientLBS(unClientBE:desErreurs);
    eval-corr client = unCLientBE;
    httpstatus = HTTP_CREATED;
    codeHTTP = HTTP_CREATED;
    httpheaders(1) = 'location: ' + url + '/' + %char(client.ID);
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **Pour le status : *impossible* de retourner un code erreur 4xx/5xx *et* des données**

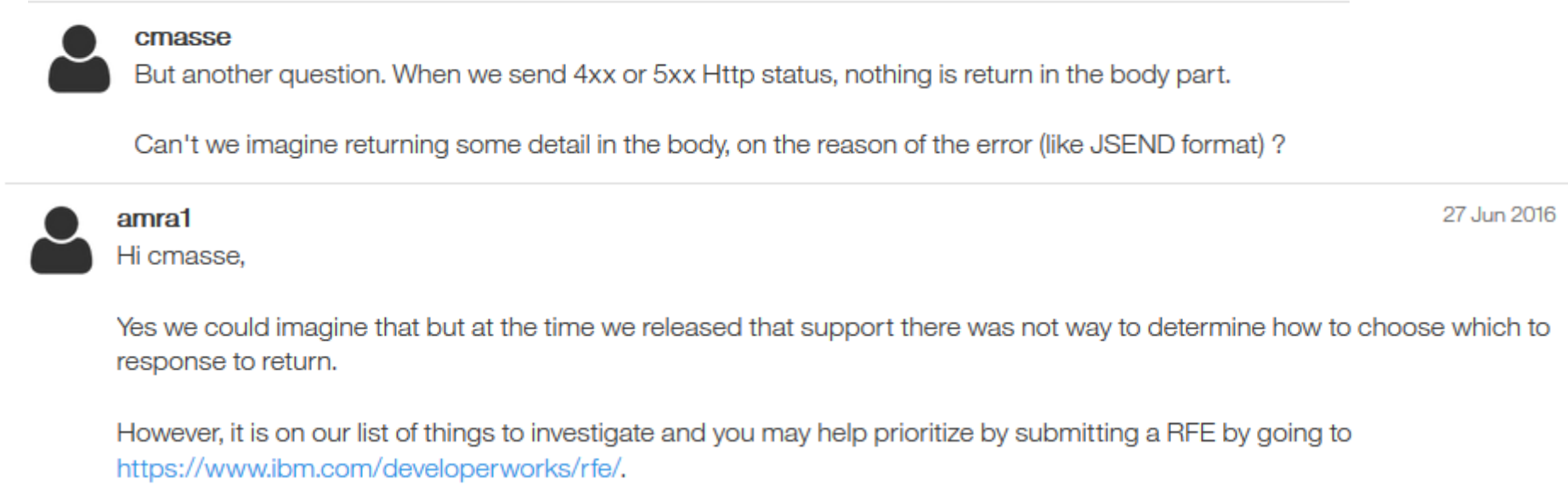
Or dans le cas d'une création par exemple, nous souhaitons pouvoir retourner une erreur ET les raisons de l'erreur (un code, un message par exemple). A aujourd'hui cela n'est pas possible, aucune données n'est retournée par la couche Java avec un status à 404 ou 500 par exemple.

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **Ce problème est connu**



cmasse

But another question. When we send 4xx or 5xx Http status, nothing is return in the body part.

Can't we imagine returning some detail in the body, on the reason of the error (like JSEND format) ?

amra1 27 Jun 2016

Hi cmasse,

Yes we could imagine that but at the time we released that support there was not way to determine how to choose which to response to return.

However, it is on our list of things to investigate and you may help prioritize by submitting a RFE by going to https://www.ibm.com/developerworks/rfe/.

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **JSEND**

OmniTI § Labs

Login | Settings | Help/Guide | About Trac

JSend

- **What?** - Put simply, JSend is a specification that lays down some rules for how JSON responses from web servers should be formatted. JSend focuses on application-level (as opposed to protocol- or transport-level) messaging which makes it ideal for use in REST-style applications and APIs.

POST /posts.json (with data body: "Trying to creating a blog post"):

```
{
    "status" : "fail",
    "data" : { "title" : "A title is required" }
}
```

!

GET /posts.json:

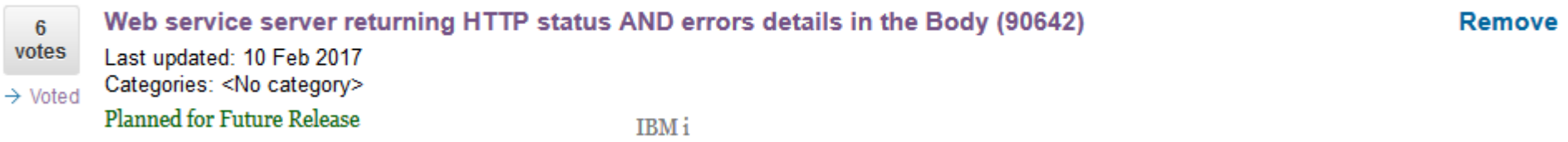
```
{
    "status" : "error",
    "message" : "Unable to communicate with database"
}
```

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

- **UNE RFE (Request for Evolution) a été créé**



Merci de voter :

https://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=90642

IBM i : API REST, retour d'expérience

•Aspects techniques

les problèmes

•PCML !

Ce langage est généré par le compilateur (mot-clé PGMINFO) et utilisé par la toolbox java pour faire le CALL, il est limité :

- **A 32 paramètres pour un *PGM**
- **A 7 pour un *SRVPGM *(sans rire ?)***
- **La gestion des dates et des timestamp n'est possible que depuis pcml6 (7.2 uniquement)**
- **Toujours pas de VARCHAR, pas d'heures !**

IBM i : API REST, retour d'expérience

- **Aspects techniques**

les problèmes

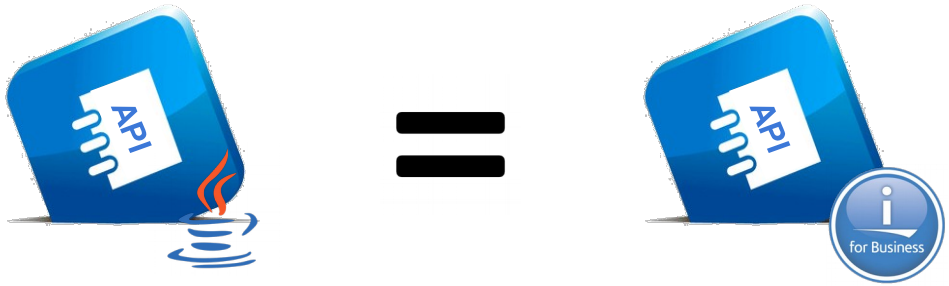
- **<u>Vu du GIE IRIS</u>**

!

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées <u>Vu du GIE IRIS</u>**

  Elles sont essentiellement dues au fait que nous voulions exactement le même fonctionnement et paramétrage pour les API JEE et IBM i



IBM i : API REST, retour d'expérience

- **Difficultés rencontrées**

et certainement aussi par manque de connaissances.

on y est allé car on sentait que c'était faisable avec l'IBM i

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *serveur Liberty***

Robustesse de la solution liberty ? ***peu d'infos***

Combien de serveurs créés sur la machine sans impact sur les performances ? cluster nécessaire ?

1 par API, 1 par application, 1 par serveur ...

Nos choix :

- ☑ **1 par API en JEE (donc 1 serveur tomcat par web service)**
- ☑ **2 par domaine fonctionnel soit max 30 liberty / partitions.**

IBM i : API REST, retour d'expérience

<context-root>

- **Difficultés rencontrées => *serveur Liberty***

```
http://<host>:<port>/<context-root>/<root-resource>/<uri-path-template>
```

When an integrated web services server is created, the default context root for all services deployed in the integrated web services server is `/web/services`. The context root for the server can be changed by modifying the server's properties.

context root = *"api-nomAPI-v1"*

Pas possible d'avoir la même chose car notre choix d'architecture ne nous permet pas d'avoir un context root différent pour chaque API. Le context root est lié au serveur donc partagé entre toutes les APIs.

Nos choix : **++ "à blanc" (possible mais seulement en modifiant les fichiers de conf, impact ?)**
+ "/api" (modification ok via la console mais ne colle pas exactement au JEE)

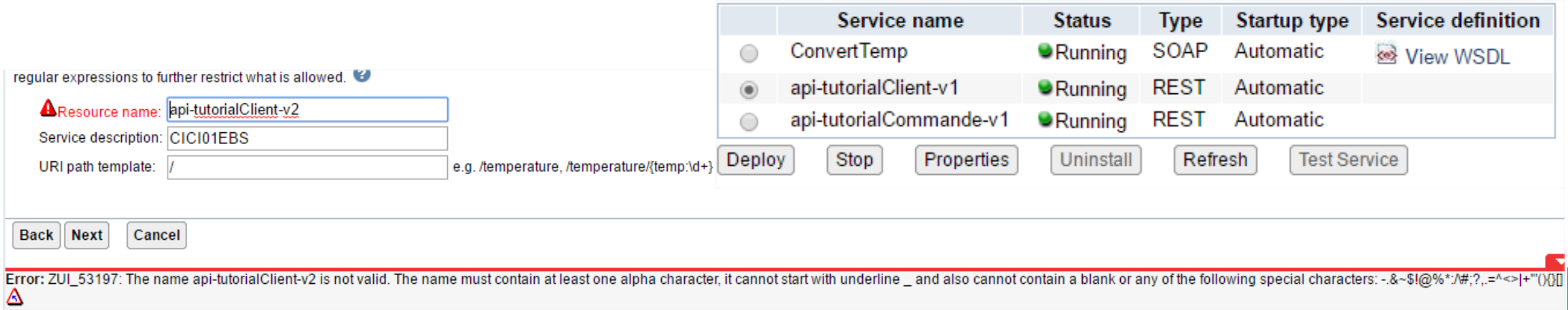
IBM i : API REST, retour d'expérience

Exploitabilité

- **Difficultés rencontrées => *serveur Liberty***

L'interface web bloque des choix de paramétrage que le script de déploiement accepte.

Dans notre cas , les "-" dans le nom du service *(Ressource name qui apparaît dans l'url)*
Pour coller au JEE, nos services sont sous la forme *"api-nomAppli-v1"*



Impact de forcer les "-" via les scripts ?

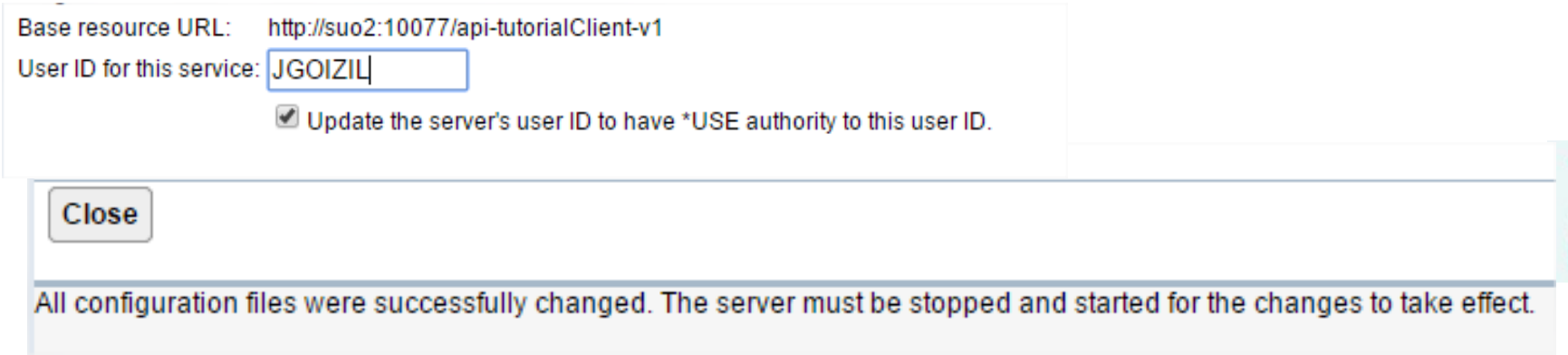
IBM i : API REST, retour d'expérience

Exploitabilité

- **Difficultés rencontrées => *serveur Liberty***

pour changer le paramétrage d'un service, il faut parfois le supprimer puis le recréer
=> *modification de l'existant impossible*
=> *pas d'annule et remplace automatique (même forcé)*

pour qu'un changement de properties sur un service soit pris en compte, il faut arrêter le *service* puis le redémarrer
=> le message parle de *server* (défaut de langage ?)



IBM i : API REST, retour d'expérience

Exploitabilité

- **Difficultés rencontrées => *serveur Liberty***

⛔ **pas facile de lire les logs et de retrouver clairement ce qui est lié à l'anomalie (si erreur avant de lancer le programme)**

```
java.lang.RuntimeException: Invocation of program failed.
AS400Message (ID: CEE9901 text: Application error.  CPF3CF2 unmonitored by QZRUCLSP at statement 0000000065, instruction X'0000'.):com.ibm.
```

il faut retrouver le job IBM i lié

```
     QZRCSRVS   QSECOFR    PJ         0,0                    TIMW
     QZRCSRVS   QSECOFR    PJ         0,0                    TIMW
5    QZRCSRVS   JGOIZIL    PJ         0,0                    TIMA
     QZSCSRVS   CNXWEB     PJ         0,0                    TIMW
     QZSCSRVS   CNXWEB     PJ         0,0                    TIMW
```

```
Client request - run program QSYS/QZRUCLSP.
Impossible d'adresser objet CICI01LBS type et sous-type X'0203' droit
  X'0000'.
Error(s) occurred during running of QleActBndPgmLong API.
Application error.  CPF3CF2 unmonitored by QZRUCLSP at statement
  0000000065, instruction X'0000'.
```

pour analyser la vraie erreur :
ici problème de bibliothèques

IBM i : API REST, retour d'expérience

gestion d'Erreurs

- **Difficultés rencontrées => *serveur Liberty***

un framework permet de renvoyer :

- **soit le flux de sortie standard**
- **soit un flux d'erreur formaté comme on le souhaite *(plus propre pour le client)***

standard

```
{
  "id": 5,
  "reference": "CLT005",
  "nom": "MINEUR_DEUX",
  "prenom": "felix",
  "dateNaissance": "2015-01-01",
  "adresse": "Nantes",
  "statutClient": "CLIENT_NORMAL"
}
```

erreur

```
{
  "retourCodeHttp": 404,
  "retourErreurs": [
    {
      "champErreur": "",
      "codeErreur": "entite.non.trouvee",
      "libelleErreur": "L'entité <99> n'existe pas."
    }
  ]
}
```

IBM i : API REST, retour d'expérience

gestion d'Erreurs

- **Difficultés rencontrées => *serveur Liberty***

le framework qui constitue le flux de retour est intégré au liberty, il renvoi :

- **soit le flux standard (httpStatus < 400)**
- **soit aucun flux si > 400**

standard

```
{
  "client": {
    "id": 5,
    "reference": "CLT005",
    "nom": "MINEUR_DEUX",
    "prenom": "felix",
    "dateNaissance": "2015-01-01",
    "adresse": "Nantes",
    "statutClient": "CLIENT_NORMAL"
  },
  "erreurs": {
    "codeHttp": 200,
    "erreurs": []
  }
}
```

Status: 200 OK

erreur

Status: 404 Not Found

IBM i : API REST, retour d'expérience

gestion d'Erreurs

- **Difficultés rencontrées => *serveur Liberty***

Solution

gérer des erreurs applicatives = structure d'erreur ajoutée au flux de retour standard

- **on force un code httpStatus à 303 (permet d'avoir un flux en retour)**
- **l'erreur applicative est gérée dans une structure spécifique du flux.**

erreur

```
{
  "client": {
    "id": 0,
    "reference": "",
    "nom": "",
    "prenom": "",
    "dateNaissance": "0001-01-01",
    "adresse": "",
    "statutClient": ""
  },
  "erreurs": {
    "codeHttp": 404,
    "erreurs": [
      {
        "champErreur": "",
        "codeErreur": "entite.non.trouvee",
        "libelleErreur": "L'entité <99> n'existe pas."
      }
    ]
  }
}
```

httpStatus

Status: 303 See Other

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *développement***

Limitation du nombre de paramètres pour un *SRVPGM

Erreur sur script d'install : IWS00244E - Number of parameters in program or procedure exceeds the maximum allowed.

```
000360   dcl-pi *n;
000361     param1 int(10);
000362     param2 char(20);
000364     structureIn likeds(commandeBE);
000365     structureRetour likeds(commandeBE);
000366     erreurs likeds(apiErreurs);
000367     httpStatus int(10);
000368     httpHeaders like(httpHeader) dim(10);
000369   end-pi;
```

Restriction liées à PCML

le nombre de paramètres est limité à :

- **7 pour un *SRVPGM**
- **32 pour un *PGM**

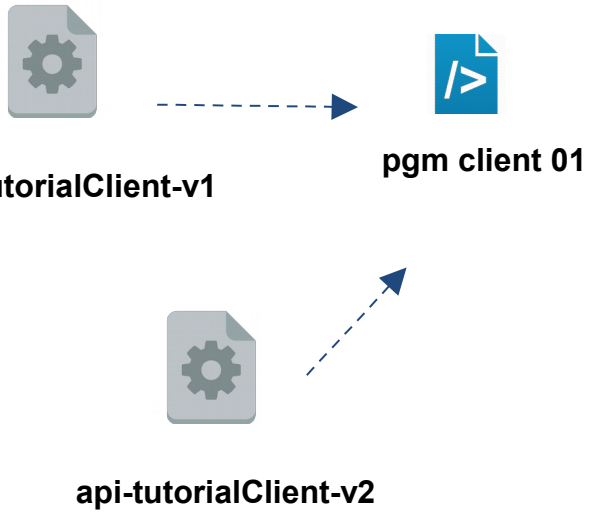
Peu de marge sachant qu'il y a des paramètres pour la gestion d'erreur, le httpStatus, le httpHeader et éventuellement un param_LENGTH pour gérer les occurrences d'une DS.

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *développement***

Comment gérer le versionning des services sur l'IBM i ?

Il faut pouvoir gérer une compatibilité lors de l'évolution d'une API (permet aux consommateurs de migrer à leur rythme)

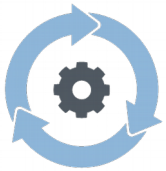


On duplique tout le code ?
On gère la compatibilité dans le v2 ?

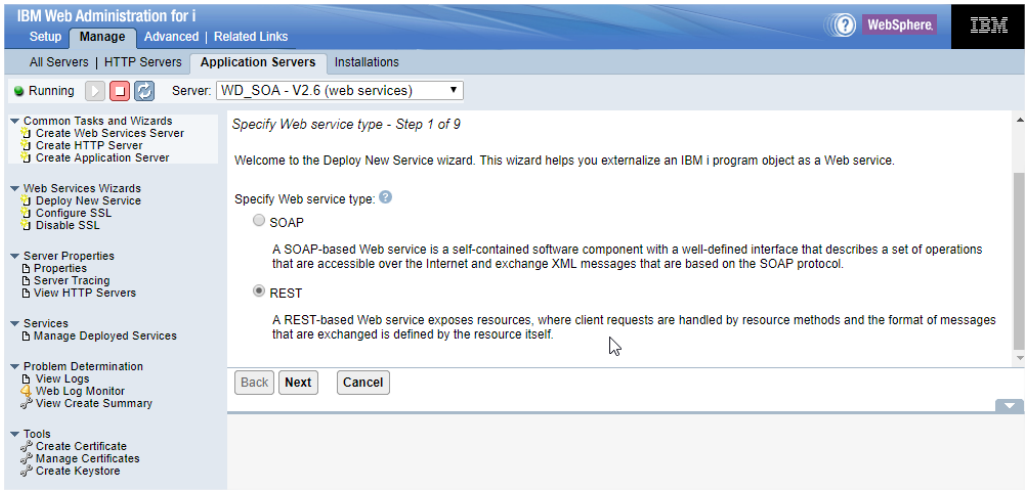
Réflexion en cours ...

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *déploiement***



2 méthodes pour déployer un service :

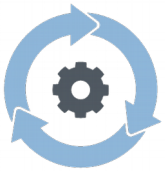


via la console HTTPAdmin

via les scripts sous QSH

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *déploiement***



Console HTTPAdmin :

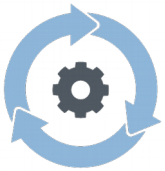
- fait le job, permet de créer un web service
- interface lourde
- droits nécessaires (délégation possible)
- manuel

Script installWebService.sh :

- fait le job, permet de créer un web service
- création rapide et automatisable
- tous les paramètres ne sont pas disponibles
- pas industrialisé (il faut créer le fichier properties et définir la commande de création)

IBM i : API REST, retour d'expérience

- **Difficultés rencontrées => *déploiement***



Outillage en cours ...

Création automatique du fichier de properties à partir du fichier source

Génération automatique de la commande de création du service

Exécution automatique du script d'installation sur les serveurs liberty

✓ **But : plus d'actions manuelles pour les intégrateurs**

IBM i : API REST, retour d'expérience

Bilan

- **Difficultés rencontrées**

- ✓ **on arrive à exposer des programmes RPG en tant qu'API REST JSON**
- ✓ **on arrive à avoir une architecture similaire JEE ⇔ IBM i**
- ✓ **IBM apporte des évolutions à chaque TR sur les API REST**

- ✗ **on est pas encore aussi souple qu'en JEE**
- ✗ **on a pas autant de framework qu'en JEE**

on compte sur IBM

En Production

API de gestion des restes à quais (volume non expédié en entrepôts)
7 serveurs Liberty (1 par entrepôt).

En Recette

API de récupération de données magasins suite à notification de modification (Queue, EAI).

En Développement

API de pilotage et d'ordonnancement de la préparation sous Android.
Une dizaine de services. (2*25 serveurs Liberty en cluster)

- Industrialisation plus poussée
- Une API Gateway interne
- Des MicroServices (~API)
  - self-contained
  - contenairisé
  - auto scalable
  - technologies multiples
- WOA (Web Oriented Architecture)
  - Marketing
    - des APIs pour les internautes ou des [illegible]ecialisés
    - monétisation