
Web Service

REST

Table des matières

I - Présentation des WS REST.....	6
1. Deux grands types de WS (REST et SOAP).....	6
1.1. Caractéristiques clefs des web-services SOAP / Xml.....	7
1.2. Caractéristiques clefs des web-services "REST" / "HTTP".....	8
2. Web Services "R.E.S.T.".....	9
2.1. Statuts HTTP (code d'erreur ou ...).....	13
2.2. Variantes classiques.....	14
2.3. Safe and idempotent REST API.....	15
II - Technologies pour WS REST / vue d'ensemble.....	16
1. Quelques API et technologies "web services REST".....	16
1.1. Vue d'ensemble sur technologies "REST" :.....	16
1.2. W.S. REST en java avec JAX-RS ou Spring-MVC.....	17
1.3. W.S. REST en javascript avec nodeJs+Express.....	20

III - Test de Web Services "REST".....	21
1. Test de W.S. REST via Postman.....	21
1.1. paramétrages "postman" pour une requête en mode "post".....	21
1.2. Exemple de réponses précises reçues et affichées par "postman".....	22
2. Test de W.S. REST via curl.....	24
IV - Appels de WS REST en javascript , CORS.....	26
1. Appels de WS REST (HTTP) depuis js/ajax.....	26
1.1. Cadre des appels.....	26
1.2. XHR (XmlHttpRequest) dans tout navigateur récent.....	27
1.3. Appel ajax via jQuery.....	30
1.4. Api fetch.....	32
1.5. Appel ajax via RxJs (api réactive).....	33
2. Limitations Ajax sans CORS.....	33
3. CORS (Cross Origin Resource Sharing).....	34
V - WS REST via nodeJs et express.....	37
1. Ecosystème node+npm.....	37
2. Express.....	37
3. Exemple élémentaire "node+express".....	38
4. Installation de node et npm.....	39
5. Configuration et utilisation de npm.....	40
5.1. Initialisation d'un nouveau projet.....	40
5.2. installation de nouveau package en ligne de commande :.....	40
5.3. Installation en mode global (-g).....	41
6. Utilisation basique de node.....	41
7. Scripts dans package.json.....	42
8. Publication de modules via npm.....	44
8.1. Publication npm vers le référentiel public par défaut.....	44
8.2. Publication npm vers un référentiel privé.....	45
9. WS REST élémentaire avec node+express.....	45
9.1. Récupérer des données entrantes au format JSON.....	45
9.2. Renvoyer des données/réponses au format JSON.....	47
9.3. Renvoyer si besoin des statuts d'erreur (http).....	47
9.4. récupération de paramètres.....	47
10. Exemple simple (CRUD) sans base de données.....	48
11. Eventuelles autorisations "CORS".....	51
12. Avec mode post et authentification minimaliste.....	51
13. Divers éléments structurants.....	54
13.1. ORDRE IMPORTANT et Routers en tant que modules annexes.....	54

13.2. Gestionnaire d'erreurs.....	56
-----------------------------------	----

VI - WS REST via JAX-RS et JavaEE.....60

1. API java pour REST (JAX-RS).....	60
1.1. Code typique d'une classe java (avec annotations de JAX-RS).....	60
1.2. Configuration de JAX-RS intégrée à un projet JEE6/CDI.....	64
1.3. Configuration de JAX-RS avec CXF intégré dans Spring.....	65
1.4. Appel de webServices REST en java via l'API "jax-rs 2".....	67
1.5. éventuelle logique d'appel RPC avec Proxy JAX-RS2.....	70

VII - WS REST via Spring-MVC.....73

1. Présentation du framework "Spring MVC"	73
1.1. Classe "controller".....	75
1.2. éventuelle génération directe de la réponse HTTP.....	75
1.3. @RequestParam (accès aux paramètres HTTP).....	76
2. Web services "REST" pour application Spring.....	76
3. WS REST via Spring MVC et @RestController.....	77
3.1. Réponse et statut http par défaut en cas d'exception.....	81
3.2. @ResponseStatus.....	82
3.3. ResponseEntityExceptionHandler (très bien).....	83
3.4. Exemples d'appels en js/ajax.....	85
3.5. Invocation java de service REST via RestTemplate de Spring.....	91
3.6. Appel moderne/asynchrone de WS-REST avec WebClient.....	95
3.7. Test d'un "RestController" via MockMvc.....	97
3.8. Test unitaire de contrôleur Rest.....	97
3.9. Test d'intégration de contrôleur Rest avec réels services.....	99

VIII - Sécurisation WS REST, Api_key , JWT ,100

1. Api Key.....	100
2. Token d'authentification.....	101
2.1. Tokens : notions et principes.....	101
2.2. Bearer Token (au porteur) / normalisé HTTP.....	103
2.3. JWT (Json Web Token).....	104
3. Jeton JWT avec Spring-Security.....	104
3.1. Vue d'ensemble sur "Spring-security".....	104
3.2. Api java pour jetons JWT.....	106
3.3. Enrobage de "JwtUtil" dans un composant "JwtTokenProvider" spécifique au contexte Spring.....	108
3.4. Exemple de web service rest d'authentification retournant un jeton "JWT" en cas de succès.....	111
3.5. Filtre web "JwtAuthenticationFilter" (basé sur spring-security) pour extraire le jeton JWT.....	115

3.6. Configuration nécessaire (Spring-security , Spring-mvc).....	116
3.7. Variante avec authentification jdbc.....	118
3.8. Exemple de WS applicatif (non public) protégé par le mécanisme d'authentification précédent:.....	120
4. Token JWT avec NodeJs.....	120

IX - Design Api REST et description swagger2.....122

1. Design d'une api REST.....	122
1.1. Rappels des fondamentaux.....	122
1.2. Retourner des réponses explicites et des statuts Http précis.....	122
1.3. Prise en compte des problématiques de sécurité.....	123
1.4. Design pattern "DTO" adapté aux web services REST.....	124
1.5. Autres considérations (bonne pratiques).....	124
2. Notion d'Api REST et description.....	125
2.1. Description détaillée d'Api REST (Swagger, RAML, ...).....	125
2.2. Fragile format YAML.....	126
3. Swagger.....	127
3.1. OpenApi (évolution "openSource" de swagger , swagger V3).....	127
4. Config swagger3 / openapi-doc pour spring.....	128

X - Délégation d'authentification , OAuth2.....132

1. OAuth2 (présentation).....	132
1.1. grant_type="code".....	135
1.2. grant_type="implicit".....	136
1.3. grant_type="password" (delegate in same organization).....	137
1.4. grant_type="client_credential" (app to app auth,no user login).....	138
2. OIDC (OpenID Connect).....	138
3. Mise en oeuvre OAuth en java.....	141
4. Mise en oeuvre OAuth avec NodeJs.....	141

XI - Annexe – Eléments de sécurité.....143

1. Eléments de sécurité (HTTP , ...).....	143
1.1. Basic Http Auth et limitations.....	143
1.2. Cryptage élémentaire via hash (MD5 , SHA) + salt.....	144
1.3. Bcrypt pour crypter les mots de passe stockés en base.....	146
1.4. Problématique "man-in-the-middle".....	146
1.5. HMAC.....	148
2. Sécurité pour WS-REST (généralités).....	150
2.1. Pseudo session avec "token" plutôt que cookie :.....	150
2.2. Responsabilités techniques coté serveur :.....	152
2.3. Service d'authentification / génération token.....	154

2.4. Intercepteur et vérification d'un jeton.....	155
3. Sécurité WS-REST avec Spring-MVC.....	156
3.1. Exemple (très basique) de WS REST d'authentification :.....	156
3.2. Exemple (basique) de WS-REST sécurisé sans intercepteur.....	157
3.3. Exemple amélioré de WS sécurisé avec intercepteur.....	157

XII - Annexe – Bibliographie, Liens WEB + TP.....	160
--	------------

1. Bibliographie et liens vers sites "internet"	160
2. TP.....	160

I - Présentation des WS REST

1. Deux grands types de WS (REST et SOAP)

2 grands types de services WEB: **SOAP/XML** et **REST/HTTP**

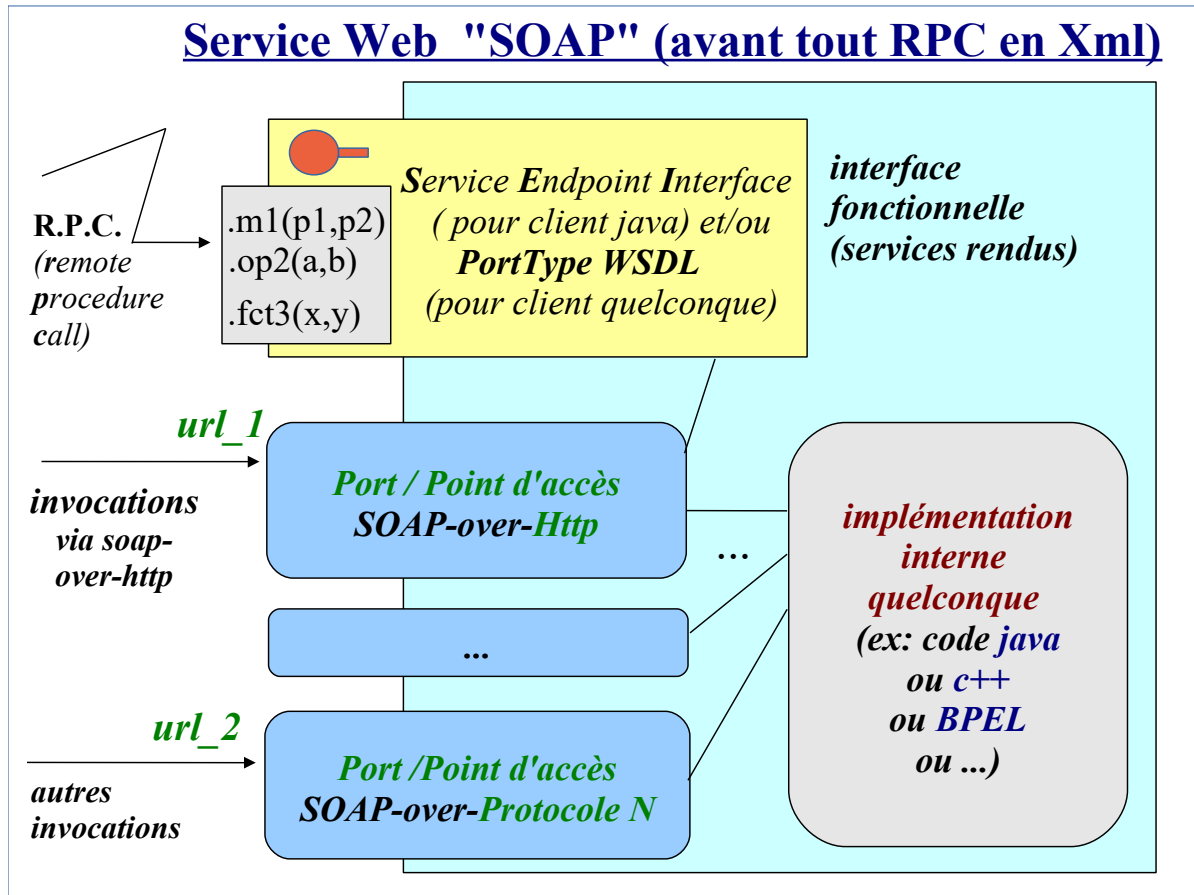
WS-* (SOAP / XML)

- "Payload" systématiquement en **XML** (*sauf pièces attachées / HTTP*)
- **Enveloppe SOAP** en XML (*header facultatif pour extensions*)
- **Protocole de transport au choix** (HTTP, JMS, ...)
- Sémantique quelconque (*appels méthodes*) , **description WSDL**
- **Plutôt** orienté Middleware SOA (*arrière plan*)

REST (HTTP)

- "Payload" au choix (XML , HTML , **JSON**, ...)
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "**CRUD**" (*modes http PUT,GET,POST,DELETE*)
- **Plutôt** orienté IHM Web/Web2 (*avant plan*)

1.1. Caractéristiques clefs des web-services SOAP / Xml



Points clefs des Web services "SOAP"

Le format "xml rigoureux" des requêtes/réponses (définis par ".xsd", ".wsdl") permet de retraiter sans aucune ambiguïté les messages "soap" au niveau certains services intermédiaires (dans ESB ou ...). Certains automatismes génériques sont applicables .

Fortement typés (xsd:string , xsd:double) les web-services "soap" conviennent très bien à des appels et implémentations au sein de **langages fortement typés** (ex : "c++", "c#", "java", "...").

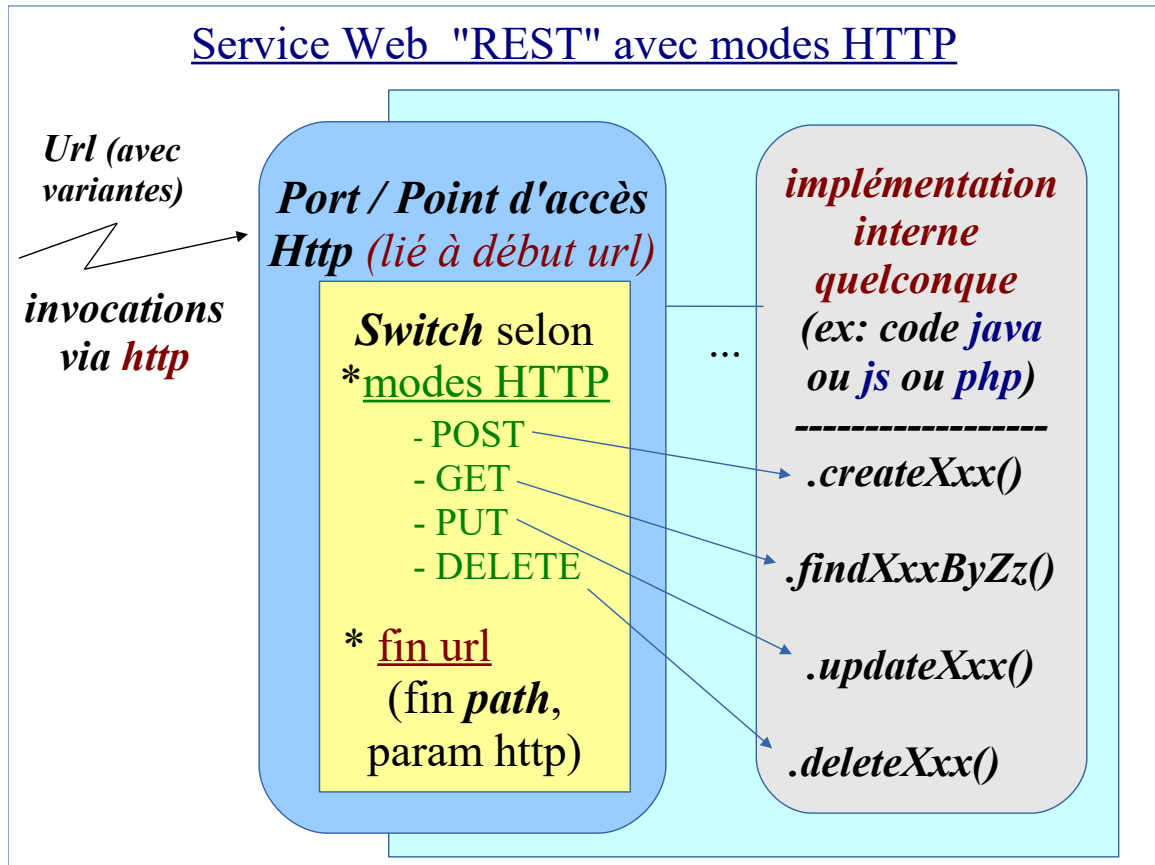
A l'inverse , des **langages faiblement typés** tels que "php" ou "js" sont **moins appropriés pour soap** (appels cependant faisables)

La **relative complexité et la verbosité "xml"** des messages "soap" engendrent des **appels moins immédiats** (mode http "post" avec enveloppe à préparer") et des **performances moyennes**.

Soap peut être utilisé en mode "envoi de document" mais c'est rare.

Les messages "soap" peuvent être véhiculés par "jms" mais c'est rare.

1.2. Caractéristiques clefs des web-services "REST" / "HTTP"



Points clefs des Web services "REST"

Retournant des données dans un format quelconque ("**XML**", "**JSON**" et éventuellement "**txt**" ou "**html**") les web-services "**REST**" offrent des **résultats qui nécessitent généralement peu de re-traitements pour être mis en forme** au sein d'une IHM web.

Le format "**au cas par cas**" des données retournées par les services REST permet peu d'automatisme(s) sur les niveaux intermédiaires.

Souvent associés au format "**JSON**" les web-services "**REST**" **conviennent parfaitement** à des appels (ou implémentations) au sein du **langage javascript**.

La **relative simplicité des URLs d'invocation des services "REST"** permet des **appels plus immédiats** (*un simple href="..." suffit en mode **GET** pour les recherches de données*).

La **compacité/simplicité des messages "JSON"** souvent associés à "**REST**" permet d'obtenir **d'assez bonnes performances**.

2. Web Services "R.E.S.T."

REST = style d'architecture (conventions)

REST est l'acronyme de **R**epresentational **S**tate **T**ransfert.

C'est un **style d'architecture** qui a été décrit par **Roy Thomas Fielding** dans sa thèse «*Architectural Styles and the Design of Network-based Software Architectures*».

L'information de base, dans une architecture REST, est appelée **ressource**.
Toute information (à sémantique stable) qui peut être nommée est une ressource: un article, une photo, une personne, un service ou n'importe quel concept.

Une ressource est identifiée par un **identificateur de ressource**. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

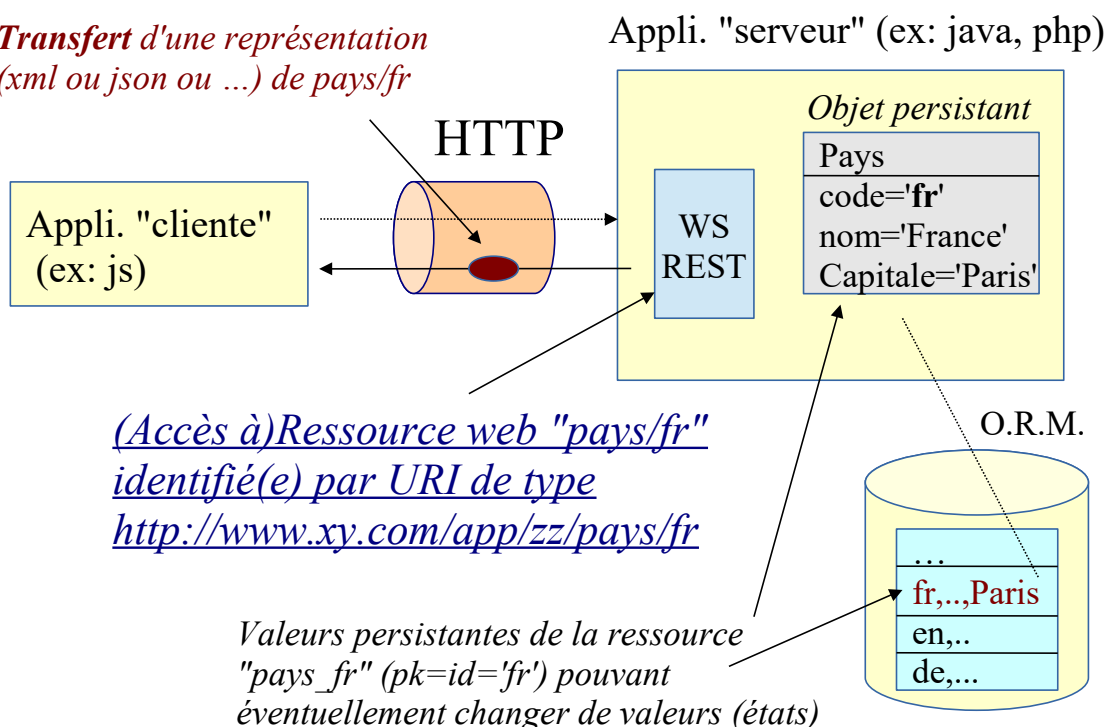
NB: dans la plupart des cas, une ressource REST correspond indirectement à un enregistrement en base (avec la *clef primaire* comme partie finale de l'uri "identifiant").

Les composants de l'architecture REST manipulent ces ressources en **transférant à travers le réseau** (via HTTP) des **représentations de ces ressources**.

Sur le web, on trouve aujourd'hui le plus souvent des représentations au format **HTML, XML ou JSON**.

REST : transferts de représentations de ressources

*Transfert d'une représentation
(xml ou json ou ...) de pays/fr*



REST et principaux formats (xml,json)

Une invocation d'URL de service REST peut être accompagnée de données (en entrée ou en sortie) pouvant prendre des formats quelconques :

text/plain , text/html , application/xml , application/json , ...

Dans le cas d'une lecture/recherche d'informations , le format du résultat retourné pourra (selon les cas) être :

- **imposé (en dur) par le code du service REST .**
- **au choix (xml , json) et précisé par une partie de l'url**
- **au choix (xml , json) et précisé par le champ "Accept :" de l'entête HTTP de la requête. (exemple: Accept: application/json) .**

Dans tous les cas, la réponse HTTP devra avoir son format précisé via le champ habituel **Content-Type: application/json** de l'entête.

Format JSON (JSON = *JavaScript Object Notation*)

Les 2 principales caractéristiques de JSON sont :

- Le principe de clé / valeur (map)
- L'organisation des données sous forme de tableau

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

```
[
  {
    "nom": "article a",
    "prix": 3.05,
    "disponible": false,
    "descriptif": "article1"
  },
  {
    "nom": "article b",
    "prix": 13.05,
    "disponible": true,
    "descriptif": null
  }
]
```

une liste d'articles

une personne

```
{
  "nom": "xxxx",
  "prenom": "yyyy",
  "age": 25
}
```

REST et méthodes HTTP (verbes)

Les **méthodes HTTP** sont utilisées pour indiquer la **sémantique des actions demandées** :

- **GET** : **lecture/recherche** d'information
- **POST** : **envoi** d'information
- **PUT** : **mise à jour** d'information
- **DELETE** : **suppression** d'information

Par exemple, pour récupérer la liste des adhérents d'un club, on peut effectuer une requête de type **GET** vers la ressource **<http://monsite.com/adherents>**

Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient **<http://monsite.com/adherents?ageMinimum=20>**

Pour supprimer numéro 4, on peut employer une requête de type **DELETE** telle que **<http://monsite.com/adherents/4>**

Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps (invisible) du message HTTP avec comme URL celle de la ressource web que l'on veut créer ou mettre à jour.

Exemple concret de service REST : "Elevation API"

L'entreprise "**Google**" fournit gratuitement certains services WEB de type REST.

"**Elevation API**" est un service REST de Google qui renvoie l'altitude d'un point de la planète selon ses coordonnées (latitude, longitude) .

La documentation complète se trouve au bout de l'URL suivante :

<https://developers.google.com/maps/documentation/elevation/?hl=fr>

Sachant que les coordonnées du Mont blanc sont :

Lat/Lon : 45.8325 N / 6.86417 E (GPS : 32T 334120 5077656)

Les invocations suivantes (du service web rest "api/elevation")

<http://maps.googleapis.com/maps/api/elevation/json?locations=45.8325,6.86417>

<http://maps.googleapis.com/maps/api/elevation/xml?locations=45.8325,6.86417>

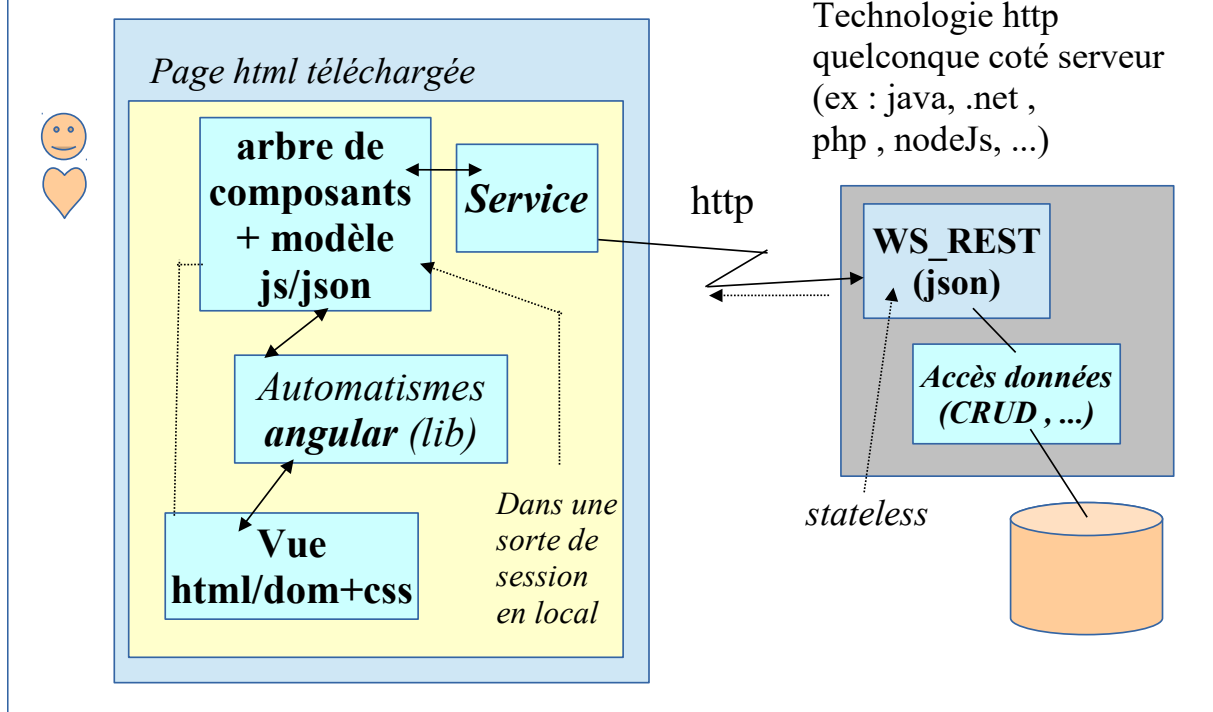
donne les résultats suivants "json" ou "xml":

```
{ "results" : [
  {
    "elevation" : 4766.466796875,
    "location" : {
      "lat" : 45.8325,
      "lng" : 6.86417
    },
    "resolution" : 152.7032318115234
  }
], "status" : "OK"
}
```

```
?xml version="1.0" encoding="UTF-8"?>
<ElevationResponse>
  <status>OK</status>
  <result>
    <location>
      <lat>45.8325000</lat>
      <lng>6.8641700</lng>
    </location>
    <elevation>4766.4667969</elevation>
    <resolution>152.7032318</resolution>
  </result>
</ElevationResponse>
```

Angular (positionnement)

Coté client (navigateur)



Conventions sur URL / Path des ressources REST

Type requêtes	HTTP Method	URL ressource(s) distante(s)	Request body	Réponse JSON
Recherche multiple	GET	.../product .../product?crit1=v1&crit2=v2	vide	Liste/tableau d'objets
Recherche par id	GET	.../product/idRes (avec idRes=1,...)	vide	Objet JSON
Ajout (seul)	POST	.../product	Objet JSON	Objet JSON avec id quelquefois calculé (incr)
Mise à jour (seule)	PUT	.../product/idRes ou .../product	Objet JSON avec .id	Objet JSON mis à jour
SaveOr Update	POST	.../product	Objet JSON	Objet JSON ajouté (auto incr id) ou modifié
suppression	DELETE	.../product/idRes	vide	Statut et message
Autres/product-action/opXy/....

2.1. Statuts HTTP (code d'erreur ou ...)

Catégories de code/statut HTTP :

1xx	Information (rare)
2xx (ex : 200)	Succès
3xx	Redirection
4xx	Erreur du client
5xx (ex : 500)	Erreur du serveur

Principaux codes/statuts en cas de succès ou de redirection:

200 , OK	Requête traitée avec succès. La réponse selon méthode de requête utilisée
201 , Created	Requête traitée avec succès et création d'un document.
204 , No Content	Requête traitée avec succès mais pas d'information à renvoyer.
301 , <i>Moved Permanently</i>	Document déplacé de façon permanente
304 , <i>Not Modified</i>	Document non modifié depuis la dernière requête

Principaux codes d'erreurs :

400 , Bad Request	La syntaxe de la requête est erronée (ex : invalid argument)
401 , Unauthorized	Une authentification est nécessaire pour accéder à la ressource.
403 , Forbidden	authentification effectuée mais manque de droits d'accès (selon rôles, ...)
404 , Not Found	Ressource non trouvée.
409 , Conflict	La requête ne peut être traitée en l'état actuel.
...	<i>liste complète sur</i> https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP
500 , Internal Server Error	Erreur interne (vague) du serveur (ex ; bug , exception , ...) .
501 , Not Implemented	Fonctionnalité réclamée non supportée par le serveur
503 , Service Unavailable	Service temporairement indisponible ou en maintenance.

2.2. Variantes classiques

Réponses plus ou moins détaillées (simple "http status" ou bien "message json")

Lorsqu'un serveur répond à une requête en mode POST , il peut soit :

- retourner le "https status" **201/CREATED** et une réponse JSON comportant toute l'entité sauvegardée coté serveur avec souvent l'id (clef primaire) automatiquement généré ou incrémenté

```
{ "id" : "a345b6788c335d56" , "name" : "toto" , ... }
```
- se contenter de renvoyer le "http status" **201/CREATED** avec aucun message de réponse mais avec le champ **Location: /type_entite/idxy** comportant au moins l'id de la ressource enregistrée au sein de l'entête HTTP de la réponse .
L'application cliente pourra alors effectuer un second appel en mode GET avec une fin d'URL en /type_entite/idxy si elle souhaite récupérer tous les détails de l'entité sauvegardée .
- combiner les 2 styles de réponses (champ Location **ET** réponse JSON)

Lorsqu'un serveur répond à une requête en mode DELETE , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message
- retourner le "https status" **200/OK** et une réponse JSON de type

```
{ "message" : "resource of id ... successfully deleted" }
```

Lorsqu'un serveur répond à une requête en mode PUT , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message
- retourner le "https status" **200/OK** et une réponse JSON comportant toutes les valeurs de l'entité mise à jour du coté serveur , exemple:

```
{ "id" : "a345b6788c335d56" , "name" : "titi" , ... }
```

*On peut éventuellement envisager que le serveur réponde **par défaut** aux modes PUT et DELETE par un simple **204/NO_CONTENT** et qu'il réponde par **200/OK + un message JSON** si le paramètre http optionnel **?v=true** ou **?verbose=true** est présent en fin de l'URL de la requête .*

Identifiant de la ressource à modifier en mode PUT placé en fin d'URL ou bien dans le corps de la requête HTTP, ou bien les deux.

Lorsqu'un serveur reçoit une requête de mise à jour en mode PUT , l'id de l'entity peut soit être précisée en fin d'URL , soit être précisée dans les données json de la partie body et si l'information est renseignée des 2 façons elle ne doit pas être incohérente .

Le serveur peut éventuellement faire l'effort de récupérer l'id de l'une ou des deux façons envisageables et peut renvoyer **400/BAD_REQUEST** si l'id de l'entité à mettre à jour n'est pas renseigné ou bien incohérent.

2.3. Safe and idempotent REST API

Une Api "Rest" désigne un ensemble de Web-services liés à un certain domaine fonctionnel (ex : gestion des stocks ou facturation ou ...)

Un appel "HTTP" vers une api-rest est dit "**safe**" s'il n'engendre pas de modifications du côté des ressources du serveur ("**safe**" = "**readonly**").

En mathématique , une fonction est dite "**idempotente**" si plusieurs appels successifs avec les mêmes paramètres retournent toujours le même résultat.

Au niveau d'une **api-rest** , une **invocation HTTP** (ex : **GET** , **PUT** ou **DELETE**) est dite "**idempotente**" si **plusieurs appels successifs avec les mêmes paramètres engendrent un même "état résultat"** au niveau du serveur .

Mais la réponse HTTP peut cependant varier .

Exemple : premier appel à "delete xyz/567" --> return "200/OK" ou "204/NO_CONTENT"

et second appel à "delete xyz/567"--> return 404 / notFound

mais dans les 2 cas , la ressource de type "xyz" et d'id=567 est censée ne plus exister .

Le DELETE est donc généralement considéré comme idempotent .

	safe	idempotent
GET (et HEAD , OPTIONS)	y	y
PUT	n	y
DELETE	n	y
POST	n	n

Intérêt de l'impotence comportementale du côté serveur :

Une application cliente doit souvent passer par des intermédiaires pour véhiculer une requête HTTP jusqu'au serveur . Certains mécanismes intermédiaires considèrent "internet / http" comme pas fiable à 100 % et vont quelquefois effectuer plusieurs retransmissions d'une requête si la première tentative échoue . il vaut mieux donc que le serveur se comporte de manière idempotente dans un maximum de cas .

Bien que le vocabulaire "~~idempotence~~" ne soit pas du tout approprié , **il est tout de même conseillé de retourner des réponses HTTP dans un format assez homogène vers le client** pour que celui-ci soit simple à programmer (pas trop de if ... else ...)

Dans tous les cas , bien documenter "comportements & réponses" d'une apit rest .

II - Technologies pour WS REST / vue d'ensemble

1. Quelques API et technologies "web services REST"

1.1. Vue d'ensemble sur technologies "REST" :

Principales API associées au web-services REST

Langage **JavaScript** ---> **nodeJs** + routes **express** , ...

Langage **PHP** --> routes \$app->get("...",...) , \$app->post("...",...)
avec "Silex" ou autre et json_encode(...)

Langage **C#** ---> ASP.NET , classe de type "ApiController" , ...

Langage **Java** --->

- API officielle(standard) "**JAX-RS**" (v1 : JEE6 , v2 : JEE7)
(@Produces , @GET, @POST, ... , @Path)
ou bien
- API efficace "**Spring-mvc**" (pour eco-système "Spring")
(@RestController , @RequestMapping() ,)

Autres Langues --> n'importe quelle combinaison de technologies
permettant de générer une réponse HTTP au
format prédominant JSON .

Compléments intéressants pour **web-services REST**

Pour la sécurité :

- * preuve d'authentification via jeton "**JWT**" (JSON Web Token)
- * **HTTPS** à utiliser et configurer en prod.
- * **OAuth** (Open Authentication : délégation d'authentification vers serveur (ou partie de serveur) spécialisé .

Pour la description/documentation et les tests:

- * **Swagger2** ou RAML ou
- * éventuel diagramme UML , ...
- * **Postman** , Tests unitaires automatisés (js, java)

1.2. W.S. REST en java avec JAX-RS ou Spring-MVC

Exemple de WS-REST codé avec "JAX-RS"

```
@Path("devises") @Produces("application/json")
public class DeviseListCtrl {
    @Inject
    private GestionDevises serviceDevises; //internal (EJB3 ou ...) local service

    @Path(value="/") @GET
    List<Devise> getAllDevises() {
        return serviceDevises.getListeDevises();
    }

    @Path("/{name}") @GET
    Devise getDeviseByName(@PathParam("name") String deviseName) {
        return serviceDevises.getDeviseByName( deviseName);
    }
}
```

→ à intégrer dans une appli web (.war)

→ nécessite une très légère configuration annexe :

```
//pour url en http://localhost:8080/myWebApp/services/rest + @Path() java
@ApplicationPath("/services/rest")
public class MyRestApplicationConfig extends Application { ... }
```

Alternative "Spring-MVC" (vis à vis de JAX-RS)

Dans le monde **java**, il est possible de programmer un web service "REST" avec **Spring-MVC** à la place de **JAX-RS**.

Inconvénients de Spring-MVC (pour WS REST):

- * ***ce n'est pas le standard officiel***, c'est du ***spécifique "Spring"***
... sachant que le standard s'est maintenant amélioré en version 2 ...
- * la technologie concurrente JAX-RS s'intègre pas trop mal dans Spring (via jersey ou cxf).

Avantages de Spring-MVC (pour WS REST):

- * **l'intégration au sein d'un projet "Spring" est plus aisée (configuration plus simple, moins de librairies ".jar" nécessaires, ...)**
- * **facilement combinable avec d'autres extensions de Spring (spring-security, spring-boot, ...)**
- * logique "MVC" pouvant être utile si l'on retourne des portions d'HTML.

Au final, si projet JEE/CDI → standard JAX-RS
si projet Spring → Spring-MVC ou bien JAX-RS

Exemple de WS-REST codé avec "Spring-MVC"

```
@RestController
@RequestMapping(value="/devises" , headers="Accept=application/json")
public class DeviseListCtrl {
    @Autowired
    private GestionDevises serviceDevises; //internal Spring local service

    @RequestMapping(value="/" , method=RequestMethod.GET)
    @ResponseBody
    List<Devise> getAllDevises() {
        return serviceDevises.getListeDevises();
    }

    @RequestMapping(value="/{name}" , method=RequestMethod.GET)
    @ResponseBody
    Devise getDeviseByName(@PathVariable("name") String deviseName) {
        return serviceDevises.getDeviseByName( deviseName);
    }
}
```

- même logique de paramétrage que JAX-RS (HTTP method , path , ...)
- annotations différentes de celles de JAX-RS (avec davantage d'attributs)
- quasiment aucune configuration annexe nécessaire (avec java-config , spring-boot)

1.3. W.S. REST en javascript avec nodeJs+Express

III - Test de Web Services "REST"

Pour tester un appel en mode de web service REST en mode GET , un simple navigateur suffit à déclencher la bonne URL.

1. Test de W.S. REST via Postman

L'application "**postman**" (téléchargeable depuis l'url <https://www.postman.com/downloads/>) existe depuis longtemps et est souvent considérée comme l'application de référence pour tester les web services "REST" .

NB1 : après le premier lancement , il n'est pas obligatoire de s'enregistrer (créer un compte) pour utiliser l'application , on peut cliquer sur un lien à peine visible plus bas que la boîte de dialogue nous invitant à nous enregistrer et l'on peut d'une manière générale fermer toutes les "popups" et créer un nouvel onglet de requête pour paramétrer et lancer un test.

NB2 : A une certaine époque , "postman" pouvait s'utiliser en tant que plugin pour le navigateur "chrome" . Ce plugin est maintenant "deprecated" (plus maintenu) .

1.1. paramétrages "postman" pour une requête en mode "post"

The screenshot shows the Postman interface for configuring a POST request. At the top, the URL bar shows 'http://localhost:8080/' with a dropdown menu set to 'POST' and the full URL 'http://localhost:8080/serverSoap/ws/rest/devise'. Below this, the 'Headers' tab is selected, showing a table with one header: 'Content-Type' with the value 'application/json'. The 'Body' tab is also visible, showing the 'raw' format selected with 'JSON (application/json)' chosen. The body content is a JSON object: { "codeDevise" : "MS2", "tauxChange" : 1.4568 }. At the bottom, there is a blue 'Send' button.

Key	Value
<input checked="" type="checkbox"/> Content-Type	application/json

Authorization	Headers (1)	Body	Pre-request Script	Tests
<div> <input type="radio"/> form-data <input type="radio"/> x-www-form-urlencoded <input checked="" type="radio"/> raw <input type="radio"/> binary JSON (application/json) </div> <pre> 1 { 2 "codeDevise" : "MS2", 3 "tauxChange" : 1.4568 4 }</pre>				

Send

Status: 200 OK Time: 82 ms Size: 156 B

1.2. Exemple de réponses précises reçues et affichées par "postman"

POST , `http://localhost:8282/login-api/public/auth` , Content-Type : `application/json` ,
request body : `{ "username": "admin1" , "password" : "pwdadmin1" , "roles" : "admin,user" }`

==> **200 ok**

et responseBody :

```
{
  "username": "admin1",
  "status": true,
  "message": "successful login",
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIqZWN0IjoieYWRtaW4xIiwicm9sZXMiOiJhZG1pbix1c2VyIiwiaWF0IjoxNTk2Nzk2MTI0LCJleHAiOjE1OTY4MDMzMjQsImZcyI6Imh0dHA6Ly93d3cubXljb21wYW55In0.wBQHJPN20VE7tZrF8vk3Cq9FItiQQOf2RETdMSB19ho",
  "roles": "admin,user"
}
```

Si requête comportant `{ "username": "admin1" , "password" : "abcdef" , "roles" : "admin,user" }`
alors réponse de ce type :

```
{
  "username": "admin1",
  "status": false,
  "message": "login failed (wrong password)",
  "token": null,
  "roles": null
}
```

DELETE , `http://localhost:8282/devise-api/private/role_admin/devise/m3875`

==> **200 ok**

et responseBody :

```
{
  "action": "devise with code=m3875 was deleted"
}
```

ou bien (suite à un second appel successif) :

==> **404 not found**

```
{  
  "errorCode": "404",  
  "message": "deleteOne exception with id=m3875"  
}
```

Par contre pour déclencher un appel en mode "put" , "post" et "delete" , un outil spécialisé tel que "postman" s'avère bien pratique pour effectuer des tests .

2. Test de W.S. REST via curl

curl (*command line url*) est un programme utilitaire (d'origine linux) permettant de déclencher des requêtes HTTP via une simple ligne de commande.

Via certaines options, curl peut effectuer des appels en mode "GET", "POST", "DELETE" ou "PUT".

Ceci peut être très pratique pour tester rapidement un web service REST via quelques lignes de commandes placées dans un script réutilisable (.bat, .sh,) .

lancer_curl.bat

```
cd /d "%~dp0"
REM instructions qui vont bien
set URL=http://localhost:8081/my-api/info/1
curl %URL%
pause
```

curl fonctionne en **mode GET par défaut** si pas de -d (*pas de data*)

```
curl %URL%
REM "verbose" (-v) très pratique pour connaître les détails de la communication réseau
curl -v %URL%
```

```
curl -o out.json %URL%
```

pour stocker la réponse dans un fichier texte (ici out.json)

curl fonctionne en **mode POST** par défaut avec data (-d ...)

curl fonctionne en **mode PUT** si **-X PUT** et **mode DELETE** si **-X DELETE**

appel au format par défaut (application/x-www-form-urlencoded)

si pas d'option -H "Content-Type: ..." au niveau de la requête
alors par défaut logique champ/paramètre de formulaire en mode POST avec

-d paramName1=valeur1 -d paramName2=valeur2 ...

Exemple :

```
set URL=clientIdPassword:secret:@localhost:8081/basic-oauth-server/oauth/token
set PWD=d8dfc382-e012-491a-8d03-ca6ad9d81083
```



```
curl %URL% -d grant_type=password -d username=user -d password=%PWD%
```

Requête au format "application/json" :

NB : en version windows , curl ne gère pas bien les simples quotes et il faut préfixer les " internes par des \

```
curl %LOGIN_URL% -H "Content-Type: application/json"
-d '{"username\":\"member1\", \"password\": \"pwd1\"}'
```

il vaut mieux donc utiliser un fichier pour les données en entrée :

```
curl %LOGIN_URL% -H "Content-Type: application/json" -d @member1-login-request.json
```

avec

member1-login-request.json

```
{
  "username": "member1",
  "password": "pwd1"
}
```

Authentification avec curl :

```
curl --user myUsername:myPassword ... permet une "BASIC HTTP AUTHENTICATION"
```

ou bien

```
curl -H "Authorization: Bearer b1094abc.._ou_autre_jeton" permet une demande d'autorisation
en mode "Bearer / au porteur de jeton" (jeton à préalablement récupérer via login ou autre )
```

IV - Appels de WS REST en javascript , CORS

1. Appels de WS REST (HTTP) depuis js/ajax

Avec ajax , ça va briller!

AJAX est l'acronyme d'Asynchronous JavaScript And XML, mais ça peut être utilisé avec Json .

1.1. Cadre des appels

Lorsqu'une requête http est initiée depuis du code javascript s'exécutant dans le contexte d'une page html , on dit que l'on effectue un appel "ajax" .

Le sigle Ajax correspondant à peu près à "asynchronous javascript activation framework" indique :

- le déclenchement non bloquant d'une requête http
- l'enregistrement d'une fonction "callback" qui sera automatiquement appelée en différé lorsque la réponse reviendra

NB : l'appel non bloquant peut être considéré comme asynchrone mais le protocole HTTP est un protocole de transport synchrone (avec timeout si la réponse ne revient pas dans un délai raisonnable).

Techniquement, un appel ajax s'effectue en s'appuyant sur un objet technique "XmlHttpRequest" fourni par tous les navigateurs pas trop anciens .

La partie Xml de XmlHttpRequest tient au fait qu'historiquement les premiers webServices normalisés (SOAP) étaient au format Xml. Bien que le terme "XmlHttpRequest" n'ait pas été changé pour des raisons de compatibilité ascendante du code javascript, il est possible de déclencher n'importe quelle requête HTTP depuis XHR , y compris des requêtes au format "JSON" .

Pour simplifier la syntaxe d'un appel ajax on peut éventuellement s'appuyer sur des bibliothèques "javascript" complémentaires ("jquery" , "fetch" , "RxJs" , ...) .

Le principe des appels "ajax" sert essentiellement à déclencher une requête HTTP suite à un événement utilisateur en vue d'obtenir des données permettant de réactualiser une partie de la page HTML courante . La page courante n'est pas entièrement remplacée par une autre. Seule une sous partie de celle ci est ajustée par code js/DOM (Document Object Model).

Certaines applications , dites "SPA: Single Page Application" sont entièrement bâties sur ce principe . Au lieu de switcher de pages html on switch de sous pages (<div ...>....</div>) .

1.2. XHR (XmlHttpRequest) dans tout navigateur récent

Exemple basique :

```
function makeAjaxRequest(callback) {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
            callback(xhr.responseText);
        }
    };
    xhr.open("GET", "handlingData.php", true);
    xhr.send(null);
}

function readData(sData) {
    if (sData!=null) {
        alert("good response");
        // + display data with DOM
    } else {
        alert("empty response");
    }
}

makeAjaxRequest(readData);
```

variations en mode "POST" :

```
xhr.open("POST", "handlingData.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send("param1=xx&param2=yy");
```

```
xhr.open("POST", "/json-handler");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.send(JSON.stringify({prenom:"Jean", nom:"Bon"}));
```

Exemple plus élaboré :

my_ajax_util.js

```
//subfunction with errCallback as optional callback :
function registerCallbacks(xhr,callback,errCallback) {
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4){
            if((xhr.status == 200 || xhr.status == 0)) {
                callback(xhr.responseText);
            }
            else {
                if(errCallback)
                    errCallback(xhr.responseText);
            }
        }
    };
}

function makeAjaxGetRequest(url,callback,errCallback) {
    var xhr = new XMLHttpRequest();
    registerCallbacks(xhr,callback,errCallback);
    xhr.open("GET", url, true);  xhr.send(null);
}

function makeAjaxDeleteRequest(url,callback,errCallback) {
    var xhr = new XMLHttpRequest();
    registerCallbacks(xhr,callback,errCallback);
    xhr.open("DELETE", url, true);  xhr.send(null);
}

function makeAjaxPostRequest(url,jsonData,callback,errCallback) {
    var xhr = new XMLHttpRequest();
    registerCallbacks(xhr,callback,errCallback);
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.send(jsonData);
}

function makeAjaxPutRequest(url,jsonData,callback,errCallback) {
    var xhr = new XMLHttpRequest();
    registerCallbacks(xhr,callback,errCallback);
    xhr.open("PUT", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.send(jsonData);
}
```

appelAjax.js

```

var traiterReponse = function (response){
    //response ici au format "json string"
    var zoneResultat = document.getElementById("spanRes");
    var jsDevise = JSON.parse(response);
    zoneResultat.innerHTML=jsDevise.change; //ou .rate
}

function onSearchDevise() {
    var zoneSaisieCode = document.getElementById("txtCodeDevise");
    var codeDevise = zoneSaisieCode.value;
    console.log("codeDevise="+codeDevise);
    var urlWsGet="./devise-api/public/devise/"+codeDevise;
    makeAjaxGetRequest(urlWsGet,traiterReponse); //non bloquant (asynchrone)
    //....
}

```

appelAjax.html

```

<html>
<head>
    <meta charset="UTF-8">
    <title>appelAjax</title>
    <script src="js/my_ajax_util.js"></script>
    <script src="js/appelAjax.js"></script>
</head>
<body>
    codeDevise :<input type="text" id="txtCodeDevise"/> <br/>
    <input type="button" id="btnSearch" onclick="onSearchDevise()"
        value="rechercher devise"/> <br/>
    Devise : <span id="spanRes"></span>
</body>
</html>

```

1.3. Appel ajax via jQuery

La syntaxe des appels "ajax" est un peu plus explicite en s'appuyant sur la bibliothèque "jquery".

Exemple :

```
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>browse-spectacles</title>
  <script src="lib/jquery-3.3.1.min.js"></script>
  <script src="js/my-jq-ajax-util.js"></script>
</script>
$(function() {

  //appel ajax pour récupérer la liste des catégories et remplir le <select>
  $.ajax({
    type: "GET",
    url: "spectacle-api/public/spectacle/allCategories",
    contentType : "application/json",
    success: function (data,status,xhr) {
      if (data) {
        var categoryList = data;
        for(categoryIndex in categoryList){
          var category=categoryList[categoryIndex];
          $('#selectCategory').append('<option value="'+ category.id +"'>'+
            category.id + ' (' + category.title + ')</option>');
        }
        //$("#spanMsg").html(JSON.stringify(data));
      }
    },
    error: function( jqXHR, textStatus, errorThrown ){
      $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
  }); //end $.ajax
});

</script>
</head>
<body>
  <h3> BROWSE Spectacles </h3>
  categorie : <select id="selectCategory"> </select><br/>
  ... <span id="spanMsg"></span> <br/>...
</body>
</html>
```

fonctions utilitaires dans [js/my-jq-ajax-util.js](#)

```
function setSecurityTokenForAjax(){

  var authToken = sessionStorage.getItem("authToken");
  //localStorage.getItem("authToken");

  $(document).ajaxSend(function(e, xhr, options) {
```

```

        //retransmission du jeton d'authentification dans l'entête http de la requete ajax
        xhr.setRequestHeader('Authorization','Bearer '+ authToken);
    });
}

function xhrStatusToErrorMessage(jqXHR){
    var errMsg = "ajax error";//by default
    var detailsMsg=""; //by default
    console.log("jqXHR.status="+jqXHR.status);
    switch(jqXHR.status){
        case 400 :
            errMsg = "Server understood the request, but request content was invalid.";
            if(jqXHR.responseText!=null)
                detailsMsg = jqXHR.responseText;
            break;
        case 401 :
            errMsg = "Unauthorized access (401)"; break;
        case 403 :
            errMsg = "Forbidden resource can't be accessed (403)"; break;
        case 404 :
            errMsg = "resource not found (404)"; break;
        case 500 :
            errMsg = "Internal server error (500)"; break;
        case 503 :
            errMsg = "Service unavailable (503)"; break;
    }
    return errMsg+" "+detailsMsg;
}

```

Variation en mode "POST" et "[application/x-www-form-urlencoded](#)"

```

var dataJsObject = { prenom : "jean", nom : "Bon", taille: 175 } ;
$.ajax({
    type: "POST",
    url: "./my-api/person",
    contentType : "application/x-www-form-urlencoded; charset=utf-8",
    data: $.param(dataJsObject),
    dataType : 'json_or_text_or_...',
    beforeSend: function (xhr) {
        xhr.setRequestHeader ("Authorization",
                               "Basic " + btoa("usernameXy" + ":" + "passwordXy"));
    },
    success: ... , error: ....
}); //end $.ajax

```

Variation en mode "POST" et "JSON" in/out :

```
//setSecurityTokenForAjax();//js/my-jq-ajax-util.js
$.ajax({
    type: "POST",
    url: "spectacle-api/spectacle",
    data : JSON.stringify(spectacleAdditionJsObject),
    dataType : "json",
    contentType : "application/json",
    success: function (data,status,xhr) {
        if (data) {
            $("#spanMsg").html(JSON.stringify(data));
        }
    },
    error: function( jqXHR, textStatus, errorThrown ){
        $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
});//end $.ajax
```

1.4. Api fetch

Api récente (syntaxe concise basé sur enchaînement asynchrone et "**Promise**") mais pas encore supporté par tous les navigateurs.

Exemple :

```
fetch('./api/some.json')
    .then(
        function(response) {
            if (response.status !== 200) {
                console.log('Problem. Status Code: ' + response.status);
                return;
            }
            // Examine the text in the response :
            response.json().then(function(data) {
                console.log(data);
            });
        }
    )
    .catch(function(err) {
        console.log('Fetch Error :-S', err);
    });
```


1.5. Appel ajax via RxJs (api réactive)

Le framework "RxJs" lié au concept de "programmation asynchrone réactive" est assez sophistiqué et permet de déclencher une série de traitements d'une façon assez indépendante de la source de données (ex : données statiques , réponse ajax , push web-socket , ...) .

RxJs peut soit être directement utilisé en tant que bibliothèque javascript dans une page HTML , soit être utilisé via "typescript et le framework Angular 2,4,5,6 ou autre) .

Attention à la version utilisée (différences significatives dans la version récente de RxJs accompagnant Angular 6) .

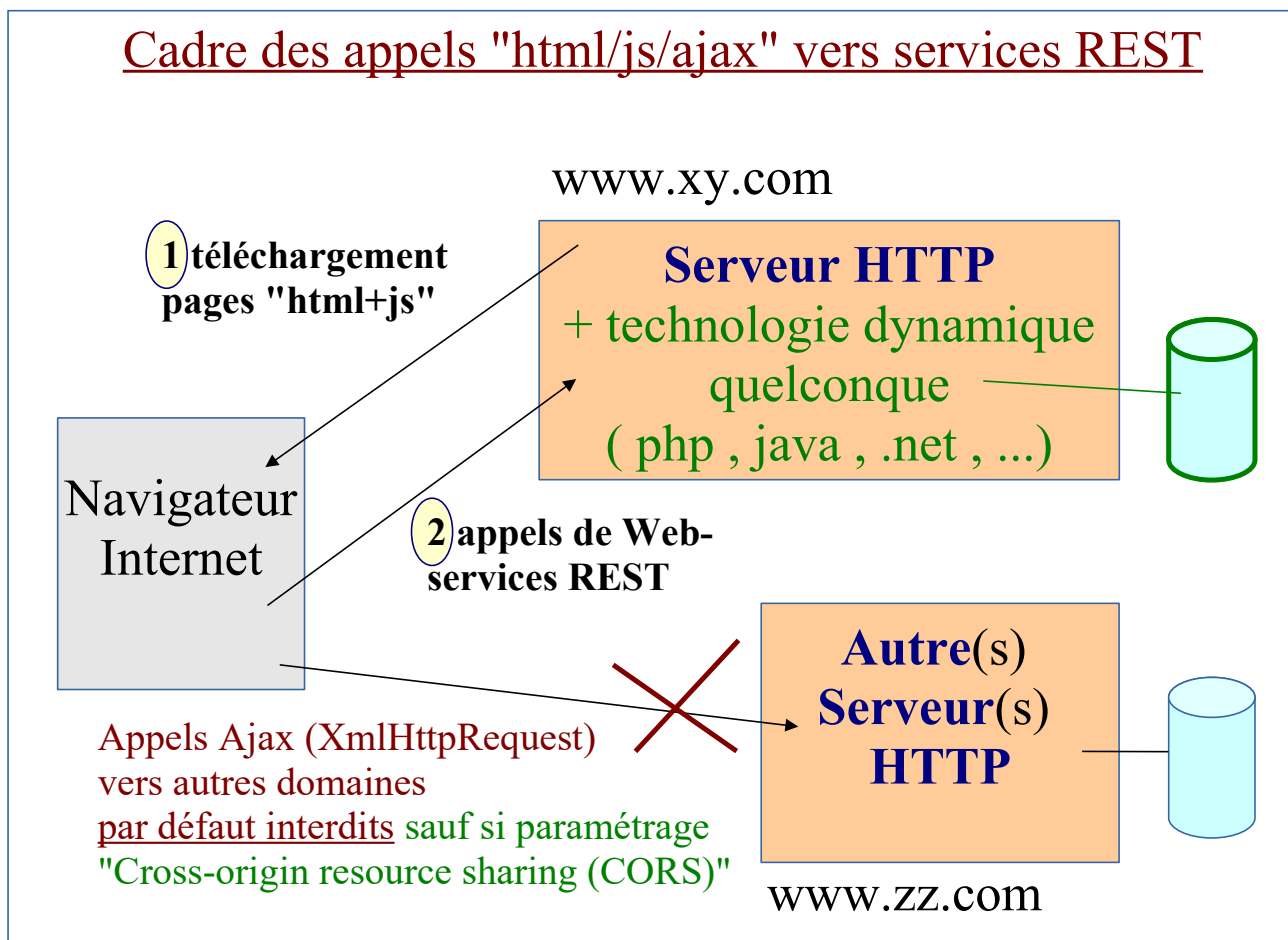
--> une bonne présentation de RxJS est accessible au bout de l'URL suivante

<https://www.julienpradet.fr/tutoriels/introduction-a-rxjs/>

--> exemples d'appel ajax via RxJs et de config proxy dans le support de cours "Angular 4,5,6"

2. Limitations Ajax sans CORS

Cadre des appels "html/js/ajax" vers services REST



3. CORS (Cross Origin Resource Sharing)

CORS=Cross Origin Resource Sharing

CORS est une **norme du W3C** qui précise certains **champs** à placer dans une **entête HTTP** qui serviront à échanger entre le navigateur et le serveur des informations qui serviront à décider si une requête sera ou pas acceptée.

(utile si domaines différents) , dans requête simple ou bien dans pré-échange préliminaire quelquefois déclenché en plus :

Au sein d'une requête "demande autorisation" envoyée du client vers le serveur :

Origin: <http://www.xy.com>

Dans la "réponse à demande d'autorisation" renvoyée par le serveur :

Access-Control-Allow-Origin: <http://www.xy.com>

Ou bien

Access-Control-Allow-Origin: * *(si public)*

→ *requête acceptée*

*Si absence de "Access-Control-Allow-Origin :" ou bien valeur différente
---> requête refusée*

CORS=Cross Origin Resource Sharing (2)

NB1: toute requête "CORS" valide doit absolument comporter le champ "**Origin** :" dans l'entête http. Ce champ est toujours construit automatiquement par le navigateur et jamais renseigné par programmation javascript.

Ceci ne protège que partiellement l'accès à certains serveurs car un "méchant hacker" utilise un "navigateur trafiqué".

Les mécanismes "CORS" protège un peu le client ordinaire (utilisant un vrai navigateur) que dans la mesure où la page d'origine n'a pas été interceptée ni trafiquée (l'utilisation conjointe de "https" est primordiale) .

NB2 : Dans le cas (très classique/fréquent) , où la requête comporte "**Content-Type: application/json**" (ou **application/xml** ou ...) , la norme "CORS" (considérant la requête comme étant "pas si simple") impose un pré-échange préliminaire appelé "**Preflighted request/response**" .

Paramétrages CORS à effectuer coté serveur

L'application qui coté serveur, fourni quelques Web Services REST , peut (et généralement doit) autoriser les requêtes "Ajax / CORS" issues d'autres domaines ("*" ou "www.xy.com").

Attention: ce n'est pas une "sécurité coté serveur" mais juste **un paramétrage autorisant ou pas à rendre service à d'autres domaines et en devant gérer la charge induite** (taille du cluster, consommation électrique, ...) .

```
// Exemple : CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); // "*" ou "xy.com , ..."
  res.header("Access-Control-Allow-Methods",
    "POST, GET, PUT, DELETE, OPTIONS"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept , Authorization");
  next();
});
```

Paramétrage "CORS" avec Spring-mvc

```

import org.springframework.web.bind.annotation.CrossOrigin;

...

@RestController
@CrossOrigin(origins = "**")
//@CrossOrigin(origins = { "http://localhost:4200" ,
//                        "http://www.partenaire-particulier.com" })
@RequestMapping(value="/rest/products" , headers="Accept=application/json")
public class ProductCtrl {...
}

```

et

```

@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

...

@Override
protected void configure(final HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/", "/favicon.ico", "**/*.png",
            "**/*.gif", "**/*.svg", "**/*.jpg",
            "**/*.html", "**/*.css", "**/*.js").permitAll()
        .antMatchers("/devise-api/public/**").permitAll()
        .antMatchers("/devise-api/private/**").authenticated()
        .and().cors() //enable CORS (avec @CrossOrigin sur class @RestController)
        .and().csrf().disable()
        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
        .and()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .addFilterBefore(jwtAuthenticationFilter,
            UsernamePasswordAuthenticationFilter.class);
    }

...
}

```

V - WS REST via nodeJs et express

1. Ecosystème node+npm

node (nodeJs) est un **environnement d'exécution javascript** permettant essentiellement de :

- compartimenter le code à exécuter en **modules** (import/export)
- exécuter du code en mode "**appels asynchrones non bloquants** + callback" (sans avoir recours à une multitudes de threads)
- exécuter directement du code javascript sans avoir à utiliser un navigateur web

npm (*node package manager*) est une sous partie fondamentale de node qui permet de :

- **télécharger et gérer des packages utiles à une application** (bibliothèques réutilisables)
- télécharger et utiliser des utilitaires pour la phase de développement (ex : grunt , jasmine , gulp , ...)
- **prendre en compte les dépendances entre packages** (téléchargements indirects)
- générer éventuellement de nouveaux packages réutilisables (à déployer)
-

node est à peu près l'équivalent "javascript" d'une machine virtuelle java.

npm ressemble un peu à maven de java : téléchargement des bibliothèques , construction d'applications.

Un **projet basé sur npm** se configure avec le fichier *package.json* et les packages téléchargés sont placés dans le sous répertoire **node_modules** .

Principales utilisations/applications de node :

- application "serveur" en javascript (répondant à des requêtes HTTP)
- application autonome (ex : StarUML2+ = éditeur de diagrammes UML , ...)
-

2. Express

Express correspond à un des packages téléchargeables via npm et exécutables via node.

La **technologie "express"** permet de répondre à des **requêtes HTTP** et ressemble un peu à un Servlet java ou à un script CGI .

A fond **basé sur des mécanismes souples et asynchrones** (avec "**routes**" et "**callbacks**") , "**express**" permet de coder assez facilement/efficacement des applications capables de :

- **générer dynamiquement des pages HTML** (ou autres)
- mettre en œuvre des **web services "REST"** (souvent au format "JSON") .
- prendre en charge les détails du protocoles **HTTP** (authentification "basic" et/ou "bearer" , autorisations "CORS" ,)
-

"express" est souvent considéré comme une technologie de bas niveau lorsque l'on la compare à d'autres technologies "web / coté serveur" telles que ASP , JSP , PHP , ...

"express" permet de construire et retourner très rapidement une réponse HTTP (avec tout un tas de paramétrages fin si nécessaire) . Pour tout ce qui touche au format de la réponse à générer , il faut

utiliser des technologies complémentaires (ex : templates de pages HTML avec remplacements de valeurs) .

3. Exemple élémentaire "node+express"

first express server.js

```
//modules to load:
var express = require('express');

var app = express();

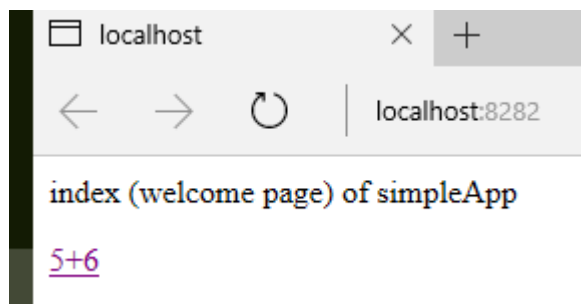
app.get('/', function(req, res , next) {
    res.setHeader('Content-Type', 'text/html');
    res.write("<html> <body>");
    res.write('<p>index (welcome page) of simpleApp</p>');
    res.write('<a href="addition?a=5&b=6">5+6</a>');
    res.write("</body></html>");
    res.end();
});

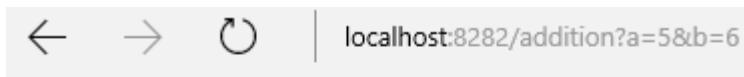
//GET addition?a=5&b=6
app.get('/addition', function(req, res , next) {
    a = Number(req.query.a);    b = Number(req.query.b);
    resAdd = a+b;
    res.setHeader('Content-Type', 'text/html');
    res.write("<html> <body>");
    res.write('a=' + a + '<br/>');    res.write('b=' + b + '<br/>');
    res.write('a+b=' + resAdd + '<br/>');
    res.write("</body></html>");
    res.end();
});

app.listen(8282 , function () {
    console.log("simple express node server listening at 8282");
});
```

lancement: node first_express_server.js

via <http://localhost:8282> au sein d'un navigateur web , on obtient le résultat suivant :





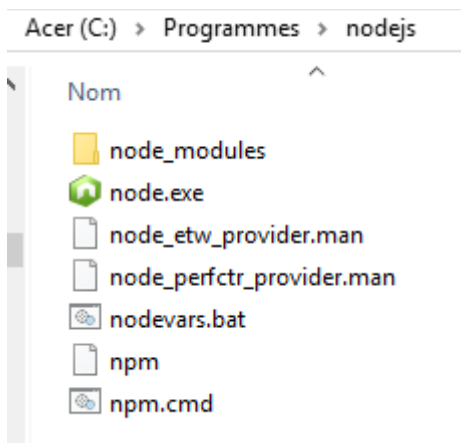
4. Installation de node et npm

Téléchargement de l'installateur **node-v16.15.0-x64.msi** (ou autre) depuis le site officiel de nodeJs (<https://nodejs.org>)

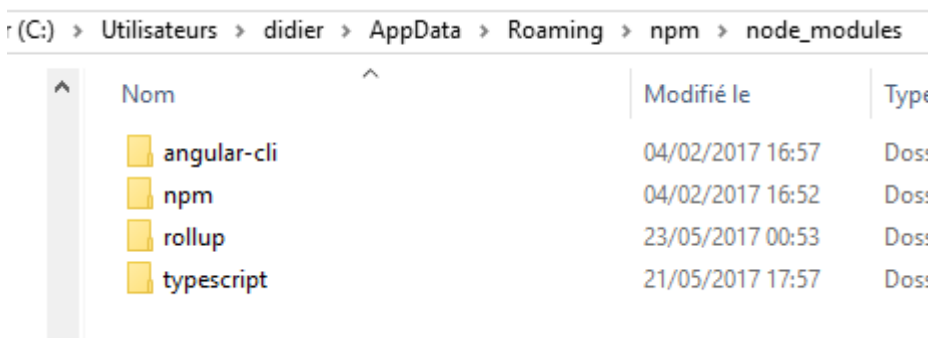
Lancer l'installation et se laisser guider par les menus.

Cette opération permet sous windows d'installer node et npm en même temps .

Sur une machine windows 64bits , nodejs s'installe par défaut dans **C:\Program Files\nodejs**



Et le répertoire pour les installations de packages en mode "global" (-g) est par défaut **C:\Users\username\AppData\Roaming\npm\node_modules**



Vérification de l'installation (dans un shell "CMD") :

node --version
v16.3.2 (ou autre)

npm --version

5. Configuration et utilisation de npm

5.1. Initialisation d'un nouveau projet

Un développeur utilise généralement npm dans le cadre d'un projet spécifique (ex : xyz). Après avoir créé un répertoire pour ce projet (ex : C:\tmp\temp_nodejs\xyz) et s'être placé dessus, on peut lancer la *commande interactive* **npm init** de façon à **générer un début de fichier "package.json"**

Exemple de fichier *package.json* généré :

```
{
  "name": "xyz",
  "version": "1.0.0",
  "description": "projet xyz",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "didier",
  "license": "ISC"
}
```

5.2. installation de nouveau package en ligne de commande :

```
npm install --save express
npm install -s mongoose
```

permet de télécharger les packages "express" et "mongoose" (ainsi que tous les packages indirectement nécessaires par analyse de dépendances) dans le sous répertoire **nodes_modules** et de mettre à jour la liste des dépendances dans le fichier **package.json** :

```
.... ,
"dependencies": {
  "express": "^4.17.1",
  "mongoose": "^6.0.12"
},
....
```

Sans l'option **--save** (ou son alias **-s**) , les packages sont téléchargés mais le fichier package.json n'est pas modifié.

Par défaut , c'est la dernière version du package qui est téléchargé et utilisé.
Il est possible de choisir une **version spécifique** en la précisant après le caractère @ :

npm install -s mongoose@4.10
ou bien (autre exemple) :
npm install -s mongodb@2.0.55

Autre procédure possible :

- 1) **éditer** le fichier **package.json** en y ajoutant des dépendances (au sein de la partie "dependencies") :
exemple :

```
"dependencies": {  
  "express": "^4.17.1",  
  "markdown": "^0.5.0",  
  "mongoose": "^6.0.12"  
}
```
- 2) lancer **npm install** (ou *npm update* ultérieurement) **sans argument**

Ceci permet de **lancer le téléchargement et l'installation de tous les packages nécessaires listés dans le fichier package.json** (ici "mardown" en plus) dans le sous répertoire **node_modules** .

Installation de packages utilitaires (pour le développement) :

Si l'on souhaite ensuite expliciter une dépendance de "développement" au sein d'un projet , on peut utiliser l'option **--save-dev** de **npm install** de façon ajouter celle ci dans la partie "devDependencies" de package.json :

npm install --save-dev grunt

```
.... ,  
"devDependencies": {  
  "grunt": "^1.0.1"  
}
```

5.3. Installation en mode global (-g)

L'option **-g** de **npm install** permet une installation en mode global : le package téléchargé sera installé dans **C:\Users\username\AppData\Roaming\npm\node_modules** sous windows 64bits (ou ailleurs sur d'autres systèmes) **et sera ainsi disponible (en mode partagé) par tous les projets** .

Le mode global est souvent utilisé pour installer des packages correspondant à des "utilitaires de développement" (ex : *grunt* ou *typescript* ou *lite-server* ou *nodemon*) .

Exemple :

npm install -g nodemon
npm install -g lite-server

6. Utilisation basique de node

hello_world.js

```
console.log("hello world");
```

```
node hello_world.js
```

7. Scripts dans package.json

Exemple :

```
npm install -g tsc
npm install -g webpack
npm install -g mocha
```

package.json

```
{
  "name": "modules-webpack-ts",
  "version": "1.0.0",
  "description": "blabla",
  "main": "index.js",
  "scripts": {
    "build": "tsc && webpack",
    "test": "mocha",
    "webpack-watch": "webpack --watch"
  },
  "author": "didier",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12"
  }
}
```

npm run *nom_script_a_lancer*

npm run **build**

npm run **webpack-watch**

npm run **test**

npm test est un raccourci pour **npm run test**

...

8. Publication de modules via npm

8.1. Publication npm vers le référentiel public par défaut

NB : **npm link** (permettant de créer un lien symbolique local entre différents projets) comporte **beaucoup de limitations/bugs** .

NB :

- Une réelle publication d'un module (ex: librairie réutilisable) vers le référentiel npm par défaut (<https://www.npmjs.com/>) nécessite de changer la version du package publié (ex 0.0.1 --> 0.0.2) .
- Tout nom de package publié doit être unique (vaut donc mieux un nom de package avec un préfixe personnel ou d'entreprise).

1) se créer un compte sur le site officiel de npm

c'est gratuit et il faut respecter des règles de bonne conduite .

2)

```
cd myprefix-libxx
npm login
...saisir username et password ...
npm publish
```

3) visualiser les packages publiés

via une URL du genre <https://www.npmjs.com/package/myprefix-libxx>

4) utiliser le package publié en tant que dépendance d'un autre projet

```
cd pyy
npm install -s myprefix-libxx
...
```

8.2. Publication npm vers un référentiel privé

En phase de développement (surtout au début) , on a souvent besoin d'un gestionnaire de référentiel npm simple et léger de manière à tester des packages localement publiés sans devoir changer la version .

Gestionnaires de référentiel npm :

- **verdaccio** : simple/léger et agissant à la fois comme un référentiel privé et un comme un proxy vers le référentiel npm public .
- Partie npm de **nexus** ≥ 3 .
-

Utilisation de Verdaccio :

npm i -g verdaccio

C:\Users\usernameXy\AppData\Roaming\verdaccio\config.yaml : config file after first launch
<http://localhost:4873/> : default verdaccio url

verdaccio

Ctrl-C

npm set registry <http://localhost:4873/>

(replace default "npm set registry https://registry.npmjs.org")

npm adduser --registry http://localhost:4873

npm login (ex: user1 or ...)

password (ex: pwd1 or ...)

email (ex: user1@worldcompany.com or)

npm whoami

npm publish

NB: si dernière version publiée sur <https://registry.npmjs.org> est 0.0.n alors en phase de dev (sur local-verdaccio) : 0.0.N+1

Pour si besoin rétablir l'utilisation du référentiel par défaut :

npm set registry <https://registry.npmjs.org>

//npm version patch //for incrementing version 0.0.n 0.0.n+1

9. WS REST élémentaire avec node+express

9.1. Récupérer des données entrantes au format JSON

La récupération des valeurs JSON véhiculés en mode **POST** ou **PUT** dans le corps (*body*) de la

requête entrante s'effectue avec la syntaxe **req.body** et nécessite la préparation et l'enregistrement d'un **"bodyParser"** :

```
//var express = require('express');
import express from 'express' ;
var app = express();

//support parsing of JSON post data
var jsonParser = express.json({ extended: true});
app.use(jsonParser);
...
//POST ... with body { "firstname" : "Jean" , "lastname" : "Bon" }
app.post('/xyz', function(req , res ) {
  let user = req.body ; //as javascript object via automatic use of jsonParser
//let user = JSON.parse(req.body); //si sans jsonParser
// ...
});
```

NB : il existe une variante de la méthode app.post() où l'on peut passer un "bodyParser" spécifique en second paramètre (pour certains cas pointus/spécifiques):

```
// POST /login gets urlencoded bodies :
app.post('/login', urlencodedParser, function (req, res) {
  res.send('welcome, ' + req.body.username)
})

// POST /api/users gets JSON bodies :
app.post('/api/users', jsonParser, function (req, res) {
  // use user in req.body
})
```

Dans le cas d'une api homogène (quasiment tout en JSON) , il est tout de même plus simple de paramétrer l'utilisation par défaut d'un bodyParser JSON via app.use() :

```
var jsonParser = express.json({ extended: true});
app.use(jsonParser)
```

9.2. Renvoyer des données/réponses au format JSON

La fonction prédéfinie `res.send(jsObject)` effectue en interne a peu près les opérations suivantes :

```
res.setHeader('Content-Type', 'application/json');
res.write(JSON.stringify(jsObject));
res.end();
```

Cette méthode `".send()"` est donc tout à fait appropriée pour retourner la réponse "JSON" à un appel de Web Service REST .

9.3. Renvoyer si besoin des statuts d'erreur (http)

Via retour direct :

```
res.status(404).json({ error : "no product to update with code=" + code });
```

ou bien via "errorHandler" ...

9.4. récupération de paramètres

```
...
// GET (array) /minibank/operations?numCpt=1
app.get('/minibank/operations', function(req, res,next) {
  let numCpt = Number(req.query.numCpt) ;
  // ...
});

// GET /minibank/compte/1
app.get('/minibank/compte/:numero', function(req, res,next) {
  let numCpt = Number(req.params.numero) ;
  // ...
});
...
```

NB :

- **req.query.pxy** récupère la valeur d'un paramètre http ayant un nom explicite en fin d'URL (`?pxy=valXy&pzz=valZz`)
- **req.params.idXy** récupère la valeur d'un paramètre logique (avec :) et sans nom explicite en fin d'URL (ex : `app.get("xyz/:idXy")` et `xyz/2`)

10. Exemple simple (CRUD) sans base de données

server.js

```
//var express = require('express');
import express from 'express';

//var produitApiRoutes = require('./produit-api-routes');
import produitApiRoutes from './produit-api-routes-memory.js';

import { dirname } from 'path';
import { fileURLToPath } from 'url';
const __dirname = dirname(fileURLToPath(import.meta.url));

var app = express();

//support parsing of JSON post data
var jsonParser = express.json({ extended: true});
app.use(jsonParser);

//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "./html"
app.use('/html', express.static(__dirname+"/html"));

app.get('/', function(req , res ) {
    res.redirect('/html/index.html');
});

//delegate REST API routes to apiRouter(s) :
app.use(produitApiRoutes.apiRouter);
//app.use(otherApiRoutes.apiRouter);

app.listen(8282 , function () {
    console.log("http://localhost:8282");
});
```

produit-api-routes-memory.js

```
//var express = require('express');
import express from 'express';
const apiRouter = express.Router();

var allProduits = [];

allProduits.push({ code : 1 , nom : 'classeur' , prix : 4.0 });
allProduits.push({ code : 2 , nom : 'cahier' , prix : 2.1 });
allProduits.push({ code : 3 , nom : 'colle' , prix : 2.4 });
allProduits.push({ code : 4 , nom : 'stylo' , prix : 1.9 });

var codeMax=4; //pour simulation auto_incr

function findProduitInArrayByCode(produits,code){
    var produit=null;
    for(let i in produits){
```



```

        if(produits[i].code == code){
            produit=produits[i]; break;
        }
    }
    return produit;
}

function removeProduitInArrayByCode(produits,code){
    var delIndex;
    for(let i in produits){
        if(produits[i].code == code){
            delIndex=i; break;
        }
    }
    if(delIndex){
        produits.splice(i,1);
    }
}

function findProduitsWithPrixMini(produits,prixMini){
    var selProduits=[];
    for(let i in produits){
        if(produits[i].prix >= prixMini){
            selProduits.push(produits[i]);
        }
    }
    return selProduits;
}

//exemple URL: http://localhost:8282/produit-api/public/produit/1
apiRouter.route('/produit-api/public/produit/:code')
.get( function(req , res , next ) {
    var codeProduit = req.params.code;
    var produit = findProduitInArrayByCode(allProduits,codeProduit);
    res.send(produit);
});

//exemple URL: http://localhost:8282/produit-api/public/produit (returning all produits)
//      http://localhost:8282/produit-api/public/produit?prixMini=1.05
apiRouter.route('/produit-api/public/produit')
.get( function(req , res , next ) {
    var prixMini = req.query.prixMini;
    if(prixMini){
        res.send(findProduitsWithPrixMini(allProduits,prixMini));
    }else{
        res.send(allProduits);
    }
});

// http://localhost:8282/produit-api/private/role-admin/produit en mode post
// avec { "code" : null , "nom" : "produitXy" , "prix" : 12.3 }

```

```

//ou bien { "nom" : "produitXy", "prix" : 12.3 } dans req.body
apiRouter.route('/produit-api/private/role-admin/produit')
.post( function(req , res , next ) {
    var nouveauProduit = req.body;
    //simulation auto_incr :
    if(nouveauProduit.code == null){
        codeMax++; nouveauProduit.code = codeMax;
    }
    console.log("POST,nouveauProduit="+JSON.stringify(nouveauProduit));
    allProduits.push(nouveauProduit);
    res.send(nouveauProduit);
});

// http://localhost:8282/produit-api/private/role-admin/produit en mode PUT
// avec { "code" : 1 , "nom" : "produit_xy", "prix" : 16.3 } dans req.body
apiRouter.route('/produit-api/private/role-admin/produit')
.put( function(req , res , next ) {
    var newValueOfProduitToUpdate = req.body;
    console.log("PUT,newValueOfProduitToUpdate="
        +JSON.stringify(newValueOfProduitToUpdate));
    var produitToUpdate =
        findProduitInArrayByCode(allProduits,newValueOfProduitToUpdate.code);
    if(produitToUpdate!=null){
        produitToUpdate.nom = newValueOfProduitToUpdate.nom;
        produitToUpdate.prix = newValueOfProduitToUpdate.prix;
        res.send(produitToUpdate);
    }else{
        res.status(404).json({ error : "no produit to update with code="
            + newValueOfProduitToUpdate.code });
    }
});

// http://localhost:8282/produit-api/private/role-admin/produit/1 en mode DELETE
apiRouter.route('/produit-api/private/role-admin/produit/:code')
.delete( function(req , res , next ) {
    var codeProduit = req.params.code;
    console.log("DELETE,codeProduit="+codeProduit);
    removeProduitInArrayByCode(allProduits,codeProduit);
    res.send({ deletedProduitCode : codeProduit } );
});

// exports.apiRouter = apiRouter; // ancienne syntaxe common-js
export default { apiRouter }; // nouvelle syntaxe es2015

```

11. Eventuelles autorisations "CORS"

```
//var express = require('express');
import express from 'express';
var app = express();
...

// CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  //ou avec "www.xyz.com" à la place de "*" en production
  res.header("Access-Control-Allow-Methods",
    "POST, GET, PUT, DELETE, PATCH"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept, Authorization");
  if(req.method === 'OPTIONS'){
    res.header('Access-Control-Allow-Methods', 'POST, GET, PUT, PATCH, DELETE');
    //to give access to all the methods provided
    return res.status(200).json({});
  }
  next();
});

...
...
app.get(...) , app.post(...) , ...

app.listen(8282 , function () {
  console.log("rest server with CORS enabled listening at 8282");
});
```

Cela peut quelquefois être pratique/utile durant la phase de développement (tant que pas d'intermédiaire "reverse-proxy" ni "api-gateway").

Attention :

A partir de cette page , tous les exemples de code sont en ancienne syntaxe (common-js ou bien typescript) .

12. Avec mode post et authentification minimaliste

```
var express = require('express');
var bodyParser = require('body-parser'); //dépendance indirecte de express via npm
var app = express();

var myGenericMongoClient = require('./my_generic_mongo_client');

var uuid = require('uuid'); //to generate a simple token
```

```

app.use(bodyParser.json()); // to parse JSON input data and generate js object : (req.body)
app.use(bodyParser.urlencoded({ extended: true}));

...

// POST /minibank/verifyAuth { "numClient" : 1 , "password" : "pwd1" }
app.post('/minibank/verifyAuth', function(req, res,next) {
  var verifAuth = req.body; // JSON input data as jsObject with ok = null
  console.log("verifAuth :" +JSON.stringify(verifAuth));
  if(verifAuth.password == ("pwd" + verifAuth.numClient) ){
    verifAuth.ok= true;
    verifAuth.token=uuid.v4();
    //éventuelle transmission parallèle via champ "x-auth-token" :
    res.header("x-auth-token", verifAuth.token);
    //+stockage dans une map pour verif ulterieure : ....
  }
  else {
    verifAuth.ok= false;
    verifAuth.token = null;
  }
  res.send(verifAuth); // send back with ok = true or false and token
});

// GET /minibank/comptes/1
app.get('/minibank/comptes/:numero',
  displayHeaders, verifTokenInHeaders /*un peu sécurisé*/,
  function(req, res,next) {
    myGenericMongoClient.genericFindOne('comptes', { '_id' : Number(req.params.numero) },
      function(err,compte){
        sendDataOnError(err,compte,res);
      });
  });

function sendDataOnError(err,data,res){
  if(err==null) {
    if(data!=null)
      res.send(data);
    else res.status(404).send(null);//not found
  }
  else res.status(500).send({error: err});//internal error (ex: mongo access)
}

//var secureMode = false;
var secureMode = true;

function extractAndVerifToken(authorizationHeader){
  if(secureMode==false) return true;
  /*else*/
  if(authorizationHeader!=null ){
    if(authorizationHeader.startsWith("Bearer")){
      var token = authorizationHeader.substring(7);

```

```

        console.log("extracted token:" + token);
        //code extrêmement simplifié ici:
        //idéalement à comparer avec token stocké en cache (si uuid token)
        //ou bien tester validité avec token "jwt"
        if(token != null && token.length>0)
            return true ;
        else
            return false;
    }
    else
        return false;
}
else
    return false;
}

// verif bearer token in Authorization headers of request :
function verifTokenInHeaders(req, res, next) {
    if( extractAndVerifToken(req.headers.authorization))
        next();
    else
        res.status(401).send(null);//401=Unauthorized or 403=Forbidden
}

// display Authorization in request (with bearer token):
function displayHeaders(req, res, next) {
    //console.log(JSON.stringify(req.headers));
    var authorization = req.headers.authorization;
    console.log("Authorization: " + authorization);
    next();
}
...
app.listen(8282 , function () {
    console.log("minibank rest server listening at 8282");
});

```

13. Divers éléments structurants

13.1. ORDRE IMPORTANT et Routers en tant que modules annexes

server.ts

```
import express from 'express';
import * as bodyParser from 'body-parser';
export const app :express.Application = express();
import { apiErrorHandler } from './api/apiErrorHandler'
import { apiRouter } from './api/apiRoutes';
import { initSequelize } from './model/global-db-model'

//PRE TRAITEMENTS (à placer en haut de server.ts) !!!!

//support parsing of JSON post data
var jsonParser = bodyParser.json() ;
app.use(jsonParser);

//ROUTES ORDINAIRES (après PRE traitements , avant POST traitements)

app.use(apiRouter); //delegate REST API routes to apiRouter
//app.use(apiRouter2); //delegate REST API routes to apiRouter2

//POST TRAITEMENTS (à placer en bas de server.ts) !!!

app.use(apiErrorHandler); //pour gérer les erreurs/exceptions
                        //pas rattrapées par try/catch et qui se propagent
                        //alors automatiquement au niveau "express appelant mon code"
                        //ou bien pour gérer les erreurs explicitement déléguées ici via next(err) ;

export const server = app.listen(8282 , function () {
  console.log("http://localhost:8282");
  initSequelize(); // ou autre initialisation
});
```

api/apiRoutes.ts

```
import { Request, Response ,NextFunction, Router } from 'express';
import { Devise } from './model/devise';
//import { ErrorWithStatus , NotFoundError, ConflictError } from './error/errorWithStatus'
import { MemoryMapDeviseService } from './dao/memoryMapDeviseService';
import { SqDeviseService } from './dao/sqDeviseService';
import { DeviseDataService } from './dao/deviseDataService';

export const apiRouter = Router();
```

```

var deviseService : DeviseDataService = new SqDeviseService();

apiRouter.route('/devise/:code')
.get( function(req :Request, res :Response , next: NextFunction ) {
    let codeDevise = req.params.numero;
    deviseService.findById(codeDevise)
    .then((devise)=> { res.send(devise) })
    .catch((err)=>{next(err)});
});

//POST ... with body { "code": "M1" , "nom" : "monnaie1" , "change" : 1.123 }
apiRouter.route('/devise').post( function(req :Request, res :Response , next: NextFunction ) {
    let devise :Devise = req.body ; //as javascript object
    //deviseService.insert(devise)
    deviseService.saveOrUpdate(devise)
    .then((savedDevise)=> { res.send(savedDevise)})
    .catch((err)=>next(err));
});

// DELETE http://localhost:8282/devise/EUR
apiRouter.route('/devise/:code')
.delete( function(req :Request, res :Response , next: NextFunction ) {
    let codeDevise = req.params.code;
    deviseService.deleteById(codeDevise)
    .then(()=> { res.status(200).send({ "action" : "devise with code="+codeDevise + " was
deleted"});})
    .catch((err)=>next(err));
});

// http://localhost:8282/devise renvoyant tout [ {} , {}]
// http://localhost:8282/devise?changeMini=1.1 renvoyant [{}] selon critere
apiRouter.route('/devise').get(function(req :Request, res :Response , next: NextFunction ) {
    let changeMini = req.query.changeMini;
    deviseService.findAll()
    .then((deviseArray)=> {
        if(changeMini){
            //filtrage selon critère changeMini:
            deviseArray = deviseArray.filter((dev)=>dev.change >= changeMini);
        }
        res.send(deviseArray)
    })
    .catch((err)=>next(err));
});

```

13.2. Gestionnaire d'erreurs

Classe(s) pratique(s) pour remonter des erreurs/exceptions avec plus de précision que `Error("message")` :

error/errorWithStatus.ts

```
// ErrorWithStatus est une version améliorée de Error (err.message)
// avec un attribut status (404,500,...) permettant une automatisation
// du retour du status http dans le "apiErrorHandler"

//NB: Error is a very special class (native)
//subclass cannot be test with instanceof, ...

export class ErrorWithStatus extends Error {
  public msg : string;
  public status : number
  constructor(message:string,status:number=500){
    super(message);    this.msg= message;    this.status=status;
  }
  public static extractStatusInNativeError(e: Error):number{
    let status=500; //500 (Internal Server Error)
    let jsonStr = JSON.stringify(e);
    let errWithStatus = JSON.parse(jsonStr);
    if(errWithStatus.status)
      status = errWithStatus.status;
    return status;
  }
}

export class NotFoundError extends ErrorWithStatus{
  constructor(message:string="not found",status:number=404){
    super(message,status);
  }
}

export class ConflictError extends ErrorWithStatus{
  constructor(message:string="conflict (with already existing)",status:number=409){
    super(message,status);
  }
}
```


gestionnaire d'erreur général (à enregistrer à la fin de server.ts) :

apiErrorHandler.ts

```
import { Request, Response, NextFunction, ErrorRequestHandler } from 'express';
...
export const apiErrorHandler : ErrorRequestHandler =
function (err: any, req: Request, res: Response, next: NextFunction) {
//console.log("in apiErrorHandler err=", err + " " + JSON.stringify(err));
//console.log("in apiErrorHandler typeof err=", typeof err);
if(typeof err == 'string'){
    res.status(500).json({errorCode:'500', message: 'Internal Server Error : ' + err});
}
else if(err instanceof Error){
    //console.log("in apiErrorHandler err is instanceof Error");
    let status = ErrorWithStatus.extractStatusInNativeError(err);
    res.status(status).json({errorCode:`${status}`, message: err.message});
}
else
    res.status(500).json({errorCode:'500', message: 'Internal Server Error'});
}
```

Le gestionnaire d'erreur ci dessus renvoie un status et un message d'erreur plus ou moins précis selon le fait que l'erreur est :

- une simple chaîne de caractère
- une instance de Error (peut être de type ErrorWithStatus et dont on essaie d'y extraire le status)
- une autre chose inconnue

Remontée des erreurs , exceptions dans apiRoutes.ts

Attention, attention : un simple throw "message erreur"

```
ou throw new Error("...")
ou throw new ErrorWithStatus("..." , 404) ;
```

ne fonctionne bien qu'en mode simpliste/synchrone .

En mode classique asynchrone, cela fait planter le serveur !!!

.../...

Solution 1 : via next(err)

```
//POST ... with body { "code": "M1", "nom": "monnaie1", "change": 1.123 }
apiRouter.route('/devise').post( function(req :Request, res :Response , next: NextFunction ) {
  let devise :Devise = req.body ; //as javascript object
  //deviseService.insert(devise)
  deviseService.saveOrUpdate(devise)
  .then((savedDevise)=> { res.send(savedDevise)})
  .catch( (err)=> { next(err) });
});
```

Solution 2 : via wrapper pour async et throw

fonction utilitaire permettant de bien remonter les erreurs/exceptions ou de renvoyer un bon résultat au format json:

```
function asyncToResp(fn : Function) {
  return function(req :Request, res :Response , next: NextFunction) {
    // Make sure to `catch()` any errors and pass them along to the `next()`
    // middleware in the chain, in this case the error handler.
    fn(req, res, next)
    .then((data:object)=> { res.send(data) })
    .catch(next);
  };
}
```

Utilisation :

```
apiRouter.route('/devise/:code')
.get( asyncToResp( async function(req :Request, res :Response , next: NextFunction){
  let codeDevise = req.params.code;
  let devise = await deviseService.findById(codeDevise)
  return devise;
}));
```

et magiquement :

- en cas d'exception --> try/catch automatique et next(err) --> apiErrorHandler(err,...) déclenché automatiquement
- quand tout va bien --> valeur retournée dans fonction async --> Promise.resolve() automatique et res.send() déclenché automatiquement via asyncToResp()

VI - WS REST via JAX-RS et JavaEE

1. API java pour REST (JAX-RS)

JAX-RS est une API java officielle/normalisée (depuis les spécifications **JEE 6**) qui permet de mettre en œuvre des services REST en JAVA :

Une classe java avec annotations JAX-RS sera exposée comme web service REST.

JAX-RS 1 se limitait à l'implémentation serveur "java".

JAX-RS 2 (JEE 7) propose maintenant une api pour les appels (du coté client "java")

Pour implémenter les services REST, on utilise principalement les annotations JAX-RS suivantes:

- **@Path** : définit le chemin (fin d'URL) de la ressource. Cette annotation se place sur la classe et/ou sur la méthode implémentant le service.
- **@GET, @PUT, @POST, @DELETE** : définit le mode HTTP (selon logique CRUD)
- **@Produces** spécifie le ou les Types MIME de la réponse du service
- **@Consumes** : spécifie le ou les Types MIME acceptés en entré du service

Les principales implémentations de JAX-RS sont :

- **Jersey**, implémentation de référence de SUN/Oracle (bien : simple/léger et efficace)
- **Resteasy**, l'implémentation interne à jboss (bien)
- **CXF** (gérant à la fois "SOAP" et "REST" , remplacer si besoin la sous couche "jettison" par "jackson" pour mieux générer du "json")
- **Restlet** (???)

1.1. Code typique d'une classe java (avec annotations de JAX-RS)

Exemple d'une classe de donnée (ex : DTO) pouvant (si besoin) être transformée en XML:

```
package pojo;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "user") /* pour préciser balise xml englobante et pour jaxb2 */
public class User {
    private Integer id; // +get/set
    private String name; // +get/set

    @Override
    public String toString() {
        return String.format("{id=%s,name=%s}", id, name);
    }
}
```

Classe d'implémentation d'un service REST:

```
package service;
```

```

import java.util.HashMap; import java.util.Map;
import javax.ws.rs.GET;      import javax.ws.rs.Path;
import javax.ws.rs.PathParam; import javax.ws.rs.QueryParam;
import javax.ws.rs.Produces; import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import pojo.User;

@Path("myservice") //service gérant des utilisateurs
//@Produces("text/xml")
@Produces("application/json") //par défaut pour toutes les méthodes de la classe
//+ éventuel @Named selon contexte (Spring ou CDI/JEE6 ou ...)
public class ServiceImpl /* implements ServiceDefn */{ //pas d'interface obligatoire

    private static Map<Integer,User> users = new HashMap<Integer,User>();

    static { //jeux de données (pour simulation de données en base)
        users.put(1, new User(1, "foo"));    users.put(2, new User(2, "bar"));
    }

    // + éventuel injection d'un service local interne pour déléguer les appels :
    // @Autowired ou @Inject ou @EJB ou ....
    // private ServiceInterne serviceInterne ;

    @GET
    @Path("users/{id}")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/users/2
    public User getUser(@PathParam("id") Integer id) {
        return users.get(id);
    }

    @GET
    @Path("users")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/users?name=foo
    // et quelquefois ...?p1=val1&p2=val2&p3=val3
    public User getUserByCriteria(@QueryParam("name") String name) {
        if(name==null) return users.values().iterator().next(); //return all if no criteria
        else .... ;
        return .... ;
    }

    @GET
    @Path("bad")
    public Response getBadRequest() {
        return Response.status(Status.BAD_REQUEST).build();
        //le comble d'un service est de ne rendre aucun service !!!! (bad: pas bien: is no good) !!!
    }

    @POST // (REST recommande fortement POST pour des ajouts )
    @Path("users")
    // pour action URL = http://localhost:8080/mywebapp/services/rest/myservice/users
    // dans form avec <input name="id" /> et <input name="name" /> et method="POST"
    public Response addNewUser(@FormParam("id") Integer id,@FormParam("name") String name) {
        users.put(id,new User(id,name));
        return Response.status(Status.OK).build();
        //une autre variante du code pourrait retourner le "User" avec une clé primaire quelquefois auto incrémentée
        // c'est ce qu'attend par défaut "Angular-Js" avec une logique @POST pour un "saveOrUpdate"
    }
}

```

<http://localhost:8080/mywebapp/services/rest/myservice/users/>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userCollection>
  <users>
    <user><id>1</id><name>foo</name></user>
    <user> <id>2</id> <name>bar</name> </user>
  </users>
</userCollection>
```

http://localhost:8080/mywebapp/services/rest/myservice/users/1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user>
  <id>1</id>
  <name>foo</name>
</user>
```

Quelques autres exemples :

```
@GET
@Path("/euroToFranc/{s}")
@Produces("text/plain")
public double euroToFranc(@PathParam("s")double s){
    return s*6.55957 ;
}
```

renvoie directement la valeur en mode texte (sans aucune balise)

---> techniquement possible mais pas conseillé (trop basique) .

```
@PUT
@Path("devises")
public Response updateDeviseChange (@FormParam("name")String name,
                                     @FormParam("change") double newChange) {
    Devise devise = mapDevises.get(name);
    if(devise!=null) {
        devise.setChange(newChange);
        return Response.status(Status.OK).build();
    }
    else return Response.status(Status.NOT_FOUND).build();
}
```

//même principe pour @DELETE

Exemple JAX-RS au format JSON :

```
import java.util.List;

import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
...

/*
 * classe java du WS REST lié aux clients (personnes)
 */

@Path("client") //avant dernière partie de l'URL
```

```

@Produces("application/json") //conv. automatique réponse java en réponse json

public class ClientRest {

//@EJB ne fonctionne pas ici ,
// il faut utiliser @Inject (plus moderne de CDI) à la place
    @Inject //@Inject de l'api CDI (Container Dependency Injection)
        // ne fonctionne que si
        //le fichier WEB-INF/beans.xml est présent dans l'application.
    private IServiceClient serviceClient;

    @Path("/{idClient}") //dernière partie de l'URL
    @GET // GET pour lecture , recherche unique par id/clef primaire
    //URL = http://localhost:8080/myappWeb/services/rest/client/1
    public Client rechercherClientQuiVaBien(
        @PathParam("idClient") Long numClient) {
        return serviceClient.rechercherInfosClient(numClient);
    }

    /* DEVINETTE :
    Soap et Rest sont sur un bateau.
    Soap glisse sur un savon et tombe à l'eau.
    Qui est qui reste ?
    */

    @Path("") //dernière partie de l'URL
        //(éventuellement vide si rien de plus)
    @POST // POST pour "ajout" ou "modif" ,
    //URL = http://localhost:8080/myappWeb/services/rest/client
    //avec en entrée { "prenom": "jean" , "nom" : "Bon" , ...}
    @Consumes("application/json")
    public Client postClient(Client cli) {
        System.out.println("requête reçue: " + cli.toString());
        //en entree cli.numClient est quelquefois à null (ou pas renseigné)
        cli = serviceClient.saveOrUpdateClient(cli);
        //en retour cli.numClient est quelquefois auto incrémenté
        return cli;
    }

    @Path("")
    @GET // recherche multiple via critère(s) de recherche
    //URL=http://localhost:8080/myappWeb/services/rest/client?nom=Therieur
    public List<Client> rechercherClients(@QueryParam("nom") String nom) {
        if(nom!=null) {
            return serviceClient.rechercherListeClientsParNom(nom);
        }else {
            return serviceClient.rechercherTousLesClients();
        }
    }
}

```

1.2. Configuration de JAX-RS intégrée à un projet JEE6/CDI

Lorsque JAX-RS est utilisé au sein d'un projet JEE6/CDI (par exemple avec ou sans "EJB3" pour JBoss7 ou bien pour Tomcat EE), on peut configurer la partie intermédiaire des URL "rest" et les classes java à prendre en charge via une configuration ressemblant à la suivante :

```
package tp.web.rest; //ou autre
import java.util.HashSet; import java.util.Set;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

//url: http://localhost:8080/myWebApp/services/rest + @Path() java
@ApplicationPath("/services/rest")
public class MyRestApplicationConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        final Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(XyServiceRest.class);
        classes.add(ServiceRest2.class);
        return classes;
    }
}
```

Ceci fonctionne avec des classes java (ex: XyServiceRest) utilisant "@Inject et avec beans.xml présent dans WEB-INF .

Variante basée sur une découverte automatique des classes de WS-Rest (comportant @Path("...")) :

```
@ApplicationPath("/rest")//partie du milieu des URLs après http://localhost:8080/appliJee-web
// et avant les valeurs de @Path() des classes java
public class MyRestApplicationConfig extends Application {
    //rien (découverte automatique via un scan automatique des packages java de l'application)
}
```

Autre variante (avec getSingletons() à préparer) :

```
public class MyRestApplicationConfig extends Application {

    @Override
    public Set<Object> getSingletons() {
        final Set<Object> singletons = new HashSet<Object>();
        singletons.add(myRestImplSingleton);
        // myRestImplSingleton peut être instancié via un new , une fabrique ou autre
        // possibilité de mixer cela avec CDI ("beans.xml" , @Named , @Inject , ....)
        return singletons;
    }
}
```


1.3. Configuration de JAX-RS avec CXF intégré dans Spring

NB: les versions récentes de CXF (≥ 2.2) implémentent maintenant JAX-RS (en plus de JAX-WS)

Attention :

- la configuration exacte diffère légèrement d'une version à l'autre (2.2 , 2.5 , 2.7 , 3.x).
- Certaines versions de CXF utilisent par défaut "jettison" en interne pour générer du "json" ce qui pose des problèmes de compatibilité avec Angular-Js (ou ...).
Il est conseillé de configurer CXF pour qu'il utilise "jackson" à la place de "jettison" .
- CXF 2.x est compatible avec jackson 1.9 (*org.codehaus.jackson*)
CXF 3.x est plutôt compatible avec jackson 2.x (*com.fasterxml.jackson*)

WEB-INF/web.xml (pour spring + cxf):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
...
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/classes/deploy-context.xml</param-value>
  </context-param>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <servlet>
    <servlet-name>CXFServlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Configuration "Spring/CXF" pour JAX-RS:

deploy-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xmlns:jaxrs="http://cxf.apache.org/jaxrs"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
```

```

http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd">
    <import resource="classpath:META-INF/cxf/cxf.xml" />

<!-- nb : par défaut , CXF 2.x utilise la techno "Jettison" pour le lien entre JAXB et JAXRS/Json. cette techno génère
des résultats "json" incompatibles avec angular-js .
il faut utiliser la technologie alternative "jackson" pour obtenir une compatibilité avec angular-js -->

<!-- 'org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider' for (old)jackson 1.9 before fasterxml -->
<bean id='jacksonJsonProvider' class='com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider' />
<bean id='jacksonXmlProvider' class='com.fasterxml.jackson.jaxrs.xml.JacksonJaxbXMLProvider' />

<!-- url complete de type "http://localhost:8080/mywebapp/services/rest/myservice/users/"
avec "services" associée à l'url-pattern de CxfServlet dans web.xml
et myservice/users associé aux valeurs de @Path() de la classe java et des méthodes
-->

<jaxrs:server id="myRestServices" address="/rest">
    <jaxrs:providers>
        <ref bean='jacksonJsonProvider' />
        <ref bean='jacksonXmlProvider' />
    </jaxrs:providers>
    <jaxrs:serviceBeans>
        <ref bean="serviceImpl" />      <!-- <ref bean="service2Impl" /> -->
    </jaxrs:serviceBeans>
    <jaxrs:extensionMappings>
        <entry key="xml" value="text/xml" />      <entry key="json" value="application/json" />
    </jaxrs:extensionMappings>
</jaxrs:server>

<bean id="serviceImpl" class="service.ServiceImpl" /> <bean id="service2Impl" class="service.Service2Impl" />
</beans>

```

Configuration maven type pour cxf :

```

<properties>
    <org.springframework.version>4.1.1.RELEASE</org.springframework.version>
    <org.apache.cxf.version>3.0.2</org.apache.cxf.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-api</artifactId>
        <version>${org.apache.cxf.version}</version>
        <scope>compile</scope>
    </dependency>

```

```

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-frontend-jaxrs</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>

<!-- to generate good json for angular-js
      from JAX-RS et CXF (voir spring conf of jaxrs:server) -->

<dependency>
  <groupId>com.fasterxml.jackson.jaxrs</groupId>
  <artifactId>jackson-jaxrs-json-provider</artifactId>
  <version>2.2.3</version>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.jaxrs</groupId>
  <artifactId>jackson-jaxrs-xml-provider</artifactId>
  <version>2.2.3</version>
</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-frontend-jaxws</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-ws-security</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-transports-http</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>
</dependencies>

```

1.4. Appel de webServices REST en java via l'API "jax-rs 2"

Appel de webServices REST en java (vue d'ensemble des possibilités)

La **version 1** de la spécification **JAX-RS** a normalisé des annotations pour implémenter un Webservice REST coté serveur mais n'avait **rien prévu pour le coté client**.

Pour le coté client des services web REST en java on peut utiliser une **API** open-source du monde *apache* spécialisée dans les appels http (en modes GET, POST,PUT, DELETE) : **httpclient** (de httpcomponents).

La technologie "**jersey**" a dès les premières versions proposé (hors spécifications JAX-RS 1) une **api cliente** pour appeler des WS REST.

Plus récemment, **la version 2 des spécifications JAX-RS a normalisé** :

- une **api java cliente et standardisée** (pour invoquer WS REST)
- la notion d'**interface java de Webservice REST** (avec annotations JAX-RS) **que l'on peut réutiliser (par copie) du coté client** de façon à **générer dynamiquement un proxy "java/objet/rpc"** vers le webservice REST distant.

Appel de webServices REST en java via l'API cliente de JAX-RS 2

Exemple simple (utilisant l'api standardisée) :

```
import javax.ws.rs.client.Client; import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity; import javax.ws.rs.client.Invocation;
import javax.ws.rs.client.WebTarget;
...
Client jaxrs2client = ClientBuilder.newClient();

String calculateurRestUrl =
    "http://localhost:8080/wsCalculateur/services/rest/calculateur";
double a=5.0 , b=6.0;
System.out.println("appel de " + calculateurRestUrl + "/addition?a=5&b=6");
WebTarget additionTarget = jaxrs2client.target(calculateurRestUrl)
    .path("addition")
    .queryParam("a", a)
    .queryParam("b", b);

double resAdd = Double.parseDouble(
    additionTarget.request(MediaType.TEXT_PLAIN_TYPE)
        .get().readEntity(String.class));
System.out.println("\t 5+6=" + resAdd);
```

Appel de webServices REST en java via l'API cliente de JAX-RS 2 (suite)

Exemple en mode **POST** (au format **JSON**) :

```

Long pk=null;          Product savedProd =null;
Product newProd = new tp.data.Product(null,"prodXy",
                                "prod Xy with good features",12.89f);

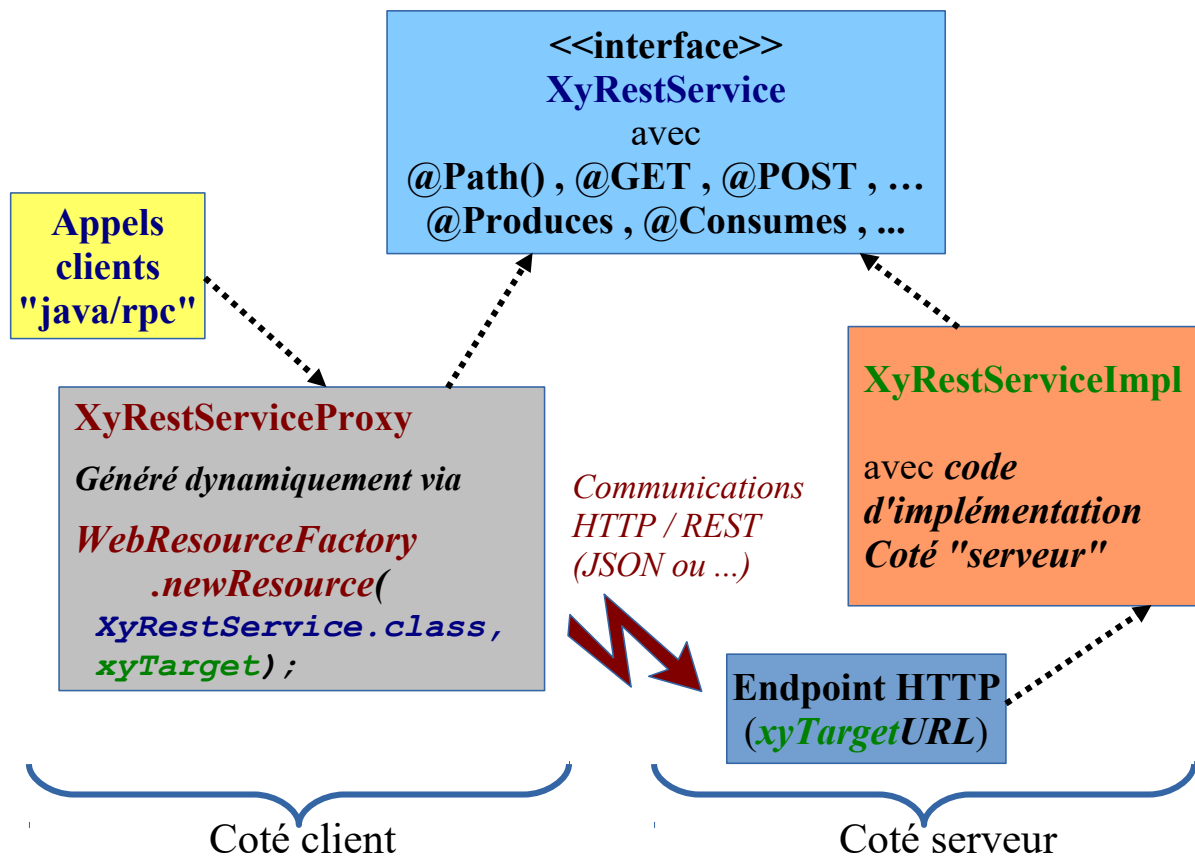
Client jaxrs2client = ClientBuilder.newClient().register(JacksonFeature.class);
WebTarget productsTarget = jaxrs2client.target(
    "http://localhost:8080/wsCalculateur/services/rest/json/products");

Response responseSaveNewProduct = productsTarget
    .path("/0") //0 or null as pk for "POST" as "save new /insert" (not "update")
    .request(MediaType.APPLICATION_JSON_TYPE)
    .post(Entity.entity(newProd, MediaType.APPLICATION_JSON_TYPE));

if(responseSaveNewProduct.getStatus() == 200 /*OK*/) {
    //String savedProductAsJsonString = responseSaveNewProduct.readEntity(String.class);
    savedProd = responseSaveNewProduct.readEntity(Product.class);
    pk=savedProd.getId();
    System.out.println("(saved) new product with auto_incremented pk = " + pk
        + "\n\t" + savedProd.toString());
} else {
    System.err.println(responseSaveNewProduct);
}

```

1.5. éventuelle logique d'appel RPC avec Proxy JAX-RS2

Appel de webServices REST avec **proxy JAX-RS 2** (principe)

Appel de webServices REST avec proxy JAX-RS 2 (exemple)

```

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import org.glassfish.jersey.client.proxy.WebResourceFactory;
import org.glassfish.jersey.jackson.JacksonFeature;
import tp.service.ProductRestJsonService; //interface avec @Annot JAX-RS2
...
Client jaxrs2client = ClientBuilder.newClient()
    .register(JacksonFeature.class);
WebTarget productsTarget = jaxrs2client.target(
    "http://localhost:8080/wsCalculateur/services/rest/json/products");

// create a new client proxy for the ProductRestJsonService :
ProductRestJsonService productRestJsonProxyService =
    WebResourceFactory.newResource(ProductRestJsonService.class,
    productsTarget);

Product prod1 = productRestJsonProxyService.getProductById(1L);
System.out.println("prod1:" + prod1.toString());

//Test creation/insert:
Product newProd = new tp.data.Product(null,"prodXy",
    "prod Xy with good features",12.89f);
savedProd = productRestJsonProxyService
    .saveOrUpdateProduct(0L, newProd); //0L for new to saved
System.out.println("new product saved : " + savedProd);

```


VII - WS REST via Spring-MVC

1. Présentation du framework "Spring MVC"

"Spring Web MVC" est une partie optionnelle du framework spring servant à gérer la logique du design pattern "MVC" dans le cadre d'une intégration "spring" .

"Spring MVC" est à voir comme un petit framework java/web (pour le coté serveur) qui peut être soit vu comme une alternative à Struts2 ou JSF2 soit être vu comme un petit framework web complémentaire à Struts2 ou JSF2.

Dépendances maven nécessaires :

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>
```

Configuration :

WEB-INF/web.xml

```
...

<servlet>
  <servlet-name>mvc-dispatcher</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/mvc-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>mvc-dispatcher</servlet-name>
  <url-pattern>/mvc/*</url-pattern>
</servlet-mapping>

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/classes/spring-mvc.xml,...</param-value>
</context-param>

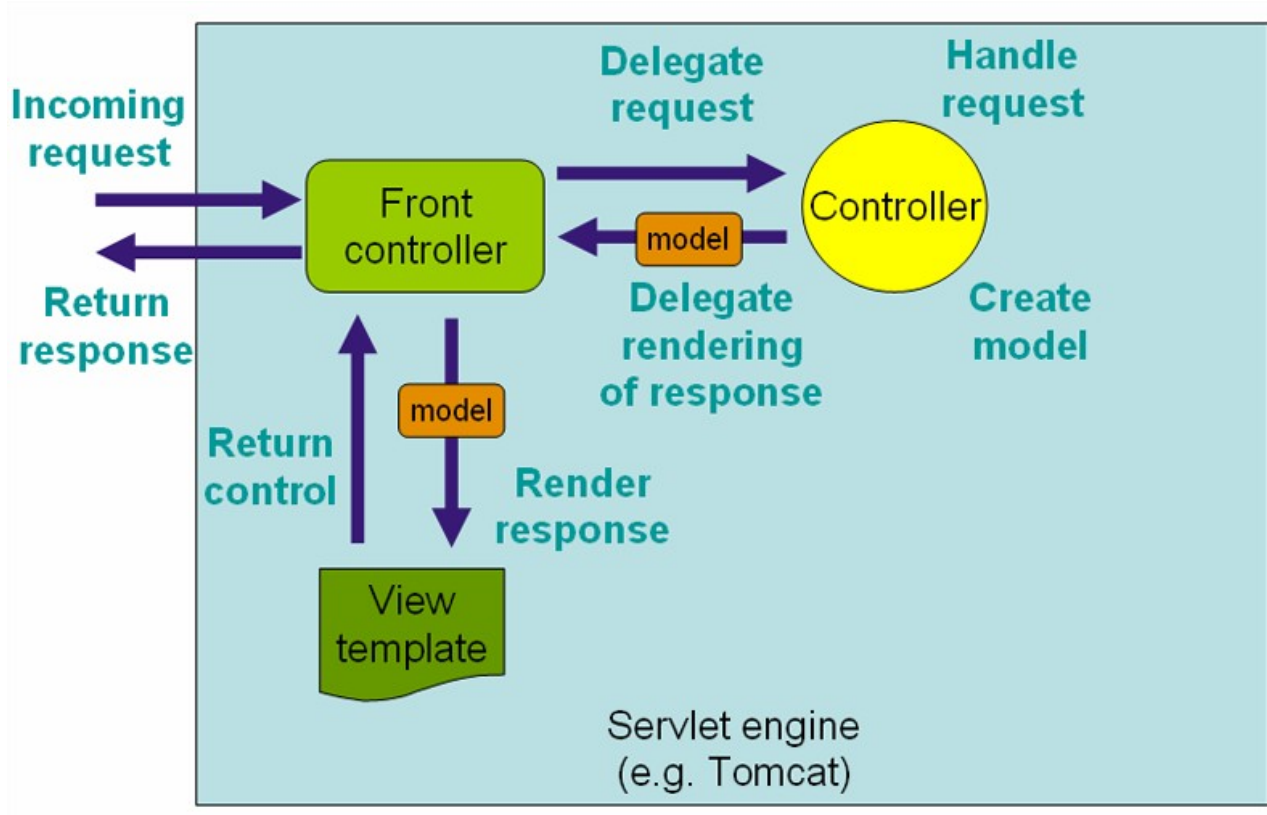
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
```

```
</listener>
```

WEB-INF/mvc-config.xml (spring mvc)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
  <context:component-scan base-package="tp.web.mvc.controller"/>
  <mvc:annotation-driven />
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- Example: a logical view name of 'showMessage' is mapped to
         '/WEB-INF/view/showMessage.jsp' -->
    <property name="prefix" value="/WEB-INF/view/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

Fonctionnement



1.1. Classe "controller"

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
```

@Controller

```
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World!");
        return "showMessage";
    }

}
```

Au niveau de showMessage.jsp, l'affichage de message pourra être effectué via \${message}.

1.2. éventuelle génération directe de la réponse HTTP

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller //but not "@Component" for spring web controller
@RequestMapping("/app")
public class WelcomeCtrl {

    @RequestMapping("/hello")
    @ResponseBody //si @ResponseBody , génération directe de la réponse ,
                  // sinon viewResolver (mvc-config.xml)
    String say_hello() {
        return "Hello World!";
    }

}
```

1.3. @RequestParam (accès aux paramètres HTTP)

conversion.jsp

```
... <form action="doConversion" method="GET_ou_POST">
    source: <select name="source" >
        <c:forEach var="d" items="${allDevises}" >
            <option value="${d.monnaie}" >${d.monnaie}</option>
        </c:forEach>
    </select> <br/>
    cible: <select name="cible" > ... </select> <br/>
    montant: <input name="montant" value="${montant}" /> <br/>
    <input type="submit" value="convertir" /> <br/>
</form>
sommeConvertie=<b>${sommeConvertie}</b> ...
```

```
@RequestMapping("/doConversion")
public String doConversion(Model model, @RequestParam(name="montant")double montant,
                                     @RequestParam(name="source")String monnaieSrc,
                                     @RequestParam(name="cible")String monnaieDest) {
....
    model.addAttribute("sommeConvertie",
                       gestionDevises.convertir(montant, monnaieSrc, monnaieDest));
    return "conversion";
}
```

2. Web services "REST" pour application Spring

Pour développer des Web Services "REST" au sein d'une application Spring , il y a deux possibilités distinctes (à choisir) :

- s'appuyer sur l'API standard **JAX-RS** et choisir une de ses implémentations (**CXF3** ou **Jersey** ou ...)
- s'appuyer sur le framework "**Spring web mvc**" et utiliser **@RestController** .

La version "JAX-RS standard" nécessite pas mal de librairies (jax-rs, jersey ou cxf , jackson et tout un tas de dépendances indirectes) .

La version spécifique spring nécessite un peu moins de librairies (spring-web , spring-mvc , jackson) et s'intègre mieux dans un écosystème spring (spring-security ,) .

Dépendances "maven" sans spring-boot :

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.2.3.RELEASE</version>
  <scope>compile</scope>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.10.2</version> <!-- to produces json -->
</dependency>

...
```

Dépendances "maven" indirecte (avec spring-boot) :

```
...
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Dans *application.properties* :

```
server.servlet.context-path=/webappXy ou ...
server.port=8181 ou 8080 ou ...
```

3. WS REST via Spring MVC et @RestController

L'annotation fondamentale **@RestController** (héritant de @Controller et de @Component) déclare que la classe*RestCtrl* correspond à l'implémentation "spring-mvc" d'un composant de l'application de type "Contrôleur de Web Service REST" .

On a par défaut @ResponseBody avec @RestController et cela signifie que la valeur de retour d'une des méthodes publiques du contrôleur sera quasi directement renvoyée au client http (sans passer par une page JSP ni un autre type de vue) .

Cependant , Lorsque la valeur de retour sera un *objet java* , *celui ci sera automatiquement transformé en JSON* (ou autre) avant d'être retourné au client http (ex : code js / appel ajax)

Exemple :

DeviseJsonRestCtrl.java

```
package tp.app.zz.web.rest;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
...
@RestController
@RequestMapping(value="/api-rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestCtrl {

    @Autowired //ou @Inject
    private GestionDevises gestionDevises; //internal business service or DAO

    //RECHERCHE UNIQUE selon RESOURCE-ID:
    //URL de déclenchement: .../webappXy/api-rest/devise/EUR
    @RequestMapping(value="{codeDevise}" , method=RequestMethod.GET)
    public Devise getDeviseByName(@PathVariable("codeDevise") String codeDevise) {
        return gestionDevises.getDeviseByPk(codeDevise);
    }

    //RECHERCHE MULTIPLE :
    //URL de déclenchement: webappXy/api-rest/devise
    //ou webappXy/api-rest/devise?name=euro
    @RequestMapping(value="", method=RequestMethod.GET)
    public List<Devise> getDevisesByCriteria(@RequestParam(value="name",required=false)
        String nomMonnaie) {
        if(nomMonnaie==null)
            return gestionDevises.getListeDevises();
        else{
            List<Devise> listeDev= new ArrayList<Devise>();
            Devise devise = gestionDevises.getDeviseByName(nomMonnaie);
            if(devise!=null) listeDev.add(devise);
            return listeDev;
        }
    }
}
```

NB :

@RequestParam avec **required=false** si paramètre **facultatif** en fin d'URL

Si l'ensemble de la classe java préfixée par @RestController comporte

@RequestMapping(value="..." , headers="Accept=application/json")

alors par défaut les valeurs en retour des méthodes publiques préfixées par **@RequestMapping** seront automatiquement converties au format **JSON** (en s'appuyant en interne sur la technologie *jackson-databind*) .

Techniquement possible mais très rare : retour direct d'une simple "String" (text/plain) :

```
//URL : webappXy/api-rest/devise/convert?amount=50&src=EUR&target=USD
@RequestMapping(value="/convert" , method=RequestMethod.GET ,
                headers="Accept=text/plain")
//@ResponseBody par défaut avec @RestController
String convert(@RequestParam("amount") double amount,
               @RequestParam("src") String src ,
               @RequestParam("target") String target) {
    double sommeConvertie=gestionDevises.convertir(amount, src, target);
    System.out.println("sommeConvertie="+sommeConvertie);
    return String.valueOf(sommeConvertie);
}
```

==> L'exemple ci-dessus est très déconseillé sur une api REST .

Un format de retour homogène (XML ou très souvent JSON) est en général attendu à la place .

Prise en charge des modes "PUT" , "POST" , "DELETE" :

NB : il est techniquement possible de convertir explicitement une "Json String" en objet java via l'api "jackson" comme le montre l'exemple inutilement long suivant (à ne pas reproduire , juste pour montrer certains mécanismes internes):

```
...
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMethod;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

@RestController
@RequestMapping(value="/api-rest/devises" , headers="Accept=application/json")
public class DeviseJsonRestController {
    ...
    @RequestMapping(value="", method=RequestMethod.PUT )
    Devise updateDevise(@RequestBody String deviseAsString) {
        Devise devise=null;
        try {
            ObjectMapper jacksonMapper = new ObjectMapper();
            jacksonMapper.configure(
                DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
            devise = jacksonMapper.readValue(deviseAsString,Devise.class);
            System.out.println("devise to update:" + devise);
            gestionDevises.updateDevise(devise);
            return devise;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    ...
}
```

Ceci dit , Spring-Mvc est capable d'effectuer de lui même automatiquement cette conversion.

L'écriture suivante (plus simple, à reproduire) assure les mêmes fonctionnalités :

```
@RestController
@RequestMapping(value="/api-rest/devise" , headers="Accept=application/json")
public class DeviseJsonRestController {
    ...
    @RequestMapping(value="", method=RequestMethod.PUT )
    Devise updateDevise(@RequestBody Devise devise) {
        System.out.println("devise to update:" + devise);
        gestionDevises.updateDevise(devise);
        return devise;
    } ....
}
```

NB : dans tous les cas , il sera souvent nécessaire de contrôler le comportement des "sérialisations/dé-sérialisations java <--> json" en incorporant certaines annotations de "jackson" au sein des classes de données (dto / payload) à véhiculer.

A ce sujet , l'annotation **@JsonIgnore** (sémantiquement équivalent à **@XmlTransient**) est assez souvent utile pour limiter la profondeur des données échangées .

Apport important de la version 4 : **ResponseEntity<T>**

Depuis "Spring4" , une méthode d'un web-service REST peut éventuellement retourner une réponse de Type **ResponseEntity<T>** ce qui permet de **retourner d'un seul coup**:

- un statut (OK , NOT_FOUND , ...)
- le corps de la réponse : objet (ou liste) T convertie en json
- un éventuel "header" (ex: url avec id si auto_incr lors d'un POST)

Exemple:

```
@RequestMapping(value="/{codeDev}" , method=RequestMethod.GET)
ResponseEntity<Devise> getDeviseByName(@PathVariable("codeDev") String codeDevise) {
    Devise dev = gestionDevises.getDeviseByPk(codeDevise);
    if(dev!=null)
        return new ResponseEntity<Devise>(dev, HttpStatus.OK);
    else
        return new ResponseEntity<Devise>(HttpStatus.NOT_FOUND);//404
}
```

ou bien

```
ResponseEntity< ?> getDeviseByName(...){
    ....
    else
        return new ResponseEntity<String> (" { \"err\" : \"devise not found\" } ",
            HttpStatus.NOT_FOUND) ;//404
}
```


Autre exemple (ici en mode DELETE) :

```
//url : http://localhost:8181/webappXy/api-rest/devise/EUR
@RequestMapping(value="/{codeDev}",method=RequestMethod.DELETE)
public ResponseEntity< ?> deleteDeviseByCode(@PathVariable("codeDev")String codeDevise){
    try {
        deviseDao.deleteDeviseBycode(codeDevise);
        return new ResponseEntity< ?>(HttpStatus.OK);
    } catch (Exception e) {
        e.printStackTrace(); //ou logger.error(e) ;
        return new ResponseEntity< ?>(HttpStatus.NOT_FOUND);
        //ou HttpStatus.INTERNAL_SERVER_ERROR
    }
}
```

NB : Bien que très finement paramétrable , un **return new ResponseEntity<?>** sera **généralement moins bien** qu'un un simple **throw new ...ClasseExceptionPréfixéePar_@ResponseStatus** plus simple et plus efficace (vu dans le paragraphe ci-après)

Eventuelles variations (simplifications):

@GetMapping(...) est équivalent à **@RequestMapping(... , method=RequestMethod.GET)**

@PostMapping(...) est équivalent à **@RequestMapping(... , method=RequestMethod.POST)**

@PutMapping(...) est équivalent à **@RequestMapping(... , method=RequestMethod.PUT)**

@DeleteMapping(...) équivalent à **@RequestMapping(..., method=RequestMethod.DELETE)**

3.1. Réponse et statut http par défaut en cas d'exception

Si une méthode d'un contrôleur REST remonte une exception java qui n'est pas rattrapée par un try/catch , la technologie Spring-Mvc retourne alors une réponse et un statut HTTP par défaut :

```
{ "timestamp" : 152....56,
  "status" : 500 ,
```

```
"error" : "Internal Server Error",
"exception" : "java.lang.NullPointerException",
"message" : ".....",
"path" : "/rest/devise/67573567" }
```

Le statut HTTP retourné par défaut dans l'entête de la réponse en cas d'exception est généralement **500** (INTERNAL_SERVER_ERROR) .

3.2. @ResponseStatus

Dans le cadre d'une remontée d'exception personnalisée il est possible de préciser le statut HTTP (pas systématiquement 500) qui sera remonté via l'annotation **@ResponseStatus** ()

Exemple :

```
@ResponseStatus(HttpStatus.NOT_FOUND) //404
public class MyEntityNotFoundException extends RuntimeException {
    public MyEntityNotFoundException() {
    }
    public MyEntityNotFoundException(String message) {
        super(message);
    }
    public MyEntityNotFoundException(Throwable cause) {
        super(cause);
    }
    public MyEntityNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
    ...
}
```

.../...

```
@RequestMapping(value="/{codeDevise}", method=RequestMethod.DELETE)
public void deleteDeviseByCode(@PathVariable("codeDevise")
                               String codeDevise) throws MyEntityNotFoundException {
    try {
```

```

        deviseService.deleteByCode(codeDevise);
    } catch (Exception e) {
        logger.error(e.getMessage());
        throw new MyEntityNotFoundException(
            "echec suppression devise pour codeDevise="+codeDevise ,e);
    }
}

```

Un appel HTTP avec une URL finissant (avec une erreur ici volontaire) par `"/devise/EURy"`

---> renvoie **404** et un message d'erreur au format JSON/spring-Web-MCV HOMOGENE :

```

{
  "timestamp": "2020-02-03T17:23:45.888+0000",
  "status": 404,
  "error": "Not Found",
  "message": "echec suppression devise pour codeDevise=EURy",
  "trace": "org.mycontrib.backend.exception.MyEntityNotFoundException:.....",
  "path": "/spring-boot-backend/rest/devise-api/private/role_admin/devise/EURy"
}

```

Dans le cadre d'un échec de validation de la requête avec `@Valid` sur le paramètre d'entrée d'une méthode d'un contrôleur REST et avec des annotations de `javax.validation` (`@Min` , `@Max` , ...) sur la classe du "DTO" (ex : `Devise`) , le statut HTTP alors automatiquement remonté dans l'entête de la réponse HTTP est **400 (Bad Request)** et le corps de la réponse comporte tous les détails sur les éléments invalides .

```

public ResponseEntity<Void> ajouterDevise(@Valid @RequestBody Devise devise) {
    ....
}

```

```

public class Devise{
    ...
    @Length(min=3, max=20, message = "Nom trop long ou trop court")
    private String nom;
}

```

3.3. ResponseEntityExceptionHandler (très bien)

ApiError.java (DTO for custom error message)

```

package tp.appliSpring.dto;

import java.time.LocalDateTime;
import org.springframework.http.HttpStatus;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @ToString
public class ApiError {

    private HttpStatus status;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy hh:mm:ss")
    private LocalDateTime timestamp;
    private String message;
    private String debugMessage;
    //private List<ApiSubError> subErrors;

    public ApiError() {
        timestamp = LocalDateTime.now();
    }

    public ApiError(HttpStatus status) {
        this();
        this.status = status;
    }

    public ApiError(HttpStatus status, Throwable ex) {
        this();
        this.status = status;
        this.message = "Unexpected error";
        this.debugMessage = ex.getLocalizedMessage();
    }

    public ApiError(HttpStatus status, String message, Throwable ex) {
        this();
        this.status = status;
        this.message = message;
        this.debugMessage = ex.getLocalizedMessage();
    }
}

```

RestResponseEntityExceptionHandler.java

```

package tp.appliSpring.web.rest;

import org.springframework.http.HttpHeaders;

```

```

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import tp.appliSpring.core.exception.ConflictException;
import tp.appliSpring.core.exception.NotFoundException;
import tp.appliSpring.dto.ApiError;

```

@ControllerAdvice

```

public class RestResponseExceptionHandler
    extends ResponseEntityExceptionHandler {

```

```

    private ResponseEntity<Object> buildResponseEntity(ApiError apiError) {
        return new ResponseEntity<>(apiError, apiError.getStatus());
    }

```

@Override

```

    protected ResponseEntity<Object>
handleHttpMessageNotReadable(HttpMessageNotReadableException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {
        String error = "Malformed JSON request";
        return buildResponseEntity(new ApiError(HttpStatus.BAD_REQUEST, error, ex));
    }

```

@ExceptionHandler(NotFoundException.class)

```

    protected ResponseEntity<Object> handleEntityNotFound(
        NotFoundException ex) {
        return buildResponseEntity(new ApiError(HttpStatus.NOT_FOUND, ex));
    }

```

@ExceptionHandler(ConflictException.class)

```

    protected ResponseEntity<Object> handleConflict(
        ConflictException ex) {
        return buildResponseEntity(new ApiError(HttpStatus.CONFLICT, ex));
    }

```

```

}

```

Et grace à cela les exceptions java retournées par les services et contrôleurs REST :

- n'ont plus besoin d'être décorées par @ResponseStatus → meilleurs séparation des couches
- seront automatiquement transformées en messages très personnalisés et accompagnés du bon statut HTTP .

3.4. Exemples d'appels en js/ajax

js/ajax-util.js

```

//fonction utilitaire pour preparer xhr en vu d'effectuer juste apres un appel ajax en mode Get ou post ou ...

```

```

function initXhrWithCallback(callback,errCallback){
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4){
            if (xhr.status == 200 || xhr.status == 0) {
                callback(xhr.responseText,xhr);
            }
            else {
                errCallback(xhr);
            }
        }
    };
    return xhr;
}

function xhrStatusToErrorMessage(xhr){
    var errMsg = "ajax error";//by default
    var detailsMsg=""; //by default
    console.log("xhr.status="+xhr.status);
    if(xhr.responseText!=null)
        detailsMsg = xhr.responseText;
    switch(xhr.status){
        case 400 :
            errMsg = "Server understood the request, but request content was invalid."; break;
        case 401 :
            errMsg = "Unauthorized access (401)"; break;
        case 403 :
            errMsg = "Forbidden resource can't be accessed (403)"; break;
        case 404 :
            errMsg = "resource not found (404)"; break;
        case 500 :
            errMsg = "Internal server error (500)"; break;
        case 503 :
            errMsg = "Service unavailable (503)"; break;
    }
    return errMsg+" "+detailsMsg;
}

function makeAjaxGetRequest(xhr,url) {
    xhr.open("GET", url, true);
    xhr.send(null);
}

```

```
function makeAjaxPostRequest(xhr,url,jsonData) {
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    //pour re-vehiculer (si necessaire) un jeton d'authentification (jwt ou pas):
    var authToken = sessionStorage.getItem("authToken");
    if(authToken != null ){
        xhr.setRequestHeader('Authorization','Bearer '+ authToken);
    }
    xhr.send(jsonData);
}
```

username :

password :

roles :

login successful with roles=admin

login.html

```
<html>
<head> <title>login</title><script src="js/ajax-util.js"></script>  <script src="js/login.js"></script>
</head>
<body>
    <h3> login (ws security) </h3>
    username : <input id="txtUsername" type='text' value="admin1"/><br/>
    password : <input id="txtPassword" type='text' value="pwdadmin1"/><br/>
    roles : <input id="txtRoles" type='text' value="admin"/><br/>
    <input type='button' value="login" id="btnLogin"/> <br/>
    <span id="spanMsg"></span> <br/>
    <hr/> <a href="index.html">retour vers index.html</a>
</body>
</html>
```

js/login.js

```
window.onload=function(){
    var spanMsg = document.querySelector('#spanMsg');
    var btnLogin=document.querySelector('#btnLogin');
    btnLogin.addEventListener("click" , function (){
        var auth = { username : null, password : null , roles : null } ;
```

```
auth.username = document.querySelector('#txtUsername').value;
auth.password = document.querySelector('#txtPassword').value;
auth.roles = document.querySelector('#txtRoles').value;

var cbLogin = function(data,xhr){
    console.log(data); //data as json string;
    var authResponse = JSON.parse(data);
    if(authResponse.status){
        spanMsg.innerHTML=authResponse.message + " with roles=" + authResponse.roles;
        //localStorage.setItem("authToken",authResponse.token);
        sessionStorage.setItem("authToken",authResponse.token);
    }else{
        spanMsg.innerHTML=authResponse.message ;
    }
} //end of cbLogin

var cbError = function(xhr){
    spanMsg.innerHTML= xhrStatusToErrorMessage(xhr) ;
}

var xhr = initXhrWithCallback(cbLogin,cbError);
makeAjaxPostRequest(xhr,"./api-rest/login-api/public/auth" , JSON.stringify(auth));

}); //end of btnLogin.addEventListener/click
} //end of window.onload
```


recherche devises selon taux mini (public)

changeMini :

- Euro , 1
- Dollar , 1.1243
- Yen , 121.6477

ajout de monnaie (after logging as ADMIN)

codeMonnaie: (ex: EUR,USD,...)
 nommonnaie: (ex: euro,dollar,...)
 tauxChange: (ex: 1, 0.85 , 1.5, ...)

 {"code":"ms","name":"monnaieSinge","change":1.23456}

appel_ajax.html

```
<html>
<head>
    <script src="js/ajax-util.js"></script>    <script src="js/appelAjax.js"></script>
    <meta charset="UTF-8"> <title>appel_ajax</title>
</head>
<body>
    <h3>recherche devises selon taux mini (public)</h3>
    changeMini : <input type="text" id="txtChangeMini" value="1" /> <br/>
                <input type="button" value="getDevises" id="btnGetDevises" /> <br/>
    <div id="divRes"></div>

    <h3> ajout de monnaie (after logging as ADMIN)</h3>
    codeMonnaie: <input type="text" id="txtCode" value="ms" /> (ex: EUR,USD,...)<br/>
    nommonnaie: <input type="text" id="txtName" value="monnaieSinge" /> (ex: euro,dollar,...)<br/>
    tauxChange: <input type="text" id="txtChange" value="1.23456" /> (ex: 1, 0.85 , 1.5, ... )<br/>
    <input type="button" id="btnPostDevises" value="sauvegarder devise" /> <br/>
    <div id="divMessage"></div>
    <hr/>
    <a href="index.html">retour index.html</a>
</body>
</html>
```

js/appelAjax.js

```

window.onload=function(){
    var inputChangeMini = document.querySelector("#txtChangeMini");
    var btnGetDevises = document.querySelector("#btnGetDevises");
    var btnPostDevise = document.querySelector("#btnPostDevise");
    var divRes = document.querySelector("#divRes");
    var divMessage = document.querySelector("#divMessage");
    var cbError = function(xhr){
        divMessage.innerHTML= xhrStatusToErrorMessage(xhr) ;
    }
    btnGetDevises.addEventListener("click" , function (){
        var changeMini = inputChangeMini.value;
        var cbAffDevises=function(texteReponse,xhr){
            //divRes.innerHTML = texteReponse;
            var listeDeviseJs = JSON.parse(texteReponse /* au format json string */)
            var htmlListeDevises = "<ul>" ;
            for(i=0; i<listeDeviseJs.length ; i++){
                htmlListeDevises = htmlListeDevises + "<li>" + listeDeviseJs[i].name + " , "
                + listeDeviseJs[i].change + "</li>";
            }
            htmlListeDevises = htmlListeDevises + "</ul>";
            divRes.innerHTML= htmlListeDevises;
        }
        var xhr = initXhrWithCallback(cbAffDevises , cbError);
        makeAjaxGetRequest(xhr,"./api-rest/devise-api/public/devise?changeMini="+changeMini );
    });//end of btnGetDevises.addEventListener/"click"

    btnPostDevise.addEventListener("click" , function (){
        var nouvelleDevise = { code : null, name : null, change : null };
        nouvelleDevise.code = document.querySelector("#txtCode").value;
        nouvelleDevise.name = document.querySelector("#txtName").value;
        nouvelleDevise.change = document.querySelector("#txtChange").value;
        var cbGererResultatPostDevise = function (texteReponse,xhr){
            divMessage.innerHTML= texteReponse;
        }
        var xhr = initXhrWithCallback(cbGererResultatPostDevise, cbError);
        makeAjaxPostRequest(xhr,"./api-rest/devise-api/private/role_admin/devise" ,
            JSON.stringify(nouvelleDevise));
    });//end of btnGetDevises.addEventListener/"click"
} //end of window.onload

```

3.5. Invocation java de service REST via RestTemplate de Spring

Utile pour une **délégation de service** ou bien pour un **test d'intégration** (automatisable via maven et intégration continue).

```

.....
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.client.RestTemplate;

/* cette classe à un nom qui commence ou se termine par IT (et par par Test)
 * car c'est un Test d'Integration qui ne fonctionne que lorsque toute l'application
 * est entièrement démarrée (avec EmbeddedTomcat ou équivalent) .*/
public class PersonWsRestIT {

    private static Logger logger = LoggerFactory.getLogger(PersonWsRestIT.class);

    private static RestTemplate restTemplate; //objet technique de Spring pour test WS REST

    //pas de @Autowired ni de @RunWith
    //car ce test EXTERNE est censé tester le WebService sans connaître sa structure interne
    // (test BOITE_NOIRE)
    @BeforeClass
    public static void init(){
        restTemplate = new RestTemplate();
    }

    @Test
    public void testGetSpectacleById(){
        final String BASE_URL =
            "http://localhost:8888/spring-boot-spectacle-ws/spectacle-api/public";
        final String uri = BASE_URL + "/spectacle/1";
        String resultAsString = restTemplate.getForObject(uri, String.class);
        logger.info("json string of spectacle 1 via rest: " + resultAsString);
    }

```

```

        Spectacle s1 = restTemplate.getForObject(uri, Spectacle.class);
        logger.info("spectacle 1 via rest: " + s1);
        Assert.assertTrue(s1.getId()==1L);
    }

    @Test
    public void testListeComptesDuClient(){
        final String villeDepart = "Paris";
        final String dateDepart = "2018-09-20";
        final String uri = "http://localhost:8080/flight_web/mvc/rest/vols/byCriteria"
            + "?villeDepart=" + villeDepart + "&dateDepart=" + dateDepart;
        String resultAsString = restTemplate.getForObject(uri, String.class);
        logger.info("json listeVols via rest: " + resultAsString);
        Vol[] tabVols = restTemplate.getForObject(uri, Vol[].class);
        logger.info("java listeComptes via rest: " + tabVols.toString());
        Assert.assertNotNull(tabVols); Assert.assertTrue(tabVols.length>=0);
        for(Vol cpt : tabVols){
            System.out.println("\t" + cpt.toString());
        }
    }

    @Test
    public void testVirement(){
        final String uri =
            "http://localhost:8080/tpSpringWeb/mvc/rest/compte/virement";
        //post/envoi:
        OrdreVirement ordreVirement = new OrdreVirement();
        ordreVirement.setMontant(50.0);
        ordreVirement.setNumCptDeb(1L);
        ordreVirement.setNumCptCred(2L);
        OrdreVirement savedOrdreVirement =
            restTemplate.postForObject(uri, ordreVirement, OrdreVirement.class);
        logger.info("savedOrdreVirement via rest: " + savedOrdreVirement.toString());
        Assert.assertTrue(savedOrdreVirement.getOk().equals(true));
    }
}

```

Exemple 2 (délégation de service) :

```
...
import java.nio.charset.Charset;
import java.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping(value="/myapi/auth" , headers="Accept=application/json")
public class LoginDelegateCtrl {

    private static Logger logger = LoggerFactory.getLogger(LoginDelegateCtrl.class);

    private static final String ACCESS_TOKEN_URL =
        "http://localhost:8081/basic-oauth-server/oauth/token";

    private static RestTemplate restTemplate = new RestTemplate();

    HttpHeaders createBasicHttpAuthHeaders(String username, String password){
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        String auth = username + ":" + password;
```

```

byte[] encodedAuth = Base64.getEncoder().encode(
    auth.getBytes(Charset.forName("US-ASCII")) );
String authHeader = "Basic " + new String( encodedAuth );
headers.add("Authorization", authHeader);
return headers;
}

@PostMapping("/login")
public ResponseEntity<?> authenticateUser(@RequestBody AuthRequest loginRequest) {
    logger.debug("/login , loginRequest:"+loginRequest);
    String authResponse="{}";
    try{
        MultiValueMap<String, String> params= new LinkedMultiValueMap<String,
String>();
        params.add("username", loginRequest.getUsername());
        params.add("password", loginRequest.getPassword());
        params.add("grant_type", "password");
        //ResponseEntity<String> tokenResponse =
        //      restTemplate.postForEntity(ACCESS_TOKEN_URL,params, String.class);
        // si pas besoin de spécifier headers spécifique .

        HttpHeaders headers = createBasicHttpAuthHeaders("fooClientIdPassword","secret");
        HttpEntity<MultiValueMap<String, String>> entityReq =
            new HttpEntity<MultiValueMap<String, String>>(params, headers);

        ResponseEntity<String> tokenResponse=
            restTemplate.exchange(ACCESS_TOKEN_URL,
                                HttpMethod.POST,
                                entityReq,
                                String.class);

        authResponse=tokenResponse.getBody();
        logger.debug("/login authResponse:" + authResponse.toString());
        return ResponseEntity.ok(authResponse);
    }
    catch (Exception e) {
        logger.debug("echec authentification:" + e.getMessage()); //for log
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body(authResponse);
    }
}

```

```

    }
}
}

```

3.6. Appel moderne/asynchrone de WS-REST avec WebClient

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <!-- pour appel de WS-REST externes , WebClient mieux que RestTemplate -->
</dependency>

```

RestClientApp.java

```

package tp.appliSpring.client;

import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;
import tp.appliSpring.dto.Currency; import tp.appliSpring.dto.LoginRequest;
import tp.appliSpring.dto.LoginResponse;

public class RestClientApp {

    public static String token="?";

    public static void main(String[] args) {
        postLoginForToken();
        posterNouvelleDevise();
    }

    private static void postLoginForToken() {
        WebClient.Builder builder = WebClient.builder();
        String baseUrl="http://localhost:8080/appliSpring/api-bank";
        WebClient webClient = builder
            .baseUrl(baseUrl)
            .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
            .build();

        LoginRequest loginRequest = new LoginRequest("admin1","pwd1");

        //envoyer cela via un appel en POST
        Mono<LoginResponse> reactiveStream = webClient.post().uri("/public/login")
            .body(Mono.just(loginRequest), LoginRequest.class)
            .retrieve()
            .bodyToMono(LoginResponse.class)
            .onErrorReturn(new LoginResponse("admin1",false,"login failed",null));
        LoginResponse loginResponse = reactiveStream.block();
    }
}

```

```

System.out.println("loginResponse=" + loginResponse.toString());
if(loginResponse.getOk())
    token = loginResponse.getToken();
}

private static void posterNouvelleDevise() {
    WebClient.Builder builder = WebClient.builder();
    String baseUrl="http://localhost:8080/appliSpring/api-bank";
    WebClient webClient = builder
        .baseUrl(baseUrl)
        .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
        .defaultHeader(HttpHeaders.AUTHORIZATION, "Bearer " + token)
        .build();

    //créer une instance du DTO Currency
    //avec les valeurs
    //{ "code" : "DDK", "name" : "couronne danoise", "rate" : 7.77 }

    Currency currencyDDK = new Currency("DDK","couronne danoise" , 7.77);

    //envoyer cela via un appel en POST
    Mono<Currency> reactiveStream = webClient.post().uri("/devise")
        .body(Mono.just(currencyDDK), Currency.class)
        .retrieve()
        .bodyToMono(Currency.class)
        .onErrorReturn(new Currency("?", "not saved !!", 0.0));

    Currency savedCurrency = reactiveStream.block();

    System.out.println("savedCurrency=" + savedCurrency.toString());
}

```

Variantes pour appel(s) en mode GET :

```

private String tempApiKey="26ca93ee7fc19cbe0a423aaa27cab235";
private String fixerApiUrl="http://data.fixer.io/api/latest"
    + "?access_key="+tempApiKey; //apiKey may be passed in header with other api

/*
Mono<String> reactiveStream = webClient.get()
    .retrieve()
    .bodyToMono(new ParameterizedTypeReference<String>() {});
String result = reactiveStream.block();
System.out.println("result="+result);
*/
//type de réponse brute attendue:
/*
{"success":true,"timestamp":1635959583,"base":"EUR","date":"2021-11-03",
"rates":{"AED":4.254663,"AFN":105.467869,..., "EUR":1 , ...}}
*/
Mono<FixerIoResponse> reactiveStream = webClient.get() //uri("/suiteUrlQuiVaBien")
    .retrieve()
    .bodyToMono(new ParameterizedTypeReference<FixerIoResponse>() {});
FixerIoResponse fixerIoResponse = reactiveStream.block();

```



```

/*
ResponseEntity<FixerIoResponse> fixerIoResponseEntity=
    webClient.get().retrieve()
        .toEntity(FixerIoResponse.class).block();

FixerIoResponse fixerIoResponse = null;
if(fixerIoResponseEntity.getStatusCode()==HttpStatus.OK) {
    fixerIoResponse=fixerIoResponseEntity.getBody();
}
*/

```

3.7. Test d'un "RestController" via MockMvc

Pour tester le comportement d'un composant "RestController" de Spring-Mvc sans avoir à préalablement démarrer l'application complète, on peut utiliser la classe **MockMvc** et l'annotation **@WebMvcTest** ou bien **@AutoConfigureMockMvc** qui sont spécialement prévues pour faire fonctionner le code d'un web service rest de spring-mvc en recréant un contexte local ayant à peu près de même comportement que celui d'un conteneur web mais sans accès réseau/http .

Deux Grandes Variantes :

- via **@WebMvcTest** : **test unitaire** avec mock de service interne
- via **@SpringBootTest** et **@AutoConfigureMockMvc** : **test d'intégration** avec réels services

3.8. Test unitaire de contrôleur Rest

```

package tp.appliSpring.rest;

import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import java.util.ArrayList;      import java.util.List;
import org.junit.jupiter.api.BeforeEach;    import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;    import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import tp.appliSpring.entity.Compte;    import tp.appliSpring.service.CompteService;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@WebMvcTest(CompteRestCtrl.class)
//NB: @WebMvcTest without security and without service layer , service must be mocked !!!
public class TestCompteRestCtrlWithServiceMock {

```

```
@Autowired
private MockMvc mvc;
```

```
@MockBean
private CompteService compteService; //not real implementation but mock to configure .
```

```
@BeforeEach
public void reInitMock() {
    //vérification que le service injecté est bien un mock
    assertTrue(Mockito.mockingDetails(compteService).isMock());
    //reinitialisation du mock(de scope=Singleton par défaut) sur aspects stub et spy
    Mockito.reset(compteService);
}
```

```
@Test //à lancer sans le profile withSecurity
public void testComptesDuClient1WithMockOfCompteService(){

    //préparation du mock (qui sera utilisé en arrière plan du contrôleur rest à tester):
    List<Compte> comptes = new ArrayList<>();
    comptes.add(new Compte(1L,"compteA",40.0));
    comptes.add(new Compte(2L,"compteB",90.0));
    Mockito.when(compteService.comptesDuClient(1)).thenReturn(comptes);

    try {
        MvcResult mvcResult =
            mvc.perform(get("/api-bank/compte?numClient=1")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(2) ))
                .andExpect(jsonPath("$.label", is("compteA") ))
                .andExpect(jsonPath("$.solde", is(90.0) ))
                .andReturn();
        System.out.println(">>>>>>>>> jsonResult="
            +mvcResult.getResponse().getContentAsString());
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

NB : Spring5 propose une variante @WebFluxTest et WebTestClient pour WebFlux .

3.9. Test d'intégration de contrôleur Rest avec réels services

```

package tp.appliSpring.rest;
import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@SpringBootTest //with all layers
@AutoConfigureMockMvc //to test controller with reals spring services implementations
@ActiveProfiles({"embeddedDb","init"}) //init profile for ...init.ReinitDefaultDataSet
public class TestCompteRestCtrlWithRealService {

    @Autowired
    private MockMvc mvc;

    @Test //à lancer sans le profile withSecurity
    public void testComptesDuClient1WithRealService() {
        try {
            MvcResult mvcResult =
                mvc.perform(get("/api-bank/compte?numClient=1")
                    .contentType(MediaType.APPLICATION_JSON))
                    .andExpect(status().isOk())
                    .andExpect(jsonPath("$", hasSize(2) ))
                    .andExpect(jsonPath("$[0].label", is("compteA") ))
                    .andReturn();
            //à adapter selon jeux de données de init.ReinitDefaultDataset
            System.out.println(">>>>>>>>> jsonResult="+
                mvcResult.getResponse().getContentAsString());
        } catch (Exception e) {
            System.err.println(e.getMessage());
            //e.printStackTrace();
        }
    }
}

```

VIII - Sécurisation WS REST, Api_key , JWT , ...

1. Api Key

Un web service hébergé par une entreprise et rendu accessible sur internet a un certain coût de fonctionnement (courant électrique , serveurs ,) .

Pour limiter des abus (ex : appel en boucle) ou bien pour obtenir un paiement en contre partie d'une bonne qualité de service , un web service public est souvent invocable que si l'on renseigne une "api_key" (au niveau de l'URL ou bien au niveau de l'entête la requête HTTP).

Une "api_key" est très souvent de type "**uuid/guid**" .

Critères d'une api_key :

- lié à un abonnement (gratuit ou payant) , ex : compte utilisateur / compte d'entreprise
- ne doit idéalement pas être diffusé (à garder secret)
- souvent lié à un compteur d'invocations (limite selon prix d'abonnement)
- doit pouvoir être administré (régénéré si perdu/volé , ...)
et les modifications doivent pouvoir être immédiatement ou rapidement prises en compte.

Exemple :

Le site **https://fixer.io** héberge un web service REST permettant de récupérer les taux de change (valeurs de "USD" , "GBP" , "JPY" , ... vis à vis de "EUR" par défaut).

Début 2018, ce web service était directement invocable sans "api_key" .

Courant 2018, ce web service est maintenant invocable qu'avec une "**api_key**" **liée à un compte utilisateur** "gratuit" ou bien "payant" selon le mode d'abonnement (options, fréquence d'invocation,).

URL d'appel sans "api_key" : **http://data.fixer.io/api/latest**

Réponse :

```
{  
  "success":false,  
  "error":{"code":101,"type":"missing_access_key",  
    "info":"You have not supplied an API Access Key. [Required format:  
      access_key=YOUR_ACCESS_KEY]"
```

```
}
}
```

URL d'invocation avec api_key valide :

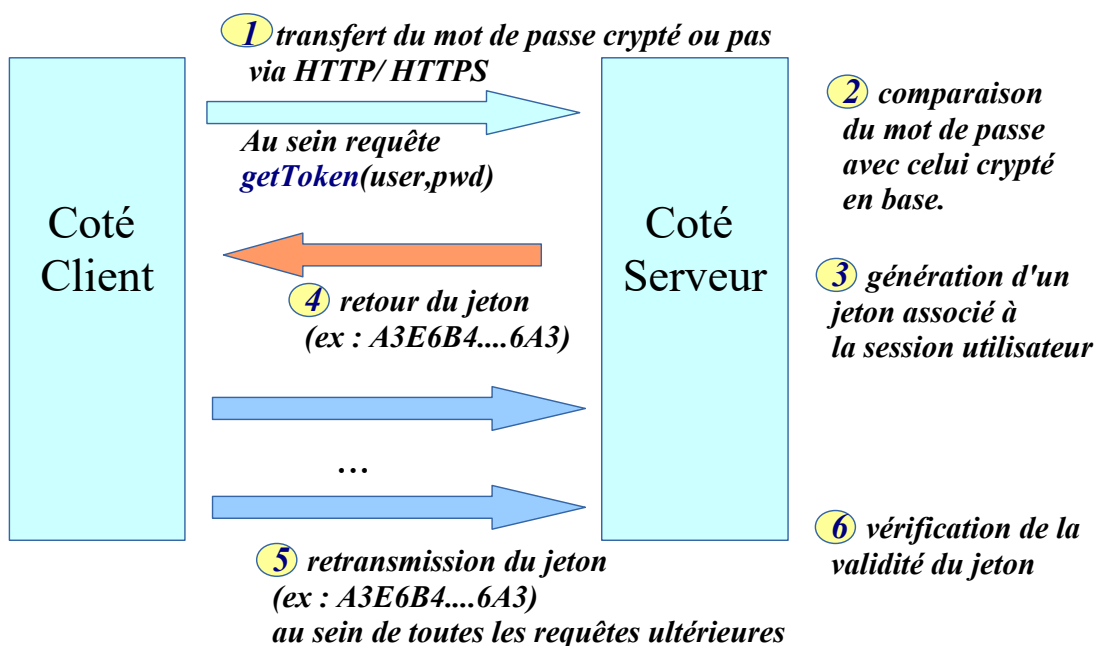
http://data.fixer.io/api/latest?access_key=26ca93ee7.....aaa27cab235

```
{
  "success":true, "timestamp":1538984646, "base":"EUR", "date":"2018-10-08",
  "rates":
  {"AED":4.224369,...,"DKK":7.460075,"DOP":57.311592,"DZD":136.091172,"EGP":20.596249,
  "ERN":17.250477,"ETB":31.695652,"EUR":1,"FJD":2.46956,"FKP":0.88584,"GBP":0.879667,.
  ..., "JPY":130.858498,...,"USD":1.15005,...,"ZWL":370.724343}
}
```

2. Token d'authentification

2.1. Tokens : notions et principes

Jeton ("token") d'authentification valide le temps d'une session utilisateur



Plusieurs sortes de jetons/tokens

Il existe plusieurs sortes de jetons (normalisés ou pas).

Dans le cas le plus simple, un **jeton est généré aléatoirement** (ex : **uuid** ou ...) et sa **validation consiste essentiellement à vérifier son existence** en tentant de le récupérer quelque part (*en mémoire ou en base*) et éventuellement à vérifier une date et heure d'expiration.

JWT (Json **W**eb **T**oken) est un **format particulier de jeton** qui **comporte 3 parties** (une entête technique , un paquet d'informations en clair (ex : username , email , expiration, ...) au format JSON et une signature qui ne peut être vérifiée qu'avec la clef secrète de l'émetteur du jeton.

2.2. Bearer Token (au porteur) / normalisé HTTP

Bearer token (jeton au porteur) et transmission

Le champ **Authorization:** normalisé d'une entête d'une requête HTTP peut comporter une valeur de type **Basic ...** ou bien **Bearer ...**

Le terme anglais "**Bearer**" signifiant "**au porteur**" en français indique que la simple possession d'un jeton valide par une application cliente devrait normalement , après transmission HTTP, permettre au serveur d'autoriser le traitement d'une requête (après vérification de l'existence du jeton véhiculé parmi l'ensemble de ceux préalablement générés et pas encore expirés).

NB: Les "bearer token" sont utilisés par le protocole "O2Auth" mais peuvent également être utilisés de façon simple sans "O2Auth" dans le cadre d'une authentification "sans tierce partie" pour API REST.

NB2 : un "bearer token" peut éventuellement être au format "JWT" mais ne l'est pas toujours (voir rarement) en fonction du contexte.

2.3. JWT (Json Web Token)



Structure jeton "JWT / Json Web Token"



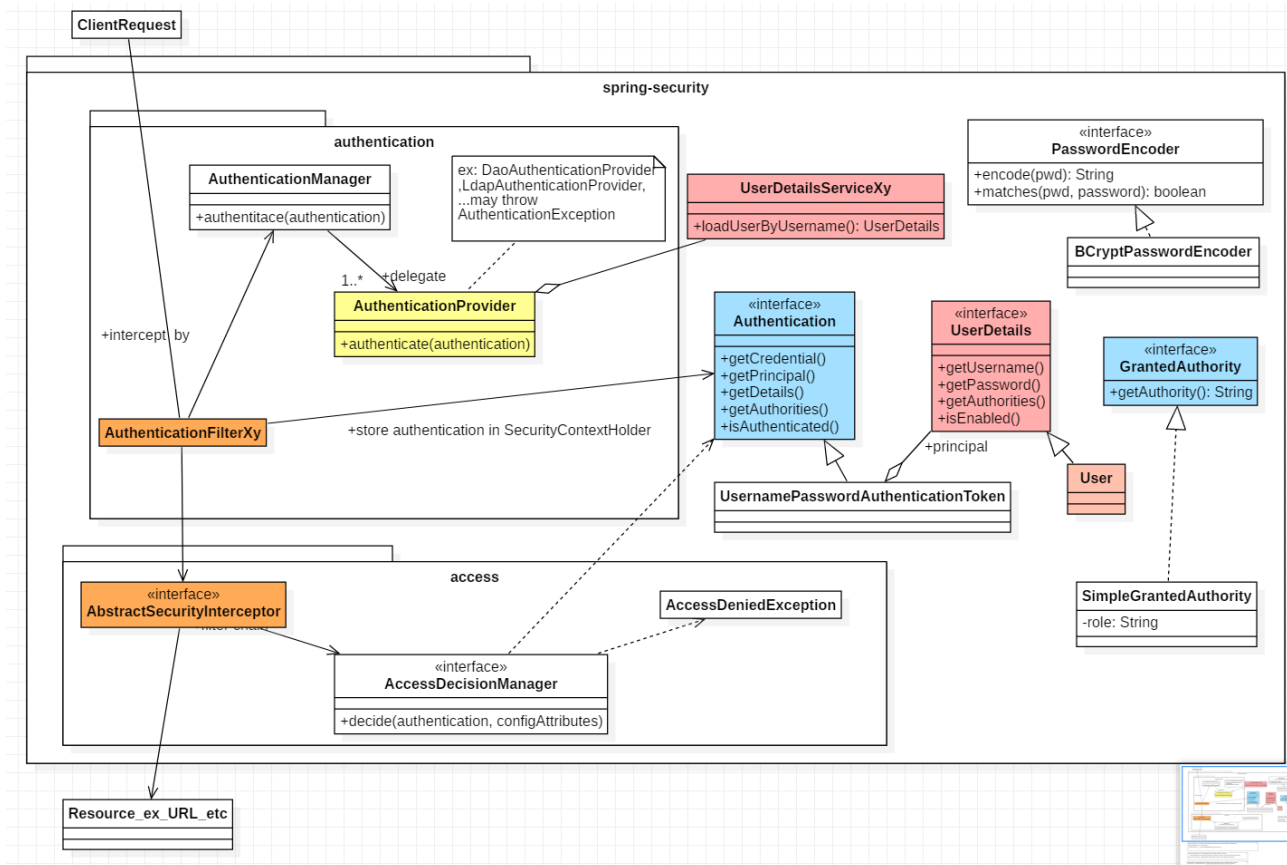
Example:

[eyJhbGeiOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0b3B0YWVwYyY29tIiwiaXhwLjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLnNvbS9qd3RfY2xhaW1zL2l2X2FkbWluIjp0cnVLJCJyb2lwYW55IjoieGV9dGFslwiYXdlc29tZSI6dHJlZX0.yRQYNwZskCZUXPwaQupWkiUzKELZ49eM7oWxAOK_ZXw](#)

NB: "iss" signifie "issuer" (émetteur) , "iat" : issue at time
 "exp" correspond à "date/heure expiration" . Le reste du "payload"
 est libre (au cas par cas) (ex : "company" et/ou "email" , ...)

3. Jeton JWT avec Spring-Security

3.1. Vue d'ensemble sur "Spring-security"



NB: https://en.wikipedia.org/wiki/Spring_Security comporte un assez bon schéma montrant les mécanismes fondamentaux de spring-security .

Lorsqu'une requête HTTP (de "login" ou autre) arrive , celle-ci est interceptée par un filtre web (prédéfini ou bien personnalisé) .

Ce filtre web va alors appeler la méthode `authenticate()` sur un objet de type "AuthenticationProvider" géré par un "AuthenticationManager" .

Authentication *authenticate*(Authentication authentication) throws AuthenticationException;

avant appel : authentication avec `getPrincipal()` retournant souvent username (String)

`getCredential()` retournant password ou autre .

après appel : authentication avec `getPrincipal()` retournant UserDetails si ok ou bien AuthenticationException sinon

En interne l'objet "AuthenticationProvider" s'appuie sur une implémentation de l'interface **UserDetailsService** avec cette unique méthode :

UserDetails *loadUserByUsername*(String username) throws UsernameNotFoundException;

Cette méthode est censée remonter les données d'un compte utilisateur depuis un certain endroit (base de données , LDAP ,) .

Ces infos "utilisateur" doivent être une implémentation de l'interface "UserDetails" (classe "User"

par exemple). L'objet "User" (ou un équivalent implémentant "UserDetails") est censée comporter le bon mot de passe.

Les mécanismes internes de Spring-security ("AuthenticationProvider" , ...) vont alors pouvoir comparer le bon mot de passe avec celui renseigné par l'utilisateur qui souhaite s'authentifier.

Dans certains cas la comparaison passe par une implémentation de "PasswordEncoder" (ex : "BCryptPasswordEncoder") lorsque les mots de passe sont cryptés dans la base de données.

Si l'authentification échoue --> AuthenticationException --> fin (pas de bras , pas de chocolat)

Si l'authentification est réussie --> la méthode authenticate() retourne un objet (implémentant l'interface "Authentication") bien complet (comportant "Roles utilisateurs" , ...).

L'objet "Authentication" est alors automatiquement stocké dans le "SecurityContextHolder" par spring-security .

Une fois l'authentification effectuée et stockée dans le contexte , on peut alors très facilement accéder aux infos "utilisateur" vérifiées via des instructions de ce type :

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
if (principal instanceof UserDetails) {
    String username = ((UserDetails)principal).getUsername();
}
```

L'objet "Authentication" comporte une méthodes **getAuthorities()** retournant un paquet d'éléments de type "GrantedAuthority" dont "SimpleGrantedAuthority" est l'implémentation la plus classique.

"SimpleGrantedAuthority" comporte un nom de rôle (ex "ROLE_ADMIN" ou "ROLE_USER" , ...)

Lorsqu'un peu plus tard , un accès à une partie de l'application sera tenté (page jsp , méthode appelée sur un contrôleur , ...) les mécanismes de la partie "contrôle d'accès" de spring-security pour alors assez facilement autoriser ou refuser les actions en comparant les rôles mémorisés dans l'objet "Authentication" du contexte avec certaines configurations du genre :

```
@PreAuthorize("hasRole('ADMIN')")
```

3.2. Api java pour jetons JWT

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.6.0</version>
</dependency>
```

L'api **jjwt** est l'une des api java spécialisée dans la gestion des jetons au format "JWT".

Cette api (indépendante de Spring) peut être également utilisée avec "JAX-RS" .

La classe utilitaire suivante (*JwtUtil.java*) peut s'avérer pratique :

```
package org.mycontrib.generic.security.jwt;
```

```

import java.util.Collection; import java.util.Date;
import org.slf4j.Logger; import org.slf4j.LoggerFactory;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.MalformedJwtException;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import io.jsonwebtoken.UnsupportedJwtException;

/* JwtUtil classe utilitaire (Helper with static methods) generic (no spring) */
public class JwtUtil {

    public static String ROLES_AUTHORITIES_CLAIM="authorities" ; // "roles" or "authorities" or "scopes"
    public static String MY_DEFAULT_JWT_ISSUER="http://www.mycompany" ;
    private static final Logger logger = LoggerFactory.getLogger(JwtUtil.class);

    public static String buildToken(String userNameOrId , long jwtExpirationInMs ,
                                   String jwtSecret , Collection<String> roleNameList) {
        //exemples: jwtExpirationInMs=60*05*1000= 300000ms pour 5minutes
        //                jwtExpirationInMs=60*15*1000= 900000ms pour 15minutes
        //                jwtExpirationInMs=60*30*1000=1800000ms pour 30minutes
        //                jwtExpirationInMs=60*60*1000=3600000ms pour 60minutes
        //                jwtExpirationInMs=60*120*1000=7200000ms pour 120minutes
        //                usernameOrId="user1"
        //                jwtSecret="MyJWTSuperSecretKey"
        //                roleNameList=[USER,ADMIN,MANAGER]

        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + jwtExpirationInMs);
        return Jwts.builder()
            .setIssuer(MY_DEFAULT_JWT_ISSUER)
            .setSubject(userNameOrId)
            .setIssuedAt(new Date())
            .claim(ROLES_AUTHORITIES_CLAIM, roleNameList.toString())
            .setExpiration(expiryDate)
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
    }

    public static Claims extractClaimsFromJWT(String token, String jwtSecret) {
        Claims claims = Jwts.parser()

```

```

        .setSigningKey(jwtSecret)
        .parseClaimsJws(token)
        .getBody();
    logger.debug("extracted claims in JWT="+claims.toString());
    return claims;
}

public static boolean validateToken(String authToken, String jwtSecret) {
    try {
        Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
        return true;
    } catch (SignatureException ex) {    logger.error("Invalid JWT signature");
    } catch (MalformedJwtException ex) {    logger.error("Invalid JWT token");
    } catch (ExpiredJwtException ex) {    logger.error("Expired JWT token");
    } catch (UnsupportedJwtException ex) {    logger.error("Unsupported JWT token");
    } catch (IllegalArgumentException ex) {    logger.error("JWT claims string is empty.");
    }
    return false;
}
}

```

3.3. Enrobage de "JwtUtil" dans un composant "JwtTokenProvider" spécifique au contexte Spring

```

package org.mycontrib.generic.security.spring.security;
import java.util.ArrayList; import java.util.List;
import org.mycontrib.generic.security.jwt.JwtUtil;
import org.slf4j.Logger; import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import io.jsonwebtoken.Claims;

```

/ Un peu comme JwtUtil mais sous forme de composant Spring */*

@Component

```

public class JwtTokenProvider {

    private static final Logger logger = LoggerFactory.getLogger(JwtTokenProvider.class);

    private String DEFAULT_SPRING_SECURITY_ROLE_PREFIX="ROLE_";

    @Value("${app.jwtSecret}") // in application.properties
    private String jwtSecret = "MyJWTSuperSecretKey"; //by default (example)

    @Value("${app.jwtExpirationInMs}") // in application.properties
    private int jwtExpirationInMs = 30*60*1000 ;//pour 30 minutes (example) par default

    public String generateToken(Authentication authentication) {

        UserDetails userPrincipal = (UserDetails) authentication.getPrincipal();

        List<String> roleNameList=new ArrayList<String>();
        for(GrantedAuthority ga : userPrincipal.getAuthorities()){
            String springSecurityRoleName=ga.getAuthority();
            String roleName=springSecurityRoleName;
            //ou bien roleName = springSecurityRoleName moins le préfixe "ROLE_" (affaire de préférence)
            /*
            if(roleName.startsWith(DEFAULT_SPRING_SECURITY_ROLE_PREFIX)){
                roleName = roleName.substring(DEFAULT_SPRING_SECURITY_ROLE_PREFIX.length());
            }*/
            roleNameList.add(roleName);
        }
        return buildToken(userPrincipal.getUsername(),roleNameList);
    }

    public String buildToken(String userNameOrId,List<String> roleNameList) {
        return JwtUtil.buildToken(userNameOrId, jwtExpirationInMs, jwtSecret,roleNameList);
    }

    //si pas de roles dans jwt claims:
    public String getUserNameOrIdFromJWT(String token) {
        return JwtUtil.extractClaimsFromJWT(token, jwtSecret).getSubject();
    }

    //si roles dans jwt claims:
    public UserDetails getUserDetailsFromJWT(String token) {
        Claims jwtClaims = JwtUtil.extractClaimsFromJWT(token, jwtSecret);
    }
}

```

```

String username = jwtClaims.getSubject();
List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
Object rolesInClaim = jwtClaims.get(JwtUtil.ROLES_AUTHORITIES_CLAIM);
    if(rolesInClaim!=null){
        String rolesInClaimAsString = (String) rolesInClaim.toString();
        //exemples: "[]" ou [USER] ou [USER,ADMIN]
        if(rolesInClaimAsString.length()>2){
            rolesInClaimAsString = rolesInClaimAsString.substring(1,
                rolesInClaimAsString.length()-1);//sans '['ni']'
            String[] tabOfRoleNames = rolesInClaimAsString.split(",");
            for(String roleName : tabOfRoleNames){
                roleName=roleName.trim();
                logger.debug("in jwt claims, found roleName="+roleName);
                String springSecurityRoleName = roleName;
                //ou bien springSecurityRoleName = "ROLE_" + roleName; si besoin :
                if(!(springSecurityRoleName.startsWith(
                    DEFAULT_SPRING_SECURITY_ROLE_PREFIX))){
                    springSecurityRoleName=DEFAULT_SPRING_SECURITY_ROLE_PREFIX
                        + roleName;
                }
                authorities.add(new SimpleGrantedAuthority(springSecurityRoleName));
            }
        }
    }
    // User(username, password, enabled, accountNonExpired, credentialsNotExpired,
        accountNonLocked, authorities)

    User user = new User(username, "unknown_in_jwt_claims_but_already_check", true
        /*account.isEnabled()*/, true, true, true, authorities);

    return user;
}

public boolean validateToken(String authToken) {
    return JwtUtil.validateToken(authToken, jwtSecret);
}
}

```

3.4. Exemple de web service rest d'authentification retournant un jeton "JWT" en cas de succès

Ce code est basé sur JwtUtil , Spring-mvc et l'implémentation *org.mycontrib.generic.security.service* des opérations "CRUD" liées aux comptes utilisateurs.

```
package org.mycontrib.generic.security.rest.payload;
...
@Getter @Setter @ToString @NoArgsConstructor
public class AuthRequest {
    private String username;
    private String password;
    //private String roles;
}
```

```
package org.mycontrib.generic.security.rest.payload;
...
@Getter @Setter @ToString @NoArgsConstructor
public class AuthResponse {
    public String authToken; //jeton d'authentification généré
    public Boolean authOk;
    private String message;
    //...
}
```

```
package org.mycontrib.generic.security.rest.spring.mvc;

import org.mycontrib.generic.security.generic.LoginAccountDetails;
import org.mycontrib.generic.security.persistence.entity.LoginAccount;
import org.mycontrib.generic.security.rest.payload.AuthRequest;
import org.mycontrib.generic.security.rest.payload.AuthResponse;
import org.mycontrib.generic.security.rest.payload.NewUser;
import org.mycontrib.generic.security.rest.payload.RegisterUserResponse;
import org.mycontrib.generic.security.service.LoginAccountService;
import org.mycontrib.generic.security.spring.security.JwtTokenProvider;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

// sign up = subscribe/register = s'inscrire
// sign in = login = se connecter

@RestController
@RequestMapping(value="/auth" , headers="Accept=application/json")
public class AuthController /*extends AbstractRestAuthWS*/ {
    private static Logger logger = LoggerFactory.getLogger(AuthController.class);

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    JwtTokenProvider tokenProvider;

    @PostMapping("/login")
    public ResponseEntity<?> authenticateUser(@RequestBody AuthRequest loginRequest) {
        logger.debug("/login , loginRequest:"+loginRequest);
        Authentication authentication = null;
        AuthResponse authResponse = new AuthResponse();
        try {
            authentication=authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    loginRequest.getUsername(),
                    loginRequest.getPassword()
                )
            ); //authenticate() soulève une exception si mauvais username ou password
            String jwt = tokenProvider.generateToken(authentication);

```



```

    authResponse.setAuthToken(jwt);
    authResponse.setAuthOk(true);
    authResponse.setMessage("login successful");
    logger.debug("/login authResponse:" + authResponse.toString());
    return ResponseEntity.ok(authResponse);
} catch (AuthenticationException e) {
    logger.debug("echec authentification:" + e.getMessage()); //for log
    authResponse.setAuthOk(false);
    authResponse.setMessage("echec authentification");
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(authResponse);
}
}
}

```

login.html (ici codé avec *jquery*) :

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>login</title>
    <script src="lib/jquery-3.3.1.min.js"></script>
    <script src="js/my-jq-ajax-util.js"></script>
    <script>
        $(function() {
            $('#btnLogin').on('click',function(){

                var auth = { username : null, password : null } ;
                auth.username = $('#txtUsername').val();
                auth.password = $('#txtPassword').val();

                $.ajax({
                    type: "POST",
                    url: "auth/login",
                    dataType : "json",
                    data : JSON.stringify(auth),
                    contentType : "application/json",
                    success: function (data,status,xhr) {
                        if (data) {
                            console.log(JSON.stringify(data));
                            var authResponse = data;
                            if(authResponse.authOk){
                                $("#spanMsg").html(authResponse.message);
                                //localStorage.setItem("authToken",authResponse.authToken);
                                sessionStorage.setItem("authToken",authResponse.authToken);
                            } else{
                                $("#spanMsg").html( authResponse.message);
                            }
                        }
                    }
                })
            })
        })
    </script>

```

```

        }
    },
    error: function( jqXHR, textStatus, errorThrown ){
        $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR)
            + "status:" + textStatus + " error : " + errorThrown );
    }
}); //end $.ajax

}); //end on click btnFindById
});
</script>
</head>
<body>
    <h3> login (ws security) </h3>
    <p>exemple of username : mycompany/myapp/employees/user1 </p>
    <p>exemple of username : mycompany/myapp/administrators/admin </p>
    username : <input id="txtUsername" type='text' value="member1"/><br/>
    password : <input id="txtPassword" type='text' value="pwd1"/><br/>
    <input type='button' value="login" id="btnLogin"/> <br/>
    <span id="spanMsg"></span> <br/>
    <hr/>
    <a href="index.html">retour vers index.html</a>
</body>
</html>

```

NB : *js/my-jq-ajax-util.js* et *register-customer.html* accessibles dans la partie

src/main/resources/static de <https://github.com/didier-mycontrib/spring-boot-spectacle-ws> .

3.5. Filtre web "JwtAuthenticationFilter" (basé sur spring-security) pour extraire le jeton JWT

Ce filtre essentiel va :

- extraire le token "jwt" dans le champ "Authorization" de l'entête HTTP de la requête
- vérifier si ce jeton est valide (déchiffable via jwtSecret , pas encore expiré, ...)
- tenter d'extraire les informations sur l'utilisateur (username , liste des roles, ...) dans la partie "claim" du jeton JWT .
- stocker les infos utilisateur dans l'objet "Authentication" de spring-security lui même placé dans le contexte de spring-security
---> ceci permettra ultérieurement aux mécanismes de spring-security d'autoriser ou pas l'accès à une ressource protégée via @PreAuthorize("hasRole('ADMIN ou ...')")

JwtAuthenticationFilter.java

```
package org.mycontrib.generic.security.spring.security;
import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtTokenProvider tokenProvider;

    @Autowired
    private DefaultCustomSpringSecurityDetailsService customUserDetailsService;

    private static final Logger logger = LoggerFactory.getLogger(JwtAuthenticationFilter.class);

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        try {
            String jwt = getJwtFromRequest(request);
            logger.info("jwt extract by JwtAuthenticationFilter in request:"+jwt);
            if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
                //plan A avec roles dans jwt claims:
                UserDetails userDetails = tokenProvider.getUserDetailsFromJWT(jwt);
                if(userDetails.getAuthorities()==null){
```

```

//plan B sans roles dans jwt claims:
String userNameOrId = tokenProvider.getUserNameOrIdFromJWT(jwt);
userDetails = customUserDetailsService.loadUserByUsername(userNameOrId);
}

UsernamePasswordAuthenticationToken authentication =
    new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
logger.info("JwtAuthenticationFilter is storing authentication:"+authentication
    + " in spring security SecurityContextHolder");
}
} catch (Exception ex) {
    logger.error("Could not set user authentication in security context", ex);
}

filterChain.doFilter(request, response);

// Clears the context from authentication
SecurityContextHolder.getContext().setAuthentication(null);
}

private String getJwtFromRequest(HttpServletRequest request) {
    String bearerToken = request.getHeader("Authorization");
    if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
        return bearerToken.substring(7, bearerToken.length());
    }
    return null;
}
}

```

3.6. Configuration nécessaire (Spring-security , Spring-mvc)

```

package org.mycontrib.generic.security.spring.security;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request,
        HttpServletResponse response,
        AuthenticationException authException) throws IOException {
        // This is invoked when user tries to access a secured REST resource without supplying any credentials
        // We should just send a 401 Unauthorized response because there is no 'login page' to redirect to
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
    }
}

```

WebSecurityConfig.java

```

package org.mycontrib.generic.security.spring.security.config;

import org.mycontrib.generic.security.spring.security.DefaultCustomSpringSecurityDetailsService;
import org.mycontrib.generic.security.spring.security.JwtAuthenticationEntryPoint;
import org.mycontrib.generic.security.spring.security.JwtAuthenticationFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
//NB: @PreAuthorize need the option prePostEnabled = true
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint unauthorizedHandler;

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Autowired
    public void globalUserDetailsInMemory(final AuthenticationManagerBuilder auth) throws Exception {
        String pwd1Crypted = passwordEncoder.encode("pwd1");
        System.out.println("pwd1Crypted via bcrypt:" + pwd1Crypted);
        auth.inMemoryAuthentication()
        .withUser("user1").password(pwd1Crypted).roles("USER").and()
        .withUser("admin1").password(passwordEncoder.encode("pwd1")).roles("ADMIN").and()
        .withUser("user2").password(passwordEncoder.encode("pwd2")).roles("USER").and()
        .withUser("admin2").password(passwordEncoder.encode("pwd2")).roles("ADMIN");
    }

    @Override
    @Bean

```

```

public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}

@Override
protected void configure(final HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/", "/favicon.ico", "/*/*.png", "/*/*.gif", "/*/*.svg",
            "/*/*.jpg", "/*/*.html", "/*/*.css", "/*/*.js").permitAll()
        .antMatchers(HttpMethod.POST, "/auth/**").permitAll()
        .antMatchers("/xyz-api/public/**").permitAll()
        .antMatchers("/xyz-api/private/**").authenticated()
        .and().cors() //enable CORS (avec @CrossOrigin sur class @RestController)
        .and().csrf().disable()
        // If the user is not authenticated, returns 401
        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
        // This is a stateless application, disable sessions
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
        // Custom filter for authenticating users using tokens
        .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
}
}

```

3.7. Variante avec authentification jdbc

JdbcAppDbGlobalUserDetailsConfig.java à adapter au contexte

```

package org.mycontrib.generic.security.config;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.authentication.configurers.provisioning.JdbcUserDetailsManagerConfigurer;
import org.springframework.security.config.annotation.authentication.configurers.provisioning.UserDetailsManagerConfigurer;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@Profile("appDbSecurity") //with jdbc
public class JdbcAppDbGlobalUserDetailsConfig {
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    private static DataSource realmDataSource;

```

```
private static void initRealmDataSource() {
    DriverManagerDataSource driverManagerDataSource = new DriverManagerDataSource();
    driverManagerDataSource.setDriverClassName("org.h2.Driver");
    driverManagerDataSource.setUrl("jdbc:h2:~/realmdb");
    driverManagerDataSource.setUsername("sa");
    driverManagerDataSource.setPassword("");
    realmDataSource = driverManagerDataSource;
}
```

```
private boolean isRealmSchemaInitialized() {
    int nbExistingTablesOfRealmSchema = 0;
    try {
        Connection cn = realmDataSource.getConnection();
        DatabaseMetaData meta = cn.getMetaData();
        String tabOfTableType[] = {"TABLE"};
        ResultSet rs = meta.getTables(null,null,"%",tabOfTableType);
        while(rs.next()){
            String existingTableName = rs.getString(3);
            if(existingTableName.equalsIgnoreCase("users")
                || existingTableName.equalsIgnoreCase("authorities")) {
                nbExistingTablesOfRealmSchema++;
            }
        }
        rs.close();
        cn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return (nbExistingTablesOfRealmSchema>=2);
}
```

@Autowired

```
public void globalUserDetails(final AuthenticationManagerBuilder auth) throws Exception {
    initRealmDataSource();
    JdbcUserDetailsManagerConfigurer jdbcUserDetailsManagerConfigurer =
        auth.jdbcAuthentication().dataSource(realmDataSource);
    if(isRealmSchemaInitialized()) {
        /*
         * jdbcUserDetailsManagerConfigurer
         * .usersByUsernameQuery("select username,password,enabled from users where username=?")
         * .authoritiesByUsernameQuery("select username, authority from authorities where username=?");
         * //by default
         * // or .authoritiesByUsernameQuery("select username, role from user_roles where username=?")
         * //if custom schema
        */
    } else {
        //creating default schema and default tables "users" , "authorities"
        jdbcUserDetailsManagerConfigurer.withDefaultSchema();
        //insert default users:
        configureDefaultUsers(jdbcUserDetailsManagerConfigurer);
    }
}
```

```
void configureDefaultUsers(UserDetailsManagerConfigurer udmc){
    udmc
        .withUser("user1").password(passwordEncoder.encode("pwduser1")).roles("USER").and()
        .withUser("admin1").password(passwordEncoder.encode("pwdadmin1")).roles("ADMIN","USER").and()
        .withUser("publisher1").password(passwordEncoder.encode("pwdpublisher1")).roles("PUBLISHER","USER").and()
        .withUser("user2").password(passwordEncoder.encode("pwduser2")).roles("USER").and()
}
```

```

        .withUser("admin2").password(passwordEncoder.encode("pwdadmin2")).roles("ADMIN").and()
        .withUser("publisher2").password(passwordEncoder.encode("pwdpublisher2")).roles("PUBLISHER");
    }
}

```

3.8. Exemple de WS applicatif (non public) protégé par le mécanisme d'authentification précédent:

```

package org.mycontrib.spectacle.rest;
...
import org.springframework.security.access.prepost.PreAuthorize;
...
@RestController
@RequestMapping(value="/spectacle-api/spectacle" , headers="Accept=application/json")
public class AdminSpectacleRestController {

    //business service vers lequel déléguer les traitements :
    @Autowired
    private SpectacleService spectacleService;

    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping("")
    public ResponseEntity<SpectacleAddition> postSpectacle(@RequestBody SpectacleAddition ajoutSpectacle)
    {
        spectacleService.addSpectacle(ajoutSpectacle.getSpectacle(), ajoutSpectacle.getCategoryId());
        return new ResponseEntity<SpectacleAddition>(ajoutSpectacle,HttpStatus.OK);
    }
}

```

==> Beaucoup d'éléments à coder/configurer pour bien sécuriser une application avec des "Web Services REST" (ici avec Spring-mvc , Spring-security , JWT , H2/JPA ,)

==> Vivement "OAuth2" pour déléguer tout ça à une application spécialisée de type "AuthorizationServer" !!!

4. Token JWT avec NodeJs

L'équivalent de "spring-security" pour NodeJs s'appelle "passportjs"

Le site officiel est "<http://www.passportjs.org/>"

passportjs s'intègre très bien dans une application "node + express" .

passportjs peut être considéré comme un mini framework très paramétrable à base plugins.

Il existe toutes sortes de **plugins** pour passportjs :

- pour jeton JWT
- pour OAuth2

....

IX - Design Api REST et description swagger2

1. Design d'une api REST

1.1. Rappels des fondamentaux

Bien respecter les "verbes"

GET pour lecture , recherche , interrogations,

POST pour ajout ou "saveOrUpdate"

PUT pour mise à jour d'un existant

DELETE pour une suppression .

Bien respecter les conventions habituelles :

fin d'url en typeEntité/**idRessource** (ex : produit/1) pour **recherche unique** (par id/pk) .

fin d'url en typeEntité?**critere1=val1&critere2=val2** pour **recherche multiple** (retournant un tableau ou liste de taille 0 ou n) .

Même URL pour recherche , suppression , mise à jour , seul le verbe (méthode HTTP) change :
GET ou DELETE ou PUT .

1.2. Retourner des réponses explicites et des statuts Http précis

Lorsque c'est possible , il vaut mieux retourner les statuts http précis :

"Unauthorized access (**401**)";
"Forbidden resource can't be accessed (**403**)";
"resource not found (**404**)";
"Internal server error (**500**)";
"Service unavailable (**503**)";

Ceux ci sont standards et récupérables via plein de technologies (xhr , jquery , jax-rs , spring-mvc, ...) . Assez "génériques" ces statuts peuvent souvent être gérés par du code hautement réutilisable (filtres ,) .

En outre **le contenu (partie "body")** de la réponse doit idéalement être très explicite.

Il est par exemple conseillé d'accompagner le résultat de la recherche avec une copie en retour des critères de recherches reçus .

Exemple (fixer.io) :

```
http://data.fixer.io/api/convert
? access_key = API_KEY
& from = GBP
& to = JPY
& amount = 25
```

Api response :

```
{
  "success": true,
  "query": {
    "from": "GBP",
    "to": "JPY",
    "amount": 25
  },
  "info": {
    "timestamp": 1519328414,
    "rate": 148.972231
  },
  "historical": ""
  "date": "2018-02-22"
  "result": 3724.305775
}
```

Il peut également être intéressant de retourner en retour une copie des données effectivement ajoutées ou mises à jours (POST , PUT) .

--> ceci permet de rassurer le client : la mise à jour s'est bien effectuée .

--> dans le cas d'un ajout (POST) , il est souvent utile de retourner une copie de l'entité complète (avec l'identifiant / clef primaire quelquefois auto incrémentée).

1.3. Prise en compte des problématiques de sécurité

.Très souvent HTTPS (rarement HTTP)

.pas de choses confidentielles à la fin d'une URL d'une requête en mode GET

.crypter si nécessaire

.Api key et/ou token d'authentification selon les cas

...

.Séparer si besoin la partie "Consultation" (read) de la partie "Mise à jour" (write)

dans deux WS complémentaires de la même api pour pouvoir simplement effectuer les paramétrages de sécurité (ex : permitAll() ou)

1.4. Design pattern "DTO" adapté aux web services REST

Bien qu'il soit techniquement possible de directement retourner des représentations "Xml" ou "Json" des entités persistantes (avec @Entity de Jpa) via des ajouts adéquats de @JsonIgnore près des @OneToMany , @ManyToMany,

il est souvent préférable de retourner des structures spécifiquement adaptées aux "contrôleurs" de "web services REST" : classes java rangées dans un package du genre "rest.dto" ou "rest.payload" ou

Il s'agit d'un glissement "web / http / rest" du design pattern "DTO = Data Transfert Object" .

Contrairement aux WS SOAP , les WS REST n'ont pas une logique "RPC" et les DTOs sont plutôt à placer près des "contrôleurs REST" que des services "métiers/business" internes) .

Dans certains cas , les DTOs sont indispensables (ex : dto.OrdreDeVirement pour déclencher serviceInterneCompte.transférer(montant,numCptDeb, numCptCred) .

D'autres fois , lorsque les structures sont (et sont censées rester) simples , une sérialisation directe des entités persistantes peut éventuellement booster les performances et/ou la rapidité de réalisation. A ce sujet, étant donné que le format JSON est très souple (comparé à Xml) , un changement de structure interne du serveur peut souvent rester transparent vis à vis du client effectuant les invocations : réadaptation plus faciles à effectuer en REST/JSON qu'en mode SOAP/XML .

En appliquant les slogans **KISS** (Keep It Simple Stupid) et **DRY** (Don't Repeat Yourself) , on peut souvent commencer simple et complexifier graduellement en fonction des besoins .

Critères importants (indépendance / non-adhérence , réutilisabilité , partage , ...)

--> à doser au cas par cas .

1.5. Autres considérations (bonne pratiques)

Etant donné que d'un point de vue externe , un WS REST est avant tout identifié par l'URL de son point d'accès (endPoint) , il est très fortement conseillé de mettre en place des URIs assez structurés/composés de type :

<https://www.domainXy.com/appliXy/rest/api-zz/public/entityXy> .

La partie "rest/api-zz" ou "api-zz" permettra d'effectuer des réglages/paramétrages de reverse-proxy HTTP dans le cadre d'une mise en production sérieuse .

Exemple :

Appli-angular 6+ (index.html + bundles js)
déposée sur serveur HTTP (Apache 2.2 ou Nginx ou ...) .

Navigateur -----> Serveur Http intermédiaire (nginx ou api-gateway ou ...)

--> partie angular (html / js) statique

---> partie "api-zz1" déléguée vers -----> Appli-SpringBoot 1

---> partie "api-zz2" déléguée vers -----> Appli-SpringBoot 2

Et du coup , moins besoin de paramétrages CORS.

2. Notion d'Api REST et description

2.1. Description détaillée d'Api REST (Swagger, RAML, ...)

Notion d' API REST

Dès le début , la structure d'un web-service "*SOAP*" a été décrite de façon standardisée via la norme *WSDL* (standard officiel du W3C).

A l'inverse les Web-services REST (qui sont basés sur de simples recommandations autour de l'usage d'HTTP) ne sont toujours pas associés à un type de description unique et standardisé.

Un **document qui décrit la structure d'un web-service REST** est généralement appelé "**API REST**" et peut être écrit en XML , en JSON ou en YAML.

Les principaux formalismes existants pour décrire une API REST sont :

- **WADL** (existant depuis longtemps en **XML** mais en perte de vitesse)
- **Swagger** (version 1.x basée sur **YAML** , v2 basée sur **JSON** que l'on peut convertir en **YAML**). Bien outillé et existant depuis plusieurs années, swagger a pour l'instant une petite longueur d'avance.
- **RAML** (pour l'instant basé sur une syntaxe **YAML** volontairement simple , pris en charge par quelques marques telles que "MuleSoft" , ...)
- **Blueprint API** : basé également sur **YAML**

Standardisation (2015-2017) --> (Swagger 3 , OpenApi)

2.2. Fragile format YAML

Format YAML

YAML (*YAML Ain't Markup Language*) n'est d'après son nom , pas un langage à balise mais se veut être un équivalent d'un point de vue fonctionnalité (à soir sérialisation/dé-sérialisation de documents informatiques arborescents).

books.yaml (exemple)

YAML est en fait **structuré via des indentations** et se veut être **facile à lire (ou à écrire) par une personne humaine** .

*De la rigueur
dans les indentations
S'impose !!!*

(fragile comme CSV)

```
/books:
  /{bookTitle}
  get:
    queryParameters:
      author:
        displayName: Author
        type: string
        description: An author's full name
        example: Mary Roach
        required: false
      publicationYear:
        displayName: Pub Year
        type: number
        description: The year released for the first time
    ...
```

YAML semble très fragile .

Principal avantage de YAML : **configuration très compacte** .

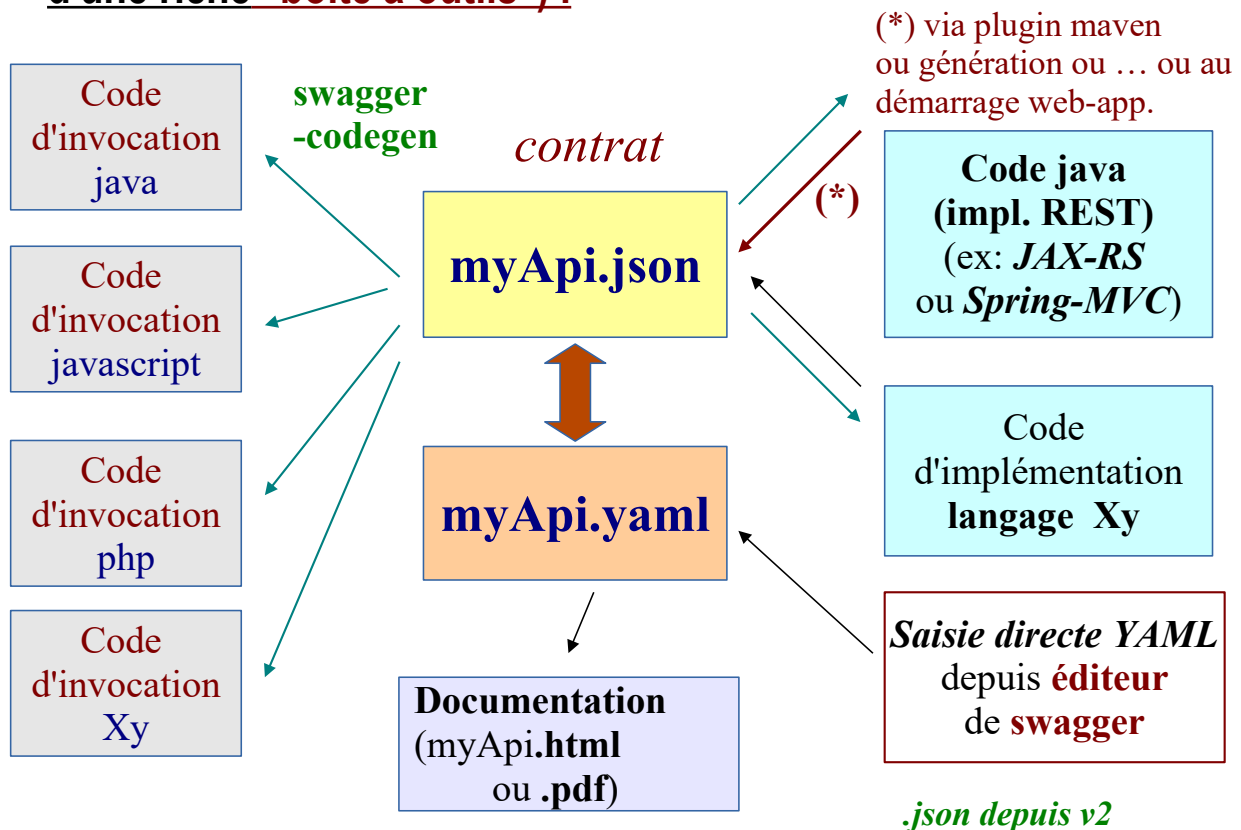
Principal inconvénient de YAML : **indentation sujette à erreurs (d'interprétation,)** .

NB : au sein d'un fichier .yaml (ou .yml) les **indentations** ne sont pas constituées par des ~~caractères spéciaux "tabulation"~~ mais par des **séries de n (ex : 2 ou 4) espaces consécutifs** .

Heureusement **Swagger 2** et **Swagger3/OpenApi** s'appuient maintenant principalement sur une description au format JSON.

3. Swagger

Swagger (spécifications pour **API REST** accompagnées d'une riche **"boîte à outils"**).



3.1. OpenApi (évolution "openSource" de swagger , swagger V3)

Swagger 1 date de 2010 , swagger 2 de 2014 (pour le tout début).

A partir de 2015 , L'**OpenApi initiative** a réuni beaucoup d'acteurs du marché (dont MuleSoft à l'origine de RAML) et un accord a été trouvé : **OpenApi en tant que standard de fait (reprenant les bonnes idées de swagger 2)**.

En 2017, swagger 3 a été un peu restructuré et peut être vu comme l'implémentation de référence de "OpenApi".

Autrement dit , swagger 3 repose sur un coeur gratuit/opensource/standardisé appelé "openApi" et la marque "SmartBear" propose également tout un tas de services annexes optionnels et payants : version commerciale de "swagger" .

Autrement dit , mieux vaut utiliser aujourd'hui swagger 3 / OpenApi que swagger 2 .

Nb : la dépendance maven **springdoc-openapi-ui** correspond à une implémentation de "swagger3/openApi" bien intégrée dans le framework spring/springBoot .

4. Config swagger3 / openapi-doc pour spring

Ancienne version à ne pas ajouter dans pom.xml

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

Version plus récente à ajouter dans pom.xml

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.1</version>
</dependency>
```

en plus de


```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```


Configuration explicite à idéalement placer dans application.properties :

```
springdoc.swagger-ui.path=/doc-swagger.html
```

dans index.html (ou ailleurs) :

```
<a href="/doc-swagger.html">documentation Api REST générée dynamiquement par swagger3/openapi</a>
```


 **swagger**

default (/v2/api-docs) 

Explore

My REST API (serverSpring MVC)

Api pour Devise

Created by DJA1
See more at www.afcepf.fr
[Contact the developer](#)
[License of API](#)

rest-devise-service : Rest Devise Service

Show/Hide | List Operations | Expand Operations

GET	/rest/devise	devisesByCriteria
POST	/rest/devise	saveOrUpdateDevise
DELETE	/rest/devise/{codeDev}	deleteDeviseByCode
GET	/rest/devise/{codeDev}	deviseByCode

ws-auth : Ws Auth

Show/Hide | List Operations | Expand Operations

ws-confidentiel : Ws Confidentiel

Show/Hide | List Operations | Expand Operations

[BASE URL: /serverSpringMvc/ws , API VERSION: API TOS]

NB : Selon le contexte applicatif , il faudra peut être paramétrer la sécurité de façon à pouvoir accéder à la documentation "swagger" générée :

@Configuration

@EnableWebSecurity

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter
```

```
//....
```

```
private static final String[] SWAGGER_AUTH_WHITELIST = {
    "/swagger-resources/**", "/swagger-ui.html", "/v2/api-docs", "/webjars/**"
};
```

```
protected void configure(HttpSecurity http) throws Exception {
    // configuration partielle à compléter:
    http.authorizeRequests()
        .antMatchers("/rest/devise-api/public/**").permitAll()
        .antMatchers(SWAGGER_AUTH_WHITELIST).permitAll()
        .anyRequest().authenticated();
}
```

Configuration de l'api via "annotations OpenApi/ swagger 3" :

Attention : les anciennes annotations de l'époque "swagger2" ([@ApiModelProperty](#) , [@ApiOperation](#) , [@ApiParam](#)) ont été changées lors de la standardisation swagger3/OpenApi ([@Schema](#) , [@Operation](#) , [@Parameter](#) , ...)

```
package org.mycontrib.backend.dto;
import io.swagger.v3.oas.annotations.media.Schema;
@Schema(description = "DTO (Result of conversion)")
public class ResConv {
    @Schema(description = "amount to convert", defaultValue = "100")
    private Double amount;

    @Schema( description = "source currency code", defaultValue = "EUR")
    private String source;
    ...
}
```

et dans une classe de `@RestController` :

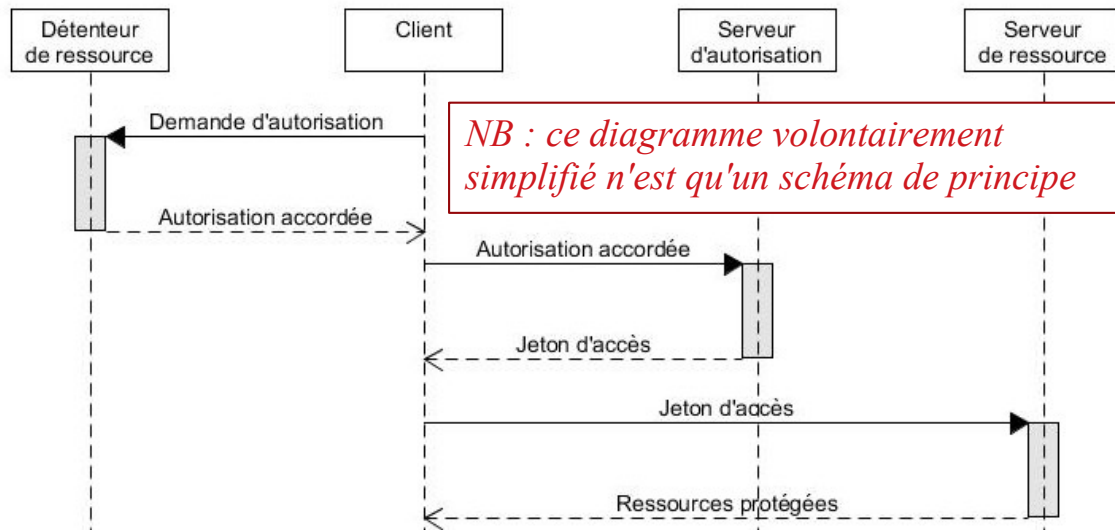
```
...
import io.swagger.v3.oas.annotations.media.Operation;
import io.swagger.v3.oas.annotations.media.Parameter;
...
@RestController
@RequestMapping(value="/rest/devise-api/public" , headers="Accept=application/json")
public class PublicDeviseRestCtrl {
...
@RequestMapping(value="/convert" , method=RequestMethod.GET)
@Operation(summary= "convert amount from source to target currency",
description = "exemple: convert?source=EUR&target=USD&amount=100")
public ResConv convertir(
    @RequestParam("amount")
    @Parameter(description = "amount to convert", ... = "100")
    Double montant,
    @RequestParam("source")
    @Parameter(description = "source currency code", ... = "EUR")
    String source,
    @RequestParam("target")
    @Parameter(description = "target currency code", ... = "USD")
    String cible) {
    Double res = convertisseur.convertir(montant, source, cible);
    return new ResConv(montant, source, cible,res);
}
....
```

X - Délégation d'authentification , OAuth2

1. OAuth2 (présentation)

Norme/Protocole "OAuth2"

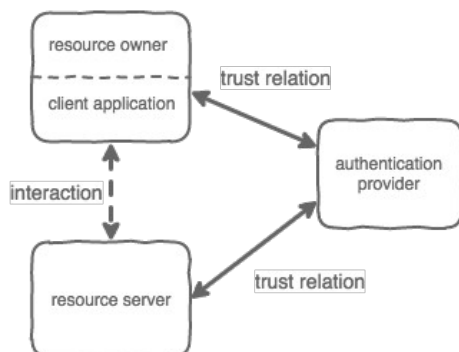
OAuth (Open Authorization) existant en versions "1" et "2" , est une norme (RFC 6749 et 6750) qui correspond à un **protocole de "délégation d'autorisation"** . Ceci permet par exemple d'autoriser une application cliente à accéder à une API d'une autre application (ex : FaceBook , Twitter , ...) de façon à accéder à des données protégées.



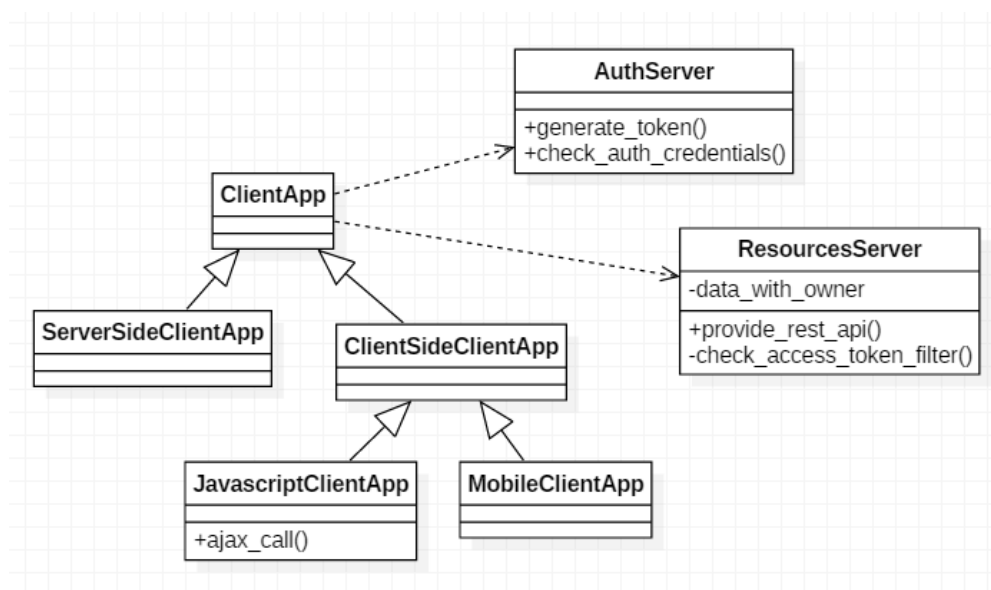
Attention, **OAuth** doit être accompagné de **HTTPS** pour être un minimum sécurisé et les autorisations doivent dans la plupart des cas être associées à une authentification basée sur un compte utilisateur pour que le jeton d'accès construit puisse véhiculer une information précise du type :

"L'utilisateur Uxyz authentifié par OrgXy est via ce jeton d'accès autorisé à accéder aux ressources accessibles via certaine(s) Api(s) "

Un jeton d'accès (souvent au format JWT) construit par OAuth2 aura un délai d'expiration court ou moyen . Dans certains cas , un autre "*refresh token*" permettra d'obtenir un nouveau jeton d'accès .



third party authentication



Les 4 modes d'autorisation (variantes) de OAuth2

* via un **code** temporaire à échanger contre un jeton d'accès : l'application cliente est un site web (avec une technologie serveur fiable) qui peut gérer des redirections d'URLs .

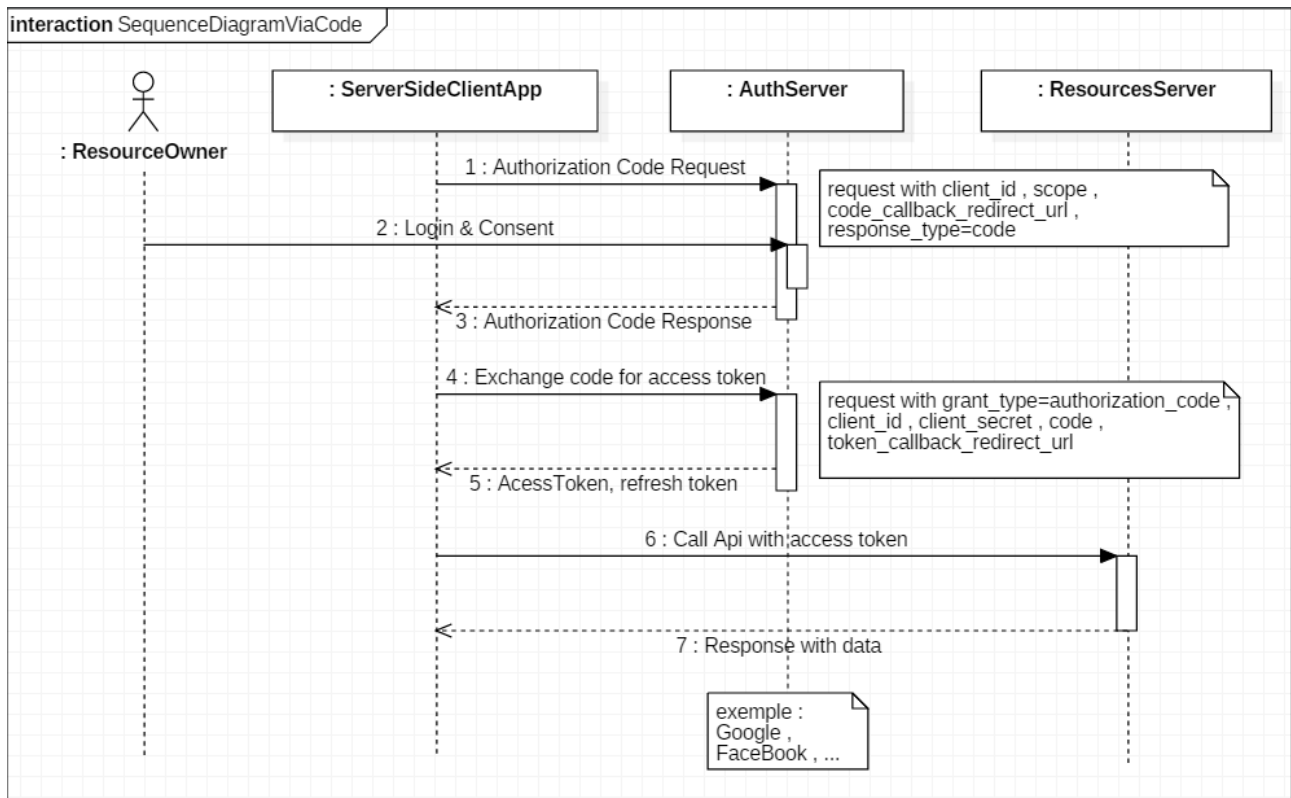
* **implicite** (jeton direct) : l'application cliente est simple et exposée (ex : javascript , mobile) [*attention : ce mode doit etre accompagné de précautions pour une bonne sécurité*]

* via **mot de passe** (retransmis) : seulement applicable si l'application cliente et le serveur d'autorisation sont gérés par la même organisation (entreprise ou ...) .

* **client App credential** : au lieu d'authentifier un utilisateur précis , c'est appli "client" qui demande une autorisation d'accès auprès d'une autre .

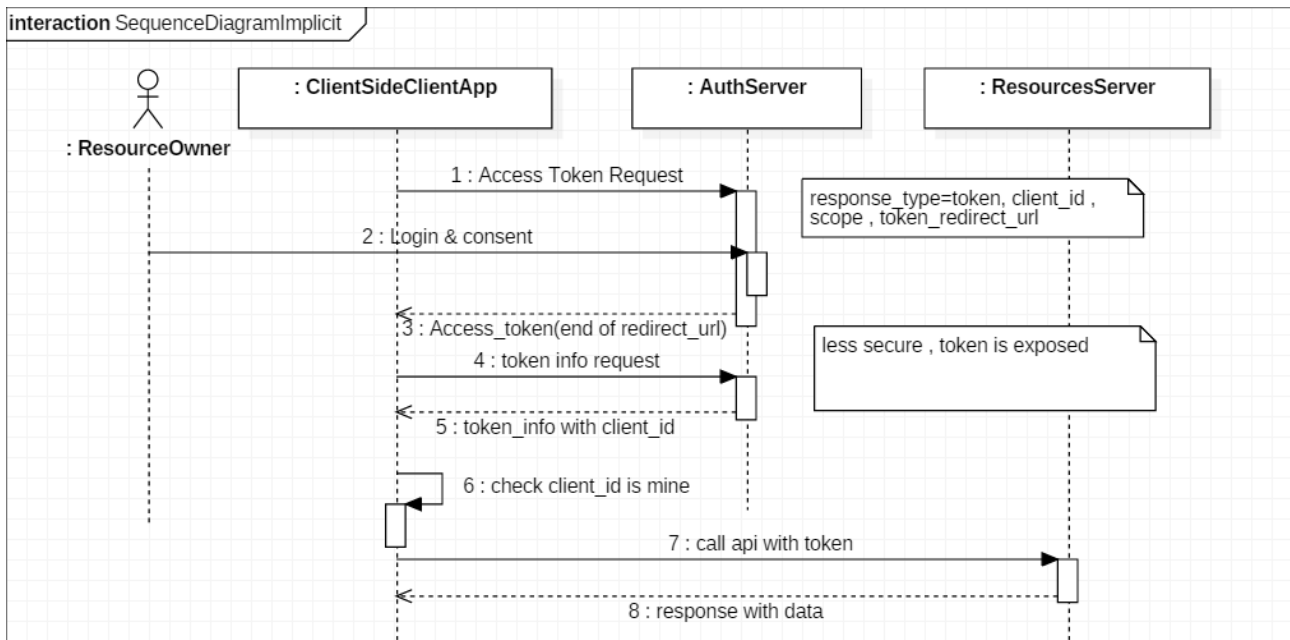
1.1. grant_type="code"

Autorisation OAuth2 par code (inter-organisations)



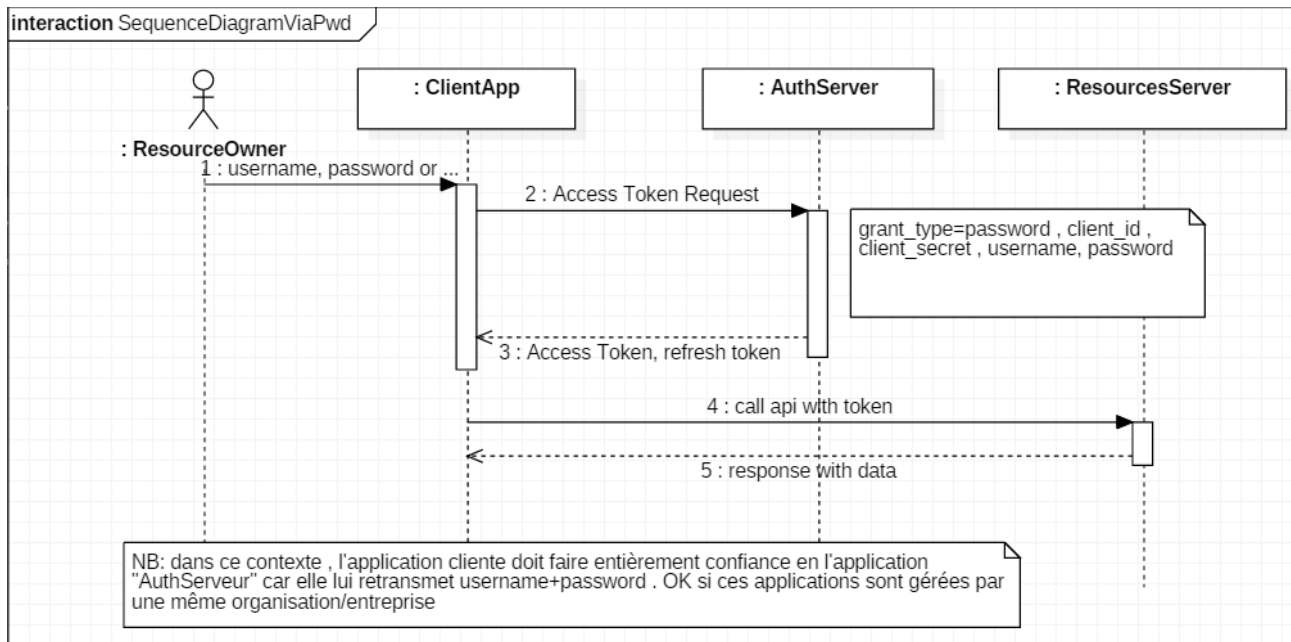
1.2. grant_type="implicit"

Autorisation OAuth2 implicite (jeton direct)



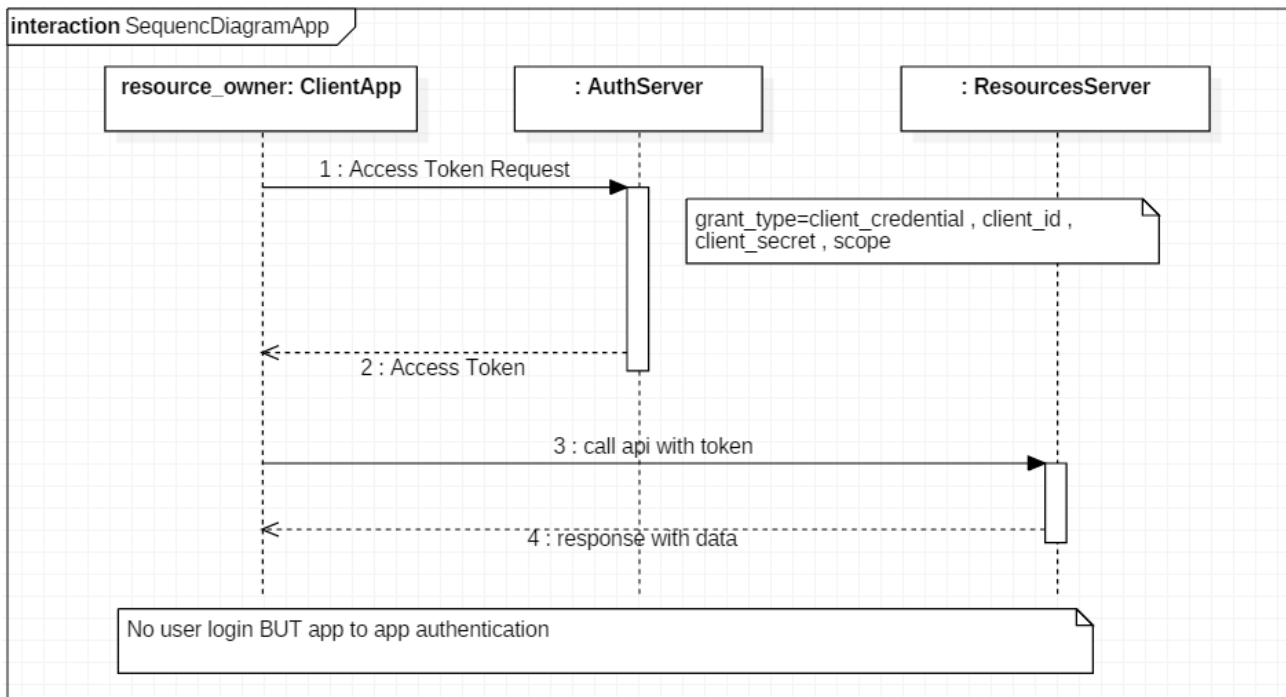
1.3. grant_type="password" (delegate in same organization)

Autorisation OAuth2 par mot de passe (intra)



1.4. grant_type="client_credential" (app to app auth,no user login)

Autorisation OAuth2 pour application utilisateur



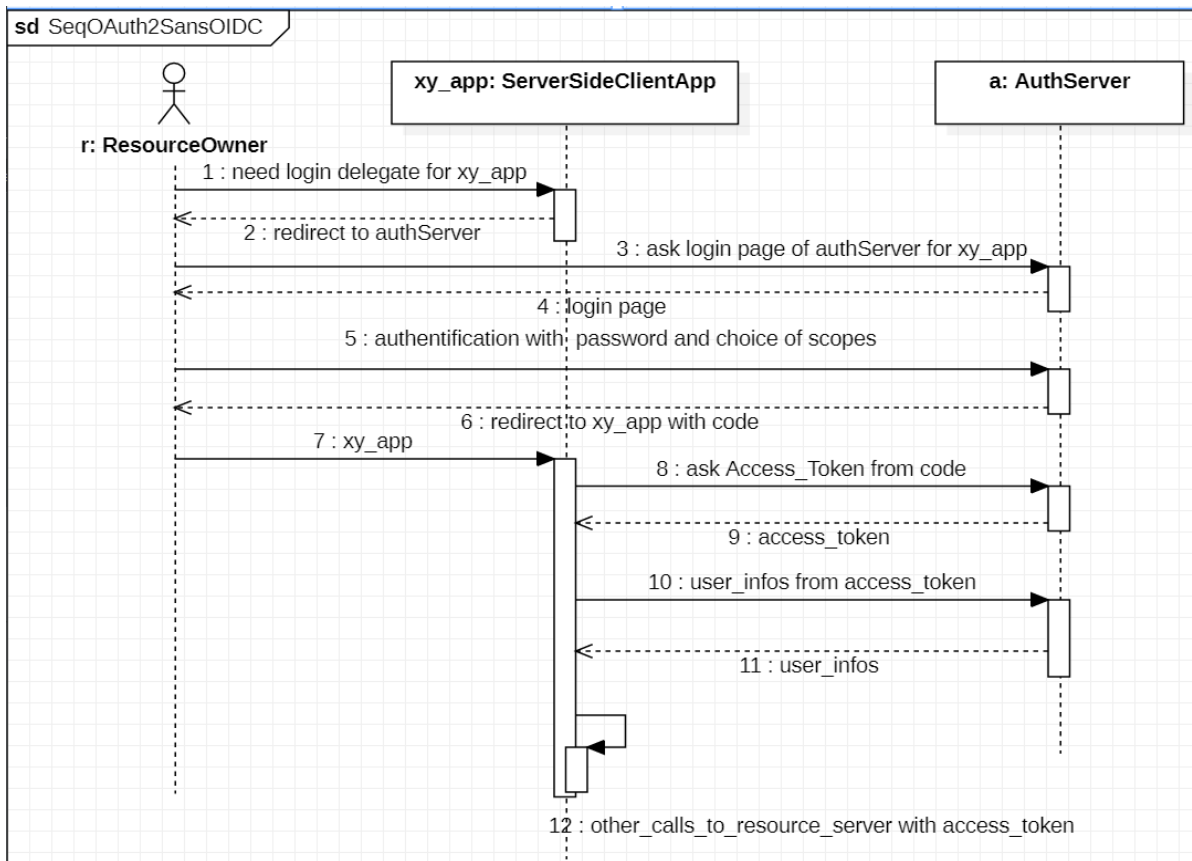
2. OIDC (OpenID Connect)

OpenId Connect

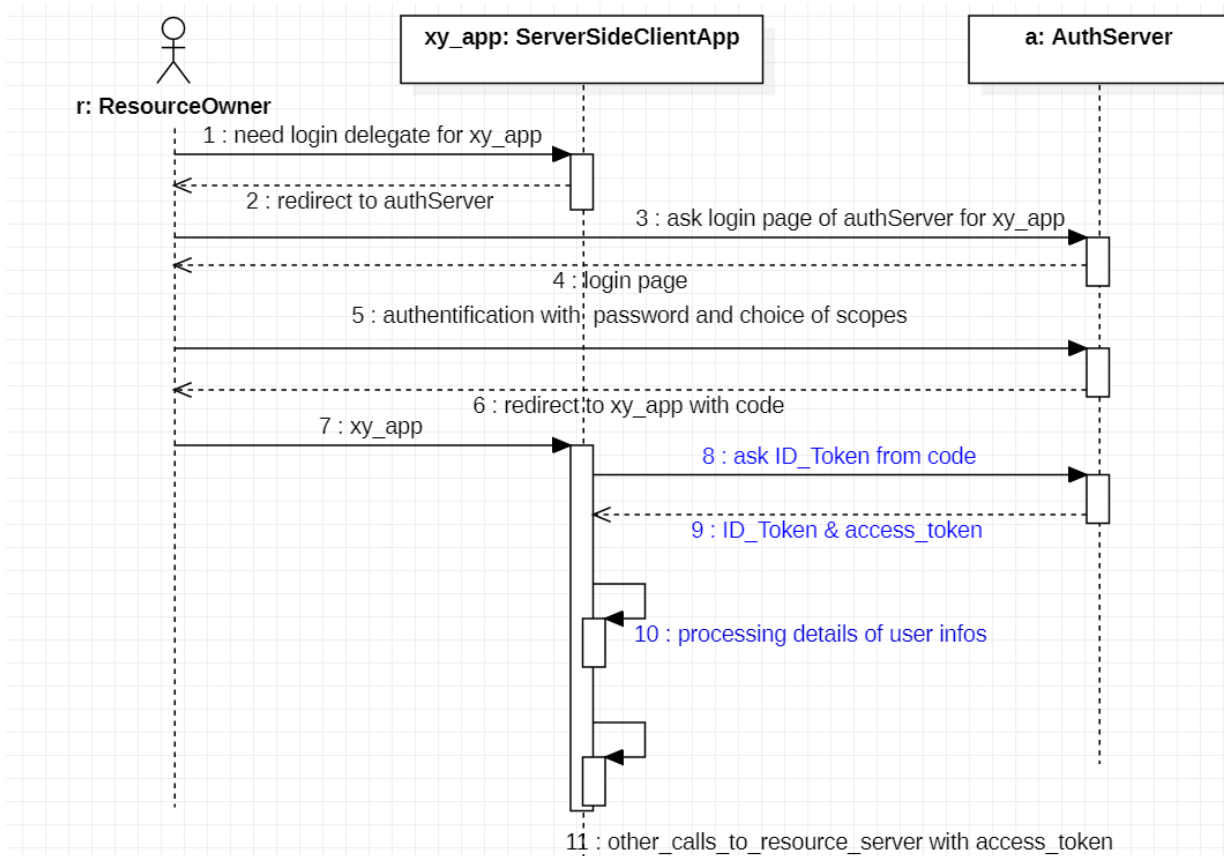
OpenID Connect (OIDC) est un protocole d'authentification forte basé sur une transmission standardisée ("**ID Token**" au format JWT) des informations sur l'utilisateur identifié .

OpenID Connect s'appuie en interne sur OAuth2 et permet d'**obtenir** simplement **des informations plus précises sur l'utilisateur à authentifier** .

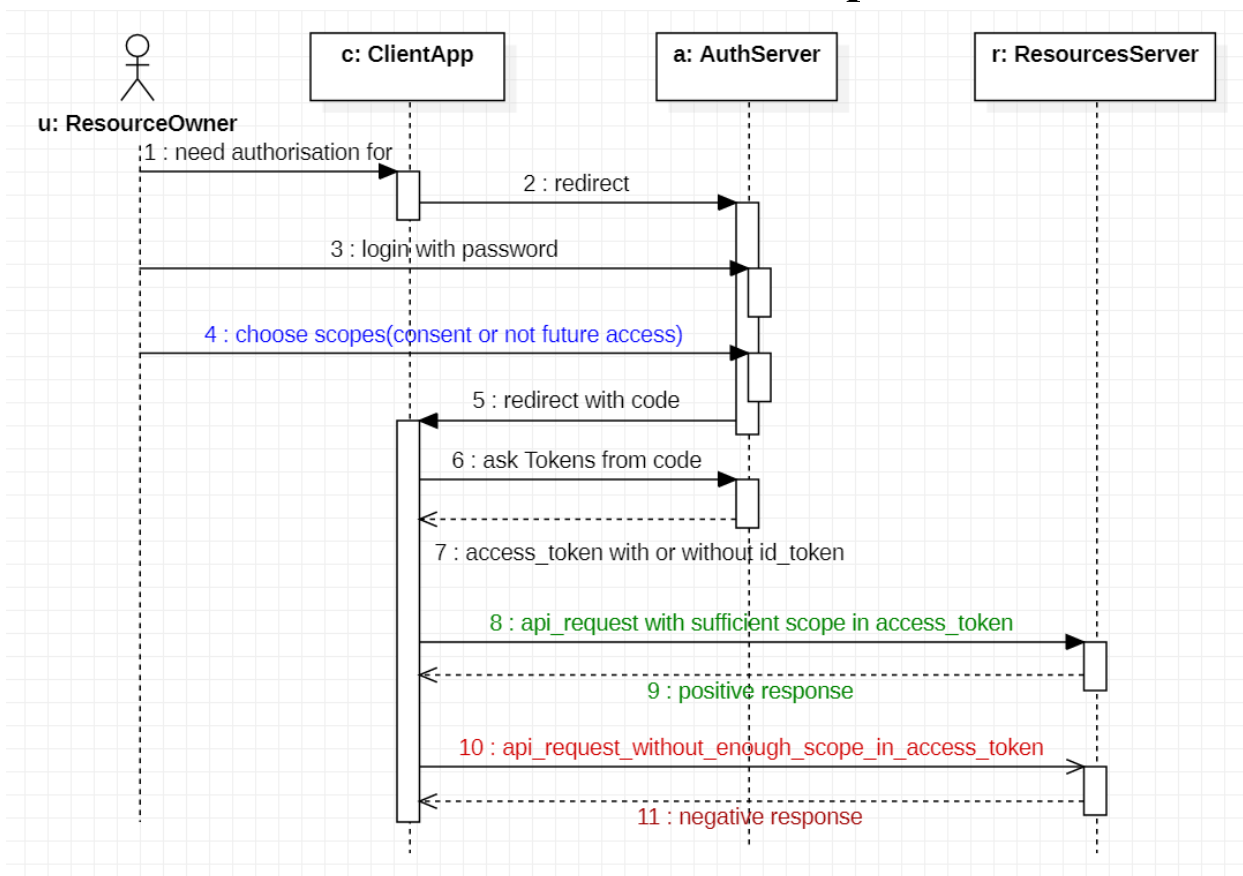
OAuth2 sans openID



avec OpenID Connect



OAuth2 or OIDC with Scopes



Quelques serveurs d'authentification OAuth2/OIDC

- Google **Hydra** (très perfectionné et très performant mais complexe car besoin de compléments à personnaliser)
- **keycloak** est un serveur d'autorisation oauth2/oidc basé sur jboss wildfly et java >=8 (relativement simple à installer). Il est par défaut basé sur une base H2 et dispose d'une ihm intégrée pour configurer des utilisateurs
- **okta** est une entreprise spécialisée dans l'identité numérique et offre des services de type "authorisation oauth2/oidc as a service"
- Les grandes plate-formes "cloud" (**Azure** , **AWS** , ...) intègrent un service OAuth2/OIDC (ex : **Azure Active Directory**, **Amazon-Cognito** , ...)
- quelques autres

3. Mise en oeuvre OAuth en java

Les deux projets ci après correspondent à des exemples de mise en oeuvre s'appuyant sur spring-security et basés sur le mode `grant_type="password"` :

- <https://github.com/didier-mycontrib/basicOAuth2AuthServer>
- <https://github.com/didier-mycontrib/basicOAuth2ResourceServer>

4. Mise en oeuvre OAuth avec NodeJs

avec **passportjs** et les plugins adéquats ...

ANNEXES

XI - Annexe – Eléments de sécurité

1. Eléments de sécurité (HTTP, ...)

1.1. Basic Http Auth et limitations

Positionnements possibles des informations d'authentification

En fin d'url :

`http://www.xy.com/zz/resource1?username=user1&password=pwd1`

Dans l'entête HTTP (avec **encodage base64** associé au standard "basic http auth") :

```
Authorization: Basic dXNlcjE6cHdkMQ==
Content-Length: 264
```

← *décodage immédiat (quasi "en clair")*

Limitations d'une authentification rudimentaire

Placer en fin d'url (ou bien dans l'entête HTTP) **une information d'authentification en clair (ou à peine cryptée via un encodage base64)** permet seulement de limiter l'accès aux utilisateurs qui connaissent le couple *username/password* .

Dans le cas où un "hacker" intercepte la requête et en récupère une copie, il connaît alors tout de suite le mot de passe et peut alors déclencher toutes les actions qu'il désire en se faisant passer par l'utilisateur "piraté" .

1.2. Cryptage élémentaire via hash (MD5 , SHA) + salt

"Basic Http Auth." ou fin d'URL avec hash(password)

Certains algorithmes standards de cryptage ("hachage") du mot de passe tels que "md5" ou "sha1" ou "..256" rendent très difficile le décryptage de celui-ci .

En véhiculant en fin d'URL (ou dans l'entête HTTP) le mot de passe haché

- celui-ci ne circule plus en clair (meilleure confidentialité).
- la base de données des mots de passe ne comporte que des informations cryptées et est donc moins vulnérable (si elle est piratée ou visualisée, les mots de passe "en clair" ne seront pas connus) .

Phases de la mise en oeuvre :

- Dès la saisie initiale du mot de passe, celui-ci est haché/crypté et stocké dans une base de données coté serveur.
- Lorsque le mot de passe est re-saisi coté navigateur , celui ci est de nouveau haché/crypté (via le même algorithme) avant d'être véhiculé vers le serveur
- Le serveur compare les deux "hachages/cryptages" pour vérifier une authentification correcte

"Basic Http Auth." avec "hash" et "salt"

- Un simple cryptage/hachage "md5" , "sha1" ou autre ne suffit souvent pas car il existe des bases de données d'associations entre mots de passe courants et les hachages correspondants "md5" ou "sha1" facilement accessibles depuis le web.
- D'autre part, si le mot de passe (en clair) peut ainsi être indirectement découvert, ceci peut être extrêmement problématique dans le cas où l'utilisateur utilise un même mot de passe pour plusieurs sites ou applications .
- De façon à prévenir les risques présentés ci-dessus, on utilise souvent un double encodage/cryptage prenant en compte une chaîne de caractère spécifique à l'application (ou à l'entreprise) appelée "salt" (comme grain de sel) .

Exemple :

`cryptedPwd = md5OuSha1("my_custom_salt" + md5OuSha1(pwd)) ;`

"hash" (+ "salt") (récapitulatif)

AlgoCryptage = **md5OuSha1(pwd)** ou bien
md5OuSha1("my_custom_salt" + md5OuSha1(pwd)) ;

②a saisie du mot
de passe en clair

②b cryptage via
algoCryptage

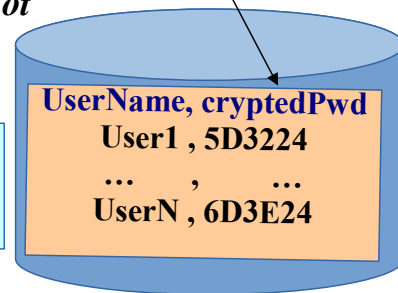
Coté client

④ comparaison du mot
de passe crypté avec
celui en base de
données

Coté
serveur

③ transfert du mot de
passe crypté via HTTP

① stockage crypté
en base de données
dès la saisie initiale



Même s'il est indéchiffrable , si le mot de passe crypté
est intercepté, l'authentification tombe à l'eau

1.3. Bcrypt pour crypter les mots de passe stockés en base

Algo. "bcrypt" pour les "password" stockés en base

Une authentification sérieuse consiste à utiliser conjointement HTTPS , des jetons et un algorithme de cryptage pour stocker les mots de passe en base.

L'algorithme "bcrypt" est tout à fait approprié pour crypter les mots de passe à stocker en base.

"Bcrypt" génère un "salt"/"clef" aléatoirement en fonction de n=10 ou autre et le résultat du cryptage n'est pas constant.

Bien que "non constant" et "avec clef jetée" , l'algorithme "bcrypt" peut déterminer si un ancien cryptage bcrypt récupéré en base correspond ou pas à un des cryptages possibles du mot de passe précisé en clair .

// cryptage avant stockage :

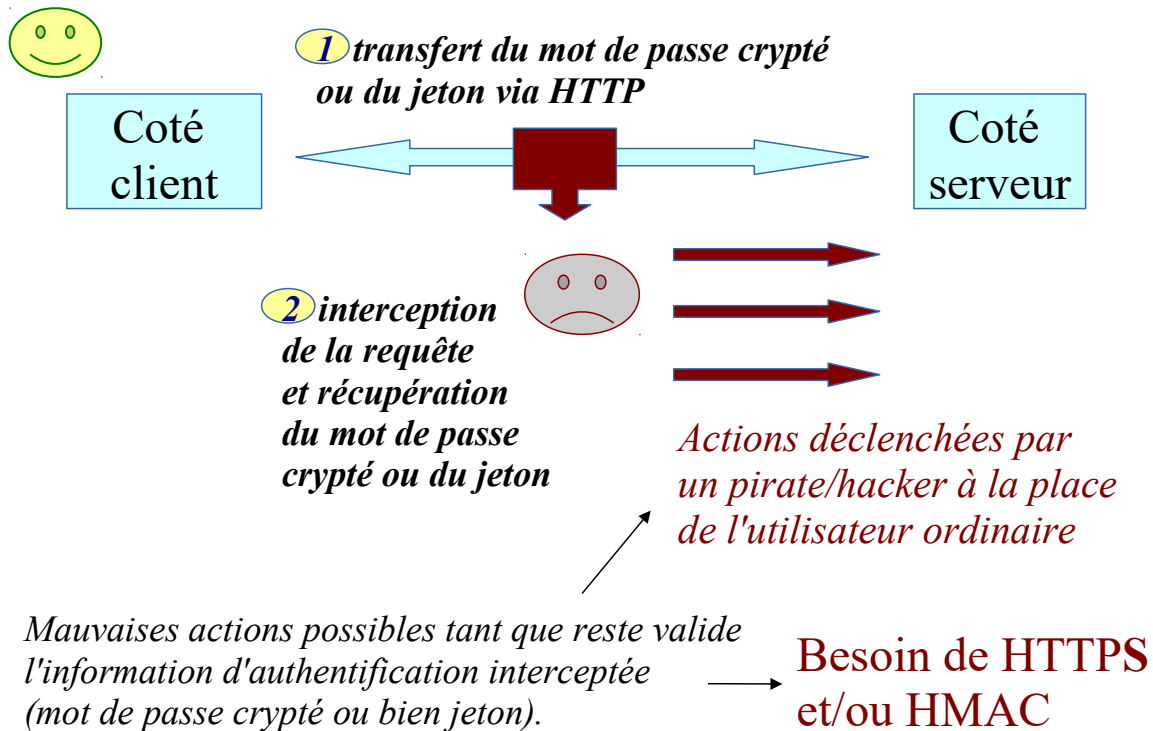
```
var bcryptedPwd =  
    bcrypt.hashSync(password, bcrypt.genSaltSync(12), null);
```

// comparaison lors d'une authentification :

```
if ( bcrypt.compareSync(password, bcryptedPwd) ) { ...} else { ...}
```

1.4. Problématique "man-in-the-middle"

Problématique "man in the middle"



1.5. HMAC

Signature des requêtes (avec clef secrète) et requête à usage unique (avec timestamp)

Pour éviter qu'une requête interceptée puisse conduire à une attaque de type "man in the middle", on peut **ajouter une signature de requête** rendant celle-ci **inaltérable (non modifiable)**.

Dans le cas, où la requête serait interceptée, le "hacker" ne pourrait que la rejouer telle quelle (sans pouvoir la modifier).

Pour, tout de même **éviter, qu'une requête puisse être relancée plusieurs fois**, il suffit d'**ajouter un "timestamp" au message à envoyer**.

NB: Ces 2 techniques sont assez souvent utilisées ensembles et la technologie d'authentification associée s'appelle **HMAC** (keyed-hash message authentication code)

HMAC avec timeStamp (partie 1 / coté "client")

1) l'application cliente prépare la requête (userName ou ... , timeStamp , paramètres , ...) . Celle-ci peut prendre la forme d'une URL en mode GET ou bien être la base d'un calcul d'empreinte en mode POST .

2) l'application cliente élabore une signature de requête :

signature=Base64(HMAC-SHA1(UTF-8-Encoding-Of(*request*), *clef*))

Exemple (javascript) :

```
var user = "powerUser";      // Récupéré depuis la page d'authentification.
var password = "topSecret";  // Récupéré depuis la page d'authentification.
const salt = "13@!azerty";  // ou autre (selon application)
var encryptedPassword = CryptoJS.SHA1(CryptoJS.SHA1(password)+salt);
var httpVerb = "GET" ;
var currentTime = +new Date(); // valeur du timeStamp
var url = "http://www.xx.yy/product?user=" + user + "&timestamp=" + currentTime;
var httpUrl = httpVerb + ":" + url;
var signature =
    CryptoJS.HmacSHA1(httpUrl,encryptedPassword).toString(CryptoJS.enc.Base64);
url = url + "&signature=" + signature; //à envoyer
```

HMAC avec timeStamp (partie 2 / coté "serveur")

1) l'application serveur reçoit la requête et en extrait le username (*en clair*).

2) l'application serveur récupère en base le mot de passe crypté (haché , salé) de l'utilisateur et s'en sert pour calculer une signature du message avec le même algorithme que du coté client .

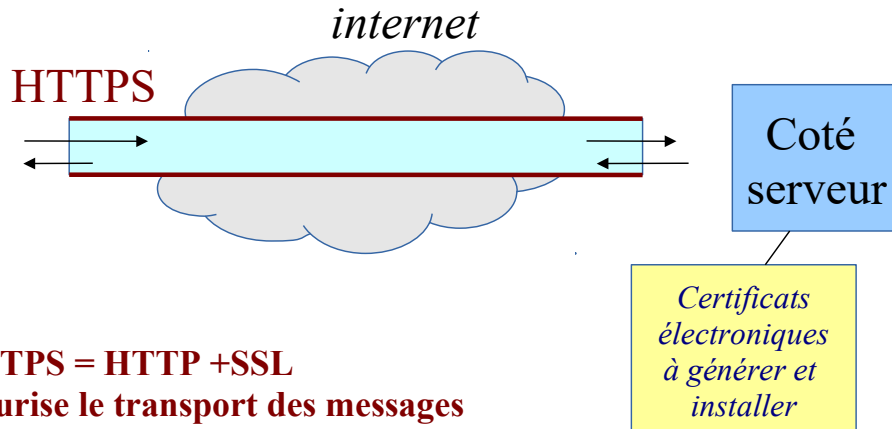
3) l'application serveur compare les deux signatures (reçue et re-calculée) pour vérifier que le message n'a pas été intercepté/modifié/altéré .

4) l'application serveur tente de récupérer en base le dernier "timeStamp" associé à l'utilisateur s'il existe (véhiculé par requête précédente).

Si le timeStamp qui accompagne la requête n'est pas inférieur ou égal au dernier "timeStamp" récupéré en base tout va bien (la requête n'a pas été lancée plusieurs fois) . Le timeStamp reçu est alors sauvegardé en base (pour le prochain test) , la requête est acceptée et traitée.

Conclusion : HMAC avec timeStamp garantit une **authentification robuste** mais ne gère pas la confidentialité des messages transmis, d'où l'éventuel besoin d'un complément HTTPS (HTTP + SSL) .

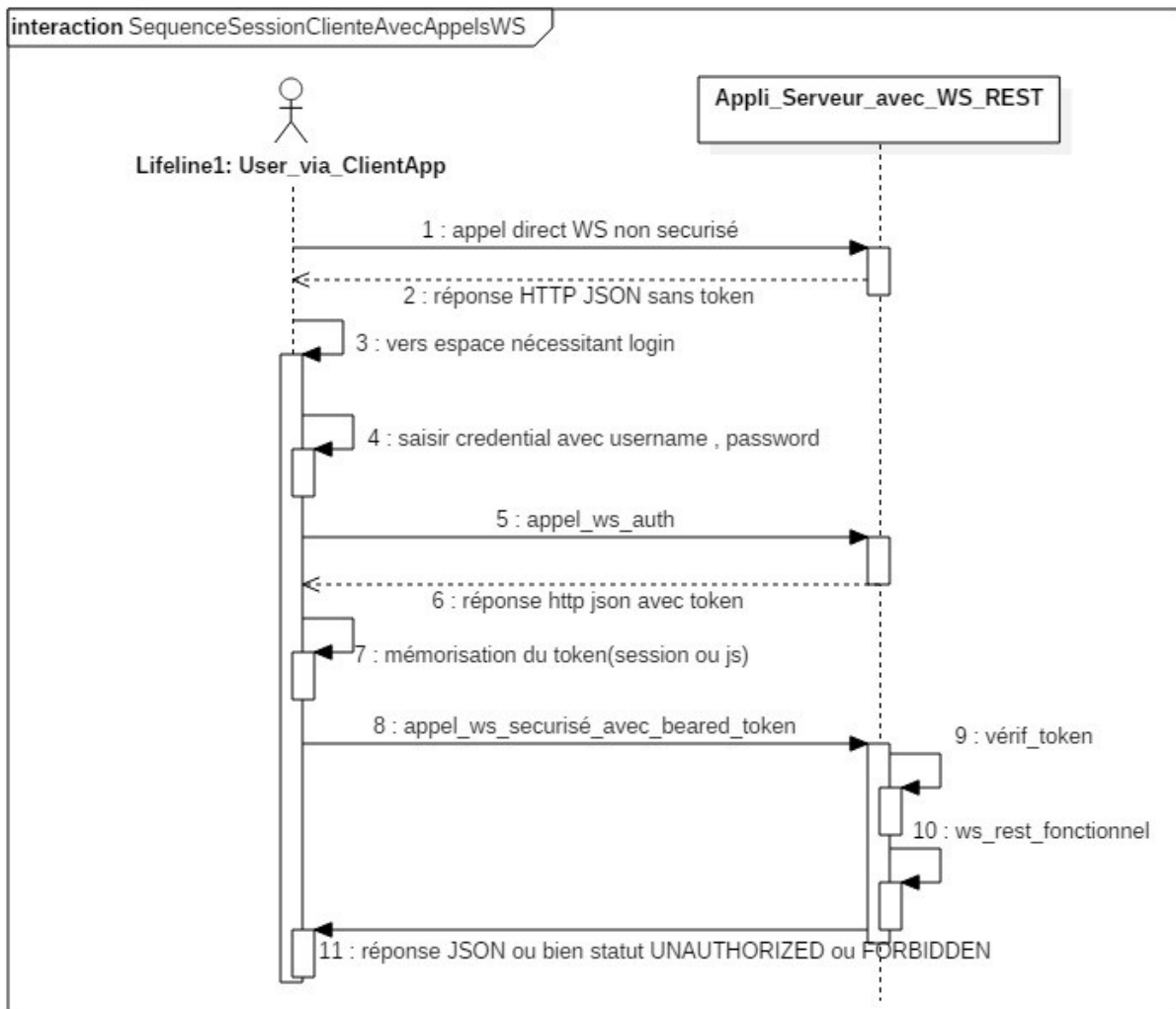
(HMAC avec timeStamp) ou ... + HTTPS



HTTPS = HTTP +SSL
sécurise le transport des messages
à travers le réseau internet
(confidentialité , protection contre
interception/modification, ...) mais
 ne gère pas (tout seul) l'authentification.

2. Sécurité pour WS-REST (généralités)

2.1. Pseudo session avec "token" plutôt que cookie :



Une application cliente qui appelle une série de WS-REST peut être développée avec des technologies très diverses :

- java standalone (swing, java-fx ,)
- java/jee (JSF+....) ou (Spring-web-mvc +)
- HTML + js (jquery ou angular ou react ou) au sein d'un navigateur avec appels ajax
- PHP , C++ , .net/C# , ...

De même , l'application serveur qui gère le WS-REST ne gère pas systématiquement une session HTTP.

Des jetons de sécurité ("token") sont généralement employés pour gérer l'authentification d'un utilisateur et d'une application dans le cadre d'une communication sous forme de WS-REST.

Le jeton de sécurité est généré si le couple (username,password) transmis est correctement vérifié coté serveur.

Ce "token" (véhiculé au format "string") pourra prendre la forme d'un uuid (universal unique id , exemple: e51cd176-a522-454c-9c0a-36ca74cdb2d0) ou bien être conforme au format JWT (Json

Web Token).

Dans le cas d'un token de type uuid , le coté serveur doit maintenir une liste ou une map des "tokens générés et valides" (éventuellement associés à certaines infos (username, role ,)).

Dans le cas d'un token sophistiqué de type jwt , le token généré comporte déjà en lui (de manière cryptée/extractible) certaines informations utiles (subject , roles ,) et donc pas besoin de map coté serveur.

Le protocole HTTP a normalisé la façon dont le token doit être retransmis au sein des requêtes émises du client vers le serveur (après l'authentification préalablement effectuée) :

Il faut pour cela utiliser le champ "Authorization :" de l'entête HTTP pas en mode "Basic" mais en mode "Bearer" (signifiant "au porteur" en français).

exemple (postman) :

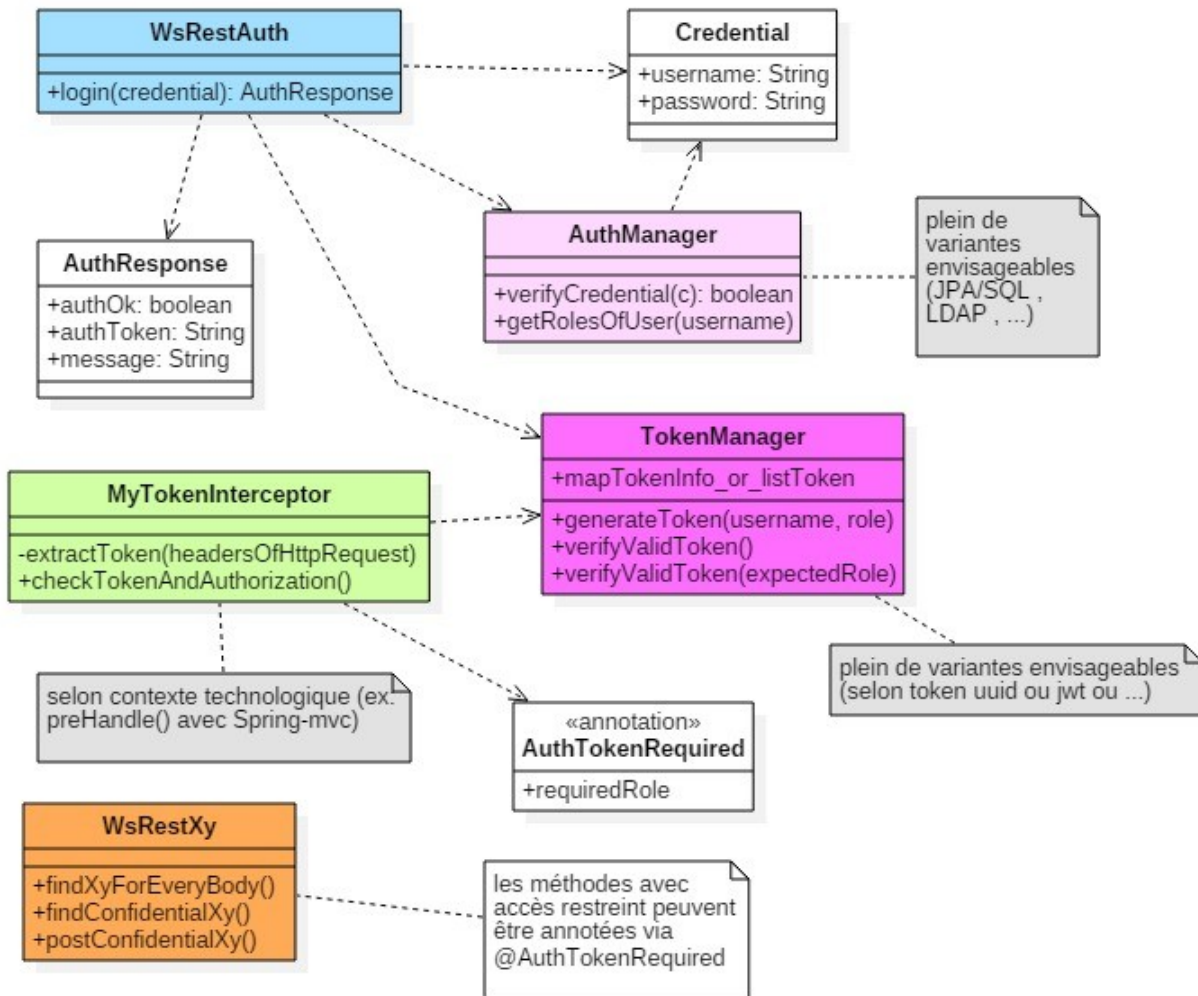
Authorization ●	Headers (1)	Body	Pre-request Script	Tests
	Key	Value		
	Authorization	Bearer e51cd176-a522-454c-9c0a-36ca74cdb2d0		
	New key	Value		

exemple (javascript / jquery) :

```
<script src="jquery-3.2.1.js"></script>
<script>
    var authToken; //token d'authentification (en mode bearer) à retransmettre
    ...

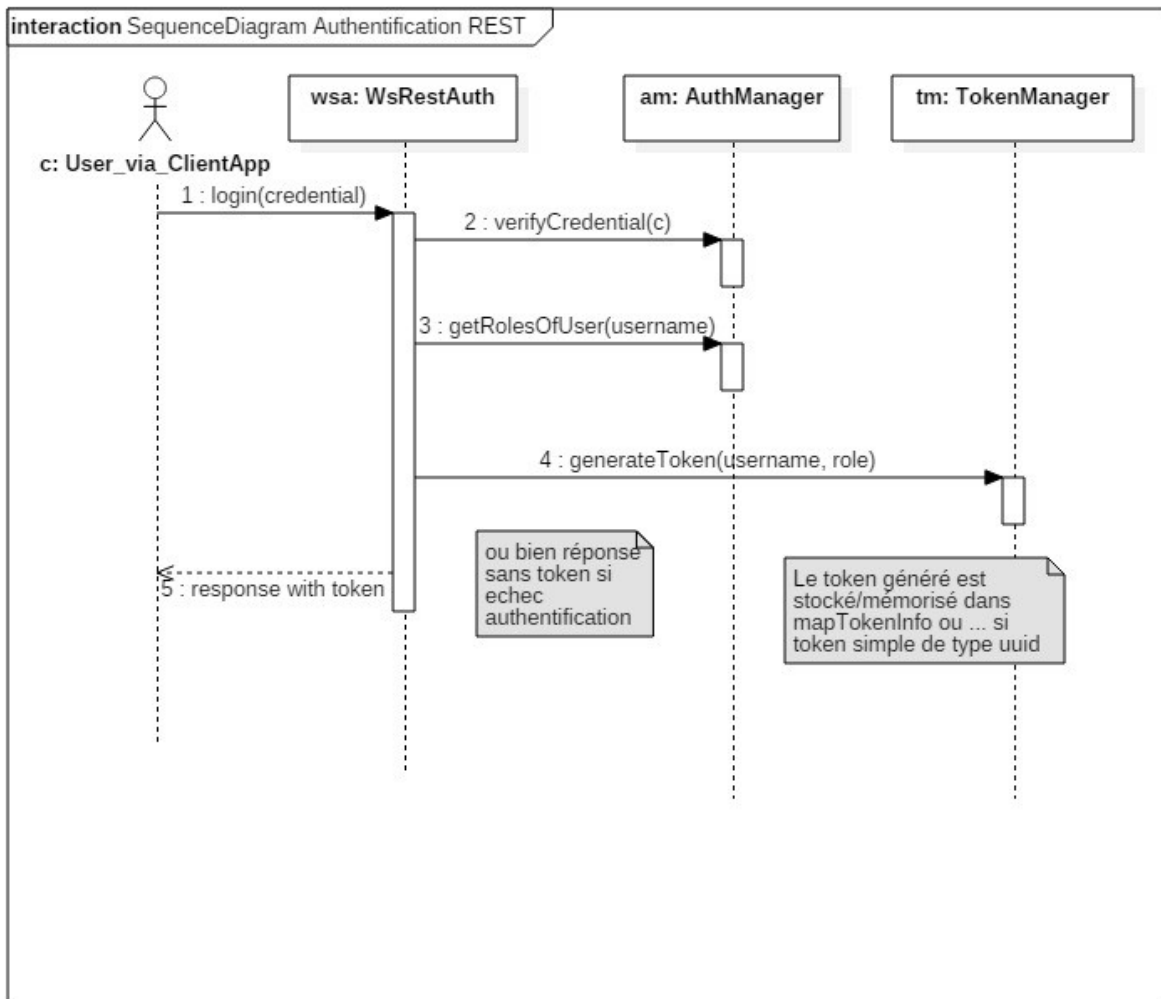
    $(document).ajaxSend(function(e, xhr, options) {
        //retransmission du jeton d'authentification
        //dans l'entête http de la requete ajax
        //xhr = XMLHttpRequest = objet technique du navigateur
        //qui déclenche les requêtes ajax
        xhr.setRequestHeader('Authorization','Bearer '+ authToken);
    });
    $.ajax({
        type: "GET",
        url: "ws/rest/confidential/news",
        contentType : "application/json" ,
        success: function (response) {
            if (response) {
                $("#spanMsg").html(JSON.stringify(response));
            }
        }
    });
    ...
```

2.2. Responsabilités techniques coté serveur :



<i>Composants</i>	<i>Responsabilités techniques</i>
AuthManager (gestionnaire d'authentification)	vérifier login/credential via dataBase ou autre
TokenManager (gestionnaire de "token")	Gérer (générer , vérifier, ...) une sorte de jeton (uuid , jwt,)
WsRestAuth (ws de login/authentification)	WS REST vérifiant login/credential et retournant token dans message de réponse global (ex : AuthResponse retourné au format JSON)
MyTokenInterceptor	Intercepteur technique (selon techno : Spring-mvc ou jax-rs ou ...) permettant de vérifier la validité du jeton véhiculée par une requête.
WsRestXy	WS REST fonctionnel avec partie en accès restreint annotée via @AuthTokenRequired

2.3. Service d'authentification / génération token



exemple (postman) :

POST ☐ http://localhost:8080/serverSpringMvc/ws/rest/auth/verifAuth

Authorization Headers (1) Body ☒ Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	application/json

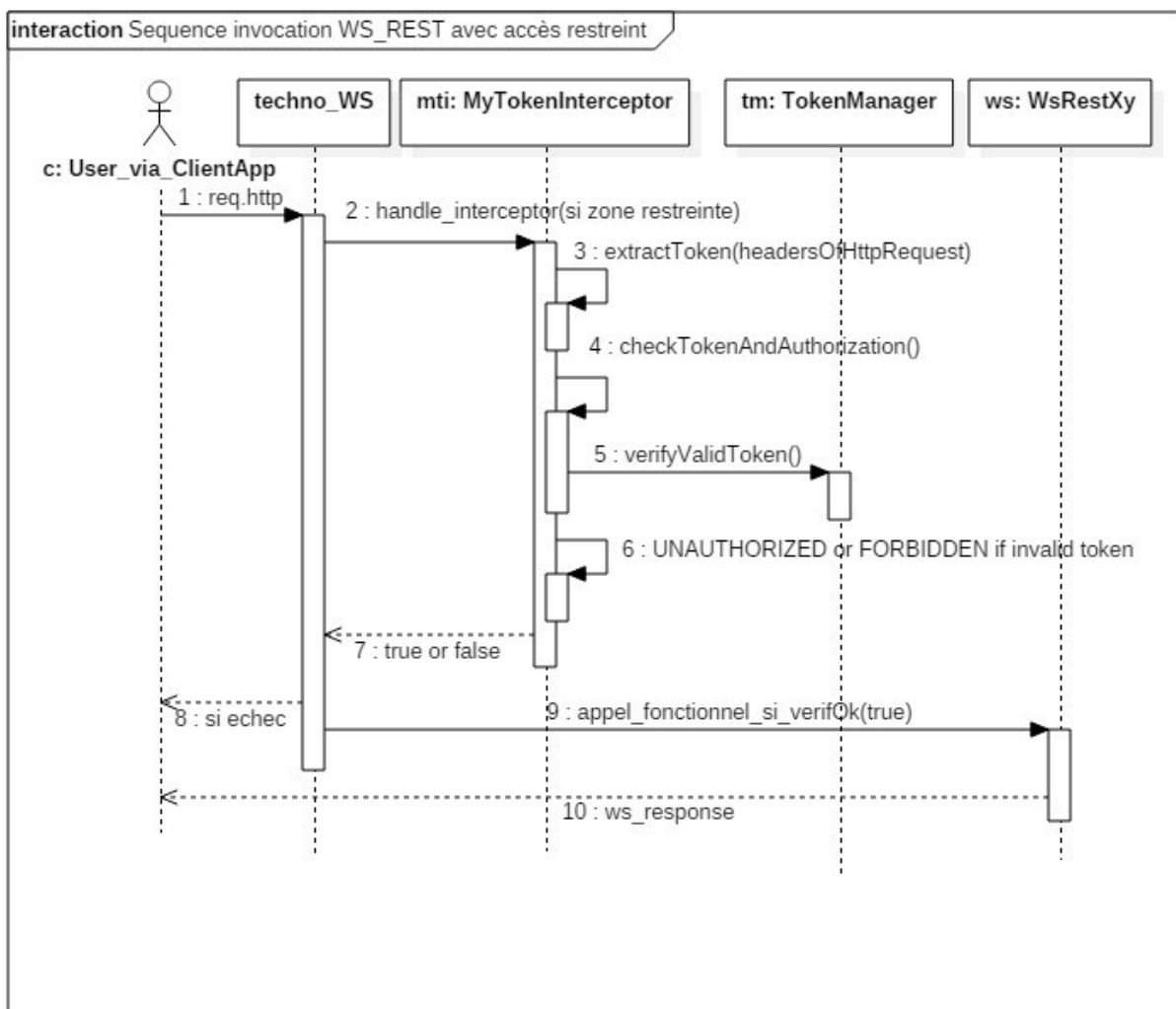
Authorization Headers (1) Body ☒ Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ☐

1 { "username" : "toto" , "password": "pwd_toto" }

```
{
  "authToken": "e51cd176-a522-454c-9c0a-36ca74cdb2d0",
  "authOk": true,
  "message": "authentification reussie"
}
```

2.4. Intercepteur et vérification d'un jeton



Status: 200 OK

si ok

Status: 403 Forbidden

ou bien si faux jeton (invalid token)

Status: 401 Unauthorized

ou bien si pas de jeton

3. Sécurité WS-REST avec Spring-MVC

3.1. Exemple (très basique) de WS REST d'authentification :

```
public class Credential {
    private String username;
    private String password;
    //...
}
```

```
public class AuthResponse {
    public String authToken;
    public Boolean authOk;
    private String message; //...
}
```

```
@RestController
@RequestMapping(value="/rest/auth" ,headers="Accept=application/json" )
public class WsAuth {
    //url = http://localhost:8080/serverSpringMvc/ws/rest/auth/verifAuth
    //avec { "username" : "toto" , "password": "pwd_toto" }
    //à tester avec POSTMAN (POST , raw , et Content-Type application/json dans header)
    @RequestMapping(value="/verifAuth" ,method=RequestMethod.POST )
    public ResponseEntity<AuthResponse> postAuth(@RequestBody Credential credential)
    {
        //code à peaufiner pour rendre plus fiable (avec gestion exception)
        AuthResponse authResponse = new AuthResponse();
        if(credential.getPassword().equals("pwd_" + credential.getUsername())){
            authResponse.setAuthOk(true);
            authResponse.setMessage("authentification reussie");
            authResponse.setAuthToken(AuthUtil.generateToken());
        }
        else{
            authResponse.setAuthOk(false);
            authResponse.setMessage("echec authentification");
        }
        return new ResponseEntity<AuthResponse>(authResponse,HttpStatus.OK);
    }
}
```

```
public class AuthUtil {
    //à peaufiner sur un vrai projet : supprimer les très anciens jetons
    private static List<String> listeJetons = new ArrayList<String>();

    public static String generateToken(){ String token=null;
        token = java.util.UUID.randomUUID().toString();
        listeJetons.add(token); return token;
    }

    public static boolean verifyToken(String token){
        return listeJetons.contains(token); //jeton considéré comme valide
        //si dans la liste des jetons générés et mémorisés
    }

    public static String extractBearerTokenFromHttpHeaders(HttpHeaders headers){
        List<String> listOfAuthorization= headers.get(HttpHeaders.AUTHORIZATION);
        if(listOfAuthorization==null || listOfAuthorization.size()==0){
            return null;
        }
        String mainAuthorisation = listOfAuthorization.get(0);
        System.out.println(mainAuthorisation);
        if(mainAuthorisation.length()<8) {
            return null;
        }
        if(mainAuthorisation.startsWith("Bearer")){
            return mainAuthorisation.substring(7);
        }
        return null;
    }
}
```

```
}
}
```

3.2. Exemple (basique) de WS-REST sécurisé sans intercepteur

```
@RestController
@RequestMapping(value="/rest/confidential" ,headers="Accept=application/json" )
public class WsConfidentiel {

    //url = http://localhost:8080/serverSpringMvc/ws/rest/confidential/news
    @RequestMapping(value="/news" ,method=RequestMethod.GET )
    public ResponseEntity<News> getNews(@RequestHeader HttpHeaders httpHeaders){
        String token = AuthUtil.extractBearerTokenFromHttpHeaders(httpHeaders);
        if(AuthUtil.verifyToken(token)){
            News news = new News();
            news.setTitre("news of the day");
            news.setTexte("you know what ? i am happy !");
            return new ResponseEntity<News>(news,HttpStatus.OK);
        }
        else {
            return new ResponseEntity<News>(HttpStatus.UNAUTHORIZED);
            //ou HttpStatus.FORBIDDEN;
        }
    }
}
```

3.3. Exemple amélioré de WS sécurisé avec intercepteur

```
/**
 * annotation permettant de paramétrer le besoin
 * en Token d'authentification sur une méthode
 * d'un @RestController
 * NB: cette annotation sera examinée (par reflection) par MyMvcAuthInterceptor
 */

@Target({ElementType.METHOD /*, ElementType.TYPE */})
@Retention(RetentionPolicy.RUNTIME)
public @interface AuthTokenRequired {
    public static final String DEFAULT_REQUIRED_ROLE="any";
    public String requiredRole() default DEFAULT_REQUIRED_ROLE;
}
```

```
public class MyMvcInterceptor implements HandlerInterceptor {

    private static Logger logger = LoggerFactory.getLogger(MyMvcInterceptor.class);

    private String extractBearerTokenFromAuthHeader(String authorizationHeader){
        logger.debug("in MyMvcAuthInterceptor, AuthorizationHeader:" + authorizationHeader);
        if(authorizationHeader.length()<8) {
            return null;
        }
        //Format de l'autorisation standard http:
        //Authorization: Bearer 1234ab344..token_au_porteur...566
        //ou bien
        //Authorization: Basic A45D3455
        if(authorizationHeader.startsWith("Bearer")){
            return authorizationHeader.substring(7);
        }
    }
}
```

```

    }
    return null;
}

```

@Override

```

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
throws Exception {
    if(!(handler instanceof HandlerMethod)) return true;
    HandlerMethod handlerMethod = (HandlerMethod) handler;
    AuthTokenRequired tokenRequiredAnnot =
        handlerMethod.getMethodAnnotation(AuthTokenRequired.class);
    if(tokenRequiredAnnot==null){
        return true; //rien à vérifier
    }else{
        String authorizationHeader = request.getHeader(HttpHeaders.AUTHORIZATION);
        if(authorizationHeader==null){
            response.setStatus(HttpStatus.UNAUTHORIZED.value());
            return false;
        }else{
            String authToken = extractBearerTokenFromAuthHeader(authorizationHeader);
            if(authToken ==null){
                response.setStatus(HttpStatus.UNAUTHORIZED.value());
                return false;
            }else{
                if(!AuthUtil.verifyToken(authToken)){
                    response.setStatus(HttpStatus.FORBIDDEN.value());
                    return false;
                }
            }
        }
        return true; //if allowed
    }
}

```

@Override

```

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView
modelAndView) throws Exception { //logger.debug("in Interceptor, MyMvcAuthInterceptor.postHandle() : ...");
}

```

@Override

```

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception
exception) throws Exception { //logger.debug("in Interceptor, MyMvcAuthInterceptor.afterCompletion() : ...");
}
}

```

Pour déclarer l'existence de cet intercepteur , on peut (dans beans.xml ou ...) insérer la config spring-mvc suivante :

```

<mvc:interceptors>
    <bean class="fr.afcepf.dja.rest.MyMvcInterceptor" />
</mvc:interceptors>

```

Configuration équivalente (en mode java-config) :

@Configuration

```

public class MyWebMvcInterceptorConfig extends WebMvcConfigurerAdapter{

    @Bean
    public MyMvcAuthInterceptor myMvcAuthInterceptor(){
        return new MyMvcInterceptor();
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {

```

```
        registry.addInterceptor(myMvcAuthInterceptor());  
    }  
}
```

NB: Beaucoup d'améliorations possibles (jwt , lien avec spring-security ,)

XII - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP