

# Presentation Api Rest, RestFull et Symfony API

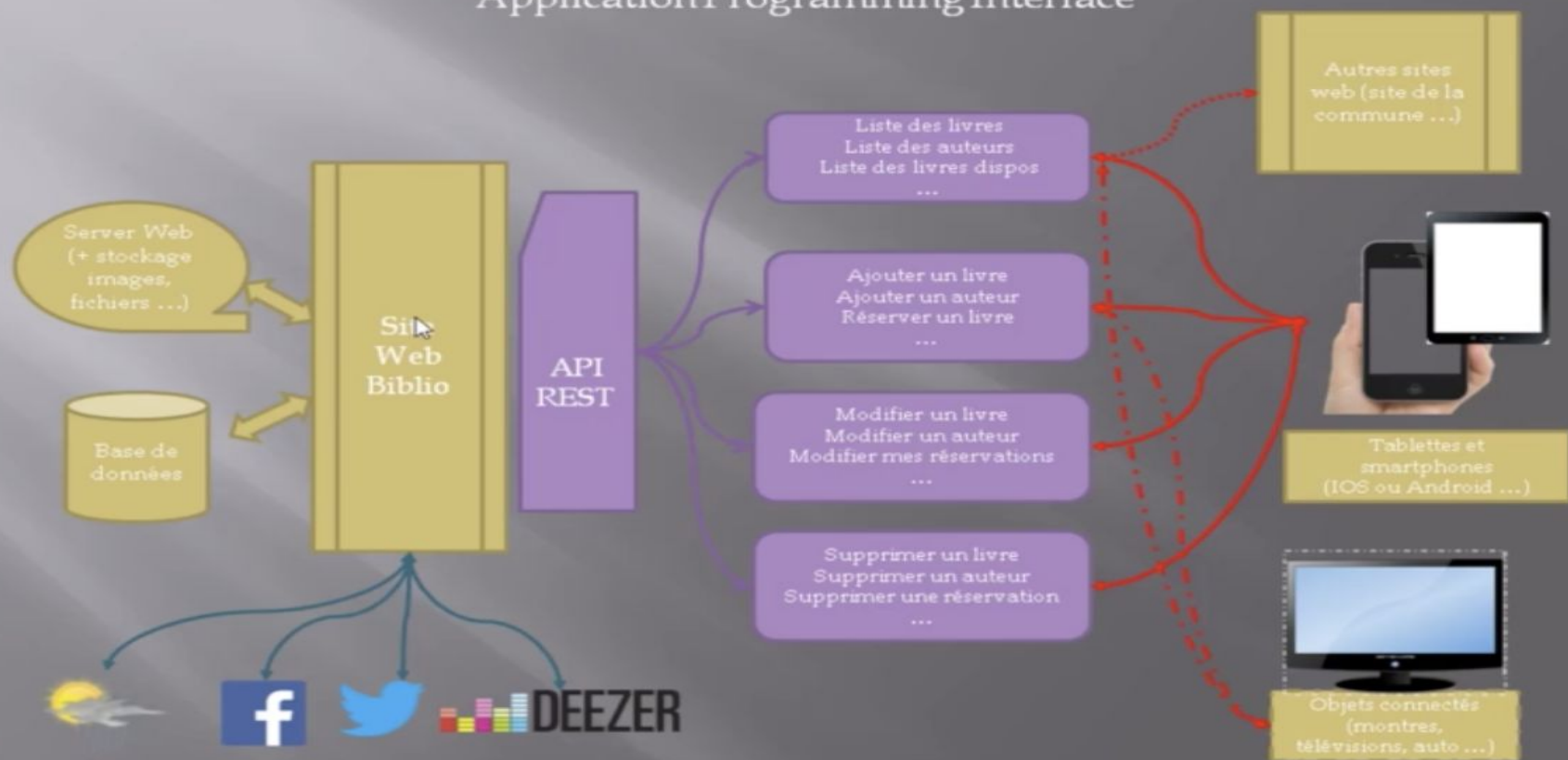
Birane Baila Wane

## **A) Section 1 :Notion Api Rest et Api RestFull**

- 1) Qu'est ce qu'une API**
- 2) Exemples D'API**
- 3) Architecture Rest basé sur le protocole HTTP**
- 4) Contrainte Architecture de API REST**
- 5) API RESTFULL: Modèle de Maturité de Richardson**

# QU'EST CE QU'UNE API

Application Programming Interface



# Raisons et Intérêts des API

1. **Besoins de multiplier les points de consommations (Site web, Smartphone, Objets connectés)**
2. **Besoin de centraliser la couche métier (accès à la base de données, Stockage fichiers, accès par authentification)**

# Quelques Exemples API

<https://geo.api.gouv.fr/>

# La requête HTTP

Ex: `http://monsite.com/auteurs?nom=martin&prenom=durant`

## Méthode HTTP

GET : pour récupérer une ressource sans la modifier (select)  
POST : pour créer une ressource (create)  
PUT : pour modifier une ressource (update)  
DELETE : pour supprimer une ressource (delete)

URI : ce qu'il y a après le nom de domaine

Version du protocole : HTTP 1.1 ou HTTP/2.0  
(les autres versions étant obsolètes)

La request line (1<sup>ère</sup> ligne)

GET /auteurs?nom=martin&prenom=durant HTTP/1.1

Les entêtes (headers)

Une ligne par entête finissant par CRLF (saut de ligne)

Host: www.monsite.fr

CRLF

← Seule ligne obligatoire

Accept: text/html

CRLF

← Type de contenu accepté (ou attendu) - voir type MIME

User-agent = Mozilla/4.0

CRLF

CRLF

Contenu de la requête : le corps  
(il n'y a pas contenu pour une requête GET)

Ici pour une requête POST ou PUT on retrouve les éléments du formulaire transmis ( Nom=martin&prenom=durant)

CRLF

# Response HTTP

## La reponse HTTP

Ex: `http://monsite.com/auteurs?nom=martin&prenom=durant`

Code Status (code à 3 chiffres) voir la liste complète :

- 1xx : information
- 2xx : succès (200 pour ok, 201 pour Created ...)
- 3xx : redirection
- 4xx : requête invalide (400, 401, 403, 404, 405)
- 5xx : erreur serveur (500, 501, 503, 505)

Version du protocole utilisé

Message: équivalent textuel du code status

La request line (1<sup>ère</sup> ligne)

HTTP/1.1 200 OK

Les entêtes (headers)

Une ligne par entête finissant par CRLF (saut de ligne)

Date: Thu, 11 Jan 2018 ...

CRLF

Server: Apache/2.0.54

CRLF

Content-Type: Application/Json

CRLF

Type de média

Content-Length: 53

CRLF

Taille de la réponse

Content-Location: /auteur/2

CRLF

URL de redirection

Cache-control: no-cache

Gestion du cache (très important pour les performances)

CRLF

Corps de la reponse

Dépend de la reponse (du html, Json, Xml ...)

# Codes Statut de la Réponse

## La reponse HTTP

Ex: <http://monsie.com/auteurs?nom=martin&prenom=durant>

### Code S

• 1xx : i  
• 2xx : s  
• 3xx : r  
• 4xx : n  
• 5xx : e

La requ

- **200 OK** • Tout s'est bien passé ;
- **201 Created** • La création de la ressource s'est bien passée (en général le contenu de la nouvelle ressource est aussi renvoyé dans la réponse, mais ce n'est pas obligatoire - on ajoute aussi un *header* **Location** avec l'URL de la nouvelle ressource) ;
- **204 No content** • Même principe que pour la 201, sauf que cette fois-ci, le contenu de la ressource nouvellement créée ou modifiée n'est pas renvoyé en réponse ;
- **304 Not modified** • Le contenu n'a pas été modifié depuis la dernière fois qu'elle a été mise en cache ;
- **400 Bad request** • La demande n'a pas pu être traitée correctement ;
- **401 Unauthorized** • L'authentification a échoué ;
- **403 Forbidden** • L'accès à cette ressource n'est pas autorisé ;
- **404 Not found** • La ressource n'existe pas ;
- **405 Method not allowed** • La méthode HTTP utilisée n'est pas traitable par l'API ;
- **406 Not acceptable** • Le serveur n'est pas en mesure de répondre aux attentes des entêtes **Accept** . En clair, le client demande un format (XML par exemple) et l'API n'est pas prévue pour générer du XML ;
- **500 Server error** • Le serveur a rencontré un problème.



# Architecture REST

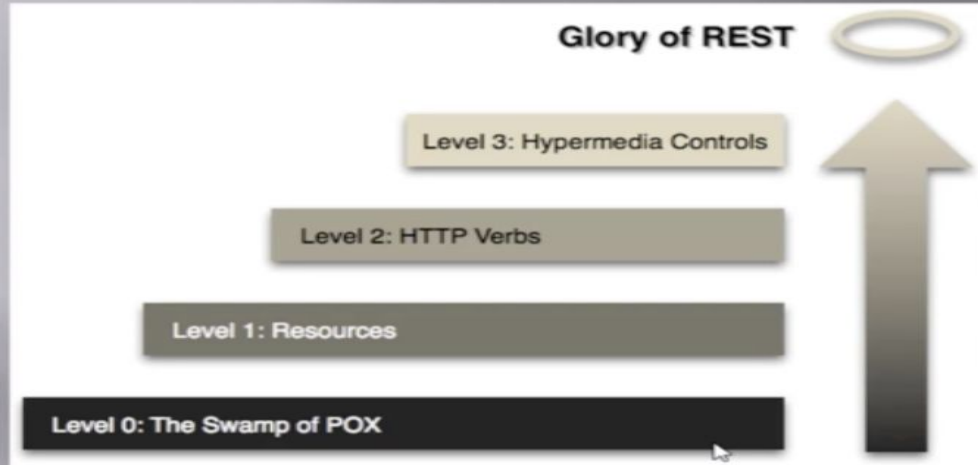
## Les 6 contraintes :

## Representational State Transfert

1. **Architecture client/serveur** : c'est le cas du protocole HTTP
2. **Stateless (sans état)** : on ne conserve pas de contexte entre les requêtes donc pas de variable de session par exemple. L'utilisateur doit donc être authentifié à chaque requête !
3. **Cacheable** : on doit pouvoir mettre en cache la ressource pour s'en resservir pour des requêtes similaires consécutives.
4. **Latered system** (système scindée en couche) : le client n'a pas à savoir comment est générée la ressource
5. **Uniform Interface** : contrainte au niveau de la ressource qui doit :
  - Posséder un identifiant unique (combinaison Methode -URI unique) :  
Ex : ces deux requêtes sont différentes
    - GET /auteurs (va récupérer la liste des auteurs )
    - POST /auteurs (va créer un auteur)
  - Avoir une représentation : il faut choisir la manière de formater la réponse et s'y tenir
  - être auto-décrite : il s'agit simplement de préciser le format de la réponse (Json, Xml, CSV ...) grâce au content-type dans le header.

```
{  
  "codeAuteur": 150,  
  "nomAuteur": "James",  
  "nationaliteAuteur": "Américain"  
}
```

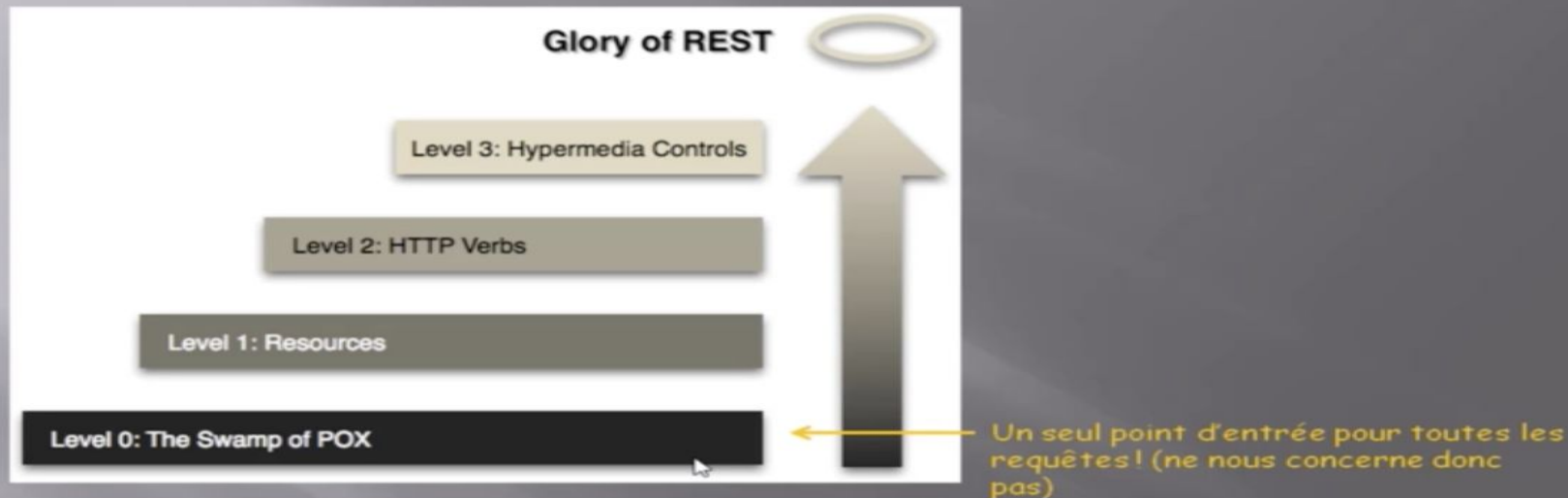
# Modèle de maturité de Richardson



Permet d'évaluer son API et son degré d'adhésion aux normes REST  
Plus les recommandations sont respectées, plus l'API est dite « RESTFul »

## Level 1

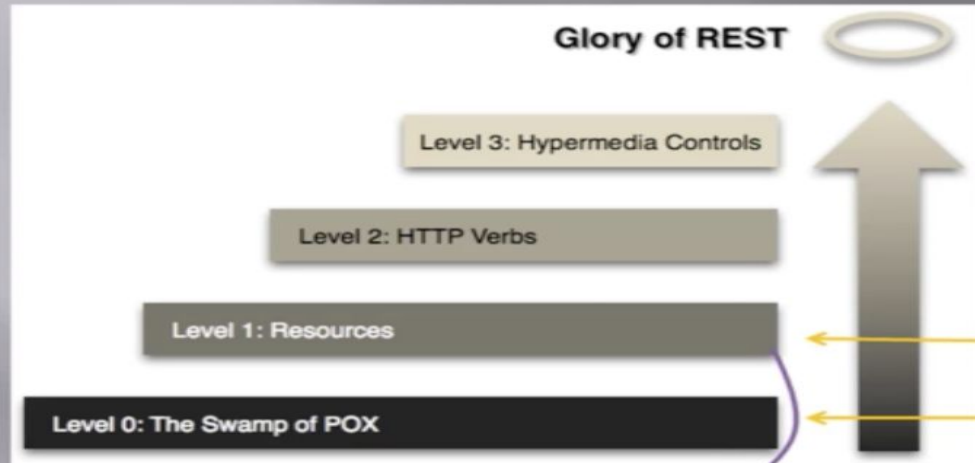
# Modèle de maturité de Richardson



Permet d'évaluer son API et son degré d'adhésion aux normes REST  
Plus les recommandations sont respectées, plus l'API est dite « RESTFul »

# Level 1

## Modèle de maturité de Richardson



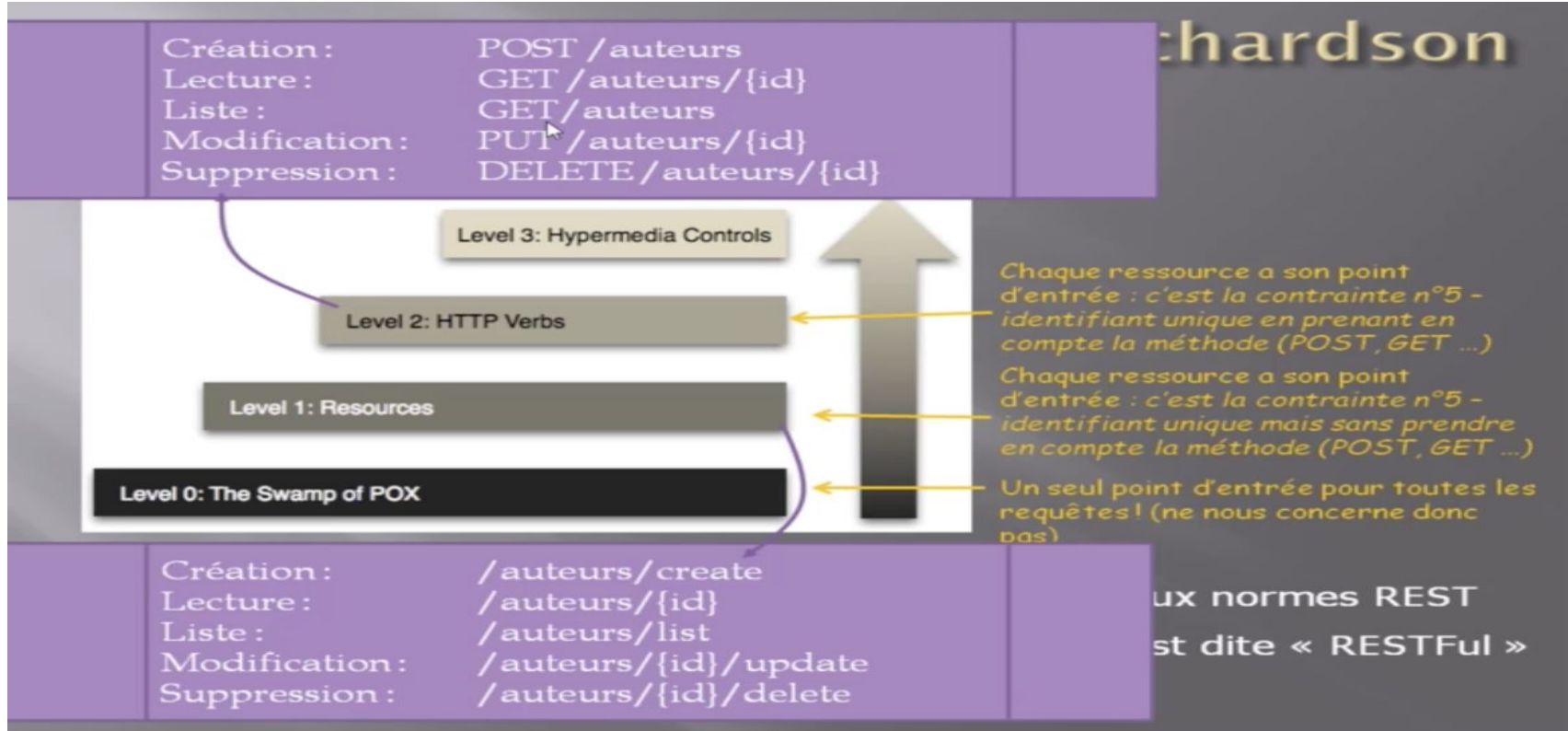
Chaque ressource a son point d'entrée : c'est la contrainte n°5 - identifiant unique mais sans prendre en compte la méthode (POST, GET ...)

Un seul point d'entrée pour toutes les requêtes! (ne nous concerne donc pas)

Création :	/auteurs/create
Lecture :	/auteurs/{id}
Liste :	/auteurs/list
Modification :	/auteurs/{id}/update
Suppression :	/auteurs/{id}/delete

aux normes REST  
est dite « RESTFul »

## Level 2



## Level 3

### Modèle de maturité de Richardson

```
{  
  "id" : 1,  
  "nomAuteur" : "James",  
  "nationaliteAuteur" : "Américain",  
  "links" : {  
    "update" : "http://domain.name/auteur/1",  
    "associated" : "http://domain.name/livre/16"  
  }  
}
```

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

Gérer les liens vers les ressources  
liées donc rendre l'API auto  
découvrable

Chaque ressource a son point  
d'entrée : c'est la contrainte n°5 -  
identifiant unique en prenant en  
compte la méthode (POST, GET ...)

Chaque ressource a son point  
d'entrée : c'est la contrainte n°5 -  
identifiant unique mais sans prendre  
en compte la méthode (POST, GET ...)

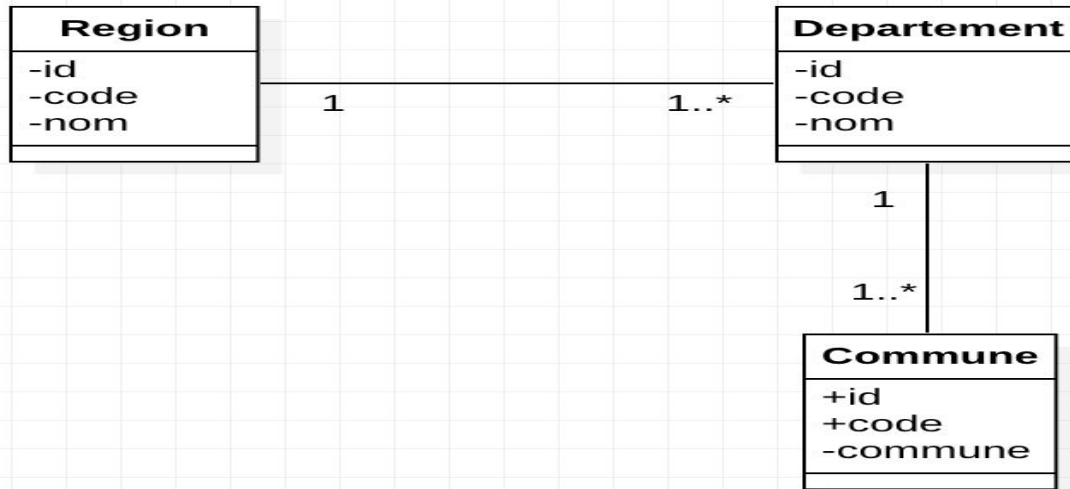
Un seul point d'entrée pour toutes les  
requêtes! (ne nous concerne donc  
pas)

Permet d'évaluer son API et son degré d'adhésion aux normes REST  
Plus les recommandations sont respectées, plus l'API est dite « RESTful »

## Section 2: Manipulation d'un Api Existence

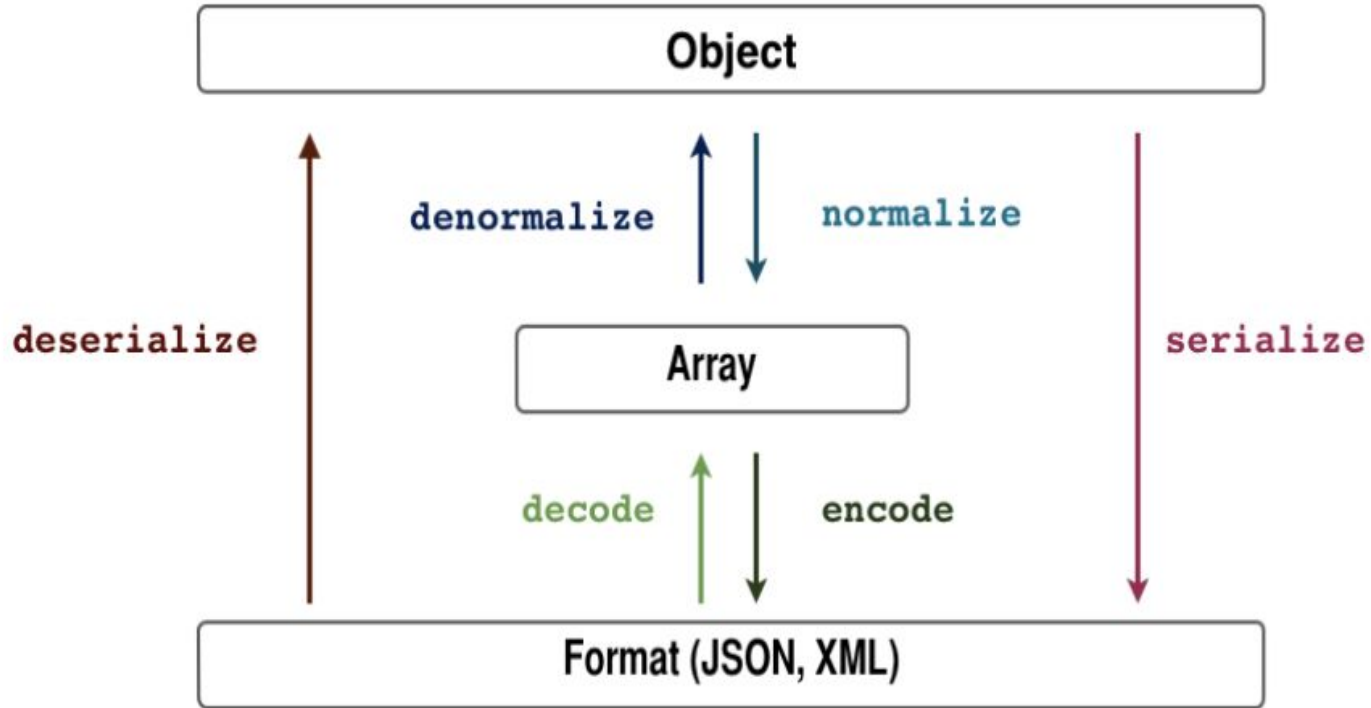
- 1) Schéma de principe de passage entre format Object et Json
- 2) Consommer une API externe
- 3) Créer une API
- 4) Insérer des Données par Fixtures
- 5) Insérer les données à partir du Bundle Easy Admin
- 6) Gestion Référence circulaire
- 7) Gestion des Objets Imbriqués

# Diagramme de Classe





# 1) Schéma de principe de passage entre format Object et Json



## **Consommer une API externe**

**Utilisation de l'api:**

**Gestion des communes**

<https://geo.api.gouv.fr/>

1) **Creer un projet Symfony Api**

**composer create-project symfony/skeleton my\_project\_name**

2) **Installation des Bundles Suivants**

**a) composer require symfony/maker-bundle --dev**

**b) composer require doctrine/annotations**

**c) composer require symfony/orm-pack**

3) **Creation de la Base donnee**

**php bin/console doctrine:database:create**

4) **Creation d'un Controller**

**php bin/console make:controller Api**

# Sérialisation et Désérialisation

## Installation du Bundle Serializer

**composer require symfony/serializer-pack**

## Création de l'entité Région

**php bin/console make:entity region**

# Application 1: Récupérer les régions de l'api et les ajouter dans la BD

Api: <https://geo.api.gouv.fr/regions> retourne du Json

//Recuperation des Regions en Json

```
$regionJson=file_get_contents("https://geo.api.gouv.fr/regions");
```

Methode 1:

a) Decode : conversion Json to Tableau

```
$regionTab=$serializer->decode($regionJson,"json");
```

## b) Dénormalisation: Conversion de Tableau vers Objet

=> Region(Tab) to Object

```
$serializer->denormalize($region, 'App\Entity\Region')
```

=> Tab Region(tab) to Tab Object

```
$serializer->denormalize($regionTab, 'App\Entity\Region[]')
```

```
$regionObject=$serializer->denormalize($regionTab, 'App\Entity\Region[]');
```

Methode 2:

## c) Déserialise : conversion Json to Object

```
$regionObject = $serializer->deserialize($regionJson, 'App\Entity\Region[]', 'json');
```

NB: Retourner un Résultat en Json

```
|  
}  
}
```

```
return new JsonResponse($resultat, 200, [], true);
```

On donne le résultat à  
envoyer

On précise le code statut

On donne éventuellement  
sous forme de tableau des  
infos sur le context (header ..)

On précise ici si \$resultat est  
déjà en json (true) ou n'est  
pas en json (false)

# Code

```
/**
 * @Route("/api/regions", name="api_all_regions",methods={"GET"})
 */
public function addRegions(SerializerInterface $serializer)
{
    //Recupere les Regions en Json
    $regionJson=file_get_contents("https://geo.api.gouv.fr/regions");

    //Methode 1:
    // $regionTab=$serializer->decode($regionJson,"json");
    // $serializer->denormalize($regionTab, 'App\Entity\Region') => Region(Tab) to Object
    // $serializer->denormalize($regionTab, 'App\Entity\Region[]') => Tab Region(tab) to Tab Object
    // $regionObject=$serializer->denormalize($regionTab, 'App\Entity\Region[]');
    // $entityManager = $this->getDoctrine()->getManager();

    //Methode 2:
    $regionObject = $serializer->deserialize($regionJson, 'App\Entity\Region[]','json');

    /**foreach($regionObject as $region){
        $entityManager->persist($region);
    }
    $entityManager->flush();
    */

    return new JsonResponse("succes",Response::HTTP_CREATED,[],true);
}
```

## Application 2: Récupérer les régions de la Base de Donnée et les afficher

```
/**  
 * @Route("/api/regions", name="api_all_region",methods={"GET"})  
 */  
public function showRegion(SerializerInterface $serializer,RegionRepository $repo)  
{  
    $regionsObject=$repo->findAll();  
    $regionsJson =$serializer->serialize($regionsObject,"json");  
    return new JsonResponse($regionsJson,Response::HTTP_OK,[],true);  
}
```



# Application 3:Ajouter une Région et validation des Données à partir de Postman

## 1) Composant Pour la Validation

**composer require symfony/validator**

## 2) Code

```
/**
 * @Route("/api/regions", name="api_add_region",methods={"POST"})
 */
public function addRegion(Request $request,ValidatorInterface $validator,SerializerInterface $serializer)
{
    $region = $serializer->deserialize($request->getContent(), Region::class,'json');
    $errors = $validator->validate($region);
    if (count($errors) > 0) {
        $errorsString =$serializer->serialize($errors,"json");
        return new JsonResponse( $errorsString ,Response::HTTP_BAD_REQUEST,[],true);
    }
    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->persist($region);
    $entityManager->flush();
    return new JsonResponse("succes",Response::HTTP_CREATED,[],true);
}
```

# Validation dans l'entité Région

## Regle de Validations

```
use Symfony\Component\Validator\Constraints as Assert;
```

### a) Le Code est obligatoire et unique

```
@Assert\NotBlank(message="Le Code est obligatoire")  
  
* @UniqueEntity(  
*     fields={"code"},  
*     message="Le code doit être unique"  
* )
```

### b) Le Nom est obligatoire

```
@Assert\NotBlank(message="Le Nom est obligatoire")
```

# Creation des Entités

- 1) Création d'un Département
- 2) Création d'une Commune

## Insérer des Données par Fixtures

### 1) Composants:

a) `composer require --dev orm-fixtures`

b) `composer require fzaninotto/faker`

Documentation sur les fixtures:

<https://github.com/fzaninotto/Faker>

### 2) Executer les fixtures

`php bin/console doctrine:fixtures:load --append`

## Code Fixtures:

### Constructeur

```
private $repo;  
  
public function __construct (RegionRepository $repo) {  
    $this->repo=$repo;  
  
}
```

## Code Fixtures: Ajouter les données de Départements et de Commune

```
public function load(ObjectManager $manager)
{
    $regions=$this->repo->findAll();
    $faker = Factory::create('fr_FR');
    //Insertion des Regions
    foreach($regions as $region){

        $departement=new Departement();
        $departement->setCode($faker->postcode)
            ->setNom($faker->city)
            ->setRegion($region);
        $manager->persist( $departement);
        //Pour chaque Département, on insère 10 Communes
        for ($i=0; $i <10 ; $i++) {
            $commune=new Commune();
            $commune->setCode($faker->postcode)
                ->setNom($faker->city)
                ->setDepartement($departement);
            $manager->persist($commune);
        }

    }
    $manager->flush();
}
```

Gestion Référence circulaire

Groupe de Sérialisation :indique les attributs à sérialisés

Composant

**composer require sensio/framework-extra-bundle**

## Référence circulaire : Dans le Controller

```
/**
 * @Route("/api/regions", name="api_all_region",methods={"GET"})
 */
public function showRegion(SerializerInterface $serializer,RegionRepository $repo)
{
    $regionsObject=$repo->find(1);
    $regionsJson =$serializer->serialize(
        $regionsObject,
        "json",
        [
            "groups"=>["listeRegionFull"]
        ]
    );

    return new JsonResponse($regionsJson,Response::HTTP_OK,[],true);
}
```



## Référence circulaire: Dans l'entité

```
class Region
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"listeRegionSimple","listeRegionFull"})
     */
    private $id;

    /**
     * @ORM\OneToMany(targetEntity=Departement::class, mappedBy="region")
     * @Groups({"listeRegionFull"})
     */
}
```

## Gestion des Objets Imbriqués:Ajout d'un Département

```
//Entité Departement
/**
 * @ORM\Entity(repositoryClass=DepartementRepository::class)
 */
class Departement
{
    /**
     * @ORM\OneToMany(targetEntity=Commune::class, mappedBy="departement",cascade={"persist"})
     * @Groups({"listeRegionFull"})
     */
    private $communes;
}
```

# Gestion des Objets Imbriqués:Ajout d'un Département

## //Controllers

```
public function addDepartement(Request $request,ValidatorInterface $validator,
                               SerializerInterface $serializer,RegionRepository $repo)
{
    //Recuperation du Contenu Json
    $departementJson = $request->getContent();
    //Transformation du contenu en Tableau
    $departementTable=$serializer->decode($departementJson,"json" );
    //Recuperation de l'objet Region
    $region =$repo->find((int)$departementTable["region"]["id"]);
    $departementsObject=$serializer->deserialize($request->getContent(), Departement::class,'json');
    $departementsObject->setRegion($region);
    $errors = $validator->validate($departementsObject);
    if (count($errors) > 0) {
        $errorsString =$serializer->serialize($errors,"json");
        return new JsonResponse( $errorsString ,Response::HTTP_BAD_REQUEST,[],true);
    }

    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->persist($departementsObject);
    $entityManager->flush();
    return new JsonResponse("succes",Response::HTTP_CREATED,[],true);
}
```

# Section 3 - Api Platform

- 1) **Mise en place du Json Web Token (JWT)**
- 2) **Paramétrage du security.yaml**
- 3) **Installation et Configuration**
- 4) **Gestion des annotations de base d'apiPlatform**
- 5) **CollectionOperations et ItemOperations**
- 6) **Normalization et Dénormalisation**
- 7) **Autorisation sur une Entité**
- 8) **Controllers Personnalisés**

# Mise en place du Json Web Token (JWT)

## Introduction

L'authentification est une problématique récurrente du développement d'applications web et mobile. Dans un monde où les API sont consommées par toutes sortes de clients, il est important de pouvoir offrir une solution commune performante et sécurisée. Pour remplir le rôle de l'authentification, les cookies sont utilisés et retournés par le serveur au client suite à une authentification réussie et accompagnent chaque requête du client afin de l'identifier. Ce pendant une API REST se doit d'être stateless c'est à dire pas de sessions coté serveur.

# Mise en place du Json Web Token (JWT)

## 1)Notion de JWT JWT

(Json Web Token) est un standard ouvert (RFC 7519) qui définit une manière compacte et autonome de transmission sécurisée d'informations entre les parties sous la forme d'un objet JSON.En d'autres termes, un jeton JWT est une chaîne de caractères que l'on va envoyer à chaque requête que l'on souhaite effectuer auprès d'une API afin de s'authentifier. Il contient toutes les informations nécessaires à notre identification.

Un JWT est constitué de trois parties séparées par un point : part1.part2.part3 .



## Structure d'un JSON Web Token

### → **Header : Entête(encodé en base64)**

L'en-tête se compose généralement d'un array JSON de deux membres :

- **Typ** : le type du jeton, qui est donc JWT
- **Alg** : l'algorithme de hachage utilisé, comme HMAC SHA256 ou RSA.

Exemple :

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

### → **Payload : Contenu (encodé en base64, JWT claims )**

Le contenu est un simple objet avec les informations à échanger entre le client et le serveur (expiration du token, identité de l'utilisateur connecté, ...).

On peut y placer librement des champs personnalisés et propres à nos besoins (public claims) et des attributs réservés définis dans la spécification (registered claims) .

Quelques "registered claims" :

- iss : Origine du token (issuer),
- sub : Sujet (subject),
- exp : Date d'expiration du token (expiration),
- iat : Date de création du token (issued at),

Exemple :

```
{ "iss": "sa.com", "name": "John Doe", "admin": true }
```

```
⇒ eyAiaXNzljogImVraW5vLmNvbSIsICJuYW1lIjogIkpvaG4gRG9lIiwgImFkbWluljogdHJ1ZSB9
```

## ✓ **Signature**

Pour générer la signature, on concatène le header et le contenu puis on encode avec l'algorithme défini dans le header.

Exemple :

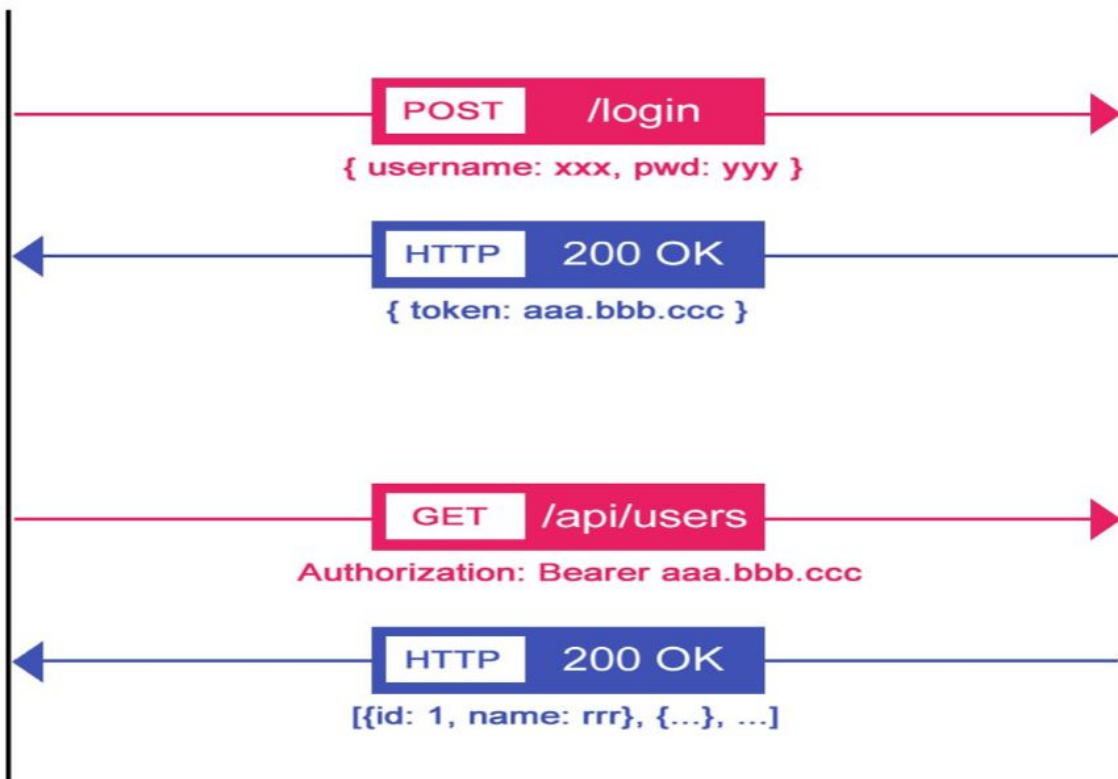
HMACSHA256(

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 + "." +

eyJAiaXNzljogImVraW5vLmNvbSIsICJuYW1lIjogIkpvaG4gRG9lIiwgImFkbWluljogdHJ1ZSB9)







# Mise en place du Json Web Token (JWT)

Installer le bundle : `composer req lexik/jwt-authentication-bundle`

Cette installation crée :

- crée le fichier `lexik_jwt_authentication.yaml` dans le dossier `package`
- modifie le fichier `.env` (il y ajoute 3 lignes)

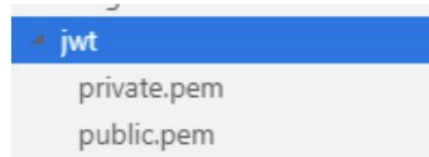
Il faut ensuite créer un dossier `jwt` dans le dossier `config`

Ensuite on va générer les deux clés (private et publique) en ligne de commande :

- `openssl genrsa -out config/jwt/private.pem -aes256 4096`
- `openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem`

Ces deux lignes de commande nécessitent d'entrée une phrase (mot de passe) qu'il faudra écrire en dur dans le fichier `.env` dans la rubrique `JWT_PASSPHRASE`

Après ceci le dossier `jwt` contiendra 2 fichiers correspondants aux deux clés



Vous pouvez également ajouter un temps de vie pour votre token en ajoutant

l'attribut `token_ttl` dans le fichier `lexik_jwt_authentication.yaml`

```
lexik_jwt_authentication:
  secret_key: '%env(resolve:JWT_SECRET_KEY)%'
  public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
  pass_phrase: '%env(JWT_PASSPHRASE)%'
  token_ttl: 3600
```

# Mise en place du Json Web Token (JWT)

## Installer le composant security

`composer require security`

## Creation de l'entity User

- `php bin/console make:user`
- `php bin/console make:migration`
- `php bin/console doctrine:migrations:migrate`

## Creation de l'entity Profil

```
private $repo;

private $encoder;

public function __construct (RegionRepository $repo, UserPasswordEncoderInterface $encoder ) {
    $this->repo=$repo;
    $this->encoder=$encoder;
}
```

# Mise en place du Json Web Token (JWT)

## Gestion des Fixtures Utilisateurs et Profils

```
$profils=["ADMIN", "FORMATEUR", "APPRENANT", "CM"];

foreach ($profils as $key => $libelle) {

    $profil =new Profil();
    $profil->setLibelle($libelle);
    $manager->persist($profil);
    $manager->flush();

    for ($i=1; $i <=3 ; $i++) {
        $user = new User();
        $user->setProfil($profil);
        $user->setLogin(strtolower($libelle).$i);
        //Génération des Users
        $password = $this->encoder->encodePassword($user, 'pass_1234');
        $user->setPassword($password);

        $manager->persist($user);

    }
    $manager->flush();
}
```

## Mise en place du Json Web Token (JWT)

### Installation du bundle

**composer require lexik/jwt-authentication-bundle**

### Générer les clés(public et privée)

**mkdir -p config/jwt**

**openssl genpkey -out config/jwt/private.pem -aes256 -algorithm rsa -pkeyopt rsa\_keygen\_bits:4096**

**openssl pkey -in config/jwt/private.pem -out config/jwt/public.pem -pubout**

# Mise en place du Json Web Token (JWT)

## Configuration fichier lexik\_jwt\_authentication.yaml et .env

Configuration fichier lexik\_jwt\_authentication.yaml et .env

```
config > packages > ! lexik_jwt_authentication.yaml
1  lexik_jwt_authentication:
2      secret_key: '%env(resolve:JWT_SECRET_KEY)%'
3      public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
4      pass_phrase: '%env(JWT_PASSPHRASE)%'
5      token_ttl: 3600
6
```

lexik\_jwt\_authentication.yaml

```
###> lexik/jwt-authentication-bundle ###
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=test
###< lexik/jwt-authentication-bundle ###
```

Fichier .env

Configuration de la route

/config/routes.yaml

```
api_login_check:
    path: /api/login_check
```

# Paramétrage du security.yaml

```
security:
# https://symfony.com/doc/current/security.html#where-do-us
encoders:
  App\Entity\Adherent:
    algorithm: bcrypt
providers:
  in_database:
    entity:
      class: App\Entity\Adherent
      property: mail
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  api:
    pattern: ^/apiPlatform
    stateless: true
    anonymous: true
    json_login:
      username_path: mail
      check_path: /apiPlatform/login_check
      success_handler: lexik_jwt_authentication.handler.authentication_success
      failure_handler: lexik_jwt_authentication.handler.authentication_failure
    guard:
      authenticators:
        - lexik_jwt_authentication.jwt_token_authenticator
access_control:
  - { path: ^/apiPlatform/login_check, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/apiPlatform$, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/apiPlatform, roles: IS_AUTHENTICATED_FULLY }
role_hierarchy:
  ROLE_MANAGER: ROLE_ADHERENT
  ROLE_ADMIN: ROLE_MANAGER
```

On définit l'algorithme de cryptage en précisant sur quelle entity et quelle propriété de l'entity il s'applique.

On crée un firewall qui interceptera les routes commençant par /apiPlatform (en accord avec le fichier `api_platform.yaml`)

On définit cet ensemble de règle pour la gestion du token : route pour se connecter, classes qui vont gérer le token, l'identifiant permettant d'authentifier l'utilisateur (ici le mail) ...

WariCustomAthenificatorAuthenticator  
todo: check the credentials inside F:\w

On paramètre ici les accès dans cet ordre afin de permettre la connexion à un utilisateur non authentifié (la seconde route permet d'accéder à la documentation de l'api)

On définit éventuellement des rôles et leur hiérarchie

On ajoute enfin un fichier `jwt.yaml` dans le dossier `route` pour déclarer la route `login_check`

```
apiPlatform_login_check:
  path: /apiPlatform/login_check
```






# Api Platform

## 1) Installation : **composer require api**

Installer le bundle en tapant : **composer req api**

3 fichiers sont créés :

`api_platform.yaml` dans le dossier `config`  
`api_platform.yaml` dans le dossier `routes`  
`nelmio_cors.yaml` dans le dossier `packages`



```
api_platform:
  resource: .
  type: api_platform
  prefix: /apiPlatform
```

The diagram shows a list of three files created during installation. An arrow points from the first file, `api_platform.yaml` in the `config` directory, to a code block showing the configuration for `api_platform`. Another arrow points to the `prefix` value in the code block.

On peut modifier ici le préfix qui sera le début de toutes les routes

## 2) Configuration des entités

Ajouter l'annotation `@ApiResponse()` sur chaque entité gérée par ApiPlatform

Ajouter l'annotation `@ApiSubresource` sur un attribut d'une entité

**Exemple :** <http://localhost:8000/api/profils/21/users>

```
{
  "@id": "/api/users/1",
  "@type": "User",
  "id": 1,
  "login": "admin1",
  "username": "admin1",
  "roles": [
    "ROLE_ADMIN"
  ],
  "password": "$argon2id$v=19$m=65536,t=4,p=1$wn7pQewZNz7jKE+8BeOTEA$dFaYkIYP6ATDnyGgZ0Bs8ABv7udn+4m5i7QsTTQmrtc",
  "profil": "/api/admin/profils?id=21"
},
```

### 3) Notion Api auto decouvrable

#### Exemple:

**Request:** `http://localhost:8000/api/profils`

#### Response:

```
{
  "@id": "/api/profils?id=21",
  "@type": "Profil",
  "id": 21,
  "libelle": "ADMIN",
  "users": [
    "/api/users/1",
    "/api/users/2",
    "/api/users/3"
  ]
}
```

# CollectionOperations et ItemOperations

## 1) CollectionOperations

### Exemple 1

```
* collectionOperations={  
*     "get","post"  
* },
```

## Exemple 2:

```
collectionOperations={  
*      "get_role_admin"={  
*          "method"="GET",  
*          "path"="/admin/profils" ,  
*      }  
* }
```

# ItemOperations

## Exemple 1:

```
* itemOperations={  
*     "get","put"  
* }
```

# ItemOperations

## Exemple 2:

```
itemOperations={  
*      "get_role_admin"={  
*          "method"="GET",  
*          "path"="/admin/profils/{id}" ,  
*      }  
* }
```

# Normalization et Dénormalisation

**Normalization :Lorsqu' on récupère des données : GET**

**1) Applicable à toute les routes sur l'entité**

**//Entity**

```
* @ApiResponse(  
*     normalizationContext={"groups":{"profil:read","profil:read_all"}}  
* )
```

**//Attribut**

```
@Groups({"profil:read_all"})
```



# Normalization et Dénormalisation

## 2) Applicable à une route de l'entité

```
*  collectionOperations={  
  
*    "get_simple"={  
*      "method"="GET",  
*      "path"="/admin/profils" ,  
*      "normalization_context"={"groups":"profil:read"}  
*    },  
*    "get_all"={  
*      "method"="GET",  
*      "path"="/admin/profils/all" ,  
*      "normalization_context"={"groups":"profil:read_all"}  
*    }  
*  }
```

# Dénormalisation

Lorsqu'on envoie des données au serveur : POST, PUT

//Entity

```
* @ApiResponse(  
*     denormalizationContext={"groups":{"profil:write"}}  
* )
```

//Attribut

```
@Groups ( {"profil:write" })
```

# Autorisation sur un Entité

## 1) Toutes L'entité

```
* @ApiResource(  
*   attributes={  
*       "security"="is_granted('ROLE_ADMIN')",  
*       "security_message"="Vous n'avez pas access à cette Ressource"  
*   }  
*)
```

# Autorisation sur un Entité

## 1) Sur une Route

```
collectionOperations={
*      "get_role_admin"={
*          "method"="GET" ,
*          "path"="/admin/profils" ,
*          "security"="is_granted('ROLE_ADMIN')",
*          "security_message"="Vous n'avez pas access à cette Ressource"
*      }
* }
```

# Controllers Personnalisés

## 1) Controllers

```
/**
 * @Route (
 *     name="apprenant_liste",
 *     path="api/apprenants",
 *     methods={ "GET" },
 *     defaults={
 *         "_controller"="\app\ControllerApprenantController::getApprenant",
 *         "_api_resource_class"=User::class,
 *         "_api_collection_operation_name"="get_apprenants"
 *     }
 * )
 */
```

## 1) Controllers

```
public function getApprenant (UserRepository $repo)
{
    $apprenants= $repo->findByProfil ("APPRENANT");
    return $this->json ($apprenants,Response::HTTP_OK,);
}
```

## Controllers Personnalisés

### 2) Définir la méthode dans le Repository

//Récupération des Utilisateurs avec le profil Apprenant

```
public function findByProfil($value)
{
    return $this->createQueryBuilder('u')
        ->innerJoin('u.profil', 'p')
        ->andWhere('p.libelle = :val')
        ->setParameter('val', $value)
        ->orderBy('u.id', 'ASC')
        ->getQuery()
        ->getResult()
    ;
}
```

## Controllers Personnalisés

### 3)Rendre visible la route sur Api PlatForm

//Entity

```
* @ApiResponse(  
*     collectionOperations={  
*         "get_apprenants"={  
*             "method"="GET",  
*             "path"="/apprenants" ,  
*             "normalization_context"={"groups":"apprenant:read"},  
*             "access_control"="(is_granted('ROLE_ADMIN') or is_granted('ROLE_FORMATEUR'))",  
*             "access_control_message"="Vous n'avez pas access à cette Ressource",  
*             "route_name"="apprenant_liste",  
*         }  
*     })
```