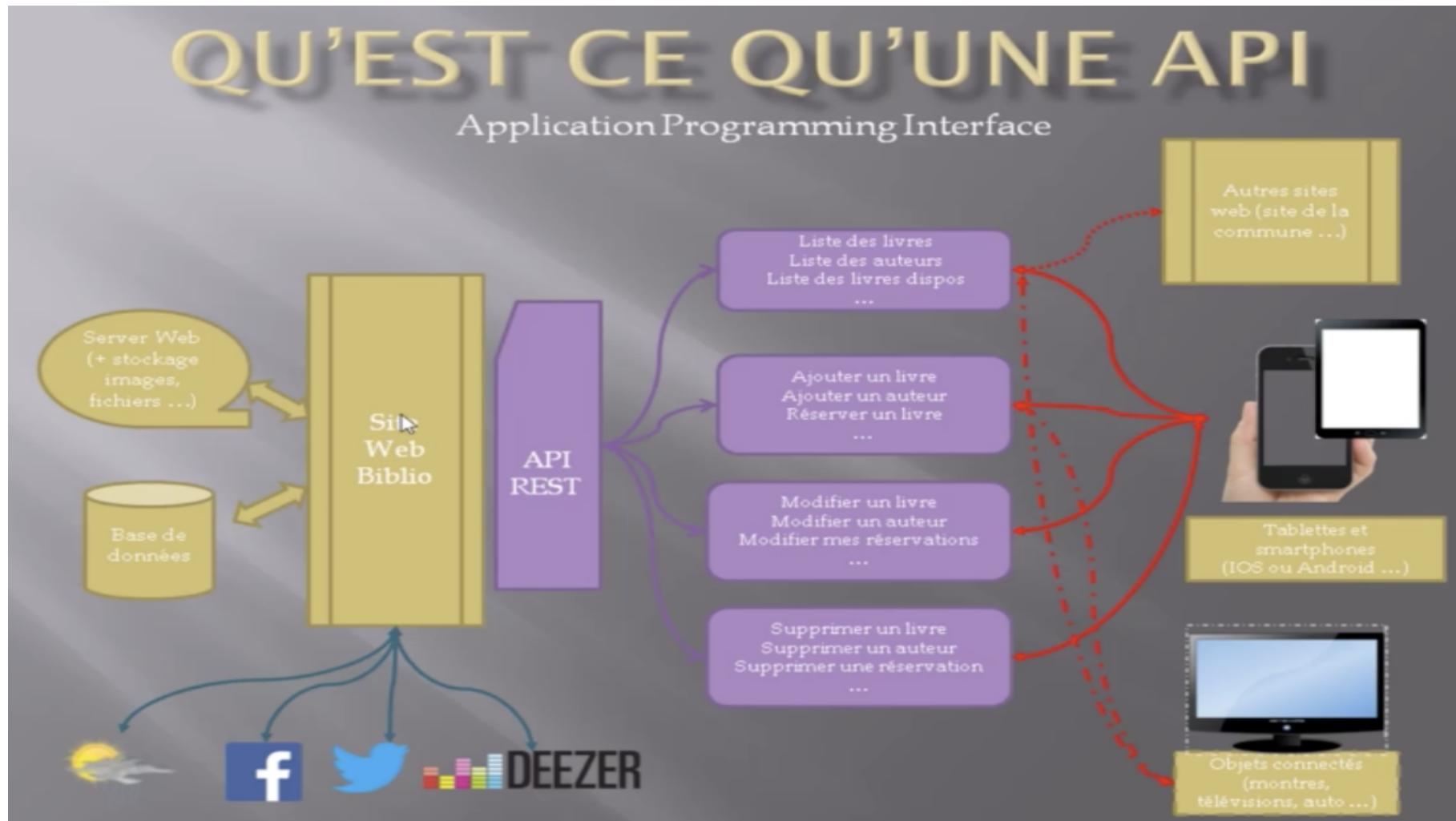


Notion Api et Api Rest

a. Qu'est ce qu'une API



b. Intérêts

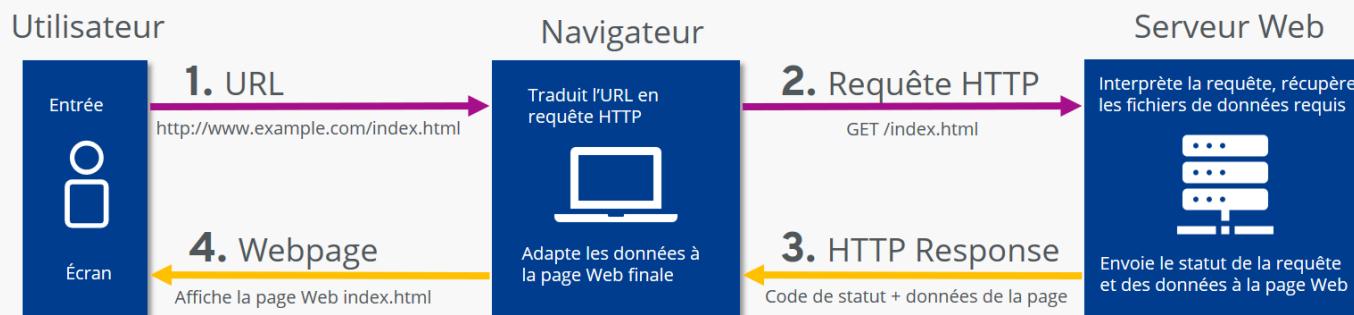
- i. Besoins de multiplier les points de consommations**
 - 1. Site web,
 - 2. Smartphone,
 - 3. Objets connectés
- ii. Besoin de centraliser la couche métier**
 - 1. accès à la base de données,
 - 2. Stockage fichiers,
 - 3. accès par authentification

c. Exemples D'API

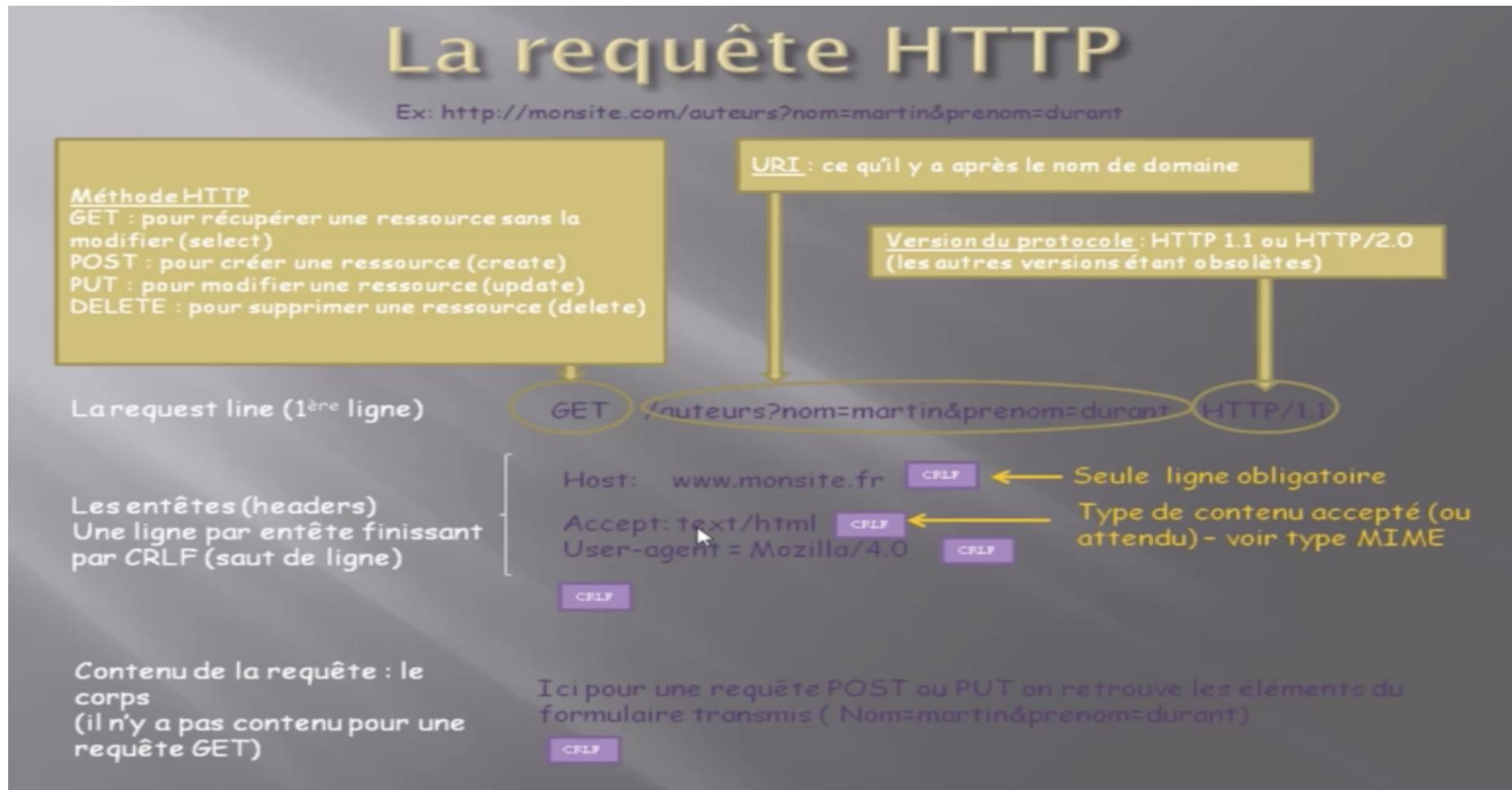
d. Architecture Rest basé sur le protocole HTTP

i. Présentation

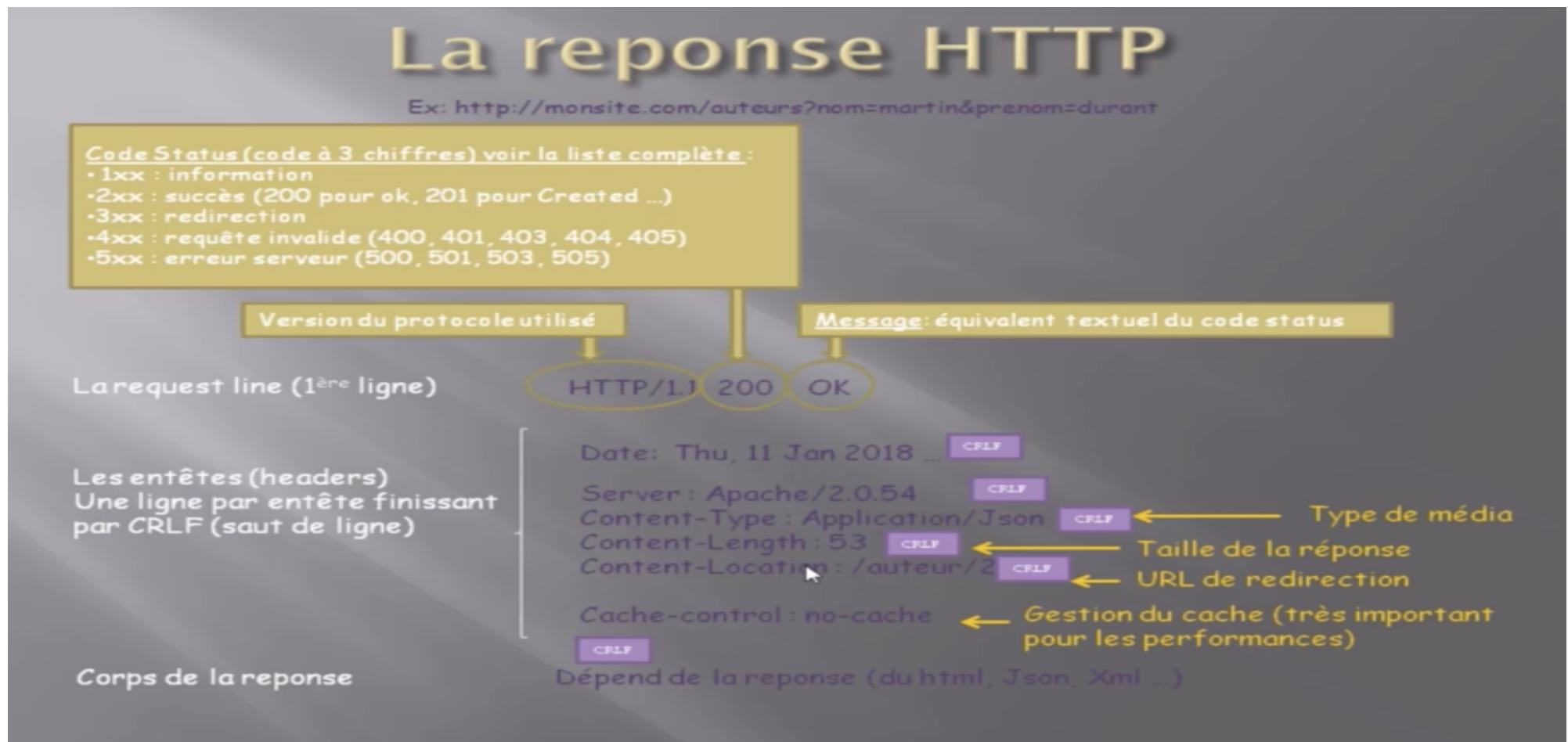
Processus de communication HTTP



ii. La Requête HTTP



iii. La Response HTTP



iv. Code de Response HTTP

La reponse HTTP

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

Code S

- 1xx : i
- 2xx : s
- 3xx : r
- 4xx : r
- 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 renvoyée 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 renvoyée 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.

e. Notion Architecture Rest : Contrainte Architecture de API REST

Architecture REST

Les 6 contraintes :

Representational State Transfert

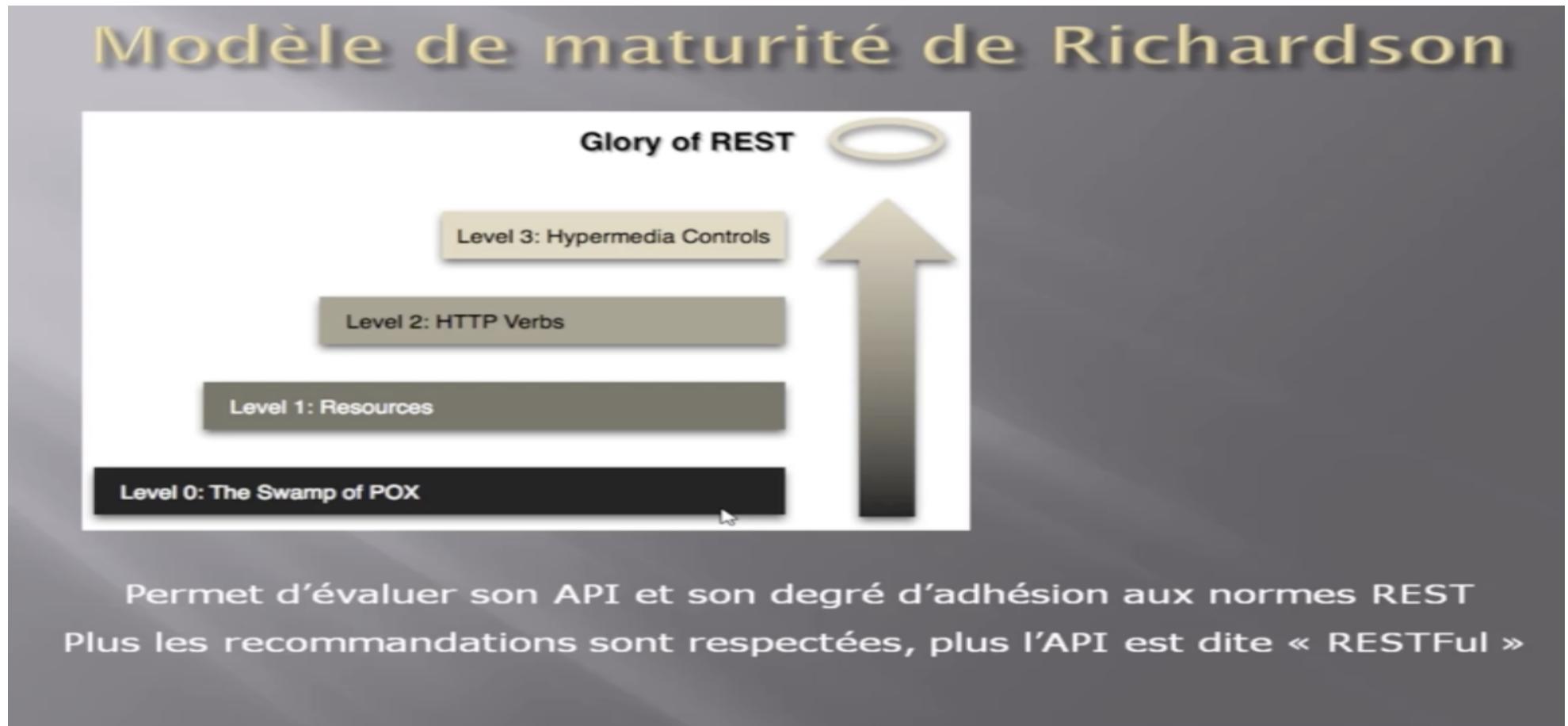
1. **Architecture client/serveur** : c'est le cas du protocole HTTP
2. **Stateless (sans état)** : on ne conserve pas de contexte entre les requêtes donc pas de variable de session par exemple. L'utilisateur doit donc être authentifié à chaque requête !
3. **Cacheable** : on doit pouvoir mettre en cache la ressource pour s'en resservir pour des requêtes similaires consécutives.
4. **Layered 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 Méthode -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-décrise : 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"  
}
```

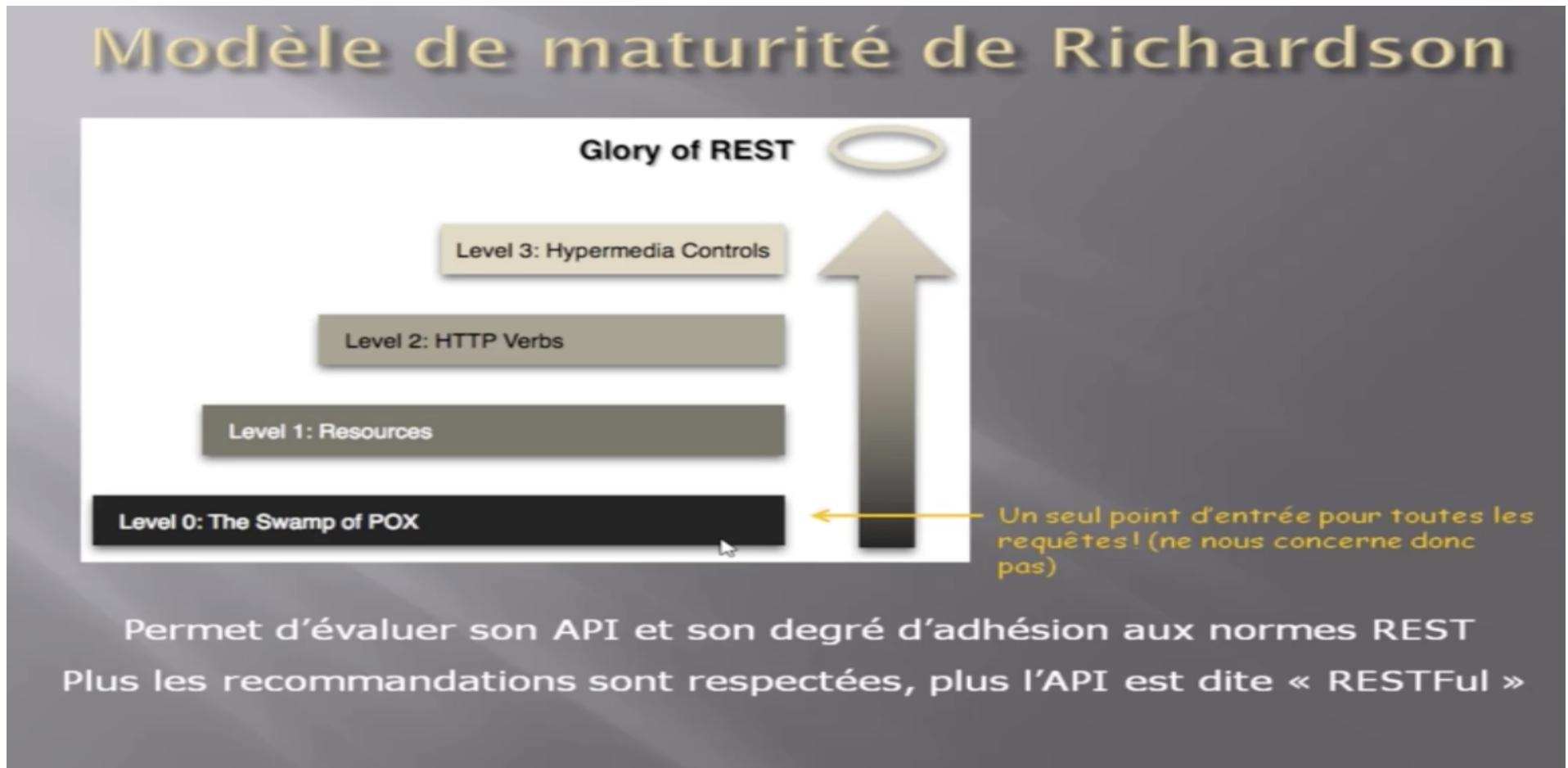
Notion Api

RestFul

i. Modèle de Maturité de Richardson



1. Level 0



Ressource Burger

End Point GET localhost:8000/api/burgers => lister => Request(Header)

[

{

id:1

"nom": "Menu Greedy",

"prix":1500

"lien": [

GET localhost:8000/api/burgers/1

PUT localhost:8000/api/burgers/1

]

,

{

id:2

"nom": "Menu Greedy",

"prix":1500

}

]

POST localhost:8000/api/burgers => ajouter Request(Header + body)

PUT localhost:8000/api/burgers/1 => ajouter Request(Header + body)

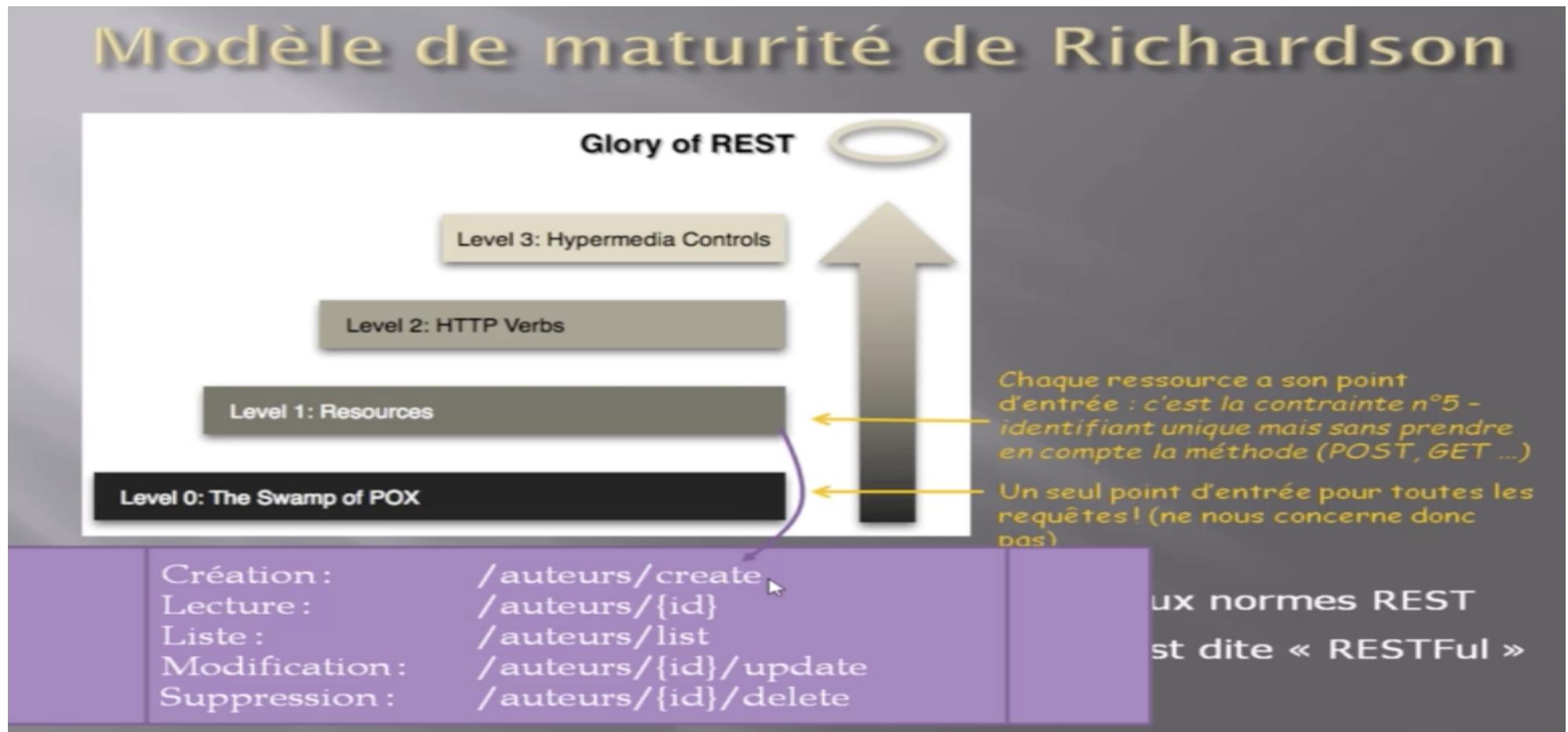
DELETE localhost:8000/api/burgers/1 => ajouter Request(Header)

Ressource User

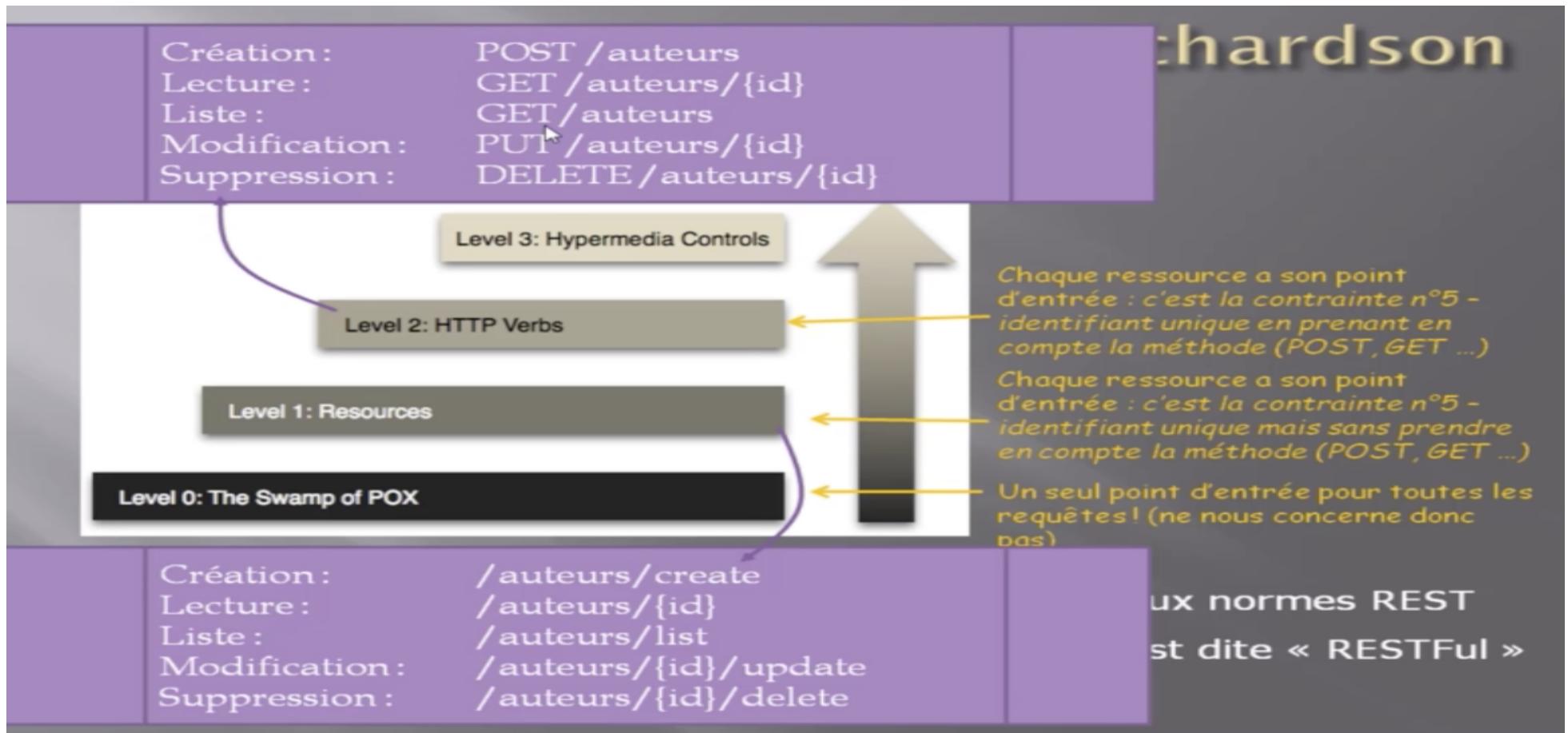
GET :localhost:8000/api/users => lister

POST :localhost:8000/api/users => ajouter

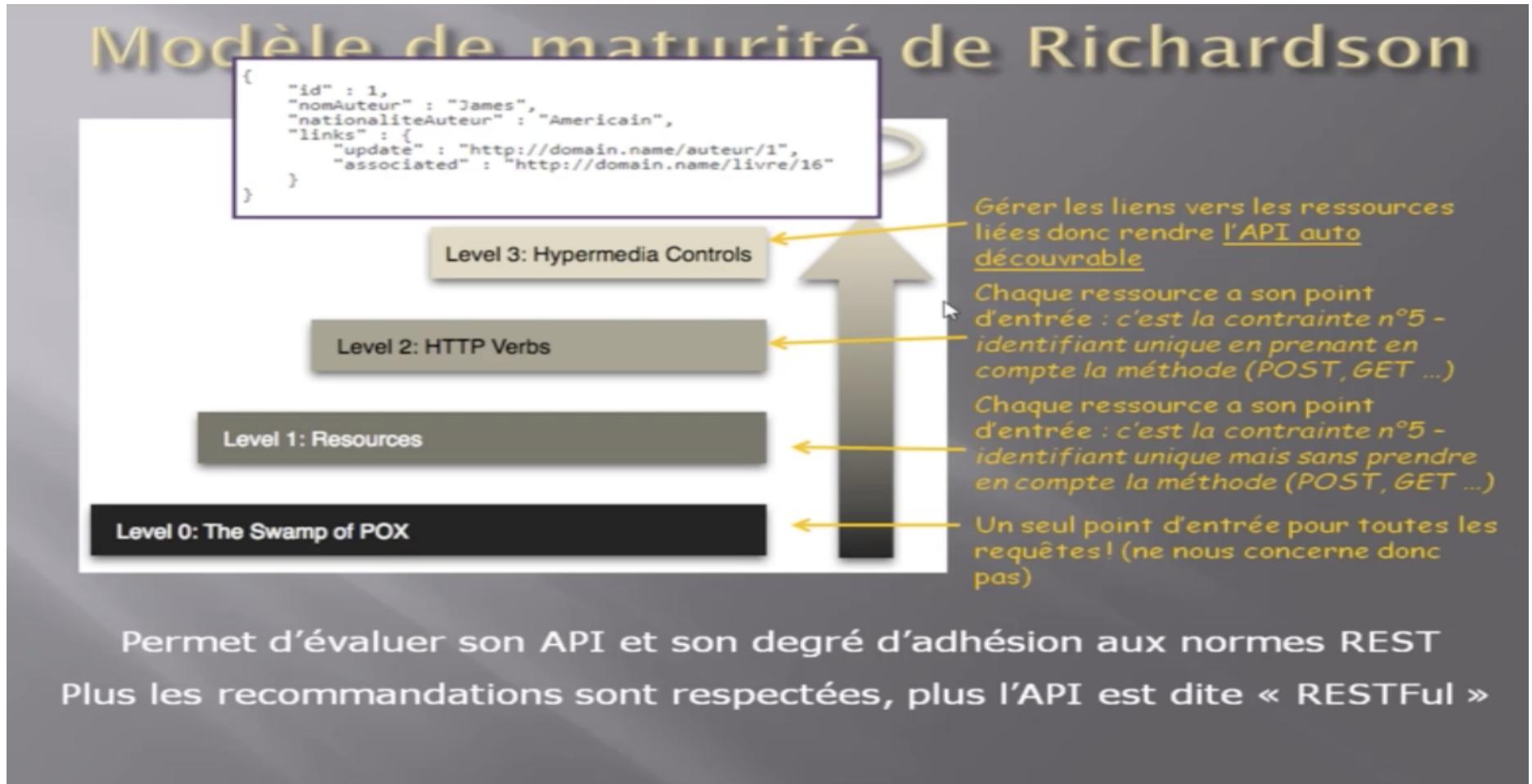
2. Level 1



3. Level 2



4. Level 3



Génération du JWT Token

I. Création du Projet et Installation des dépendances

A. Creer un projet Symfony Api

```
composer create-project symfony/skeleton my_back
```

2. Installation des Dépendances

- a) composer require symfony/maker-bundle --dev
- b) composer require doctrine/annotations
- c) Composer require symfony/orm-pack

3. Configurer le fichier .env

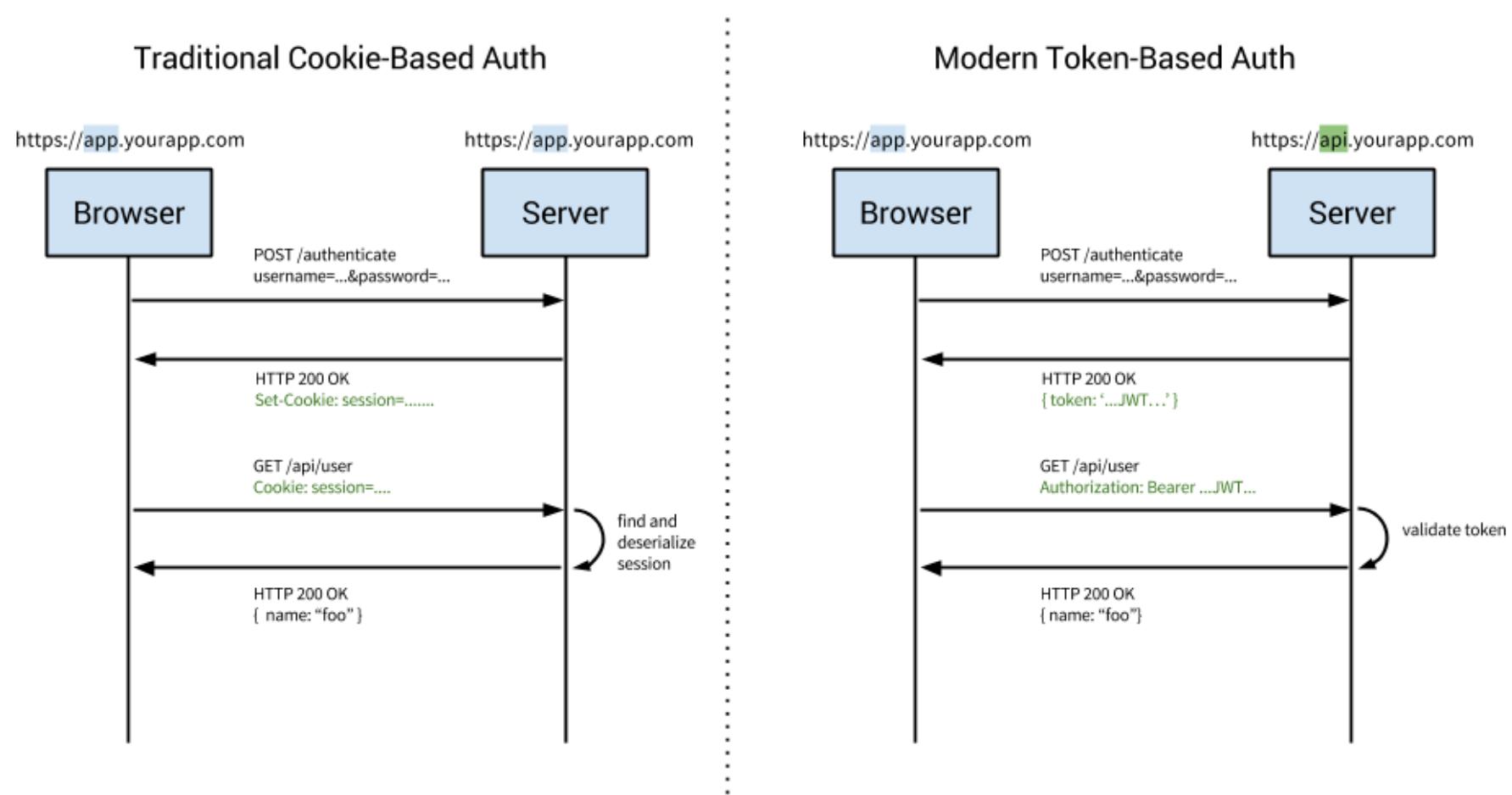
```
DATABASE_URL=mysql://root:root@127.0.0.1:8889/my_back
```

4. Crédation de la Base donnée

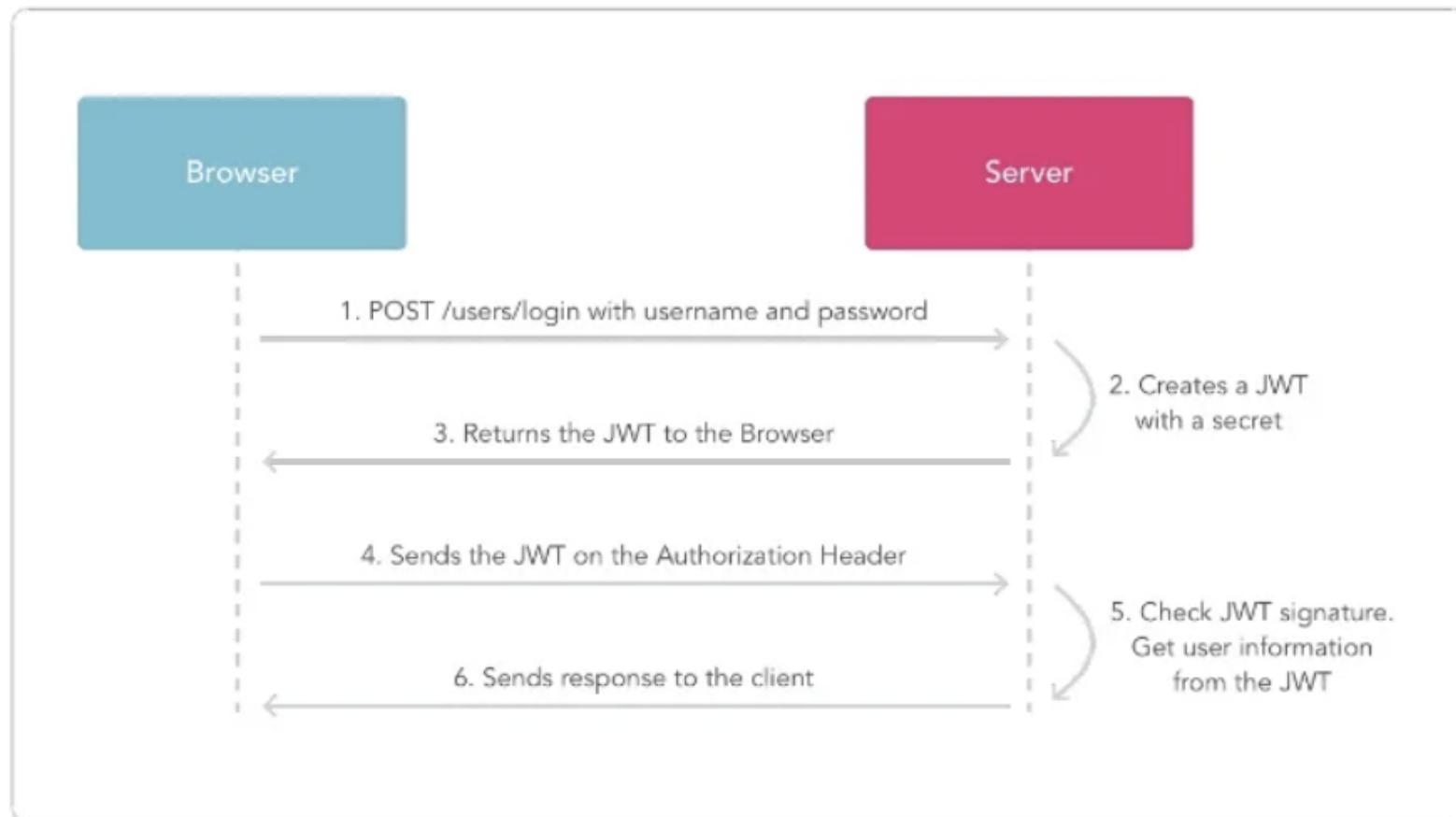
```
php bin/console doctrine:database:create
```

II. Authentification

A. Authentication StateFull vs StateLess



B. Séquence de Connexion



C. Notion de JWT Token

D. Étapes de Génération du Token

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`

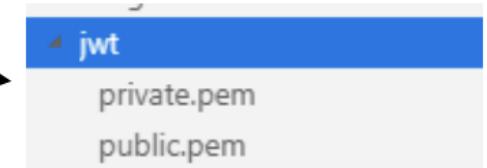
Ensuite on va générer les deux clés (private et publique) en ligne de commande :

- `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
```

E. Installation de la Sécurité

composer require security

F. Création de l'entity User

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

G. Installation des dependance du token

composer require lexik/jwt-authentication-bundle

H. 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**

```
[  
‘secutity’=>[  
‘password_hashers’=>‘Symfony\Component’,  
]  
]
```


I. 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-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`

`lexik_jwt_authentication.yaml`

Configuration de la route `/config/routes.yaml`

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

J. Paramétrage security.yaml

```
security:
    # https://symfony.com/doc/current/security.html#where-do-users-log-in
    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) ...

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`

WariCustomAuthenticatorAuthenticator
todo: check the credentials inside F:\w...

apiPlatform_login_check:
path: /apiPlatform/login_check

```
api:
    pattern:  ^/api
    stateless: true
    json_login:
        username_path: login
        check_path: /api/login_check
        success_handler: lexik_jwt_authentication.handler.authentication_success
        failure_handler: lexik_jwt_authentication.handler.authentication_failure
```

```
access_control:
    - { path: ^/api/login_check, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/api$,roles: IS_AUTHENTICATED_FULLY }
```

K. Création des Fixtures

1. Installation des Dépendances

- composer require --dev orm-fixtures
- composer require symfony/password-hasher

2. Définition des Fixtures

```
public function load(ObjectManager $manager): void
{
    $user=new User();
    $user->setLogin('client@gmail.com');
    $hashedPassword = $this->passwordHasher->hashPassword(
        $user,
        'passer'
    );
    $user->setPassword($hashedPassword);
    $user->setRoles(['ROLE_CLIENT']);
```

```

$user1=new User();
$user1->setLogin('gestionnaire@gmail.com');
$hashedPassword = $this->passwordHasher->hashPassword(
    $user1,
    'passer'
);
$user1->setPassword($hashedPassword);
$user1->setRoles(['ROLE_GESTIONNAIRE']);
$manager->persist($user);
$manager->persist($user1);
$manager->flush();
}

```

L. Request de Connexion

1. Request

a) path : Post http://localhost:8000/api/login_check

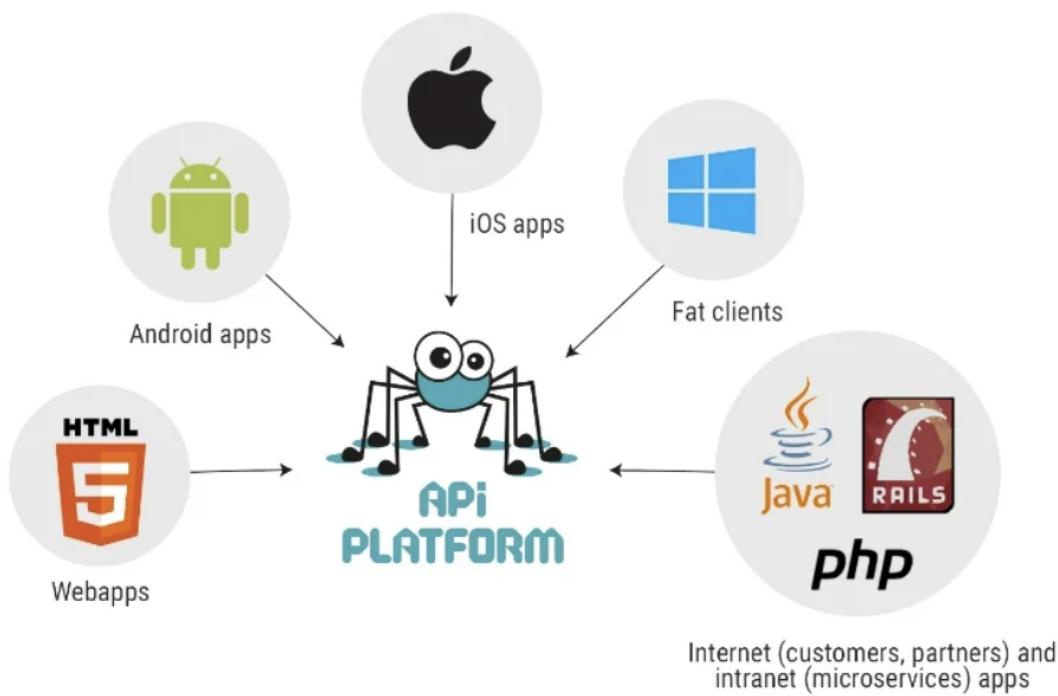
b) body :

```
{
    "login":"client@gmail.com",
    "password":"passer"
}
```

2. Response

```
{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9eyJpYXQiOjE2NTU3ODIzMjQsImV4cCI6MTY1NTc4NTk2NCwicm9sZXMiOlsiUk9MRV9VU0VSII0sInVzZXJuYW1ljoIY2xpZW50QGdtYWlsLmNvbSJ9.NxBn-PdnQ9L3Ys3_miWJCIAzLsyMNj_4MVi_DbJ8xpKVfworMlyPw-TRNtT9GNALaHhmhBagNd_ggPK-9zNAxof2V-4EgSIGzQHNB0ccFoa7k7UFu7o1BTm73FbUtw-hjBmr8C-AoEAYQqhlyLSiOceRRj7Go6Hf9zeJ68F7OWQ1cfGKg3d_v4EjaF7Rvm6AbiaQJqTS3YUbhGiir6H93c5KBjYcOhyOhriKdCrQjizloKQ4LV-Mb7kxA9D0XLjETyN84uiqUUrY4U_Uf5h23mvysO2wpwjDSln5N9YmhgRaF07ZAwdoPeQCRv003QfnqNi-Bx-KUvst6ay8FpCkFmXgPq5-MRkXuEOfcXOvzUBk3OKMuEOKa9Di2YMqQeZA9Q-OsUWox6DuCJhNCguojBDTwJPAnwIR3qBlvg2PSgdsODEbPVMJsS3eR3j9yjVOc9-Nsqj_Ewk_rHNuwXfuUJAjlTebK2GDunT8zCX2zQgIY-nb6EVse-qIBMmOonZzICU7vqneqx_7n4BU1XIEoonsGGNjMYuPaXhqBEPxBfg_sz52sRWOeCHbeH98cGQv70QldbOS1BxJRFehL4hcwvz9uXgU4JqdAUAv4ksKQS7MHPySpg3tbaLvQMLa6OmPkI6XLxWdMs7QxEFxojRW5UvgXSefLdy2B8DUm_o"}
```

Réalisation Api avec Api PlatForm



I. Présentation

API Platform est un framework web utilisé pour générer des API REST et GraphQL, se basant sur le patron de conception MVC. La partie serveur du framework est écrite en PHP et basée sur le framework Symfony.

II. Installation des dépendances

composer require api

III. Documentation swagger

path : Post <http://localhost:8000/api>

IV. Configuration des entités

Ajouter l'annotation **[*ApiResource()*]** sur chaque entité gérée par Api PlatForm

Exemple :

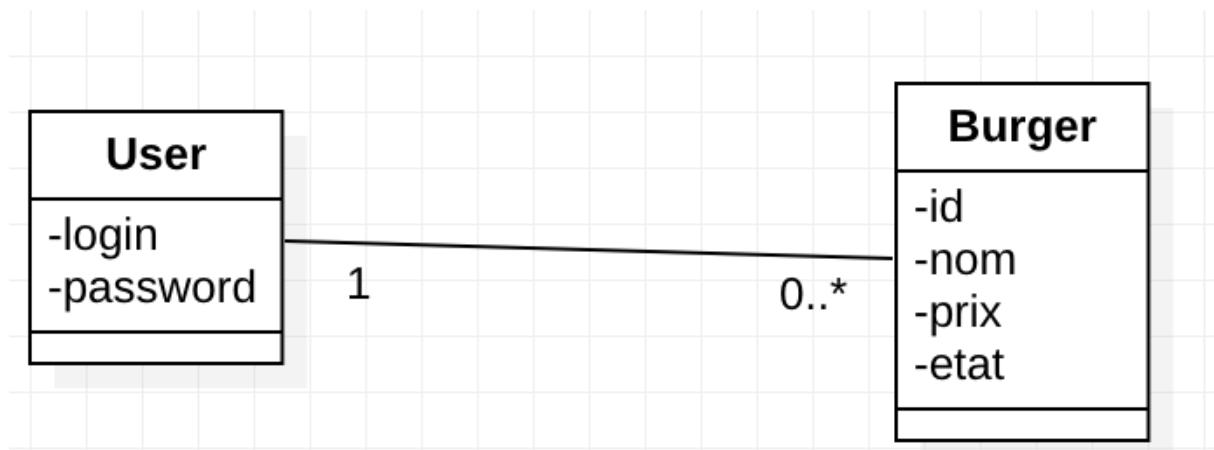
```
use ApiPlatform\Core\Annotation\ApiResource;

#[ApiResource()]

class User implements UserInterface, PasswordAuthenticatedUserInterface{}
```

V. Réalisation des Endpoint des Ressources suivantes

A. Diagramme de Classe



B. Use Case Diagramme

C. Use Case

Use Case	Operation	Verbe	path	Autorisation
Lister les Burger(on affiche pas l'état)	Collection	GET	/burgers	ROLE_USER
Lister un burger(on affiche l'état)	Item	GET	/burgers/id	ROLE_USER
Ajouter Burger	Collection	POST	/burgers	ROLE_GESTIONNAIRE
Modifier Burger	Item	PUT	/burgers/id	ROLE_GESTIONNAIRE
Lister les burger creer par un User	Collection	GET	/users/id/burgers	ROLE_GESTIONNAIRE
Créer un User(DataPersister)	Collection	POST	/register	ROLE_USER
Créer un Burger et lui affecte le User Connecter	Controller Personnalisée	POST	/add	ROLE_GESTIONNAIRE
	EventSubscriber	POST	/burgers	ROLE_GESTIONNAIRE
	• id	SearchFilter		

Filtrer Burger par	<ul style="list-style-type: none">● user● nom				
	<ul style="list-style-type: none">● prix	NumericFilter			

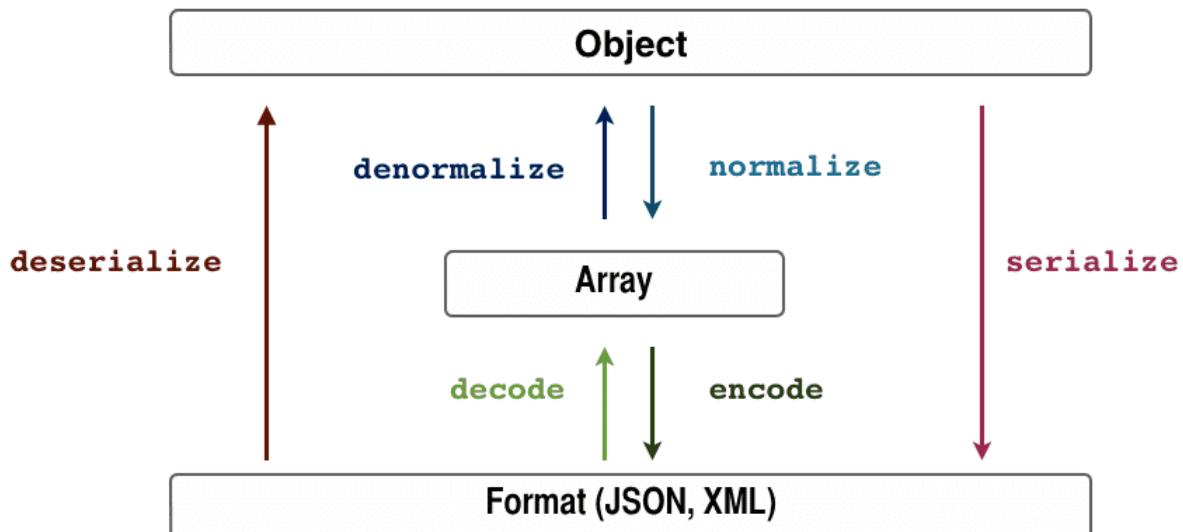
NB : ROLE_USER est ROLE_CLIENT ou ROLE_GESTIONNAIRE

D. Réalisation

1. Création de la Ressource

```
php bin/console make:entity Burger -a
```

2. Conversion de format



3. Fixer les Opération possible sur le burger

Collection operations

METHOD	MANDATORY	DESCRIPTION
GET	yes	Retrieve the (paginated) list of elements
POST	no	Create a new element

Item operations

METHOD	MANDATORY	DESCRIPTION
GET	yes	Retrieve element
PUT	no	Update an element
DELETE	no	Delete an element

Next Slide

```
# [ApiResource(  
    collectionOperations: ["get", "post"]  
    itemOperations: ["put", "get"]  
)
```

4. Use Lister les Burger

a) Toutes les propriétés

```
collectionOperations: [  
    "get" => [  
        'method' => 'get',  
        'status' => Response::HTTP_OK,  
    ]  
,
```

b) Notion de Normalisation

#Sur l'entité

NB: La Normalisation permet de définir le format de la donne de la réponse avec la notion de Groups.

```
"get" => [  
    'method' => 'get',  
    'status' => Response::HTTP_OK,  
    'normalization_context' => ['groups' => ['simple']],  
]
```

#Sur les attributs

```
# [Groups(["simple"])]
```

5. Use Lister un burger

#Sur l'entité

```
itemOperations: [  
    "get" => [  
        'method' => 'get',  
        'path' => "/burgers/{id}",  
        'requirements' => ['id' => '\d+'],  
        'normalization_context' => ['groups' => ['all']],  
    ],  
]
```

#Sur les attributs

```
#[Groups(["simple", "all"])]  
    private $id;  
#[Groups(["all"])]  
private $isEstat;
```

6. Use Case ajouter un burger

a) sans Denormalisation

NB: La Denormalisation permet de définir le format du body de la requête (PUT,POST ou PATCH) avec la notion de Groups.

```
collectionOperations: [
    "post" => [
        'status' => Response::HTTP_CREATED,
    ],
]
```

b) Avec Denormalisation

```
"post" => [
    ,
    'status' => Response::HTTP_CREATED,
    'denormalization_context' => ['groups' => ['write:simple','write:all']],
],
]
```

c) Avec Denormalisation + Normalisation

```
ectionOperations:[
    "post" => [
        // "access_control" => "is_granted('ROLE_GESTIONNAIRE')",
        // "security_message"=>"Vous n'avez pas access à cette Ressource",
        'status' => Response::HTTP_CREATED,
        'denormalization_context' => ['groups' => ['write:simple','write:all']],
        'normalization_context' => ['groups' => ['all']],
        // "path"=>"/bugers"
    ]
]
```

7. Use Case Modifier burger

a) Modifier les Attributs

```
"put"=>[  
    'method' => 'put',  
    // "security" => "is_granted('ROLE_GESTIONNAIRE')",  
    // "security_message"=>"Vous n'avez pas access à cette Ressource",  
    'status' => Response::HTTP_OK,  
]
```

b) Affecter User

- (1) IRI
- (2) Denormalisation
- (3) Denormalisation+Normalisation

8. Use Case Lister les Burger créer par un user

a) Sans Normalisation

#Dans User

NB: Les SubRessource.

```
#[ApiSubresource]
private $burgers;
```

9. Use Case Créer un User

#securite.yaml

```
- { path: ^/api/register$, roles: IS_AUTHENTICATED_FULLY }
```

#User.php

```
collectionOperations:[
    "get",
    "post_register" => [
        "method"=>"post",
        'status' => Response::HTTP_CREATED,
        'path'=>'register/',
        'denormalization_context' => ['groups' => ['user:write']],
        'normalization_context' => ['groups' => ['user:read:simple']]
    ],
]
```

#src/DataPersister/DataPersister.php

NB: Un Data Persister permet de redéfinir les opérations de persist(insert,update) ou de remove(suppression)

```
class UserDataPersister implements DataPersisterInterface
{
    private UserPasswordHasherInterface $passwordHasher;
    private EntityManagerInterface $entityManager;
    private ?TokenInterface $token;
    public function __construct(UserPasswordHasherInterface $passwordHasher,
                                EntityManagerInterface $entityManager,
                                TokenStorageInterface $tokenStorage)
    {
        $this->passwordHasher = $passwordHasher;
        $this->entityManager = $entityManager;
        $this->token = $tokenStorage->getToken();
    }
}
```

```
}

public function supports($data): bool
{
    return $data instanceof User;
}

/**
 * @param User $data
 */
public function persist($data)
{
    $hashedPassword = $this->passwordHasher->hashPassword(
        $data,
        'passer'
    );
    $data->setPassword($hashedPassword);
    $this->entityManager->persist($data);
    $this->entityManager->flush();
}

public function remove($data)
{
    $this->entityManager->remove($data);
    $this->entityManager->flush();
}
}
```

10. Créer un Burger et lui affecte le User Connecter

a) Controller Personnalise

#Entity Burger

```
collectionOperations: [
    "add" => [
        'method' => 'Post',
        "path"=>"/add",
        "controller"=>BurgerController::class,
    ]
]
```

#Entity Burger Validator

1. Dépendance

composer require symfony/validator doctrine/annotations

2. NameSpace

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

3. Contraintes

```
#[Assert\NotBlank(message:"Le nom est Obligatoire")]
private $nom;
```

```
#[Assert\NotBlank(message:"Le prix est Obligatoire")]
private $prix;
```

#Controller Burger

```
class BurgerController extends AbstractController
{
    public function __invoke(Request $request,
        ValidatorInterface $validator,
        TokenStorageInterface $tokenStorage,
        SerializerInterface $serializer,
        EntityManagerInterface $entityManager) : JsonResponse
    {
    }
}
```

```

        $burger = $serializer->deserialize($request->getContent(),
Burger::class, 'json');

        $errors = $validator->validate($burger);

        if (count($errors) > 0) {
            $errorsString = $serializer->serialize($errors, "json");

            return new JsonResponse($errorsString
, Response::HTTP_BAD_REQUEST, [], true);
        }

        $burger->setUser($tokenStorage->getToken()->getUser());
        $entityManager->persist($burger);
        $entityManager->flush();

        $result = $serializer->serialize([
            'code' => Response::HTTP_CREATED,
            'data' => $burger
        ], "json", [
            "groups" => ["all"]
        ]);
    }

    return new JsonResponse($result, Response::HTTP_CREATED, [], true);
}
}

```

b) EventSubscriber #Controller UserSubscriber

```

class UserSubscriber implements EventSubscriberInterface
{
    private ?TokenInterface $token;

    public function __construct(TokenStorageInterface $tokenStorage)
    {
        $this->token = $tokenStorage->getToken();
    }
}

```

```

public static function getSubscribedEvents(): array
{
    return [
        Events::prePersist,
    ];
}

private function getUser()
{
    //dd($this->token);

    if (null === $token = $this->token) {
        return null;
    }

    if (!is_object($user = $token->getUser())) {
        // e.g. anonymous authentication
        return null;
    }

    return $user;
}

public function prePersist(LifecycleEventArgs $args)
{
    if ($args->getObject() instanceof Burger) {
        $args->getObject()->setUser($this->getUser());
    }
}

```

#service.yml

```

services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true      # Automatically injects dependencies in your services.
        autoconfigure: true # Automatically registers your services as commands, e

    App\EventSubscriber\TokenSubscriber:
        tags:
            - { name: doctrine.event_listener, event: prePersist}

```

11. Sécurité

1. Sécurité Global

```
#[ApiResource(  
    attributes: [  
        "security" => "is_granted('ROLE_GESTIONNAIRE')",  
        "security_message"=>"Vous n'avez pas access à cette Ressource",  
    ],  
)
```

2. Sécurité Dans une Opération

```
collectionOperations: [  
    "add" => [  
        "method" => 'Post',  
        "path"=>"/add",  
        "controller"=>BurgerController::class,  
        "security"=>"is_granted('ROLE_GESTIONNAIRE')",  
        "security"=>"Vous n'avez pas access à cette Ressource",  
    ],
```

12. Filter

Filters

- Search filter (partial, start, end, exact, ipartial, iexact)
- Date filter (?property[<after|before>]=value)
- Boolean filter (?property=[true|false|1|0])
- Numeric filter (?property=int|bigint|decimal)
- Range filter (?property[lt] | [gt] | [lte] | [gte] | [between]=value)
- Order filter (?order[property]=<asc|desc>)
- ...
- Custom filters



#Dans Burger.php

```
#[ApiFilter(SearchFilter::class,  
properties: ['id' => 'exact',  
            'user' => 'exact',
```

```
'nom'=>'ipartial'])]
```

Exemple End Point :

- localhost:8000/api/burgers?nom=viande
- localhost:8000/api/burgers?id=1
- localhost:8000/api/burgers?user=1

```
# [ApiFilter(NumericFilter::class, properties: ['prix'])]
```

- localhost:8000/api/burgers?prix=15000
- localhost:8000/api/burgers?prix[between]=2000..3000

13. Pagination

Pagination

```
# app/config/config.yml
api_platform:
# ...
    collection:
        pagination:
            items_per_page: 30 # Default value
            client_items_per_page: true # Disabled by default
            items_per_page_parameter_name: itemsPerPage # Default value
            client_enabled: true # optional
            enabled_parameter_name: pagination # optional
            page_parameter_name: _page # optional
```

