

PHP framework



CODEIGNITER



PHP > 7.2



MySQL > 5.1

# CodeIgniter 4.0.4

## API REST (Service Web)

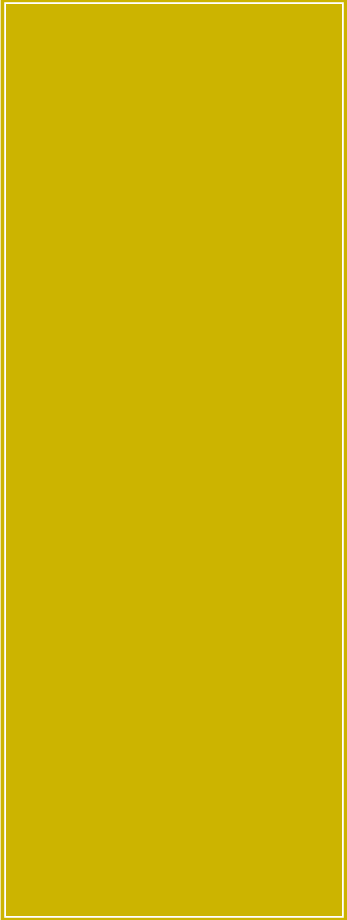


CI4\_5

Gwénaél LAURENT – [glaurent001@gmail.com](mailto:glaurent001@gmail.com)

# Plan

2

- 
- Qu'est-ce qu'une API REST ?
  - Le protocole HTTP
  - Echange des données en JSON
  - Les requêtes vers une API REST
  - Codelgniter 4 pour les API REST
  - Les requêtes de lecture GET
  - Les requêtes d'écriture POST
  
  - Les requêtes de suppression DELETE
  - Les requêtes de modification PUT
  - Les requêtes RESTful particulières

3

# Qu'est-ce qu'une API REST ?



# Qu'est-ce qu'une API ?

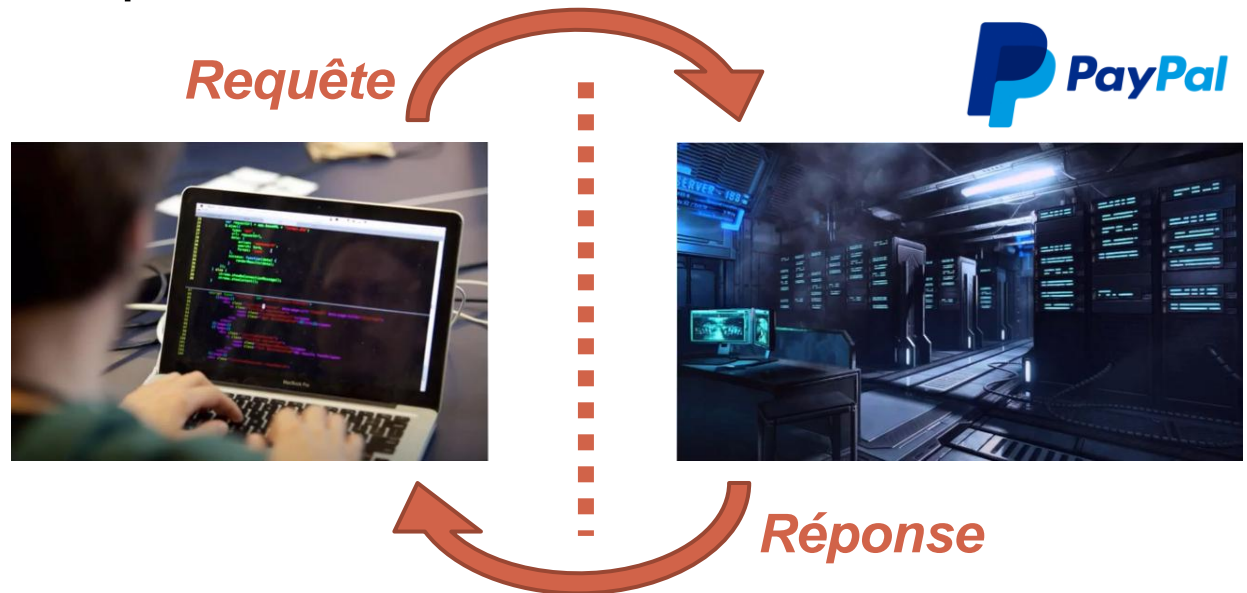
4

**API** : Application Programming Interface

**Interface** utilisée par des programmes pour interagir avec une autre application

Une façon de communiquer

Si 2 applications doivent se parler, elles ont besoin d'une interface



# Qu'est-ce qu'une API REST ?

5

**REST** : Representational State Transfer

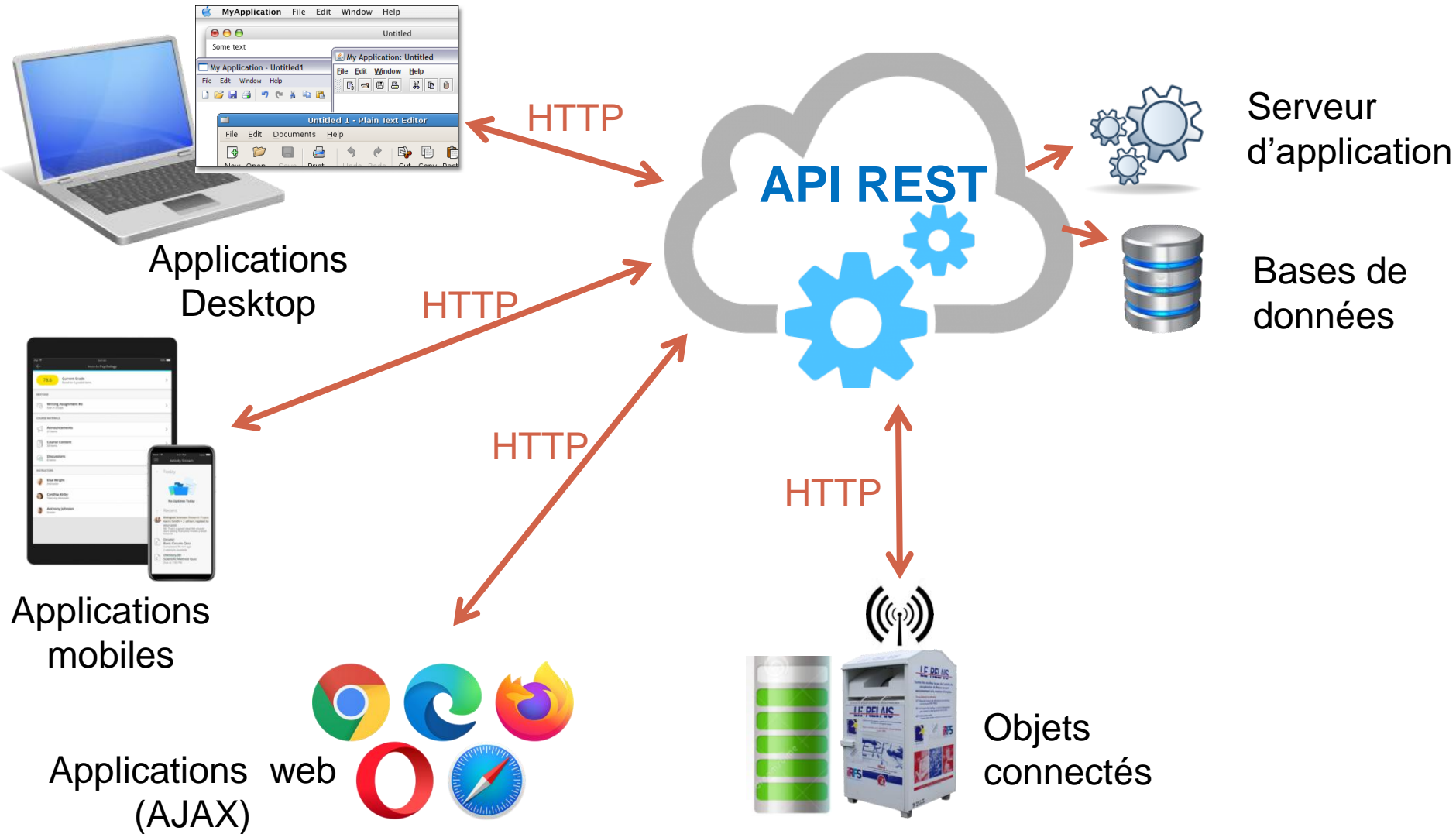
**Interface « normalisée »** pour simplifier l'échange d'information (architecture logicielle)

- **API distante** : les ressources exposées par l'API ne se trouvent pas sur l'ordi qui envoie les requêtes
- **API Web** : Utilise le **protocole HTTP** pour les requêtes / réponses

API Web = Web Services

# Exemples d'utilisation des API REST

6



# Accès aux API

7

- API privées :
  - ▣ Utilisables qu'en interne
- API partenaires :
  - ▣ Partagées avec certains partenaires de l'entreprise
- API publiques :
  - ▣ Accessibles à tous

En fonction du domaine d'utilisation, il faut adapter :

- Le choix des données exposées (ressources)
- Le choix des actions sur les ressources (modification, suppression)
- La gestion de la sécurité

8

# Le protocole HTTP



**http://**



# Le protocole HTTP en 2 parties

9

## Modèle TCP/IP

4	Application	HTTP
3	Transport	TCP
2	Internet	IP
1	Accès réseau	Ether.

*n° port*

*@ IP*

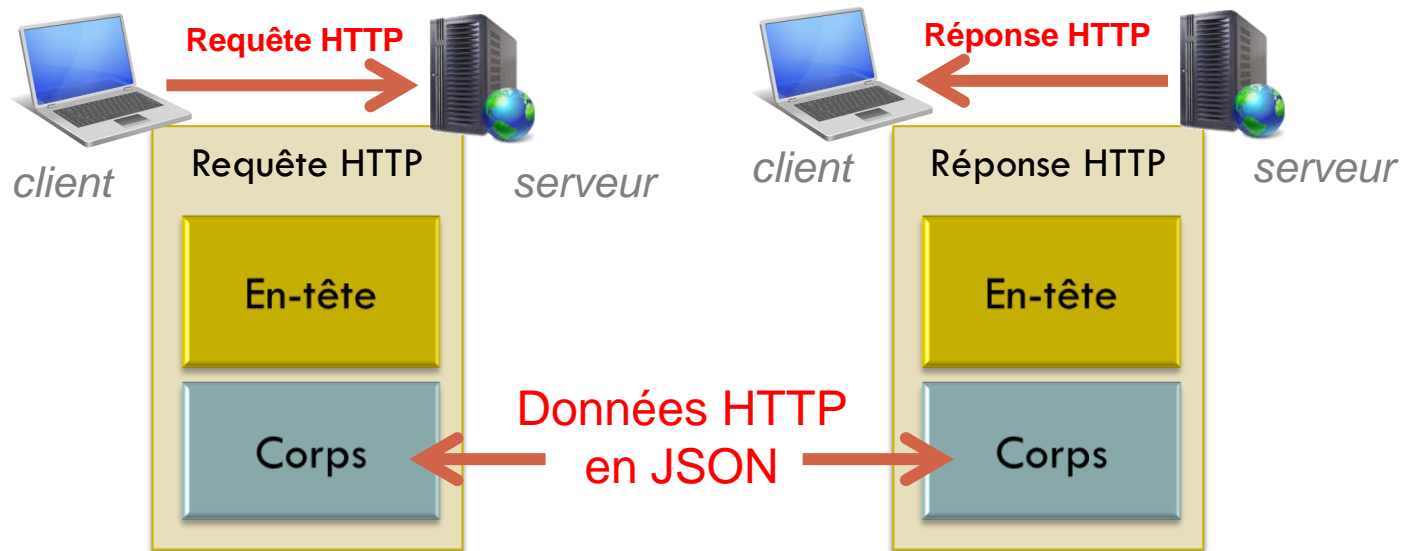
*@ MAC*

## La trame envoyée sur le réseau

*Les données sont « encapsulées » par les protocoles*



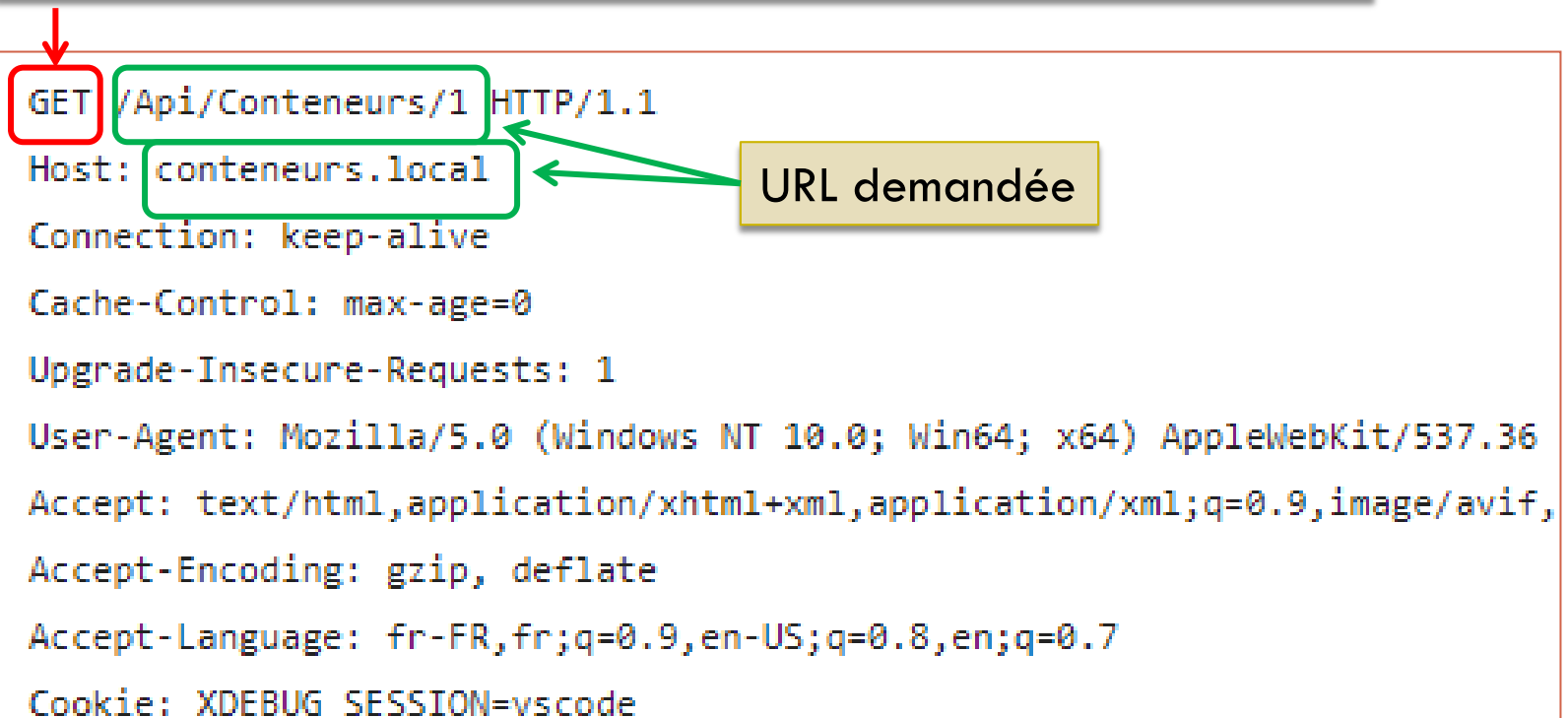
## La trame HTTP est composé de 2 parties



# L'en-tête d'une requête HTTP

10

**Type de requête HTTP** = « verbes » HTTP  
Il en existe plusieurs : GET, POST, PUT, DELETE

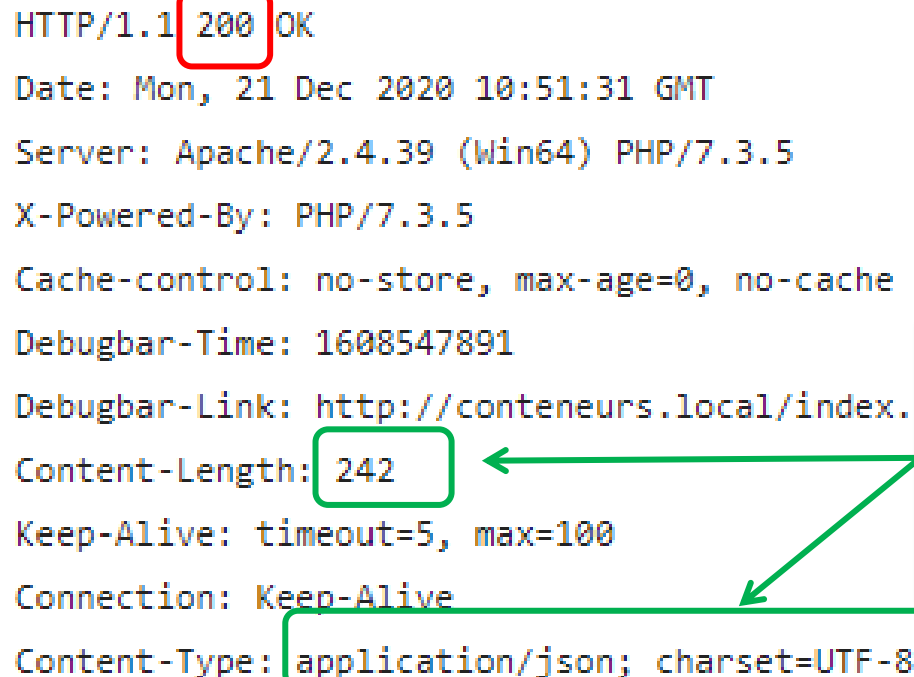


```
GET /Api/Conteneurs/1 HTTP/1.1
Host: conteneurs.local
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: XDEBUG_SESSION=vscode
```

# L'en-tête d'une réponse HTTP

11

**Code de statut de la réponse.** Il en existe plusieurs :  
200 : Le serveur web a trouvé la page demandée  
404 : la page demandée n'existe pas

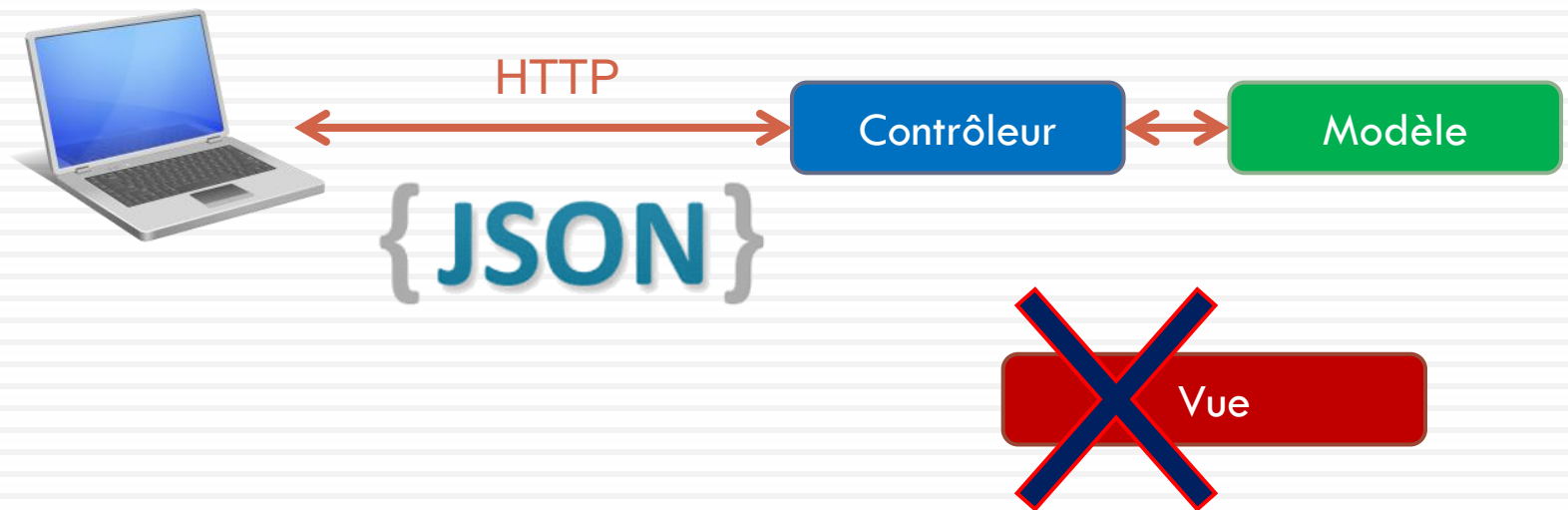


```
HTTP/1.1 200 OK
Date: Mon, 21 Dec 2020 10:51:31 GMT
Server: Apache/2.4.39 (Win64) PHP/7.3.5
X-Powered-By: PHP/7.3.5
Cache-control: no-store, max-age=0, no-cache
Debugger-Time: 1608547891
Debugger-Link: http://conteneurs.local/index.
Content-Length: 242
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

Informations sur la donnée  
contenue dans le corps :

- 242 octets
- Format JSON

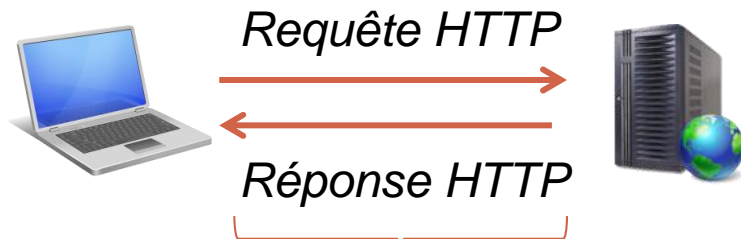
# Echange des données en JSON



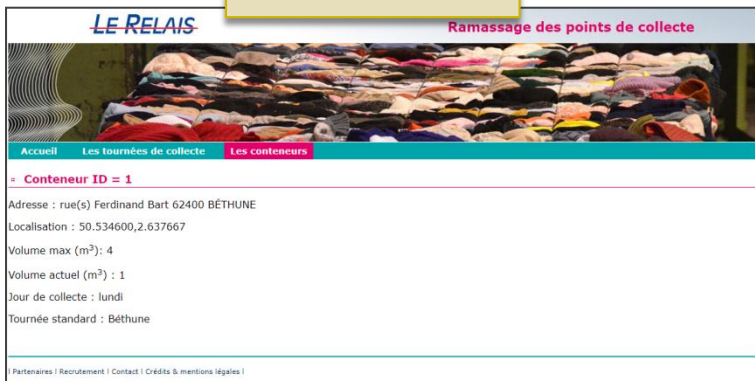
# HTTP pour site web ou API REST

13

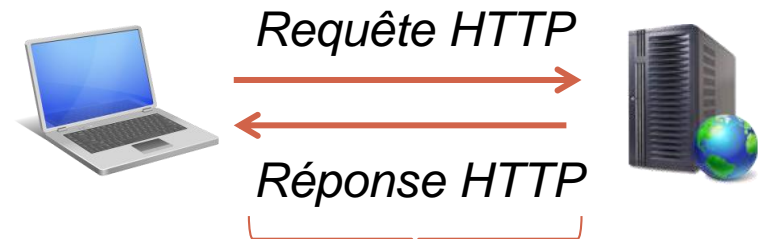
## Site web



**Code HTML**



## API REST



**Données**

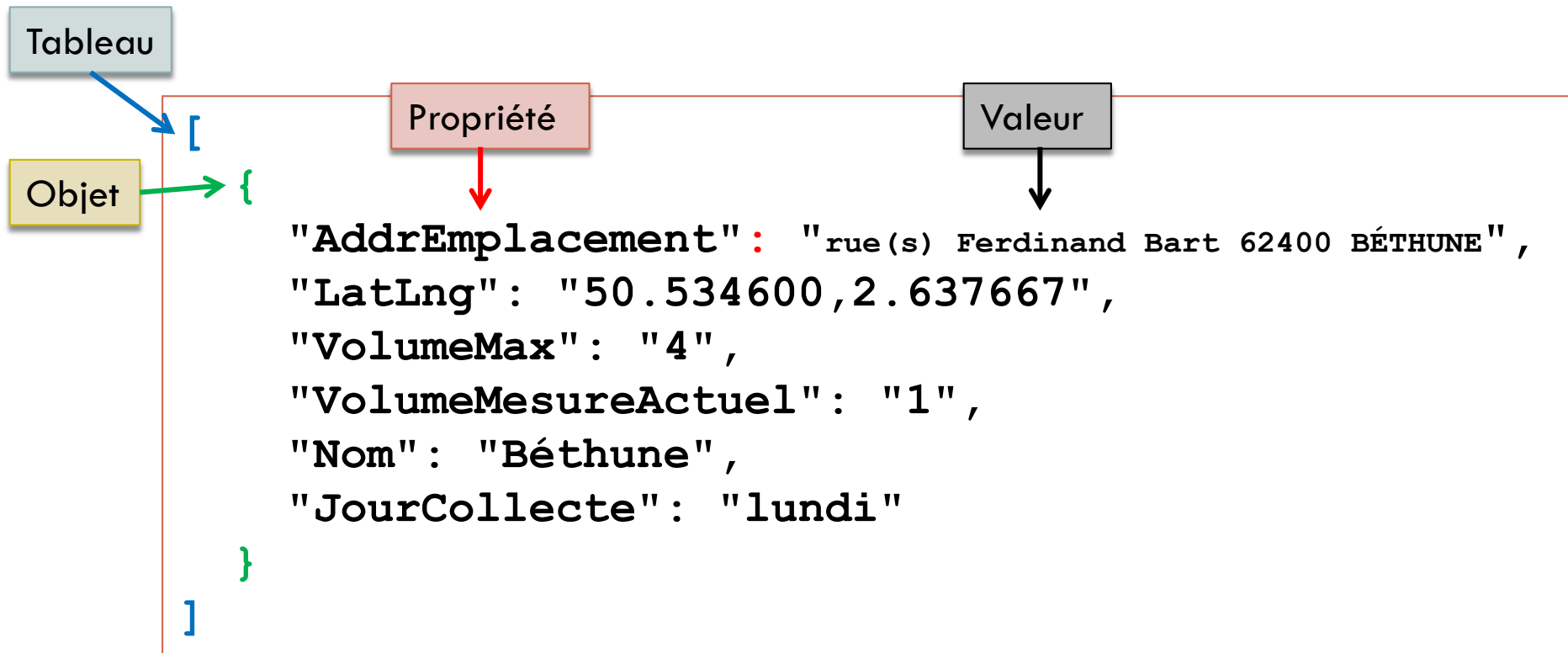
```
[
  {
    "AddrEmplacement": "rue(s) Ferdinand Bart 62400 BÉTHUNE",
    "LatLng": "50.534600,2.637667",
    "VolumeMax": "4",
    "VolumeMesureActuel": "1",
    "Nom": "Béthune",
    "JourCollecte": "lundi"
  }
]
```

**{JSON}**

# Formats d'échange des données

14

- Les données renvoyées par une API REST sont souvent écrites au **format JSON** (*JavaScript Object Notation*)



# Les requêtes vers une API REST

**RESTful API**  
**GET POST PUT DELETE**

# Fondamental d'une API REST

16

- **2 choses importantes pour une API REST** (ou RESTful)
  - ▣ Les **Ressources** sont identifiables par leur URI (*Uniform Resource Identifier*) et accessibles en HTTP  
[http://point\\_terminal/nom\\_de\\_ressource/](http://point_terminal/nom_de_ressource/)
  - ▣ Le type de requête HTTP (en-tête) indique l'action à effectuer sur la ressource (les **verbes**)
    - GET : lire
    - PUT : modifier
    - POST : écrire
    - DELETE : supprimer

Les verbes HTTP correspondent au type d'accès aux données de la base : **CRUD**



# Ex: Gérer les livres d'une bibliothèque

17

- L'adresse de notre bibliothèque représente le « point terminal » (endpoint) :  
<http://bibliotheque/>
- les livres pourront être manipulés à une URI formée par convention de la sorte :  
`http://point_terminal/nom_de_ressource/`  
<http://bibliotheque/livres/>

# Ex: Manipulation sur ces livres

18

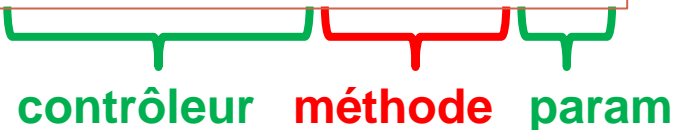
- **lire** : Requête de type **GET** sur  
[http://bibliotheque/livres/ID\\_DU\\_LIVRE](http://bibliotheque/livres/ID_DU_LIVRE)
- **écrire** : Requête de type **POST** sur  
<http://bibliotheque/livres/>  
Le corps du message POST représente le contenu du nouveau livre à créer.
- **modifier** : Requête de type **PUT** sur  
[http://bibliotheque/livres/ID\\_DU\\_LIVRE](http://bibliotheque/livres/ID_DU_LIVRE)  
Le corps du message PUT représente le contenu modifié du livre d'identifiant ID\_DU\_LIVRE.
- **supprimer** : Requête de type **DELETE** sur  
[http://bibliotheque/livres/ID\\_DU\\_LIVRE](http://bibliotheque/livres/ID_DU_LIVRE)

# CodeIgniter n'est pas RESTful ...

19

- Les URLs standards de **CodeIgniter** :

`http://conteneurs.local/Cconteneur/detail/2`

  
contrôleur méthode param

- En **RESTful**, on devrait avoir :

**GET** `http://conteneurs.local/Api/Cconteneur/2`

  
contrôleur param

... mais CodeIgniter fournit ce qu'il faut pour créer des API RESTful !

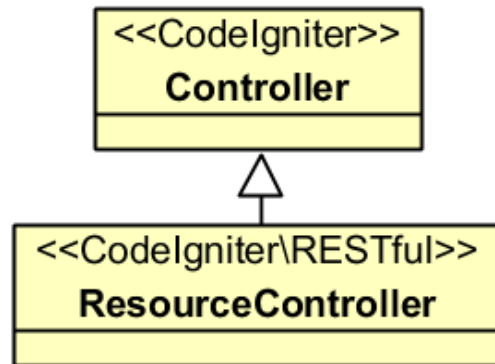
# CodeIgniter 4 pour les API REST



# CodeIgniter 4 pour les API REST

21

- CodeIgniter permet de créer des API RESTful avec des URL conformes à l'architecture REST grâce à :
  - ▣ La classe *ResourceController*



- ▣ Un système automatisé de routage des URL

## Routage des URL :

faire correspondre une URL  
avec une méthode d'un contrôleur

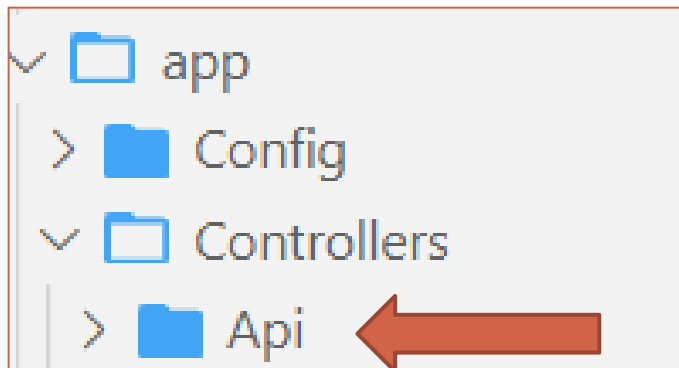
# Emplacement des contrôleurs REST



22

**Attention :** les contrôleurs de l'API REST ne doivent pas se trouver dans le même dossier que les autres contrôleurs

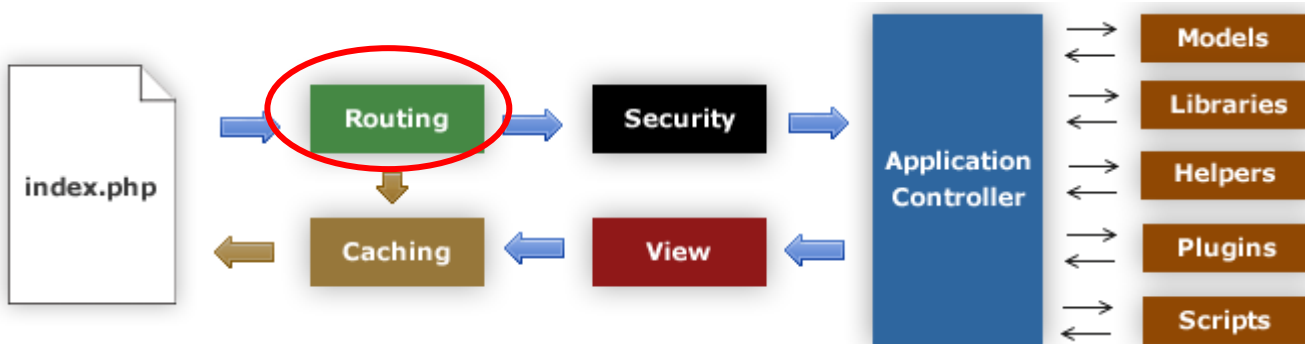
- Créez le dossier `/app/Controllers/Api`
  - ▣ Tous les contrôleurs REST devront être créés dans le dossier « Api »
  - ▣ Un contrôleur pour chaque ressource exposée par l'API



# Routage dans CodeIgniter

23

- **Routage** : faire correspondre une URL avec une méthode d'un contrôleur



- CodeIgniter intègre un routage « standard » pour les pages MVC que nous avons créées.
- Nous l'avons déjà utilisé pour définir la page d'accueil

```
$routes->get('/', 'Caccueil::index');
```



# Routage pour l'API REST

24

- Le routage de CodeIgniter se configure dans **/app/Config/Routes.php**
- Ajoutez le routage de l'API REST
  - ▣ Pour la ressource « Conteneurs » (dans le sous dossier « Api »)

```
$routes->resource('Api/Conteneurs');
```

Cette seule ligne est équivalente à la création de toutes ces routes :

```
$routes->post('Api/Conteneurs', 'Api/Conteneurs::create');  
$routes->get('Api/Conteneurs', 'Api/Conteneurs::index');  
$routes->get('Api/Conteneurs/(:segment)', 'Api/Conteneurs::show/$1');  
$routes->put('Api/Conteneurs/(:segment)', 'Api/Conteneurs::update/$1');  
$routes->delete('Api/Conteneurs/(:segment)', 'Api/Conteneurs::delete/$1');
```



# Résumé du routage pour la ressource « Conteneurs »

25

Operation	Method	Controller Route	Controller Function
<b>Create</b>	POST	Api/Conteneurs	create()
<b>List</b>	GET	Api/Conteneurs	index()
<b>Show</b>	GET	Api/Conteneurs/(:segment)	show(\$id = null)
<b>Update</b>	PUT	Api/Conteneurs/(:segment)	update(\$id = null)
<b>Delete</b>	DELETE	Api/Conteneurs/(:segment)	delete(\$id = null)

# Exemple : Routage REST pour lister les conteneurs

26

- Pour une requête envoyée en GET sur cette URL

Segment 1 Segment 2  
`http://conteneurs.local/Api/Conteneurs`

contrôleur

- **Le routage** appellera la méthode `index()` du contrôleur `Api/Conteneurs`

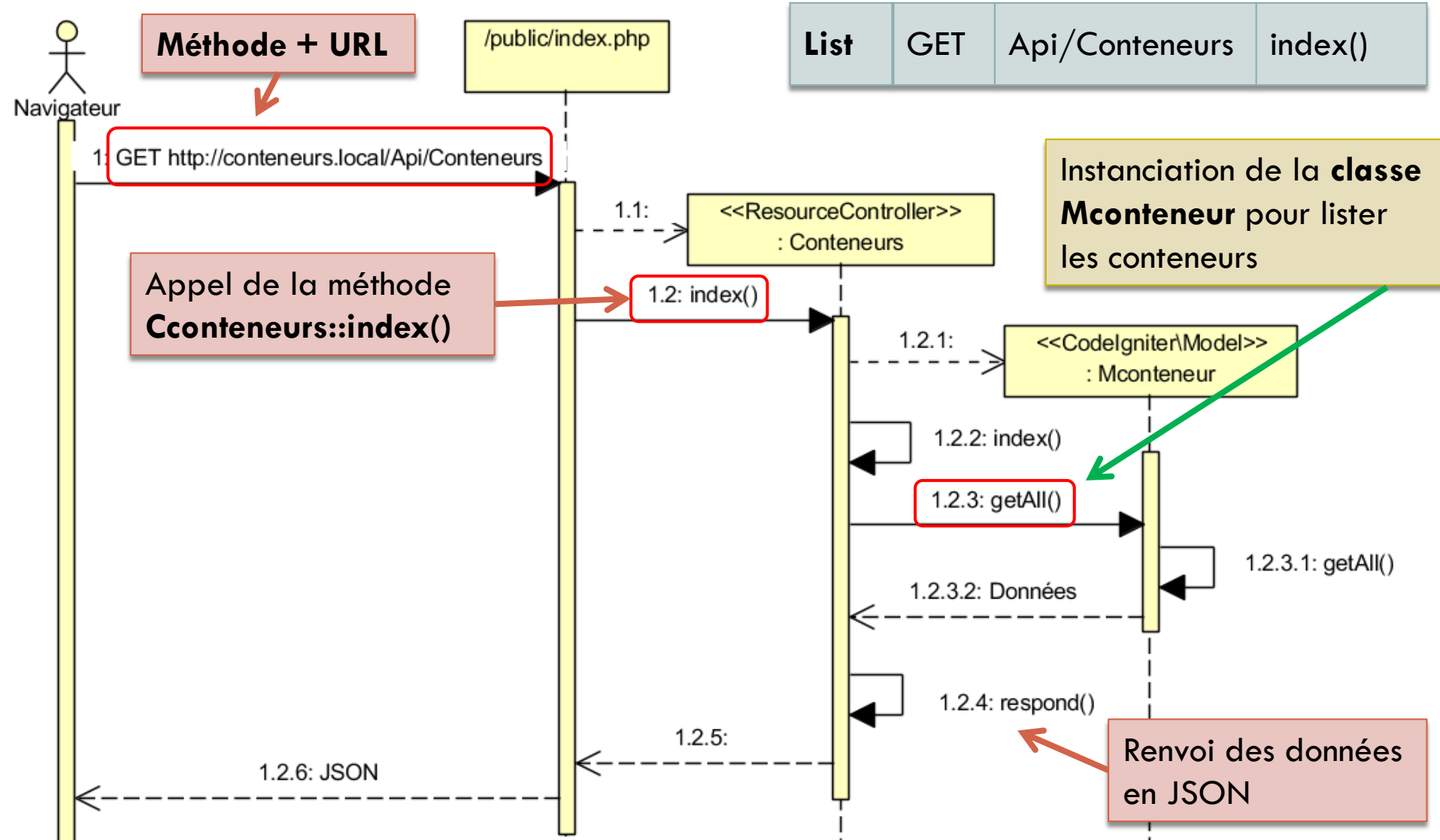
```
namespace App\Controllers\Api;  
class Conteneurs extends ResourceController  
{  
    public function index()  
    {
```

## Conteneurs

+create()
<b>+index()</b>
+show(\$id = null)
+update(\$id = null)
+delete(\$id = null)

# UML : Routage REST pour lister les conteneurs

27



# Créer le contrôleur REST pour la ressource Conteneurs



28

/app/Controllers/Api/Conteneurs.php

```
<?php
namespace App\Controllers\Api;
use CodeIgniter\RESTful\ResourceController;
```

```
class Conteneurs extends ResourceController
```

```
{
```

```
    protected $modelName = 'App\Models\Mconteneur';
    protected $model;
```

```
    protected $format = 'json';
```

```
    public function index()
```

```
{
```

```
        $result = $this->model->getAll();
        return $this->respond($result);
```

```
}
```

```
}
```

## Tous les contrôleurs REST doivent :

- être créés dans le dossier /app/Controllers/Api
- faire partie du namespace App\Controllers\Api
- importer le namespace CodeIgniter\RESTful\ResourceController
- dériver de ResourceController

Définir \$modelName permet d'instancier automatiquement le modèle dans \$model

\$result est un tableau associatif

Les tableaux associatifs retournés avec \$this->respond() sont automatiquement converties au format JSON

# Tester les API REST (1 / 2)

29

- Pour les requêtes envoyées en GET, un navigateur suffit (URL dans la barre d'adresse)

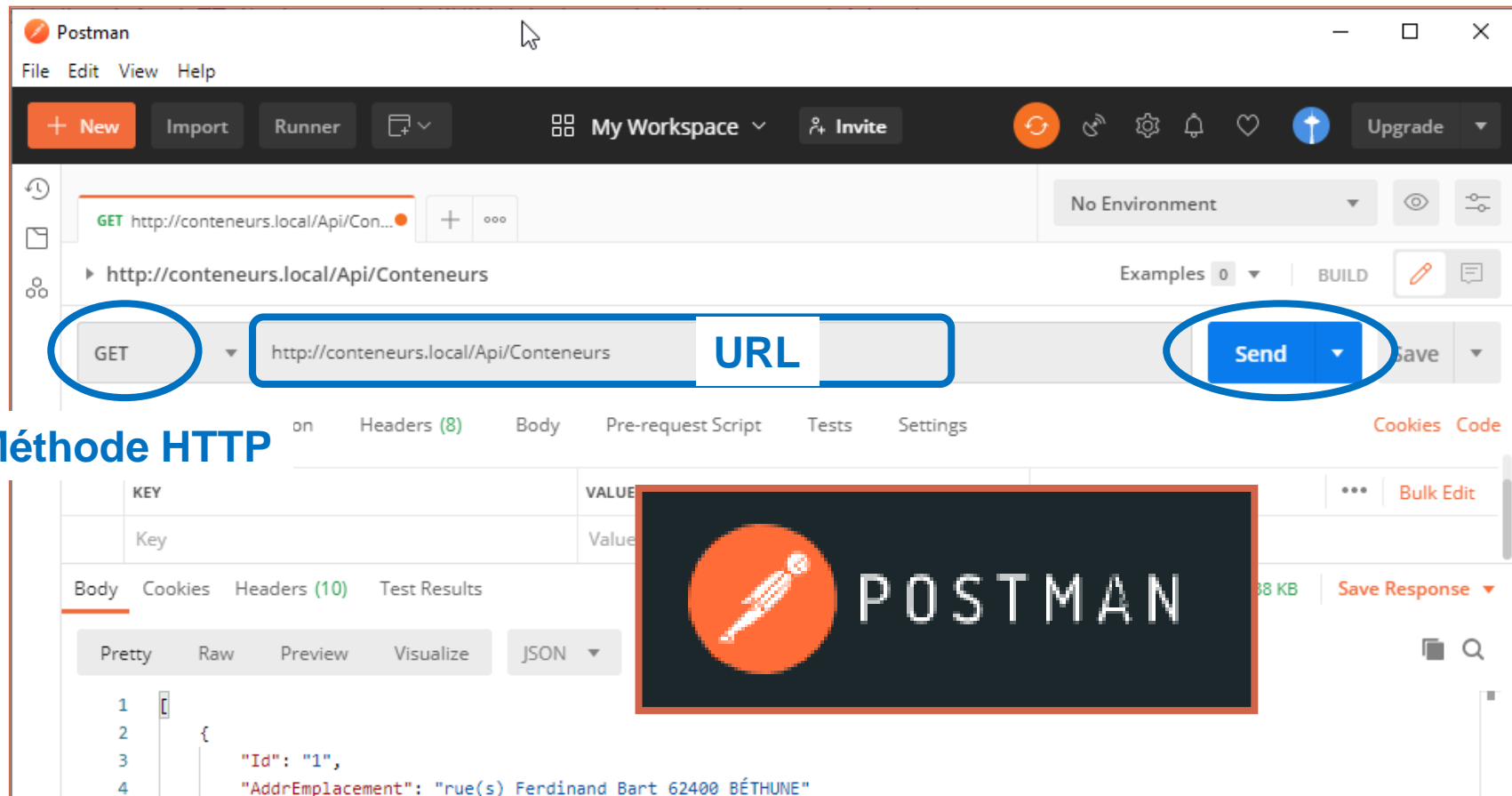


- Pour les autres méthodes HTTP :
  - ▣ POST : il faut coder un formulaire HTML
  - ▣ PUT et DELETE : il faut coder une requête AJAX en JS

# Tester les API REST (2/2)

30

- La meilleure méthode est d'utiliser le logiciel Postman  
<https://www.postman.com/downloads/>



# Les requêtes de lecture GET

RESTful API

**GET** POST PUT DELETE

# Les requêtes GET



32

- Les requêtes envoyées en HTTP GET vers l'API sont dédiées à la lecture des données
- Ces requêtes concernent la ressource « Conteneurs »
- Elles correspondent à la requête **SQL SELECT**
- Elles sont de 2 types (Liste/détail) :
  - ▣ **Lister** les conteneurs
  - ▣ Obtenir le **détail** d'un conteneur
- CodeIgniter utilise 2 routes différentes :

List	GET	Api/Conteneurs	index()
Show	GET	Api/Conteneurs/(:segment)	show(\$id = null)



# Les requêtes GET

GET

En-tête HTTP

~~Données HTTP~~

33

- Les requêtes GET sont routées vers 2 méthodes différentes du contrôleur

**GET**

`/Api/Conteneurs`



Contrôleur REST

`index()`

Status=200 'OK'

renvoie la liste de tous les conteneurs

**GET**

`/Api/Conteneurs/2`



Contrôleur REST

`show(2)`

Status=200 'OK'

renvoie le détail du conteneur ID=2

**Testez votre API avec Postman.** Si vous créez un compte gratuit, vous pouvez même enregistrer vos requêtes !

Conteneurs

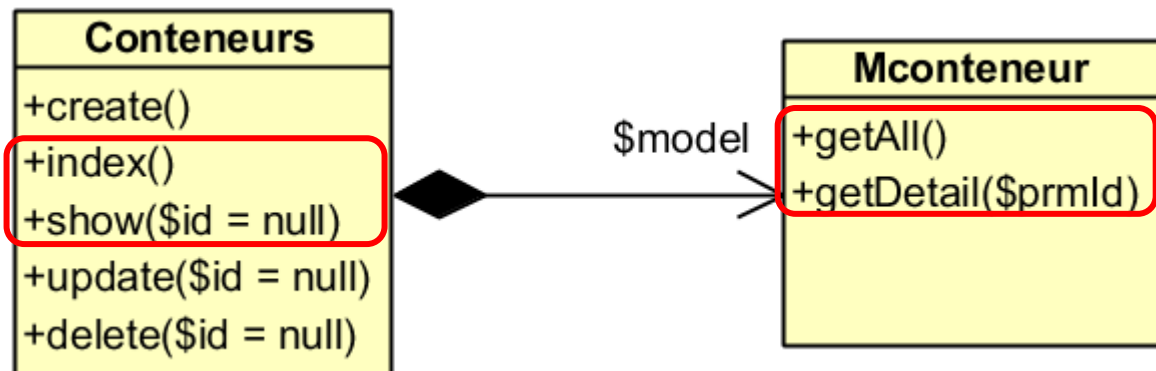
+create()  
+index()  
+show(\$id = null)  
+update(\$id = null)  
+delete(\$id = null)



# Codage des requêtes GET

34

- Modifiez le contrôleur REST Conteneurs pour qu'il réponde aux 2 types de requêtes GET
  - ▣ Lister les conteneurs : renvoie **un tableau d'objets** JSON
  - ▣ Obtenir le détail d'un conteneur : renvoie **un objet** JSON
- Servez vous des méthodes déjà codées dans le modèle Mconteneur



# Les « status code » HTTP

35

- Pour la plupart des réponses, les contrôleurs REST renvoient **implicitement le bon code** de statut.
- On peut néanmoins choisir explicitement le code à renvoyer :
  - ▣ Valeur numérique :

```
return $this->respond($result, 400);
```

- ▣ ou en utilisant le tableau de codes :

```
$this->codes['invalid_request']
```

=

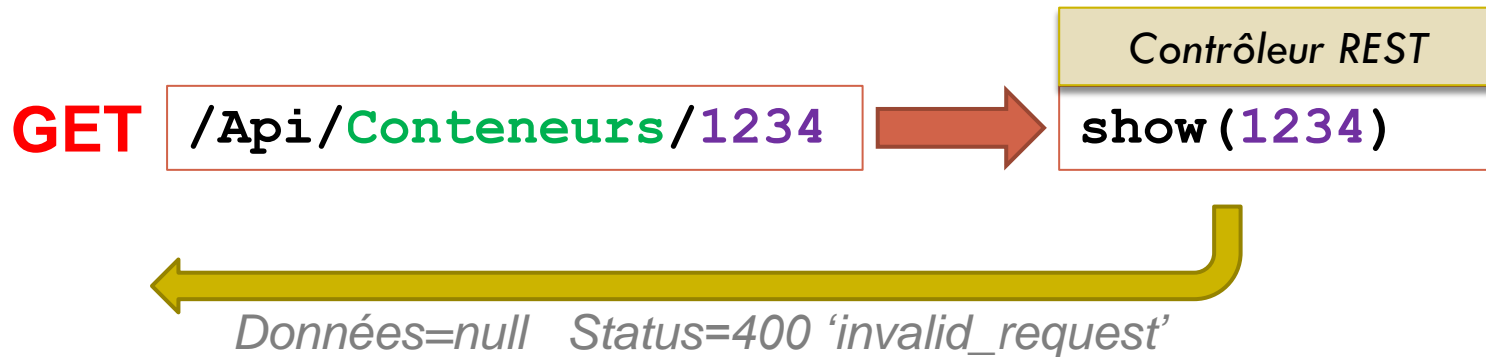
```
400
```

# Gestion des erreurs



36

- Modifiez le code du contrôleur pour renvoyer une erreur 400 quand l'ID du conteneur n'existe pas



**Remarque** : Si CodeIgniter est configuré en mode développement (`CI_ENVIRONMENT = development` dans le fichier `/.env`), l'API renvoie quand même des données (c'est le code javascript normalement destiné au navigateur). Désactivez ce mode pendant la mise au point de l'API REST : **`CI_ENVIRONMENT = production`**

# Tester le service REST en GET



37

Postman

File Edit View Help

+ New Import Runner

POSTMAN

GET http://conteneurs.local/Api/Co... X + ...

http://conteneurs.local/Api/Conteneurs/1 Examples 0

GET http://conteneurs.local/Api/Conteneurs/1 Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 31 ms Size: 463 B

```
1 [
2   {
3     "AddrEmplacement": "rue(s) Ferdinand Bart 62400 BÉTHUNE".
```

# Les requêtes de création POST

**RESTful API**  
**GET POST PUT DELETE**

<https://codeigniter4.github.io/userguide/models/model.html>

<https://codeigniter4.github.io/userguide/database/helpers.html>

[https://codeigniter4.github.io/userguide/database/query\\_builder.html#inserting-data](https://codeigniter4.github.io/userguide/database/query_builder.html#inserting-data)

# Les requêtes POST



39

- Les requêtes envoyées en HTTP POST vers l'API sont dédiées à la création de nouvelles données
- Ces requêtes concernent la ressource « Conteneurs »
- Elles correspondent à la requête **SQL INSERT**
- Une requête POST insère **une seule nouvelle ligne** dans la base de données (*fonctionnement par défaut*):
  - ▣ Les données de la nouvelle ligne sont envoyées comme donnée POST au format JSON (1 seul objet JSON)
- Codelgniter utilise une route :

Create

POST

Api/Conteneurs

create()

# Données JSON envoyées en POST

40

- Les données envoyées seront de ce type :

```
dto={  
  "AddrEmplacement": "rue Jules Lebleu 59280 Armentières",  
  "LatLng": "50.688937, 2.871912",  
  "VolumeMax": "3"  
}
```

- L'objectif est d'enregistrer ces données dans une **nouvelle ligne de la table MySQL « conteneur »**

- Les champs non listés dans **dto** resteront vide (null)
- Après insertion, on récupère la valeur de l'Id du conteneur ajouté

Table : conteneur	
Id	
TourneeStandardId	
AddrEmplacement	
LatLng	
RefSigfox	
CoefSurface	
VolumeMax	
VolumeMesureActuel	
DateDerniereCollecte	



# Les requêtes POST

POST

dto={ }

En-tête HTTP

Données HTTP

41

- Les requêtes POST sont routées vers une seule méthode du contrôleur

**POST**

/Api/Conteneurs



dto={ ...données\_JSON... }

Contrôleur REST

create()

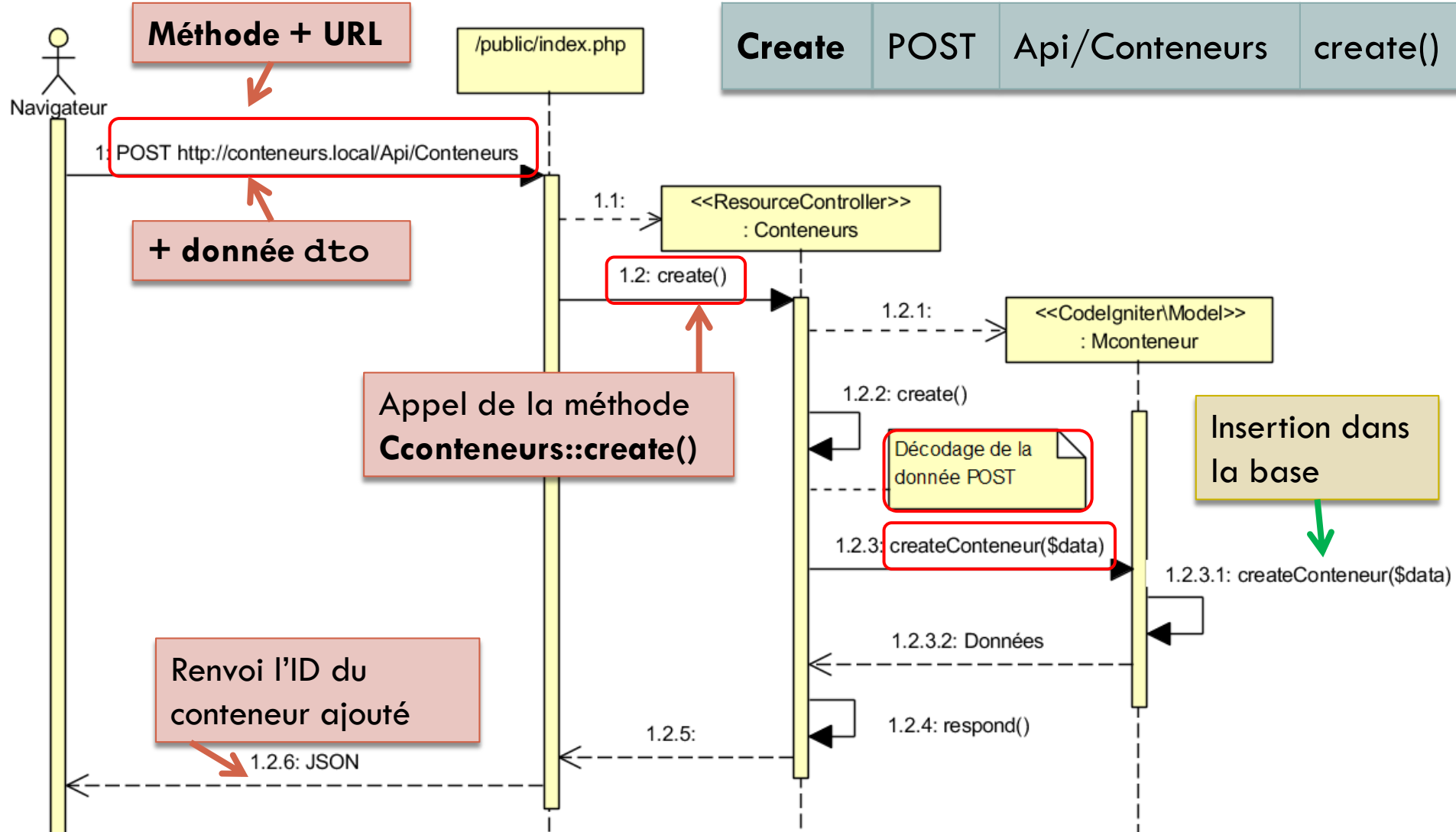
renvoie l'ID du  
conteneur ajouté

*Status=201 created*

Note : La donnée envoyée en POST peut porter le nom que vous voulez. J'ai choisi dto pour Data Transfer Object.

# UML : Routage REST pour créer un conteneur

42



# Codage des requêtes POST

## Le contrôleur



43

### □ Codage du contrôleur, méthode create()

#### ▣ Décodage de la donnée reçue en POST (dto)

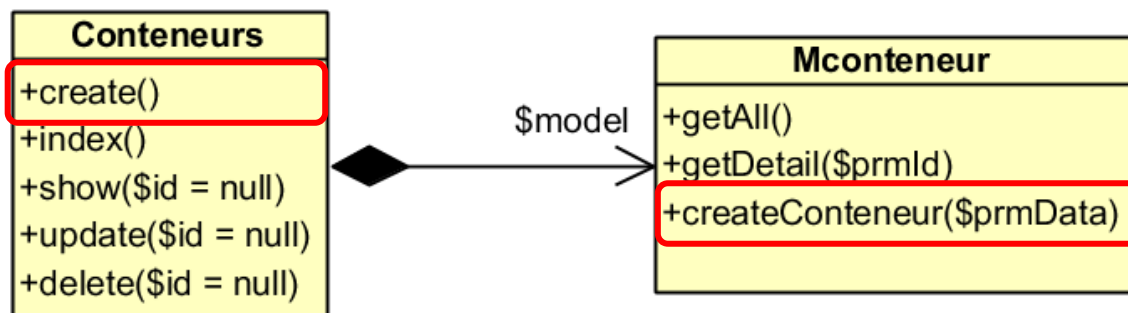
```
$data = $this->request->getPost('dto');  
$data = json_decode($data, true);
```

#### ▣ Si dto existe :

- Enregistrer la donnée dans la base

- Renvoyer l'Id du nouveau conteneur + Status Code 201

#### ▣ Sinon : Renvoyer un status Code 400



# Codage des requêtes POST

## Le modèle



44

### □ Codage du modèle, méthode createConteneur(\$prmData)

/app/Models/Mconteneur.php

```
public function createConteneur($prmData)
{
    //nom des colonnes qui peuvent être modifiées par cette requête
    //(obligatoire pour les requêtes INSERT et UPDATE)
    $this->allowedFields = ['AddrEmplacement', 'LatLng', 'VolumeMax'];

    $this->insert($prmData);

    //ID du nouvel enregistrement retourné dans un tableau associatif
    $retour['lastInsertId'] = $this->insertID('Id');
    return $retour;
}
```

*Requête SQL correspondante :*

```
INSERT INTO `conteneur` (`AddrEmplacement`, `LatLng`, `VolumeMax`)
VALUES ('rue(s) Ferdinand Bart 62400 BÉTHUNE', '50.534600,2.637667', '4')
```

# Tester le service REST en POST



45



POSTMAN interface showing a REST client setup for a POST request.

**Request Details:**

- Method: **POST** (highlighted with a blue circle)
- URL: **http://conteneurs.local/Api/Conteneurs** (highlighted with a blue circle)
- Body Type: **x-www-form-urlencoded** (highlighted with a blue circle)

**Body Data (Key-Value Pairs):**

KEY	VALUE
<input checked="" type="checkbox"/> dto	{ ... "AddrEmplacement": "rue Jules Lebleu 59280 Armentières", ... "LatLng": "50.688937505876254, 2.871912513621717", ... "VolumeMax": "3"
Key	

**Response Details:**

- Status: **201 Created**
- Time: **5.09 s**
- Size: **315 B**
- Body Type: **JSON** (highlighted with a red circle)
- Response Content: **"lastInsertId": 65** (highlighted with a red circle)