

# REST

## REpresentational State Transfer



# AU MENU

## SOAP / REST

Environnement de développement

Rest – Présentation

Méthodes HTTP

HTTP status codes

Exemples de code

Bonnes pratiques

Documenter/tester votre API avec Swagger

Outils pour tester l'API

Sécurité des API

- ◇ Deux grandes familles de services WEB :
- ◇ **SOAP (Simple Object Access Protocol)** est un protocole qui impose un ensemble de règles de communication. Il repose sur des messages de requête et de réponse, généralement transmis via HTTP, SMTP ou TCP.
- ◇ **REST (Representational State Transfer)** est un style architectural qui ne dicte pas de protocole particulier. Il tire parti des protocoles existants, en utilisant principalement des méthodes HTTP telles que GET, POST, PUT et DELETE.
- ◇ SOAP est un protocole standard qui transfère des informations **axées sur les fonctions**, alors que REST a un style architectural avec une approche plus **axée sur les données**.

## ◇ SOAP (Simple Object Access Protocol)

- ◇ Protocole conçu pour l'échange de données.
- ◇ Ses avantages sont liés à **l'ensemble de règles et de normes** qui doivent être respectées pour assurer la réussite des interactions client/serveur.
- ◇ Les **demandes SOAP sont transmises par des enveloppes** qui doivent contenir toutes les informations nécessaires au traitement de la demande.
- ◇ Une enveloppe de demande SOAP contient généralement un en-tête facultatif et un attribut de corps obligatoire.
  - ◇ L'attribut En-tête contient des informations telles que les informations d'identification de sécurité et d'autres métadonnées.
  - ◇ L'attribut Corps est réservé au traitement des données réelles et des erreurs éventuelles.
- ◇ Spécifications du W3C.



<https://www.w3.org/TR/soap12/>

- ◇ SOAP (Simple Object Access Protocol) : principales spécifications
  - ◇ WS-Security (Web Services Security) : spécification qui standardise la manière dont les messages sont sécurisés et transférés via des identifiants uniques appelés jetons.
  - ◇ WS-ReliableMessaging : spécification qui standardise la gestion des erreurs entre les messages transférés par le biais d'une infrastructure informatique non fiable.
  - ◇ WS-Addressing (Web Services Addressing) : spécification qui ajoute les informations de routage des paquets en tant que métadonnées dans des en-têtes SOAP, au lieu de les conserver plus en profondeur dans le réseau.
  - ◇ WSDL (Web Services Description Language) : décrit la fonction d'un service web ainsi que ses limites.

# SOAP

- ◇ SOAP : transport d'un message
- ◇ SOAP utilise des requêtes HTTP POST dans le corps de la demande.



# SOAP

## ◇ SOAP : structure d'un message



SOAP définit un format pour l'envoi des messages. Messages sont structurés en un document XML représentant une **enveloppe**, Une enveloppe comprend :

- Un en-tête (facultatif) : contient les données opérationnelles
- Un corps (obligatoire)
  - La structure du corps est applicative
  - Contient les données fonctionnelles
  - Peut transporter le détail d'une erreur

- ◇ REST (Representational State Transfer)
  - ◇ REST est un ensemble de principes architecturaux adapté aux besoins des services web et applications mobiles légers. La mise en place de ces recommandations est laissée à l'appréciation des développeurs.
  - ◇ L'envoi d'une requête de données à une API REST se fait par le protocole HTTP (Hypertext Transfer Protocol).
  - ◇ À la réception de la requête, les API développées selon les principes REST (appelées API ou services web RESTful) peuvent renvoyer des messages dans différents formats : HTML, XML, texte brut et JSON.
  - ◇ Le format JSON (JavaScript Object Notation) est le plus utilisé pour les messages, car, en plus d'être léger, il est lisible par tous les langages de programmation ainsi que par les humains. Les API RESTful sont ainsi plus flexibles et plus faciles à mettre en place.



# SOAP vs. REST



## ◇ SOAP vs. REST

- ◇ SOAP transmet des messages au format XML standard en raison de son état, tandis que l'API REST ne suit pas de format de données en raison de son absence d'état.
- ◇ SOAP fonctionne avec WSDL (Web Service Description Language) grâce au format XML, tandis que REST utilise des requêtes telles que GET, PUT, POST et DELETE.
- ◇ SOAP peut fonctionner avec n'importe quel protocole de communication, comme HTTP, HTTPS, TCP et SMTP, tandis que REST utilise uniquement le protocole HTTP (ou HTTPS) pour communiquer.
- ◇ SOAP est plus structurée, tandis que REST utilise des données sous forme volumineuse.
- ◇ SOAP est le meilleur choix pour les grandes organisations qui recherchent la sécurité, comme les banques, tandis que REST fonctionne efficacement avec les appareils mobiles.
- ◇ SOAP nécessite une bande passante élevée, alors que REST nécessite une bande passante minimale.
- ◇ SOAP ne peut pas utiliser REST parce que c'est un protocole, alors que REST peut utiliser SOAP en raison de son style architectural.

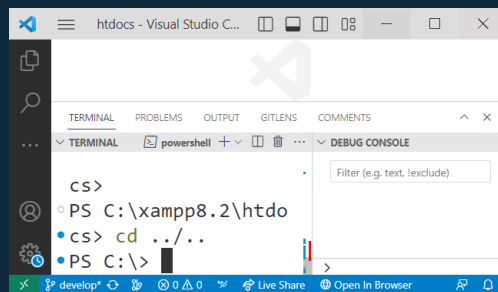
# SOAP vs. REST



## ◇ SOAP vs. REST

- ◇ SOAP a des normes de sécurité élevées, alors que REST utilise le protocole SSL (Secure Socket Layer) et HTTPS pour la sécurité. Les organisations qui ont besoin de protocoles de haute sécurité utilisent des SOAP pour sécuriser les informations sensibles des utilisateurs. Par exemple, les banques utilisent SOAP pour sécuriser les données des utilisateurs, comme le numéro de carte et le code confidentiel.
- ◇ SOAP supporte le format de données XML, tandis que REST supporte le texte brut, XML, HTML, JSON, etc.
- ◇ SOAP est un **protocole standard qui transfère des informations axées sur les fonctions**, alors que **REST a un style architectural avec une approche plus axée sur les données**.
- ◇ SOAP ne peut pas mettre en cache les appels. **REST peut mettre en cache tous les appels**, ce qui le rend plus rapide que SOAP.

# ENVIRONNEMENT DE DEVELOPPEMENT



# Environnement de développement

- ◇ **Visual Studio Code**
- ◇ **Xampp** (PHP + Apache + MySQL)  
<https://www.apachefriends.org/fr/index.html>
- ◇ **Composer** : pour installer des packages  
<https://getcomposer.org/Composer-Setup.exe>
- ◇ **Postman** : pour tester l'API  
<https://www.postman.com/product/rest-client/>
- ◇ **Curl** : pour tester l'API (ligne de commandes)  
<https://curl.se/>
- ◇ **Swagger-PHP** : pour annoter votre API Rest et générer un fichier au format JSON  
Télécharger plus tard via git
- ◇ **Swagger-UI** : pour accéder à la documentation de votre API Rest sous forme d'une page Web et pour tester votre API  
Installer plus tard via Composer

# ENVIRONNEMENT DE DEVELOPPEMENT

---

## Composer


# Composer

- ◇ Composer est un **gestionnaire de dépendances** sous licence libre (GPL v3) utilisé en ligne de commande.
- ◇ Il est écrit en PHP.
- ◇ Si vous souhaitez installer un package, Composer se chargera d'installer les autres paquets dont il dépend.
- ◇ Création du fichier **composer.json** : fichier contenant la liste des bundles du projet, très utile car il suffit juste d'avoir le **composer.json** pour importer tous les bundles d'un projet.

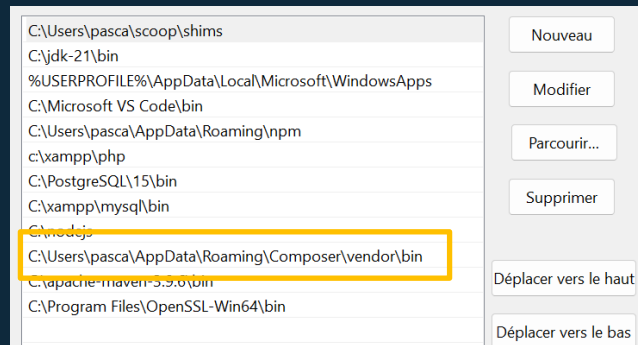
Fichier composer.json : exemple

```
{  
  "require": {  
    "zircote/swagger-php": "^3.0"  
  }  
}
```

# Installation de composer

1. Installez PHP sur votre ordinateur (via XAMPP)
2. Téléchargez la dernière version de Composer.  
 <https://getcomposer.org/Composer-Setup.exe>
3. Lancez l'assistant d'installation de Composer. Lorsqu'il vous demande d'activer le mode développeur, ignorez-le et poursuivez le processus d'installation.
4. Une autre fenêtre s'ouvre et vous demande de localiser la ligne de commande PHP. Par défaut, elle se trouve dans C:/xampp/php/php.exe. Après avoir spécifié le chemin, cliquez sur Suivant.

Note : **Le programme d'installation ajoute automatiquement Composer à la variable d'environnement PATH.** Vous pouvez maintenant ouvrir l'invite de commande et exécuter le logiciel depuis n'importe quel répertoire.



# Utilisation de composer

- ◇ Afin de fonctionner, Composer se base sur 2 fichiers de configuration :
  - ◇ Fichier **composer.json** : contient la liste des principales dépendances nécessaires au fonctionnement du projet et précise toutes les configurations nécessaires à leur installation et à leur utilisation, comme leur numéro de version ou leur répertoire d'installation, par exemple. L'élément principal constituant le fichier composer.json est sa propriété "require", qui liste toutes les dépendances nécessaires ainsi que leur numéro de version.
  - ◇ Fichier **composer.lock** : contient toutes les informations et configurations de chacune des dépendances actuellement installées dans le projet. Composer se base sur ce fichier pour installer ses dépendances, s'il est présent dans l'arborescence du projet. Il permet donc d'assurer une cohérence de version et de configuration entre différentes instances d'un même projet, que ce soit sur des postes de développement différents ou sur différents serveurs.
- ◇ Note : Composer vérifie la compatibilité des versions avec votre projet. Cela signifie que si vous utilisez un ancien paquet, Composer vous le fera savoir afin d'éviter tout problème ultérieur.



# Composer : principales commandes

Mise à jour de toutes les dépendances

```
C:\composer update
```

Ajout d'un bundle

```
C:\composer require bundle1
```

Exemple : ajout du bundle http-foundation

```
C:\composer require symfony/http-foundation
```

Ajout de plusieurs bundles

```
C:\composer req bundle1 bundle2 bundle3
```

# Composer : principales commandes

Supprimer un bundle

```
C:\composer remove nom_bundle
```

Installer les dépendances : lit le fichier composer.json, résout les dépendances et les installe

```
C:\composer install
```

# REST

---

## Présentation

# Historique



- ◇ Issue d'une Thèse de Roy T. Fielding
- ◇ Roy T. Fielding
  - ◇ Un des auteurs de la spécification HTTP
  - ◇ Co-fondateur du projet Apache HTTP Server
  - ◇ Premier président de la fondation Apache
- ◇ Date de la thèse : 2000
- ◇ SOAP (Simple Object Access Protocol) apparaît vers 1998

SOAP est un protocole d'échange d'information structurée dans l'implémentation de services web bâti sur XML. Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur.

# REST



- ◇ REST est l'acronyme de **REpresentational State Transfert**
- ◇ REST est un style d'architecture où **chaque URL est une représentation d'une ressource sur le web.**
- ◇ Un web service REST est une **entité conceptuelle identifiée par une URL**, sollicitée par une requête.
- ◇ **Basée sur les opérations HTTP Standards**
  - ◇ GET pour obtenir sa représentation, POST pour sa création, PUT pour sa modification et DELETE pour sa suppression.
- ◇ Le format de représentation de la ressource est libre (XML, **JSON**, HTML, GIF, JPEG, etc.).

# Caractéristiques



- ◇ Les services Web REST sont **sans états** (Stateless)
  - ◇ Chaque requête envoyée vers le serveur doit contenir toutes les informations à leur traitement
  - ◇ Minimisation des ressources systèmes, pas de session ni d'état
- ◇ Les services Web REST fournissent une **interface uniforme basée sur les méthodes HTTP** : GET, POST, PUT et DELETE
- ◇ Les architectures orientées REST sont **construites à partir de ressources qui sont uniquement identifiées par des URIs** (Uniform Resource Identifier)

# Points forts



- ◇ Simplicité : Pas de norme à apprendre / utiliser.
- ◇ Légèreté : Pas de gros framework.
- ◇ Utilisation de l'infrastructure existante : **Un serveur HTTP suffit.**

# Points forts



- ◇ Les bénéfices de REST sont ceux de HTTP :
  - ◇ Politique de mise en cache des réponses
  - ◇ Utilisation des clients HTTP existants
  - ◇ Système de redirection
  - ◇ Réémission des requêtes
  - ◇ Codes d'erreurs standardisés
  - ◇ Contrairement à SOAP, REST est lié au protocole HTTP
  - ◇ Pas de standard pour décrire l'interface du service



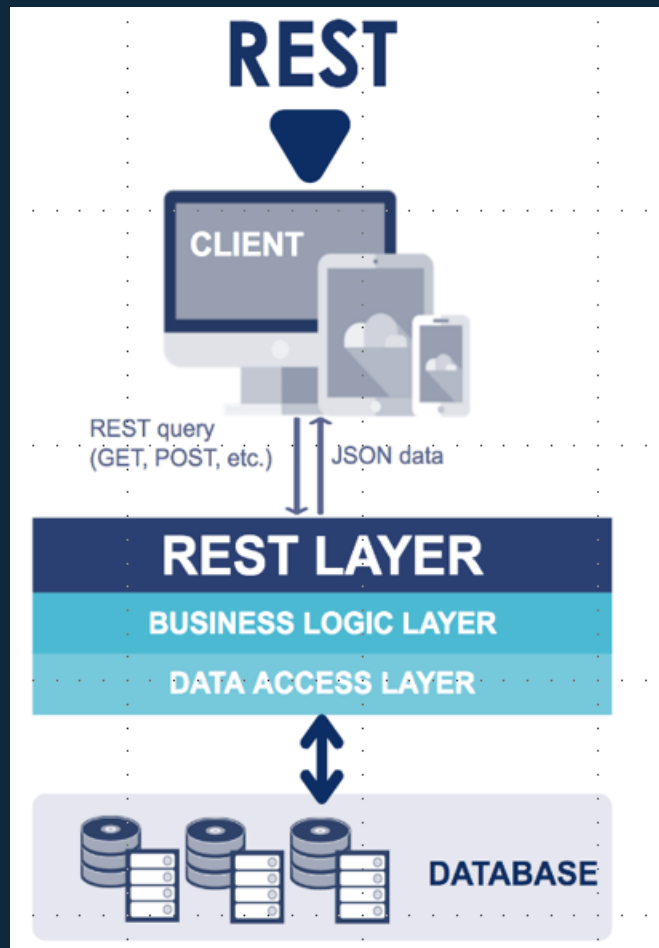
# Notion de ressource



- ◇ Une application expose des ressources
- ◇ Chaque ressource
  - ◇ Est identifiée par une URI (Uniform Resource Identifier)
  - ◇ Est adressable (se trouve à une URI donnée et fixe)
  - ◇ Est manipulable par une interface uniforme
    - ◇ Les verbes HTTP : GET, PUT, DELETE, POST, OPTIONS, HEAD
  - ◇ A un état indépendant des clients
- ◇ L'état d'une ressource peut être :
  - ◇ Téléchargé sous un format appelé « représentation » (xml, json, txt, png, pdf, ...)
  - ◇ Modifié puis retransmis par un client pour être mis à jour

- ◇ Vocabulaire basé sur 3 notions :
  - ◇ Les **méthodes (verbes) HTTP** comme identifiant des opérations
  - ◇ L'**URI** (Uniform Resource Identifier) comme identifiant des ressources
  - ◇ Les **codes de retour** (HTTP Status Code)
- ◇ Les verbes sont définis dans la norme HTTP
  - ◇ OPTIONS – GET – HEAD – POST – PUT – DELETE – TRACE – CONNECT
- ◇ Les plus importants pour REST sont **GET, POST, PUT et DELETE**

# ARCHITECTURE



# REST

---

## Méthodes HTTP

- ◇ La méthode GET signifie "récupérer" le contenu quel qu'il soit de la ressource (sous forme d'une entité) identifiée par l'URL visée.
- ◇ Si l'URL visée identifie un processus générant dynamiquement des données, ce sont les données produites qui sont renvoyées dans l'entité au lieu du source de l'exécutable appelé, sauf si ce texte lui-même est la sortie du processus.
- ◇ La sémantique de la méthode GET introduit une notion conditionnelle si la requête contient le champ d'en-tête If-Modified-Since.
  - ◇ Une méthode GET conditionnelle ne récupère la ressource que si celle-ci a été modifiée depuis la date associée au champ If-Modified-Since.
  - ◇ La méthode GET conditionnelle est utilisée pour réduire la charge du réseau en permettant à des ressources en cache d'être rafraîchies sans multiplier les requêtes ou transférer des données inutiles

- ◇ Utilisée pour indiquer au serveur de soumettre l'entité contenue dans le message à la ressource identifiée par l'URI visée.
- ◇ POST est destinée à fournir un moyen uniforme pour les opérations suivantes :
  - ◇ Annotation de ressources existantes
  - ◇ Envoi d'un message vers une édition en ligne, un groupe de nouvelles, une liste d'adresse, ou listes similaires
  - ◇ Fournir un bloc de données, par exemple, un résultat de formulaire, à un processus de gestion de données
  - ◇ **Ajout d'éléments à une base de données**
- ◇ La résolution d'une méthode POST ne signifie pas forcément la création d'une entité sur le serveur origine, ni une possibilité d'accès futur à ces informations.
- ◇ Le résultat d'un POST n'est pas nécessairement une ressource associable à une URI. Dans ce cas, une réponse de code 200 (OK) ou 204 (pas de contenu) est la réponse la plus appropriée, suivant que la réponse contient ou non une entité pour décrire le résultat.

- ◇ Si une ressource a été créée sur le serveur origine, la réponse sera du type 201 (créé) et contiendra l'entité (de préférence du type "text/html") qui décrira les informations sur la requête et contiendra une référence sur la nouvelle ressource.
- ◇ Un champ Content-Length valide est demandé dans toute requête POST HTTP/1.0. Un serveur HTTP/1.0 répondra par un message 400 (requête incorrecte) s'il ne peut déterminer la longueur du contenu du message.
- ◇ Les applications ne peuvent enregistrer en cache les réponses à des requêtes POST dans la mesure où il n'est absolument pas certain qu'une réponse ultérieure émise dans les mêmes conditions produise le même résultat.

- ◆ **PUT demande au serveur de stocker la ressource sous l'URL fournit.**
- ◆ Sémantique proche de la méthode POST
- ◆ Si le Request-URI référence une ressource déjà existante, la ressource envoyée doit être considérée comme une nouvelle version de la ressource sur le serveur.
- ◆ Si le Request-URI ne fait pas référence une ressource déjà existante et si l'URI peut être créé, le serveur stocke la ressource.
- ◆ Si la ressource est créée, un code retour 201 est retourné (Created)
- ◆ Si une ressource a été modifiée un code 200 (OK) ou 204 (No Content) doit être retourné.
- ◆ Si une ressource ne peut pas être créée avec le Request-URI demandé, une erreur appropriée doit être renvoyée en fonction du problème rencontré.



# DELETE



- ◆ DELETE demande au serveur de supprimer une ressource identifiée par le Request-URI.
- ◆ Le serveur doit indiquer un succès de l'opération sauf si au moment de la réponse, l'opération n'a pas encore été effectuée.
- ◆ Une réponse en cas de succès doit être :
  - ◆ 200 (OK).
  - ◆ 202 (Accepted) si la suppression n'est pas effective.
  - ◆ 204 (No Content) si la réponse ne contient pas de contenu.

- ◇ HEAD permet de tester si un document existe sans le télécharger et de vérifier son type, sa date de modification, ...
- ◇ La sémantique de la méthode HEAD est la même que celle de la méthode GET sauf que seuls les headers sont renvoyés dans la réponse.

- ◇ Ces 4 méthodes permettent d'effectuer les opérations classiques que vous retrouvez dans toutes les applications Web : **CRUD**

Action	REQUETE SQL	METHODE HTTP
Create	Insert	POST
Read	Select	GET
Update	Update	PUT
Delete	Delete	DELETE

# IDEMPOTENCE



- ◇ Le concept de l'idempotence signifie qu'une opération a le même effet qu'on l'applique une ou plusieurs fois, ou encore qu'en la réappliquant on ne modifiera pas le résultat.
- ◇ Une application sur une ressource est dite idempotente si l'effet d'une requête est identique à l'effet d'une série de la même requête

HTTP	IDEMPOTENCE
PUT	Idempotent
GET	Idempotent
POST	Non idempotente (plusieurs créations de la même ressource)
DELETE	Idempotent


Les méthodes PUT et DELETE permettent d'illustrer cette notion d'idempotence. Si un client crée une ressource avec PUT, la création de la même ressource avec PUT ne modifiera pas son état. Pareil pour DELETE, si un client supprime une ressource, les requêtes suivantes de même objectif que la première ne modifient plus l'état du serveur.

# REST

---

## HTTP status codes

# HTTP Status Code

- ◇ Les codes de retour (HTTP Status Code) sont utilisés pour communiquer avec les serveurs.
- ◇ Ces codes sont répartis en 5 grandes classes :
  - ◇ Information 1xx : réponses informatives
  - ◇ Succès 2xx : réponses de succès
  - ◇ Redirection 3xx : messages de redirection
  - ◇ Erreur client 4xx : erreurs du client
  - ◇ Erreur serveur 5xx : erreurs du serveur
- ◇ Ces codes font partie de la spécification HTTP.  
 <https://developer.mozilla.org/fr/docs/Web/HTTP/Status>

# HTTP Status Code

100 => 'Continue'

101 => 'Switching Protocols'

200 => 'OK'

201 => 'Created'

202 => 'Accepted'

203 => 'Non-Authoritative Information'

204 => 'No Content'

205 => 'Reset Content'

206 => 'Partial Content'

300 => 'Multiple Choices'

301 => 'Moved Permanently'

302 => 'Found'

303 => 'See Other'

304 => 'Not Modified'

305 => 'Use Proxy'

306 => '(Unused)'

307 => 'Temporary Redirect'

400 => 'Bad Request'

401 => 'Unauthorized'

402 => 'Payment Required'

403 => 'Forbidden'

404 => 'Not Found'

405 => 'Method Not Allowed'

406 => 'Not Acceptable'

407 => 'Proxy Authentication Required'

408 => 'Request Timeout'

409 => 'Conflict'

410 => 'Gone'

411 => 'Length Required'

412 => 'Precondition Failed'

413 => 'Request Entity Too Large'

414 => 'Request-URI Too Long'

415 => 'Unsupported Media Type'

416 => 'Requested Range Not Satisfiable'

417 => 'Expectation Failed'

500 => 'Internal Server Error'

501 => 'Not Implemented'

502 => 'Bad Gateway'

503 => 'Service Unavailable'

504 => 'Gateway Timeout'

505 => 'HTTP Version Not Supported'

# HTTP Status Code : exemples



- ◇ **Information 1xx** : Les codes de statut de réponse commençant par 1 indiquent une réponse informative du serveur. Ils ne signalent ni une erreur ni un succès, mais délivrent simplement une information au client.
- ◇ **100 (Continue)** : indique au client que tout est OK. Ce code l'invite à poursuivre sa requête envers le serveur ou bien à quitter si celle-ci est terminée.
- ◇ **101 (Switching Protocol)** : Il constitue une réponse à la requête Upgrade du client. Il sert à lui indiquer par quel protocole le serveur passe pour récupérer les données.
- ◇ **103 (Processing)** : informe le client du fait que le serveur a bien reçu la requête et qu'il est en train de la traiter. Il s'affiche lorsqu'aucune réponse de succès ou d'échec n'est encore disponible.



# HTTP Status Code : exemples

{REST}

- ◆ **Succès 2xx** : Les codes de statut de réponse commençant par 2 indiquent que la requête du client a été correctement transmise, interprétée, et exécutée.
- ◆ **200 (OK)**
  - ◆ La requête a abouti. L'information retournée en réponse dépend de la requête émise :
  - ◆ GET : Une entité correspondant à l'URI visée par la requête est renvoyée au client
  - ◆ HEAD : La réponse au client ne doit contenir que les champs d'en-tête à l'exclusion de tout corps d'entité
  - ◆ POST : Une entité décrivant le résultat de l'action entreprise
- ◆ **201 (Created)**
  - ◆ La requête a abouti, et une nouvelle ressource réseau en résulte.
  - ◆ La nouvelle ressource créée est accessible par l'URI(s) renvoyée dans l'entité de la réponse.
  - ◆ Si la création ne peut pas être opérée immédiatement, le serveur devra indiquer dans la réponse quand la ressource deviendra disponible ou retourner un message de type 202 (acceptée).

# HTTP Status Code : exemples

- ◇ **Erreur client 4xx** : Les codes HTTP débutant par 4 servent à indiquer une erreur du côté du client
  - ◇ **400 (Bad Request)** : indique au client une mauvaise syntaxe dans la requête. 401 informe l'utilisateur de l'obligation de s'identifier auprès du serveur pour accéder aux ressources. Le code 403 est assez proche de 401 : le client est dans ce cas identifié, mais n'a pas les droits d'accès au contenu en question.
  - ◇ **404 (Not Found)** : fait partie des plus courants, notamment sur les sites web. Dans ce cas, le serveur n'a pas trouvé la ressource demandée par l'utilisateur.
  - ◇ **405 (Method Not allowed)** : utilisé par le serveur pour indiquer que la méthode utilisée par le client pour formuler la requête n'est pas autorisée. Dans ce cas, le serveur connaît la méthode utilisée, mais n'est pas en mesure de l'accepter.

# HTTP Status Code : exemples



- ◇ **Erreur serveur 5xx** : Les codes de statut de réponse HTTP 500 servent à informer d'une erreur du côté du serveur.
- ◇ **500 (Internal Error)** : prévient d'une erreur interne. Le serveur n'est pas en mesure de traiter la situation qu'il rencontre.
- ◇ **501 (Not implemented)** : utilisé quand le serveur ne supporte pas la méthode utilisée pour exprimer la requête. Il faut dans ce cas privilégier les méthodes GET et HEAD.
- ◇ **503 (Service Unavailable)** : informe le client que le serveur n'est pas prêt pour traiter sa requête. Il se peut que le serveur soit surchargé ou tout simplement éteint.

# REST

---

## Exemples de code

# Exemples de code

## ◇ Headers

```
// Headers requis
// Accès depuis n'importe quel site ou appareil (*)
header("Access-Control-Allow-Origin: *");

// Format des données envoyées = JSON
header("Content-Type: application/json; charset=UTF-8");

// Méthode autorisée = GET
header("Access-Control-Allow-Methods: GET");

// Durée de vie de la requête
header("Access-Control-Max-Age: 3600");

// Entêtes autorisées
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
```

# Exemples de code

- ◇ Vérifier que la méthode utilisée est correcte : exemple GET

```
// Méthode autorisée = GET
header("Access-Control-Allow-Methods: GET");

// On vérifie que la méthode utilisée est correcte
if ($_SERVER['REQUEST_METHOD'] == 'GET'){
...
}
else
{
...
}
```

# Exemples de code

- ◇ Méthodes

- ◇ Récupérer les données au format JSON

```
$donnees = json_decode(file_get_contents("php://input"));
```

- ◇ Envoyer un status code

```
http_response_code(code);
```

- ◇ Envoyer un message : On encode en JSON et on envoie le message

```
echo json_encode(["message" => "Produit a été créé"]);
```

- ◇ Envoyer un tableau : On encode en JSON et on envoie le tableau

```
echo json_encode($tableauProduits);
```

# Exemples de code

## ◇ Exemple de Méthode POST - Create

```
<?php
require_once '../Autoloader.php';
use modele\entites\Produit as Produit;
use modele\dao\ProduitDao as ProduitDao;
// Headers requis
header("Access-Control-Allow-Origin: *"); // Accès depuis n'importe quel site ou appareil (*)
header("Content-Type: application/json; charset=UTF-8"); // Format des données envoyées = JSON
header("Access-Control-Allow-Methods: POST"); // Méthode autorisée = POST
header("Access-Control-Max-Age: 3600"); // Durée de vie de la requête
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With"); // Entêtes autorisées
// SI la méthode est autorisée
if($_SERVER['REQUEST_METHOD'] == 'POST'){
    // On instancie un produit
    // On instancie la dao produit
    // On récupère les données envoyées au format JSON
    $donnees = json_decode(file_get_contents("php://input"));
    // On hydrate notre objet avec les données reçues
    // SI la création est ok
        // Status code = 201 : created
        http_response_code(201);
        // On encode en json et on envoie le message
        echo json_encode(["message" => "Produit a été créé"]);
    // SINON création NON ok
        // Status code = 503 : Service Unavailable
        http_response_code(503);
        // On encode en json et on envoie le message
        echo json_encode(["message" => "Impossible de traiter la requête"]);
    // SINON méthode non autorisée
        // Status code = 405 : Method Not Allowed
        http_response_code(405);
        // On encode en json et on envoie le message
        echo json_encode(["message" => "Méthode non autorisée"]);
    // SIFIN
}
```



# REST

---

## Outils pour tester l'API

# Pour consommer des services...



- ◇ Des outils pour tester des services REST :

- ◇ CURL :

 <https://curl.se/>



- ◇ RESTClient (plugin Firefox) :

 <https://addons.mozilla.org/fr/firefox/addon/restclient/>

- ◇ SOAPUI

 [www.soapui.org](http://www.soapui.org)



- ◇ Postman REST Client

 <https://www.postman.com/product/rest-client/>



- ◇ Swagger-UI

- ◇ Hoppscotch

 <https://hoppscotch.io/>

# EVALUATION

---

A VOUS DE JOUER...



# A vous de jouer...



PHP - REST - TP1.pdf

Objectif 1

Objectif 2

Objectif 3