bisgambiglia@univ-corse.fr

24/09/2025

Cours de programmation web

PAUL-ANTOINE BISGAMBIGLIA

2019-2020

HTTP://PAUL-ANTOINE-BISGAMBIGLIA.UNIV-CORSE.FR/

@PABISGAMBIGLIA

Licence

- Cours en licence libre
- Certaines images ont été récupérées sur google image avec le filtre « réutilisation utilisée sans but commercial »
- Certains exemples de code sont issus de sites web référencés en fin de chapitre via leurs urls

Objectifs

- Approfondir les notions de programmation web
- Apprendre à créer un site internet riche

- Technologies utilisées
  - HTML5
  - CSS3
  - JavaScript
  - PHP
  - MySQL
  - Des frameworks

Prérequis

Master 1

- Cours de L3
  - Algorithmique et programmation
  - Notion de réseau
    - Communication client serveur
    - Protocole HTTP
  - Techo web : HTML/CSS/JS/PHP

Licence 3

- Notion d'algorithmique et de programmation
- Notion de réseau
  - Communication client serveur
  - Protocole HTTP

Programme

- Introduction et révisions
  - Le web, et ses techno
- HTML 5
  - Langage de balise *<body> </body>*
- CSS 3 avec bootstrap
  - Mise en forme
  - Ergonomie du web
- JavaScript avec jQuery
  - Dynamisme coté client
  - Angular.js
  - Node.js
- PHP
  - Dynamisme coté serveur
  - Les frameworks (Zend, Symfony, cakephp)
- Le +
  - les Design Patterns pour le web
  - Sécuriser son code
  - Django le framework web en python
  - Framework html 5 pour mobile (IONIC, Mobile Angular UI, Intel XDK, Appcelerator Titanium, Sencha Touch)
  - Apache Cordova, Meteor

PHP

1. PHP ET POO
2. PATTERNS
3. WEB SERVICES
4. FRAMEWORKS
    1. phacon (framework php / C)
    2. Laravel
5. OAUTH PHP OAUTH GOOGLE FACEBOOK TWITTER

PHP 7

- http://blog.teamtreehouse.com/5-new-features-php-7
- https://www.tutorialspoint.com/php7/php7_introduction.htm

PHP et POO

- *Les modifications entre la v5 et la v7 sont présentées ici : http://php.net/manual/fr/migration70.php*
- *Il n'y a pas trop de modification sur la partie objet*

POO en PHP

PHP 5 et 7

PHP et POO

- En PHP 5, il y a un nouveau model objet. La gestion des objets a été complètement réécrite, permettant de meilleurs performances ainsi que plus de fonctionnalités.
- *Chaque définition de classe commence par le mot-clé **class** , suivi par le nom de la classe, qui peut être quelconque à condition que ce ne soit pas un mot réservé du PHP. Suivent une paire d'accolade contenant la définition des membres et des méthodes. Une pseudo-variable **$this** est disponible lorsqu'une méthode est appelée depuis un contexte objet.*

PHP 5

- **Définition simple d'une classe :**

```
<?php
class SimpleClass
{
   // déclaration d'un membre
   public $var = 'une valeur par défaut';
   // déclaration de la méthode
   public function displayVar()
   {
     echo $this->var;
   }
}
?>
```

PHP

- **Le mot clé new :**
  - Pour créer une instance d'un objet, un nouvel objet doit être créé et assigné à une variable. Un objet doit toujours être assigné lors de la création d'un nouvel objet à moins qu'un l'objet ait un constructeur défini qui lance un exception en cas d'erreur. Création d'une instance :

```
<?php
$instance = new SimpleClass();
?>
```

PHP

- **Le mot clé extends**

*Une classe peut **hériter** des méthodes et des membres d'une autre classe en utilisant le mot clé **extends** dans la déclaration. Il n'est pas possible d'étendre de multiples classes, une classe peut uniquement hériter d'une seule classe de base.*

PHP

- *Les méthodes et membres hérités peuvent être **surchargés**, à moins que la classe parent ait défini une méthode comme **final**. Pour surcharger, il suffit de redéclarer la méthode avec le même nom que celui défini dans la classe parent. Il est possible d'accéder à une méthode ou un membre surchargé avec l'opérateur **parent::***

PHP

```php
<?php
class SimpleClass
{
   // déclaration d'un membre
   public $var = 'une valeur par défaut';
   // déclaration de la méthode
   public function displayVar() {
     echo $this->var;
   }
}
```

```php
// extension de la classe
class ExtendClass extends SimpleClass
{
  // Redéfinition de la méthode parent
  function displayVar()
  {
    echo "Classe étendue\n";
    parent::displayVar();
  }
}
$extended = new ExtendClass();
$extended->displayVar();
?>
```

PHP

- **Constructeurs** : void __construct (mixed args , ...)
  - Les classes qui possèdent une méthode constructeur appellent cette méthode à chaque création d'une nouvelle instance de l'objet.
- **Exemple** :

```
<?php
class BaseClass {
   function __construct() {
      print "In BaseClass constructor\n";
   }
}
class SubClass extends BaseClass {
   function __construct() {
      parent::__construct();
      print "In SubClass constructor\n";
   }
}
$obj = new BaseClass();
$obj = new SubClass();
?>
```

PHP

- **Destructeurs : void __destruct ( )**
  - PHP 5 introduit un concept de destructeur similaire aux autres langages orientés objet, comme le C++ . La méthode destructeur doit être appelée aussitôt que toutes les références à un objet particulier sont effacées ou lorsque l'objet est explicitement détruit. Exemple avec un Destructeur

```
<?php
class MyDestructableClass {
   function __construct() {
      print "In constructor\n";
      $this->name = "MyDestructableClass";
   }
   function __destruct() {
      print "Destruction de " . $this->name .
    "\n";
  } } $obj = new MyDestructableClass(); ?>
```

PHP

- **Visibilité :** la visibilité d'une propriété ou d'une méthode peut être définie en préfixant la déclaration avec un mot-clé : **public** , **protected** ou **private** .
- Les éléments déclarés publics (public) peuvent être utilisés par n'importe quelle partie du programme.
- L'accès aux éléments protégés (protected) est limité aux classes et parents hérités.
- L'accès aux éléments privés (private) est uniquement réservé à la classe qui les a définis.

```php
<?php
/**
 * Définition de MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this-> public;
        echo $this->protected;
        echo $this->private;
    }
}
```

PHP

- **L'opérateur de résolution de portée (::)**
  - Il fournit un moyen d'accéder aux membres statiques ou constants ainsi qu'aux éléments redéfinis par la classe.
  - Lorsque vous référencez ces éléments en dehors de la définition de la classe, utilisez le nom de la classe.

```
<?php
class MyClass {
    const CONST_VALUE = 'Une valeur constante';
}

echo MyClass::CONST_VALUE;
?>
```

PHP

```php
<?php
class MyClass {
    const CONST_VALUE = 'Une valeur constante';
}

echo MyClass::CONST_VALUE;
?>
```

```php
:: depuis la définition de la classe
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'variable statique';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}
OtherClass::doubleColon();
?>
```

PHP

- **Statique :**
  - Le fait de déclarer des membres ou des méthodes comme statiques vous permet d'y accéder sans avoir besoin d'instancier la classe. Un membre déclaré comme statique ne peut être accédé avec l'objet instancié d'une classe (bien qu'une méthode statique le peut).
  - La déclaration **static** doit être faite après la déclaration de visibilité. Pour des raisons de compatibilité avec PHP 4, si aucune déclaration de visibilité n'est utilisée, alors le membre ou la méthode sera traité comme s'il avait été déclaré comme public.
  - Comme les méthodes statiques sont appelables sans instance d'objet créée, la pseudo variable $this n'est pas disponible dans la méthode déclarée en tant que statique.

PHP

Exemple avec un membre statique

```
<?php
class Foo
{
    public static $my_static = 'foo';
    public function staticValue() {
        return self::$my_static;
    }
}
class Bar extends Foo
{
    public function fooStatic() {
        return parent::$my_static;
    }
}

print Foo::$my_static . "\n";
$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";
// propriété my_static non définie
// $foo::my_static n'est pas possible
print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

PHP

- **Constantes de classe :**

Il est possible de définir des valeurs constantes à l'intérieur d'une classe, qui ne seront pas modifiables. Les constantes diffèrent des variables normales du fait qu'on n'utilise pas le symbole $ pour les déclarer ou les utiliser. Tout comme pour les membres statiques , on ne peut pas accéder aux valeurs constantes depuis une instance de l'objet (en utilisant $object::constant).

```
<?php
class MyClass
{
  const constant = 'valeur constante';
  function showConstant() {
    echo  self::constant . "\n";
  }
}
echo MyClass::constant . "\n";
$class = new MyClass();
$class->showConstant();
// echo $class::constant;  n'est pas autorisé
?>
```

```
<?php
abstract class AbstractClass
...
```


- **Abstraction de classes**
  - PHP 5 introduit les classes et les méthodes abstraites. Il n'est pas autorisé de créer une instance d'une classe définie comme abstraite. Toutes les classes contenant au moins une méthode abstraite doivent également être abstraites. Pour définir une méthode abstraite, il faut simplement déclarer la signature de la méthode et ne fournir aucune implémentation.
  - Lors de l'héritage depuis une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parent doivent être définies par l'enfant ; de plus, ces méthodes doivent être définies avec la même (ou plus faible) visibilité . Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie en tant que protégée ou publique.

PHP

- **Interfaces**
  - Les interfaces objet vous permettent de créer du code qui spécifie quelles méthodes et variables une classe peut implémenter, sans avoir à définir comment ces méthodes seront gérées.
  - Les interfaces sont définies en utilisant le mot clé interface , de la même façon qu'une classe standard mais sans aucun contenu de méthode.
  - Toutes les méthodes déclarées dans une interface doivent être publiques.
- **implements**
  - Pour implémenter une interface, l'opérateur implements est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise. Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule.

PHP

```php
<?php
// Declaration de l'interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
```

```php
// Implémentation de l'interface
// Ceci va fonctionner
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }
        return $template;
    }
}
```

PHP

- **Surcharge**
  - Les appels de méthodes et l'accès aux membres peuvent être surchargés via les méthodes __call , __get et __set . Ces méthodes ne seront déclenchées que si votre objet, hérité ou non, ne contient pas le membre ou la méthode auquel vous tentez d'accéder. Toutes les méthodes surchargées doivent être définies en tant que public .
  - Depuis PHP 5.1.0, il est également possible de surcharger les fonctions isset et unset via, respectivement, les méthodes __isset et __unset.

- **Mot clé 'final'**
  - PHP 5 introduit le mot-clé " final " qui empêche les classes filles de surcharger une méthode en en préfixant la définition par le mot-clé " final ". Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.

PHP

- **Parcours d'objets**
  - PHP 5 fournit une façon de définir les objets de manière à ce qu'on puisse parcourir une liste de membres avec une structure foreach . Par défaut, toutes les propriétés visibles seront utilisées pour le parcours.

```
$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
```

```
<?php
class MyIterator implements Iterator
```

  - Comme nous le montre l'affichage, ***l'itération foreach*** affiche toutes les variables visibles disponibles. Pour aller plus loin, vous pouvez implémenter l' interface interne de PHP 5 nommée Iterator . Ceci permet de déterminer comment l'objet doit être parcouru.

PHP

- **Méthodes magiques**

Les noms de fonction __construct , __destruct, __call , __get , __set , __isset , __unset, __sleep , __wakeup , __toString , __set_state , __clone et __autoload sont magiques dans les classes PHP.

Vous ne pouvez pas utiliser ces noms de fonction dans aucune de vos classes sauf si vous voulez modifier le comportement associé à ces fonctions magiques.

PHP

- Le but de **__sleep** est de clore toutes les connexions aux bases de données que l'objet peut avoir, valider les données en attente ou effectuer des tâches de nettoyage.
- Le but de **__wakeup** est de rétablir toute connexion base de données qui aurait été perdue durant la linéarisation et d'effectuer des tâches de réinitialisation.
- La méthode **__toString** détermine comment la classe doit réagir lorsqu'elle est convertie en chaîne de caractères

PHP

- **Auto-chargement de classes**
  - De nombreux développeurs qui créent des applications orientées objet, créent un fichier source par définition de classe. L'inconvénient majeur de cette méthode est d'avoir à écrire une longue liste d'inclusions de fichier classes au début de chaque script : une inclusion par classe.
  - En PHP 5, ce n'est plus nécessaire. Vous pouvez définir la fonction **__autoload** qui va automatiquement être appelée si une classe n'est pas encore définie au moment de son utilisation. Grâce à elle, vous avez une dernière chance pour inclure une définition de classe, avant que PHP ne déclare une erreur.

PHP

- Class MaClasse [extends SuperClass]

```
function MaClasse(parametre1, parametre2) {
    this.attributPublic = parametre1;
    var attributPrive = parametre2;
    this.methodePublique = function() { }
    var methodePrivee = function() { }
}
```

- Instanciation
  - var objetDeMaClasse = new MaClasse("valeur1", "valeur2");
- Appel des méthodes
  - $obj = new Classe(); //Instanciation de classe
  - $obj->methodeInstance(); //Application de méthode

PHP

```
class Livre extends Produit {
    private $auteur;
    private $editeur;
    private $nb_pages;
    function _construct($auteur,
        $editeur, $nb_pages) {
            $this->auteur = $auteur;
            $this->editeur = $editeur;
            $this->nb_pages =
                $nb_pages; }
    function
        changer_nb_pages($nb){…}
    }
$livre = new Livre(…);
$livre->changer_nb_pages(150) ;
```

PHP

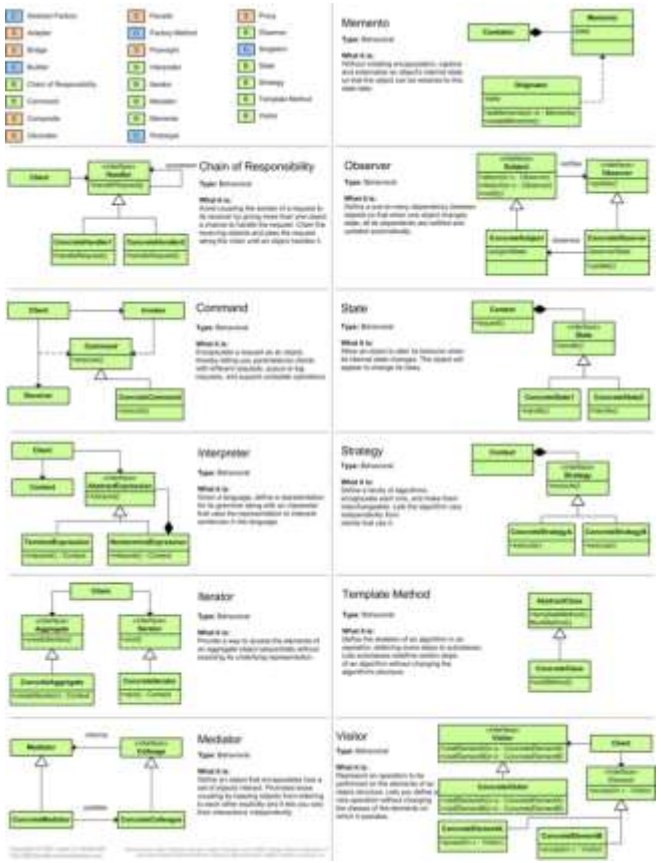
- **Patterns**

Les patterns sont un moyen de décrire les meilleures pratiques et les bonnes conceptions. Ils proposent une solution flexible aux problèmes habituels de programmation.

PHP

- **MVC** (design pattern) http://bpesquet.developpez.com/tutoriels/php/evoluer-architecture-mvc/
- **Singleton** (classe qui sera instanciée qu'une seule) https://www.grafikart.fr/formations/programmation-objet-php/singleton
- **Factory** (https://openclassrooms.com/courses/programmez-en-oriente-objet-en-php/les-design-patterns et http://ir2.php.net/manual/fr/language.oop5.patterns.php)
- **Observer**
- **Strategy**

Factory



- Factory est avec le Singleton l'un des Design Patterns les plus aisé à comprendre et à mettre en œuvre.
- **Pour** fournir un objet prêt à l'emploi, configuré correctement, en libérant le code client de toute responsabilité (choix de l'implémentation, configuration, instanciation, …).

http://www.croes.org/gerald/blog/la-fabrique-factory-en-php/24/

Factory

classes dont la finalité est de créer d'autres objets

- A utiliser :

Vous avez plusieurs objets respectant la même *interface* ou héritant de la même *classe abstraite*, mais chacun adapté à un contexte différent (par exemple une interface IDB avec une classe par base de données DBMySql, DBOci, DBPgSql, …)

Vous voulez centraliser le code en charge du choix de l'objet à créer

Le pattern factory permet l'instanciation d'objets durant l'exécution. Il est appelé "pattern d'usine" puisqu'il est responsable de la "fabrication" d'un objet. Une méthode d'usine reçoit le nom de la classe pour l'instancier en tant qu'argument.

http://www.croes.org/gerald/blog/la-fabrique-factory-en-php/24/

http://ir2.php.net/manual/fr/language.oop5.patterns.php

Singleton

- **Objectif** : s'assurer que seule et une seule instance d'une classe soit utilisée.
- **Solution :** rendre le constructeur de la classe privé puis créer une méthode statique dont la charge est de toujours fournir le même objet.
- Le pattern singleton est souvent implémenté pour des classes de bases de données, des journaliseurs, des objets de requête, de réponse ou encore des contrôleurs frontaux.

http://www.croes.org/gerald/blog/le-singleton-en-php/17/

http://ir2.php.net/manual/fr/language.oop5.patterns.php

MVC

- Un design pattern ou modèle de conception répond à des problèmes d'ordre architectural au niveau des développement informatiques.

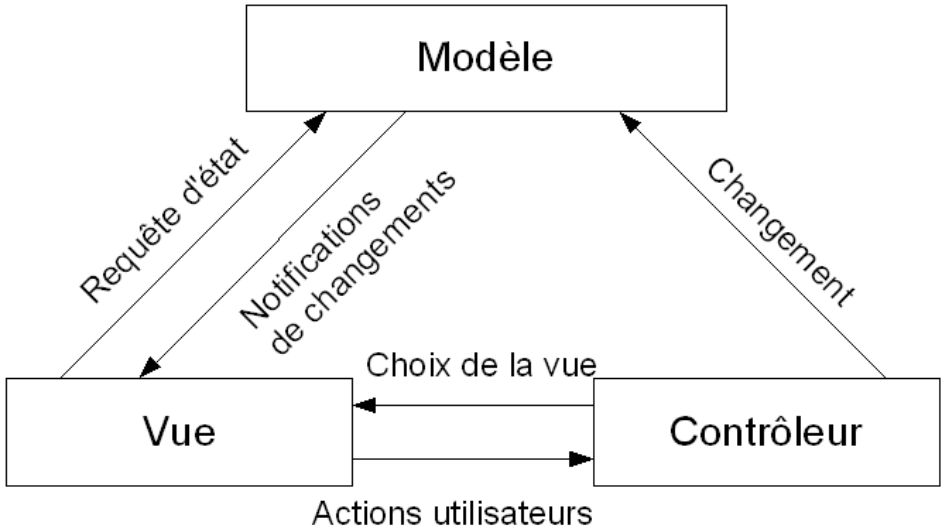
- Un Design pattern est intimement lié à la POO. Il consiste en un ensemble de règle de bonne pratique visant à améliorer l'organisation du projet en privilégiant la réutilisabilité du code. Une telle organisation permet de catalyser les temps de développements, et améliore considérablement la maintenance de l'application

MVC

- Le pattern Modèle-Vue-Contrôleur organise l'interface Homme-machine d'une application logicielle en
  - un modèle (objet métier, modèle de données),
  - une vue (présentation, interface utilisateur) et
  - un contrôleur (logique de contrôle, gestion des événements, traitement),

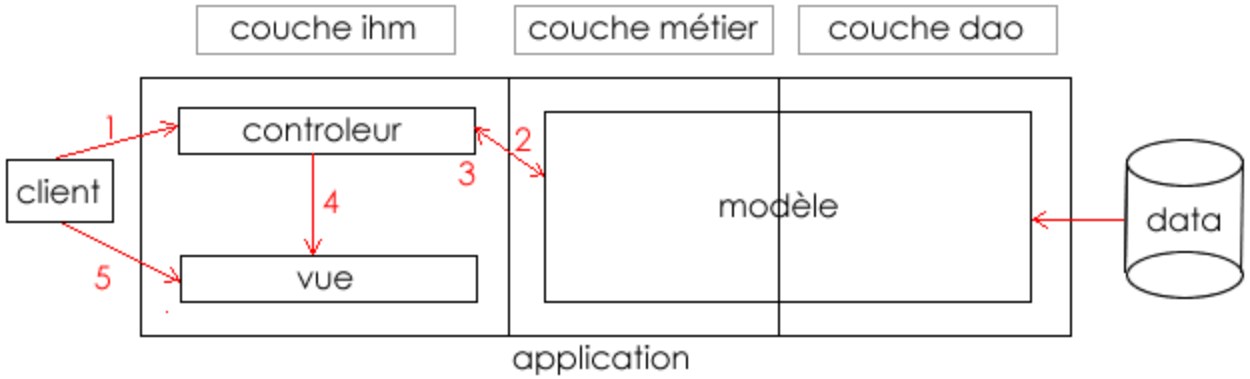
MVC

1. la requête est analysée par le contrôleur
2. le contrôleur demande au modèle approprié d'effectuer les traitements
3. le contrôleur renvoie la vue adaptée.



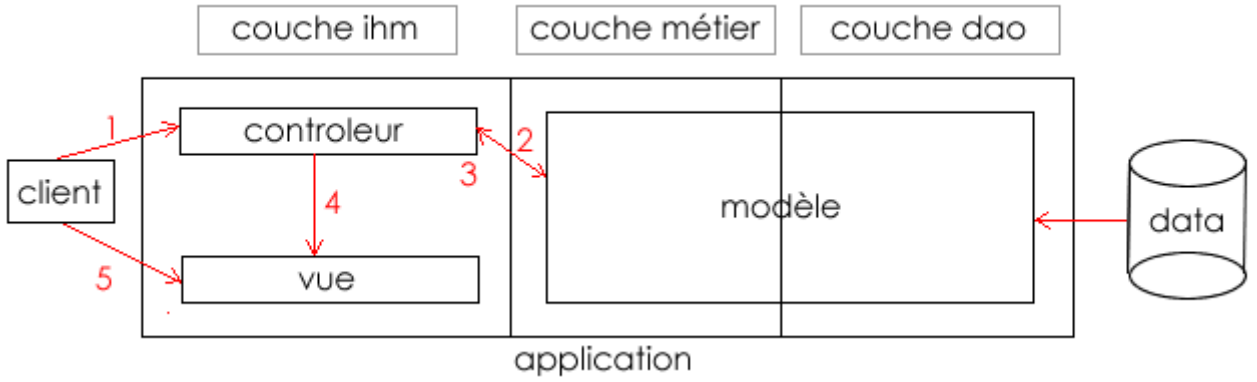
MVC

- MVC est bien adaptée à des applications web écrites avec des langages orientés objet, elles tirent ainsi le meilleur parti du PHP5 ou 7.

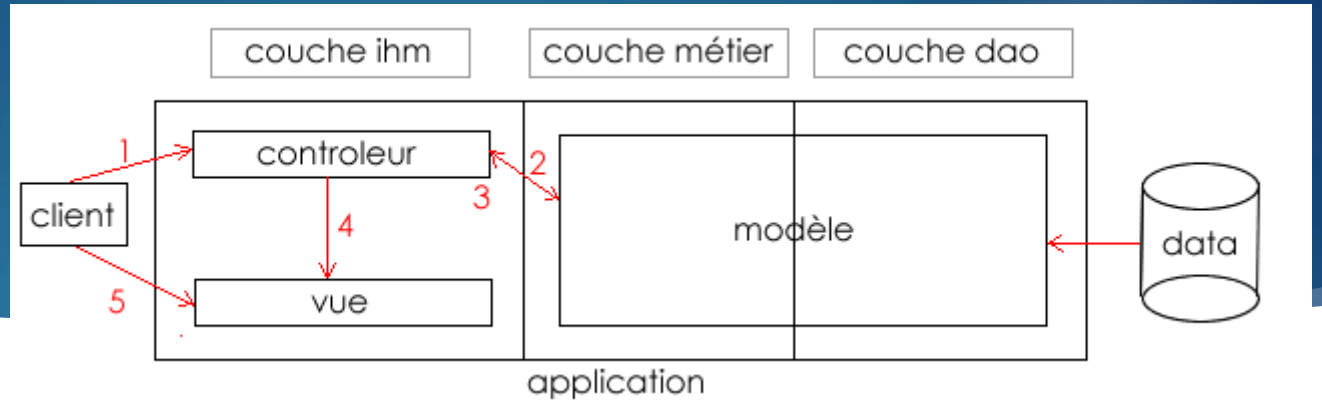


MVC

- couche ihm: c'est l'interface utilisateur encore appelé interface homme machine
- couche métier : c'est le coeur de l'application où réside les objets traités par l'application
- couche dao : couche d'accès aux données (data access object). Cette couche permet une indépendance de la logique métier et du stockage des données associées



MVC



1. le client fait une demande au contrôleur. Ce contrôleur voit passer toutes les demandes des clients
2. le contrôleur traite la demande. Pour ce faire, il peut avoir besoin de l'aide de la couche métier, et éventuellement de la couche dao.
3. le contrôleur reçoit une réponse de la couche métier et effectue les actions demandées par l'utilisateur
4. le contrôleur sélectionne et nourrit une vue pour présenter les résultats de l'action qui vient d'être effectuée
5. la vue est enfin envoyée au client

MVC

- +++
  - http://profgra.org/lycee/activite_intro_MVC_PHP.html

les patterns ne sont pas des solutions figées, que les patterns sont un outils de réflexion…

Source : http://www.croes.org/gerald/blog/la-strategie-strategy-en-php/93/

Web services

Webservice PHP

- Un Service WEB est un ensemble de protocoles permettant à des applications de communiquer entre elles, et ce indépendamment de la structure et du langage employés.
- C'est un programme sur Internet qui permet la communication et l'échange de données entre applications. Ce sont des fonctionnalités, des services mis à disposition sur Internet (ou Intranet) et accessibles par tout le monde.
- C'est interopérable, c'est basée sur HTTP (donc pas de soucis au niveau des parefeux), on utilise des standards et des protocoles ouverts.

http://vivien-brissat.developpez.com/tutoriels/php/soap/

Webservice PHP

- Architecture orientée service
- L'objectif :
  - Chercher (annuaire)
  - Publier (fournisseur de service)
  - Consommer (application)
- ***Service ?***

http://mbaron.developpez.com/cours/soa/intro/

Webservice PHP

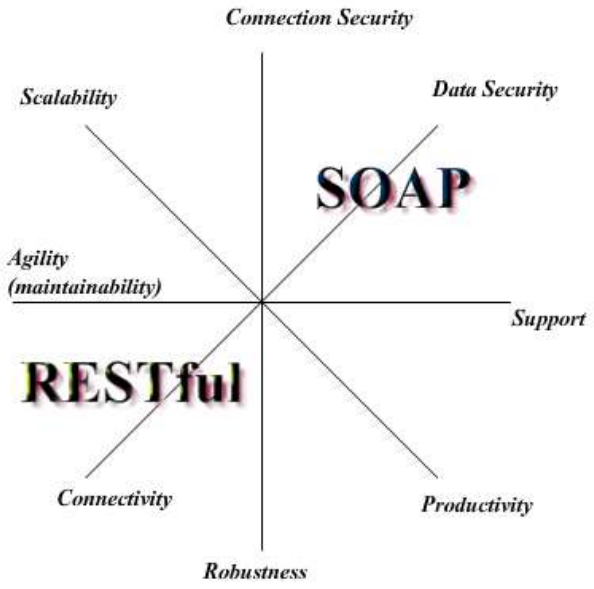
- **WSDL**

Web Services Description Language est un langage de description standard. C'est l'interface présentée aux utilisateurs. Il indique comment utiliser le service Web et comment interagir avec lui. WSDL est basé sur XML et permet de décrire de façon précise les détails concernant le service Web tels que les protocoles, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.

https://openclassrooms.com/courses/les-services-web

Webservice PHP

SOAP



- **SOAP**

Simple object Access Protocol est un protocole standard de communication. C'est l'épine dorsale du système d'interopérabilité. SOAP est un protocole décrit en XML et standardisé par le W3C. Il se présente comme une enveloppe pouvant être signée et pouvant contenir des données ou des pièces jointes. Il circule sur le protocole HTTP et permet d'effectuer des appels de méthodes à distance.

https://openclassrooms.com/courses/les-services-web

Webservice REST

- REST est un style d'architecture défini dans la thèse de Roy Fielding dans les années 2000, ce n'est donc ni un protocole ni un format. Les implémentations sont donc multiples et différentes. Cependant, on retrouve souvent le principe dans les API HTTP, comme c'est le cas de GitHub et Twitter par exemple.
- **Les contraintes**
  - Le serveur et le client sont indépendants. L'interface utilisateur est situé côté client (une application mobile par exemple) et le stockage est située côté serveur (une base de données).
  - Contrairement à l'accès web classique, aucune variable de session ou autre état volatile ne doit être enregistré côté serveur. Chaque requête vers le serveur est donc indépendante.
  - Mise en cache : le serveur indique au client s'il peut mettre en cache les données qu'il reçoit. Cela permet d'éviter des requêtes inutiles et ainsi préserver la bande passante.
  - Une interface uniforme.
    - On accède à chaque ressource de façon unique. Il n'y a qu'une seule façon d'y accéder.
    - Les ressources sont manipulées via des représentations définies, elles sont accompagnées de données permettant sa compréhension (vous comprendrez mieux dans l'exemple).
    - Auto-description. L'encodage, par exemple, est précisé de façon claire ainsi le client peut comprendre le document et appeler le service correspondant à cet encodage.
    - Hypermédia comme moteur d'application (HATEOAS) : la ressource indique comment accéder aux états suivants (suppression, édition etc…), le client peut ainsi connaître quelles actions sont possibles sur la ressource en l'obtenant
  - Hiérarchie par couche : les ressources sont individuelles. On peut imaginer que nos ressources embarquent des ressources provenant de serveurs distants ou de mise en cache par des serveurs intermédiaires.
  - (Facultatif) Exécution de scripts côté client obtenus par le serveur. Cela permet de rendre le client plus léger et plus générique.

https://zestedesavoir.com/tutoriels/299/la-theorie-rest-restful-et-hateoas/

Webservice REST

Prise de RDV (client)

```
POST/cabinetMedecinHTTP/1.1
Content-Type:application/json
#en-têtes HTTP...
  {"prendreRDV":
      {"date":"2015-10-21",
      "medecin_id":"doc"}}
```

Créneaux dispo (serveur)

```
HTTP/1.1200 OK
Content-Type:application/json
...
{"creneaux":
   [{"debut":"14h00","fin":"14h50",

"medecin":{"id":"doc","autre_propriete":"v
aleur"}},

    {"debut":"16h00","fin":"16h50",

"medecin":{"id":"doc","autre_propriete":"v
aleur"}
}
]
}
```

Choix d'horaire (client)

```
POST/cabinetMedecinHTTP/1.1
Content-Type:application/json
...
{
"demandeRDV":
{"creneau":
    {"medecin_id":"doc",
     "debut":"14h00",
      "fin":"14h50"},
       "patient":{"id":"marty"}
}
}
```

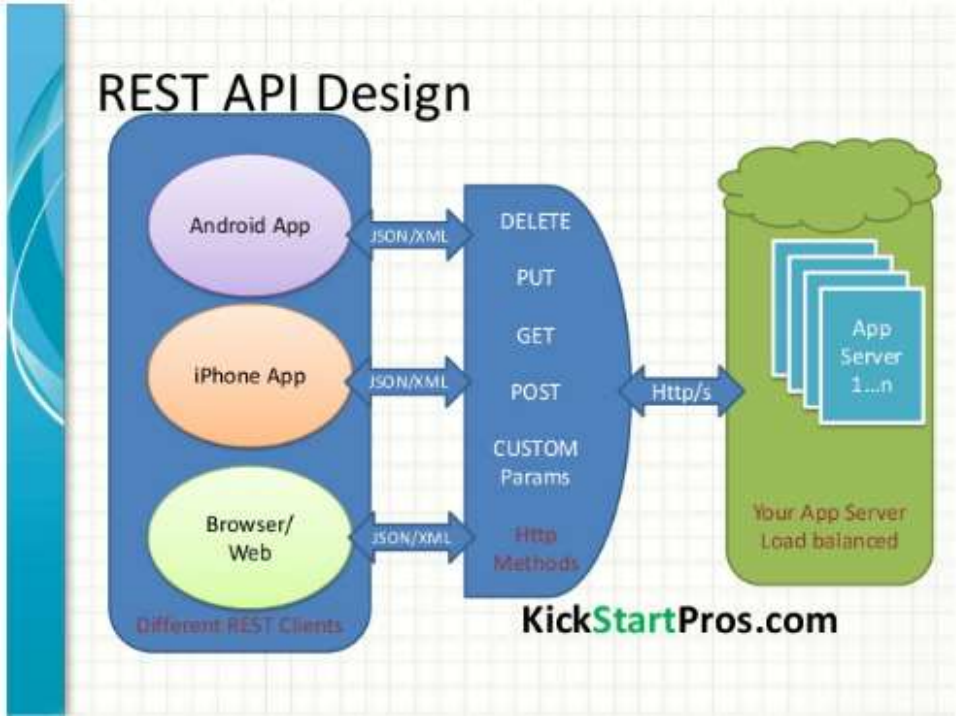
Validation (serveur)

```
HTTP/1.1200OKContent-Type:application/json
...
{
"RDV":
{"creneau":...,"patient":
...}}
```

https://zestedesavoir.com/tutoriels/299/la-theorie-rest-restful-et-hateoas/

API RESTful

- Une API est RESTful quand elle respecte le principe d'architecture REST. Ce principe d'architecture s'applique aux services Web. La particularité principale de cette architecture est que la partie serveur (l'API) et la partie client communiquent sans que le client ne connaisse la structure et le contenu des informations stockées sur le serveur.
- une API regroupe un ensemble de fonctions ou méthodes, leurs signatures et ordre d'usage pour obtenir un résultat.



https://www.journaldunet.fr/web-tech/developpement/1203011-qu-est-ce-que-restful/

https://www.supinfo.com/articles/single/5642-qu-est-ce-qu-une-api-rest-restful

API RESTful

- Les applications REST s'appuient sur les verbes fournis par le protocole HTTP :
  - POST
  - GET
  - PUT
  - DELETE
  - PATCH
- Ce sont des mots-clés qui définissent l'action que l'on souhaite effectuer sur une ressource.
- Le média peut être par exemple un fichier JSON ou XML.

https://www.journaldunet.fr/web-tech/developpement/1203011-qu-est-ce-que-restful/

API RESTful

- Pour commencer la communication, le client va appeler l'URL racine de l'API, /. Requête :
- `GET / Accept: application/json`
- Le serveur va alors lui retourner comme réponse différentes possibilités d'interactions
- Le client saura alors qu'il existe deux interactions possibles

```
200 OK
Content-Type: application/json
{
version: 1.0,
liens: [
{ href: /utilisateur, rel: lister,
methode: GET },
{
href: /utilisateur,
rel: creer,
methode: POST } ] }
```

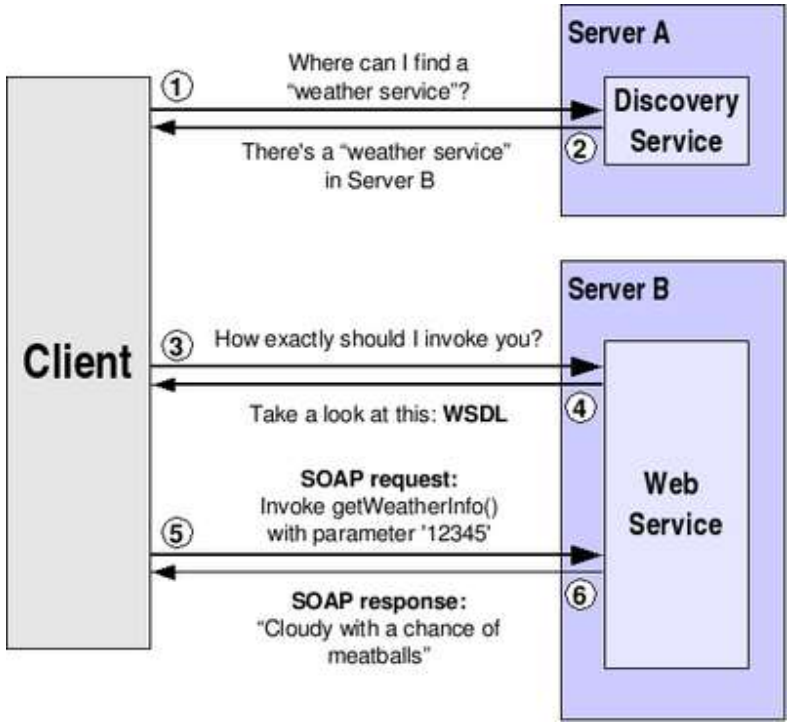
https://www.journaldunet.fr/web-tech/developpement/1203011-qu-est-ce-que-restful/

API RESTful

```
GET /user
Accept: application/json
Réponse : {
    users: [
    { id: 1, nom: Dupont, prenom: Jean, liens:
        [ { href: /utilisateur/1, rel: afficher, methode: GET },
    { href: /utilisateur/1, rel: modifier, methode: PUT },
    { href: /utilisateur/1, rel: supprimer, methode: DELETE }
] }
] }
```

https://www.journaldunet.fr/web-tech/developpement/1203011-qu-est-ce-que-restful/

Webservice PHP



En 1, Il va demandé à un annuaire de webservices (UDDI) où il peut trouver un tel service.

- En 2, l'annuaire lui indique où est le service qui pourra le renseigner.

- En 3, le client demande au service comment l'invoquer (c'est-à-dire comment bien lui demander un service).

- En 4, le service renvoit sa déscription (WSDL) où est indiqué comment invoquer ce service.

- En 5, le client invoque le service en envoyant une requête SOAP.

- En 6, le service renvoit la réponse au client dans une réponse (toujours en SOAP).

Webservice PHP

- Exemple de webservice simple avec deux méthodes :
  - *hello()* et retournera « Hello world! ».
  - *donne()*, prendra une variable en attribut et la retournera.

```
class Donne {
  function donne($i)   {
    return $i;   }
  function hello()   {
    return "hello world";   }
}
```

Webservice PHP

L'URI qui représente l'espace de nom, est différente de la Location du service qui est l'URL à interroger

- Création d'un objet **SoapServer**
- arguments un fichier WSDL (*null* ici) et une liste de paramètres
- classe qui gère les requêtes SOAP par la méthode*setClass()*.
- La méthode *handle()* fait le nécessaire pour gérer une requête SOAP.

```
Try {
  $server = new SoapServer(null, array('uri'
=> 'http://monserveur/ws/mon_premier_
webservice.php'));
  $server->setClass("Donne");
  $server->handle();
}
catch(Exception $e) {
  echo "Exception: " . $e;
}
```

Webservice PHP

Client

```php
<?php
    try {
      $clientSOAP = new SoapClient( null,
        array (
          'uri' => 'http://monserveur/ws/mon_premier_webservice.php',
          'location' => 'http://monserveur/ws/mon_premier_webservice.php',
          'trace' => 1,
          'exceptions' => 0
      ));
      $ret = $clientSOAP->__call('hello', array());
      echo $ret;
      echo '<br />';
      $ret = $clientSOAP->__call('donne', array('i'=>5));
      echo $ret;
    }
    catch(SoapFault $f) {
      echo $f;
    }
    ?>
```

Webservice PHP

- Exemples à tester :
  - https://benjion.wordpress.com/2013/04/22/php-web-services-comment-creer-un-service-web-php-avec-soap/
  - http://vivien-brissat.developpez.com/tutoriels/php/soap/
  - http://www.willdurand.fr/decouverte-des-webservices-en-php/

RESTFULL + PHP

- **Exemple simple**
  - https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php
- Doc fichier .htaccess
  - https://www.mauricelargeron.com/parametrer-les-acces-a-son-serveur/
- Autre exemple
  - https://www.pulsar-informatique.com/actus-blog/entry/mise-en-place-api-rest-en-php

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

api.php

```php
<?php
header("Content-Type:application/json");
require "data.php";
if(!empty($_GET['name'])) {
    $name=$_GET['name'];
    $price = get_price($name);
    if(empty($price)) {
        response(200,"Product Not Found",NULL);
    }
    else {
        response(200,"Product Found",$price);
    }
}
else {
    response(400,"Invalid Request",NULL);
}

function response($status,$status_message,$data) {
    header("HTTP/1.1 ".$status);
    $response['status']=$status;
    $response['status_message']=$status_message;
    $response['data']=$data;
    $json_response = json_encode($response);
    echo $json_response;
}
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

```php
<?php
    function get_price($name) {
        $products = [ "book"=>20,
                      "pen"=>10,
                      "pencil"=>5 ];

        foreach($products as $product=>$price) {
            if($product==$name) {
                return $price; break;
            }
        }
}
```

data.php

Tester :
http://localhost/webservice/api.php?name=pen
Modification du **.htaccess**
RewriteEngine On # Turn on the rewriting engine
RewriteRule ^api/([0-9a-zA-Z_-]*)$ api.php?name=$1 [NC,L]
Tester => http://localhost/projectname/api/pen

Sous windows ajouter
Options - MultiViews

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

utilisation

```
<!DOCTYPE html>
<html lang="en"><head><title>Rest API Client Side Demo</title>
<meta charset="utf-8"> <meta name="viewport" content="width=device-width, initial-
scale=1"><link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"> <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script> <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script> </head>
<body><div class="container"> <h2>Rest API Client Side Demo</h2>
<form class="form-inline" action="" method="POST">
<div class="form-group"><label for="name">Name</label>
<input type="text" name="name" class="form-control" placeholder="Enter Product Name"
required/></div>
<button type="submit" name="submit" class="btn btn-default">Submit</button> </form>
<p> </p> <h3>
<?php
if(isset($_POST['submit'])) {
    $name = $_POST['name'];
    $url = "http://localhost/webservice/api/".$name;
    $client = curl_init($url);
    curl_setopt($client,CURLOPT_RETURNTRANSFER,true);
    $response = curl_exec($client);
    $result = json_decode($response);
    echo $result->data; }
?>
</h3> </div> </body> </html>
```

https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php

RESTFULL + PHP

- **Exemple de service REST avec PHP**
  - https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html
  - implémentation d'un service REST qui exposera les données de la table de contact appelée CARNET
  - Un contact sera accessible à partir de la route du type: /api/v1/contact/12 qui permettra aux clients de récupérer le contact en JSON employant la méthode HTTP GET dans la version 1 de notre API.
  - Dans cet exemple, le contact constitue la *ressource* manipulée dans notre API.
  - La méthode GET sera employée pour récupérer des éléments *individuellement* ou par *Collections*.

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

RESTFULL + PHP

Méthode	Action réalisée	URI
GET	Récup. tous les liens	/api/v1/
GET	Récuperation un Element	/api/v1/contact/{id}
GET	Récupération Collection	/api/v1/contact
POST	Creation d'Elements	/api/v1/contact
DELETE	Effacer Element	/api/v1/contact/{id}
PUT	Modifier un Element	/api/v1/contact/{id}
PATCH	Modif. partielle d'Elt.	/api/v1/contact/{id}

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

RESTFULL + PHP

- La route /api/v1/ en GET renverra la liste des URLs des contacts plutôt que la liste de tous les contacts avec tous leurs détails. Ceci permet d'avoir un serveur REST auto-documenté où la récupération d'une première URL permet en suivant d'obtenir la liste des ressources présentes sur le service avec leurs URLs respectives.
- On pourra également paginer les réponses pour ne pas manipuler trop de données simultanément.
- Pour assurer le routage simplement nous allons continuer avec Silex (à télécharger framework php et à installer avec composer)
- Nous pouvons donc modifier le fichier index.php (fichier de Silex) déjà mis en place comme suit:

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

- SILEX https://silex.symfony.com/download
- Tuto https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/mvc2.html#
- Composer https://getcomposer.org/download/

- Créer un projet SILEX
  - composer require silex/silex "~2.0"
  - Silex/silex/src/Silex/api

index.php

```
<?php
require_once __DIR__.'/vendor/autoload.php’;
 require_once 'modele.php’;
$app = new Silex\Application();
$app['debug']=true;
$app->get('/contact', function () {
    $content ='<ul>’;
    $amis=get_all_friends();
    foreach ($amis as $ami){
        $content.='<li>'.$ami['NOM'].'</li>’;
    }
    $content.='</ul>’;
    return $content;
});
$app->get('/api/', function () {
    $amis=get_all_friends_links();
    return json_encode($amis); });
$app->get('/api/contact', function () {
    $amis=get_all_friends();
    return json_encode($amis); });
?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

model.php

```php
<?php
function get_all_friends_links() {
    $connexion=connect_db();
    $amis=Array();
    $sql="SELECT * from CARNET";
    $data=$connexion->query($sql);
    while($pers=$data->fetch(PDO::FETCH_ASSOC)) {
        $res=Array();
        $res['NOM'] = $pers['NOM'];
        $res['URL']=
            $_SERVER["REQUEST_SCHEME"].'://'.
            $_SERVER['HTTP_HOST'].
            $_SERVER['CONTEXT_PREFIX'].
            '/silex/api/contact/'.$pers['ID'];
        $amis[] = $res;
    }
    return $amis;
}
?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

A ajouter a index.php

```php
<?php
$app->get('/api/contact/{id}', function($id) use ($app) {
    $ami = get_friend_by_id($id);
    if (!$ami) $app->abort(404, "Contact inexistant");
    else return json_encode($ami,JSON_PRETTY_PRINT);
}); ?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

À ajouter à model.php

```php
<?php
function get_friend_by_id($id) {
    $connexion=connect_db();
    $sql="SELECT * from CARNET where ID=:id";
    $stmt=$connexion->prepare($sql);
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);
    $stmt->execute();
    return $stmt->fetch(PDO::FETCH_OBJ);
}

?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

A ajouter a index.php

```php
<?php
$app->delete('/api/contact/{id}', function($id) use ($app) { $ami =
get_friend_by_id($id);
if (!$ami)
    $app->abort(404, "Contact inexistant");
else {
    delete_friend_by_id($id);
    return json_encode($ami,JSON_PRETTY_PRINT);
} });
?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

À ajouter à model.php

```php
<?php
    function delete_friend_by_id($id) {
        $connexion=connect_db();
        $sql="Delete from CARNET where ID=:id";
        $stmt=$connexion->prepare($sql);
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_OBJ);
    }
?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

```
<?php
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$app->before(function (Request $request) {
    if (0 === strpos($request->headers->get('Content-Type'),
'application/json')) {
        $data = json_decode($request->getContent(), true);
        $request->request->replace(is_array($data) ? $data : array());
}
});
$app->post('/api/contact', function (Request $request) use ($app) { $data =
$request->request->all();
add_friends($data);
return new Response(json_encode($data), 200, array('Content-Type' =>
'application/json'));
});
?>
```

A ajouter a index.php

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

À ajouter à model.php

```
<?php
function add_friends($data) {
    $connexion=connect_db();
    $sql="INSERT INTO CARNET(NOM,PRENOM,NAISSANCE,VILLE) values (?,?,?,?)";
    $stmt=$connexion->prepare($sql);
    return $stmt->execute(array($data['NOM'], $data['PRENOM'],
$data['NAISSANCE'],$data['VILLE']));
}
?>
```

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/rest2.html

How to use PHP to build microservice?

https://itnext.io/how-to-use-php-to-implement-microservice-94957206abc6

Les microservices sont un style architectural qui structure une application sous la forme d'un ensemble de services :

- Hautement maintenable et testable
- Faiblement couplé
- Déploiement indépendant
- Organisé autour des capacités métier
- Administré par une petite équipe

https://microservices.io/patterns/microservices.html

L'architecture des microservices permet la livraison rapide, fréquente et fiable de grandes applications complexes.

Cela permet également à une organisation de faire évoluer sa pile technologique.