

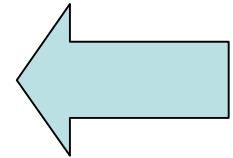
Module MI3

PHP5 et architecture MVC
pour les applications web

2016-2017

Plan du cours

1. Rappels sur la Programmation Web
2. Présentation de PHP 5
3. Vers une plus grande Structuration du Code
4. PHP 5 et l'accès aux Bases de Données
5. **Principe d'une architecture MVC dans les applications web**



Module MI3

Principe d'une architecture MVC dans les applications Web

Usage d'un modèle : conclusion

- Une partie de la complexité de l'application passe dans le modèle
- Plus facile de faire évoluer séparément le modèle et l'interface d'interaction
 - Ex: passer d'une persistance fichier à une BD (tp2)

Mais...

- Toujours un peu de mélange entre le design (HTML, affichage) et l'appel au modèle (PHP)
- Le pré-calcul a des limites (exemple du déplacement)

Solution : séparer le « contrôle » des « vues »

- Matérialiser l'action à réaliser par une variable
- Supprimer les pré-calculs
- Limiter au maximum l'usage de PHP dans le design
 - Un designer n'est pas un programmeur !
 - Changer facilement de design

On pense naturellement à...

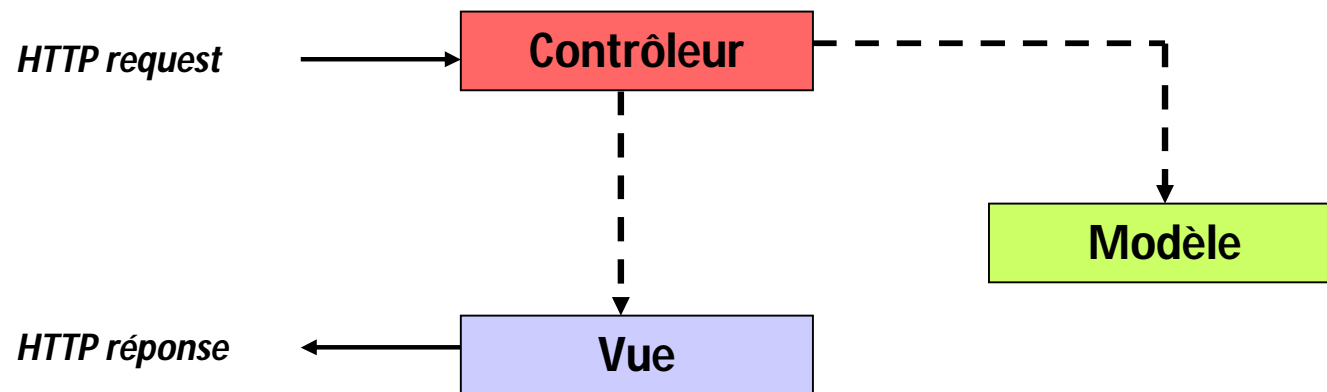
- ➔ Une partie du code pour gérer les demandes d'actions ➔
Le contrôleur
- ➔ Une partie du code pour gérer l'affichage ➔ La vue

Le motif MVC : **M**odèle-**V**ue-**C**ontrôleur

- Le but du motif MVC: bien identifier et séparer les couches d'une application (3 couches distinctes, au minimum)
 - ⇒ trois parties fondamentales dans la conception d'une application : le **modèle**, la **vue** et le **contrôleur**
 - ⇒ sépare les données, l'interaction et la présentation
- Remarques:
 1. tend à devenir un standard dans le développement d'applications web
 2. une « bonne » pratique de conception qui permet de séparer chaque rôle de l'application, de produire du code clair, évolutif, de programmer à plusieurs,...
 3. pour une petite application web, la complexité en terme de code, apportée par MVC, ne sera pas justifiée.
 4. de nombreux frameworks tel que Zend Framework, **Symfony**, ou CakePHP propose un motif MVC (le leur!)

Motif MVC

- Motif MVC => comment diviser une application en 3 couches logiques, liées entre elles:
1. Le **modèle** : gestion des données
 2. La **vue** : affichage de l'interface
 3. Le **contrôleur** : gestion des demandes d'actions de l'utilisateur



Le C du motif MVC

- Le **Contrôleur** : c'est le « chef d'orchestre »
 1. Il Analyse la requête HTTP du client et récupère les paramètres
 2. Il appelle le modèle en lui passant les paramètres
 3. Il détermine la vue à afficher et demande son affichage
- Une manière classique de faire :
 - Un contrôleur principal (index.php) qui répartit les demandes de l'utilisateur à d'autres contrôleurs spécialisés

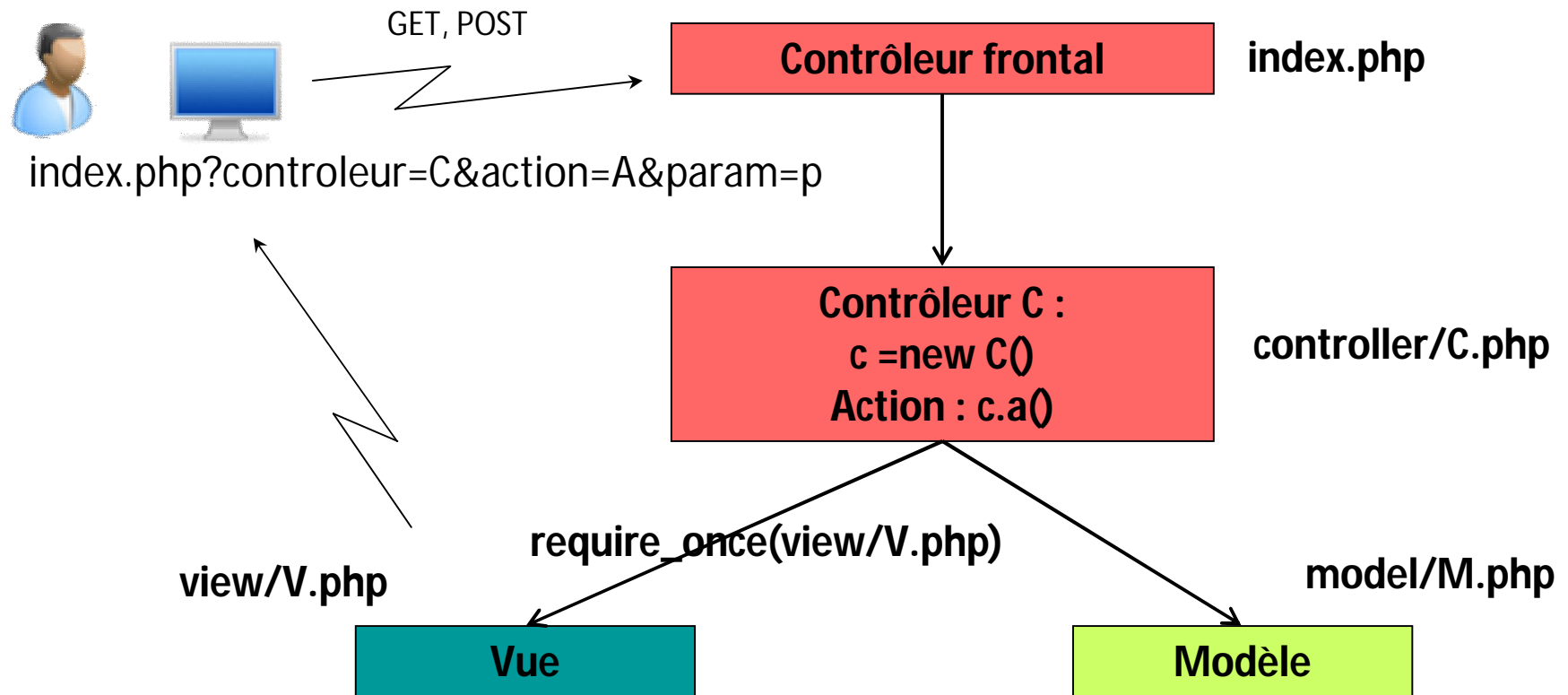
Le **M** du motif MVC

- Le **Modèle** :
 - Il est complètement indépendant du reste; il ne connaît ni le (les) contrôleur(s) ni les vues
 - Le plus souvent la source de données est une base de données
 - Regroupe les classes métier, les classes d'accès aux données
 - Il existe de nombreuses méthodes d'accès aux données

Le V du motif MVC

- La **Vue** :
 - Reçoit les actions de l'utilisateur et les transmet au contrôleur
 - Elle affiche les données passées par le contrôleur (notre cas) ou récupérées directement du modèle

Schéma global du fonctionnement



MVC en PHP et en pratique

- Un contrôleur est une classe
- Il est capable d'actions
 - des méthodes de la classe
- Il utilise le modèle
 - il suffit d'inclure son code (des classes) et de l'utiliser
- Il choisit et déclenche l'interface
 - il suffit d'inclure son code

C'est simple



Contenu du répertoire en MVC

index.php
controller/
 Display.php
view/
 add_news_form.php
 main.php
 icons/
 blog.css
Model/
 blog.php
 news.php
 blog.txt

Point d'entrée

Nouveau

Contrôleurs

Nouveau

Design Interface :
les vues

Modèle & Données

Illustration sur l'application Blog

- Demo Appli blog...
- Le contrôleur frontal index.php
- Le contrôleur display.php (constructeur, getParam, action forward)
- Les variables globales : les paramètres (\$from,\$mode), les données passées à la vue (\$data)
- La vue main.php
- La vue add_news_form.php

Codage MVC en PHP : remarques

- Variables d'état : plus simple d'être globales
- Passage de ce qu'il faut afficher à la vue
 - Un seul tableau \$data global
- Combien de contrôleurs ?
 - Un ensemble d'actions partagent un contrôleur si elles partagent les variables d'état.
 - Un contrôleur représente aussi un état dans le graphe d'interactions.
- Exploiter la modélisation objet
 - Factoriser les codes communs
 - Rendre inaccessible (protected) les codes utilitaires

Codage MVC : conclusion

- Avantages
 - Séparation claire du traitement (modèle), du design (les vues), et du graphe de l'interaction (contrôleur), évolution, maintenance facilitée
 - Constance de cette structure : facile d'entrer dans du code inconnu,
 - Récupérer du code si on utilise un framework
- Inconvénients
 - Abstraction MVC à maîtriser
 - Maîtriser l'objet
 - Semble "lourd" pour des petits projets, moins performant

NB : structure proche de <http://codeigniter.com/>

Créer efficacement des applications Web ?

- Quelle architecture générale d'applications Web ?
 - Pour gagner en structuration, lisibilité du code
 - Pour définir "proprement" les couches de l'application
 - Pour gagner en fiabilité
 - Pour la maintenance de l'application
 - ...

⇒ Modèle MVC

⇒ se créer son propre cadre de travail (framework)

⇒ utiliser un cadre existant : Symfony (module SIL-5),
Zend, CakePHP,...

Mise en pratique

- tp3 : Passer l'application image de la version Modèle à la version MVC **en s'appuyant sur l'exemple de blog**. Il ne s'agit que de réorganiser le code ! (*tp3.pdf* sur l'intranet)
- projet : Développer de nouvelles fonctionnalités en continuant de respecter le principe d'une architecture MVC (enrichissement de la vue, du modèle et du contrôleur) (*projet.pdf* sur l'intranet)