UN EXEMPLE ÉLÉMENTAIRE D'APPLICATION MVC EN PHP

KRISTEN LE LIBOUX

JUILLET 2013

Introduction

Une liste de choses à faire

Deux pages-types :

- Page d’accueil (liste des choses, cliquables)
- Détails d’une chose (nom, à faire avant le...)

Objectif

Concevoir une architecture pour ce site élémentaire

- Générique, réutilisable, extensible
- Avec le modèle de conception MVC :
  Modèle - Vue - Contrôleur
- Et avec URL rewriting

Mockups : liste

TODO.COM

A faire :

- Ranger les courses
- Faire le ménage
- Dormir

http://localhost/todolist

Mockups : détails

TODO.COM

Ranger les courses :
A faire avant le
17/01/2013 11:00:00
Retour

copyright...

~~http://localhost/todolist/item.php?id=1~~

http://localhost/todolist/item/ranger-courses

Rappels sur MVC

Modèle MVC

Principe de conception d'applications
basé sur la séparation de 3 fonctions essentielles
en composants distincts :

- **Modèle** : gérer les données
- **Vue** : gérer l'affichage, l'UI
- **Contrôleur** : agir

Modèle MVC en PHP

Modèle :

- Gère les échanges avec la BDD
- Une classe par entité de la BDD
  (utilisateur, article, catégorie, produit, etc...)
- On peut utiliser un ORM (Doctrine, Propel) pour cela

Modèle MVC en PHP

Vue :

- Affichages HTML (parfois JSON si Ajax)
- On injecte des parties variables (provenant du modèle par exemple) : détails d'un article, d'un produit, etc
- On peut utiliser un moteur de templates (Smarty, Twig)

Modèle MVC en PHP

Contrôleur :

- Implémente les actions
- Autant de classes que nécessaires, regroupées par entités logiques (Articles, Utilisateurs, Produits, Commande, etc...)
- Et aussi pour les erreurs (404, 403...)
- Chaque contrôleur a ses actions (lister, afficher, insérer, éditer, supprimer, etc)
- Chaque contrôleur a ses vues associées

Modèle MVC en PHP

Noyau (*kernel*) : le chef d'orchestre

- Analyse la requête du client
- Instancie le contrôleur correspondant
- Exécute l'action
- Affiche la vue, etc.

Modèle MVC en PHP

Point d'entrée du site (unique, index.php) :

- Premier script appelé, quelle que soit la requête
- Initialise la configuration
- Appelle le noyau

Impose de configurer Apache, URL rewriting

Schéma récapitulatif (1/4)

Supposons que le client (internaute) appelle l'URL :
http://localhost/todolist/item/ranger-courses

- Il s'agit de l'URL de la page
  « détails de l'élément "Ranger les courses" »
- Observons le processus côté serveur.

http://localhost/todolist/item/ranger-courses

client

réécriture

point d'entrée

```
index.php?query=item/ranger-courses
```

noyau

```
Kernel::run()
```

contrôleur et action

```
$cont = new ItemController
                ("ranger-courses");
$cont->display();
```

modèle

```
Item::find("ranger-courses");
```

vue

```
$view = new View("item-details.html");
echo $view->render();
```

Schéma récapitulatif (3/4)

1. Le client requiert une URL http://.../item/ranger-courses
2. Redirection Apache vers index.php?query=item/ranger-courses
3. Exécution de index.php (point d'entrée unique) :
   3.1. Initialisation de la configuration (ROOT, HOME)
   3.2. Inclusion de «kernel.php» (classe statique)
   3.3. Exécution du noyau avec Kernel::run()

Schéma récapitulatif (4/4)

3.3.1 Le noyau analyse la requête (fait appel à un routeur) :

Contrôleur : ItemController
Action : display
Paramètres : slug = "ranger-courses"

3.3.2 Le noyau instancie ce contrôleur, lui passe les paramètres et exécute l'action :

- Instanciation du modèle nécessaire (Item)
- Recherche de l'élément correspondant au slug
- Génération de la vue et envoi au client

Pourquoi une telle complexité ?

Modularité

- Facilite la coopération de différents développeurs
- Les fichiers HTML sont le moins modifiés possible par rapport aux livraisons de l'intégrateur

Fiabilité et maintenabilité

- Chaque classe fait peu de choses, mais le fait parfaitement
- Ou sinon, c'est facile de localiser les erreurs
- Le code est factorisé, jamais dupliqué

Extensibilité

- Ajouter une action ou une fonctionnalité entière est facile

Architecture des répertoires

Répertoires et fichiers à créer

app/

- kernel/
- controller/
- model/
- view/

www/

- .htaccess
- index.php
- css, images, js, etc (aucun ici...)

Classes à créer

app/kernel/

- Kernel
- Controller
- Model
- View
- Router
- Database

app/controller/

- IndexController
- ItemController

app/model/

- Item

Vues à créer

En général on crée une vue par action.

Ici il y a deux contrôleurs avec action «display»

app/view/

- index/display.html
- item/display.html
- error/404.html

Discussion

Extensions possibles

1. Plusieurs modèles de vues : système de thèmes
2. Vues modulables (header, footer, sidebar, etc)
3. Présence d'un back- et d'un front-office :
   Il faut identifier les éléments communs ou spécifiques
4. Notion de session (droits d'accès)
5. Routeur plus élaboré (expressions régulières, extraction automatisée des paramètres)
6. Gestion des formulaires

Télécharger les sources

https://github.com/kleliboux/code-samples

MERCI

@NOVLANGUE SUR TWITTER

COMMENTAIRES, DISCUSSIONS, QUESTIONS