Extensions modulaires - HMVC (une version fork avec un meilleur routage des modules, optimisation de la vitesse)

UPDATE 2018
codeigniter_mx, php 7.2, Nouvelle version Issue du projet InvoicePlane https://invoiceplane.com/ implémentée par BYOOS http://byoos.net
gbobard@gmail.com

Vous pouvez trouver le repo principal ici :
https://bitbucket.org/wiredesignz/codeigniter-modular-extensions-hmvc

C'est un fork via bitbucket commit #868e975 à ce github repo.
Il y a deux raisons pour avoir ceci sur GitHub par opposition à un repo Mercurial sur bitbucket. Le premier était d'avoir un repo sur GitHub que je peux facilement extraire pour mes projets, et le second était d'ajouter le support pour un meilleur routage personnalisé à l'intérieur des modules.
Wiredesignz a fait un travail impressionnant avec l'extension HMVC, cependant, il ne prendrait pas les fichiers de routage personnalisés dans les modules si le premier segment de l'URI ne correspondait pas au nom de répertoire du module. Avec mes modules, je ne veux pas que l'itinéraire commence toujours par le nom du module pour le premier segment. Permettez-moi de vous donner un exemple :
Si l'URI demandée est ceci :
premier-segment/deuxième-segment
Ce qui se passe normalement avec l'implémentation de Wiredesignz, c'est que le routage est d'abord vérifié :
application/config/routes.php
Si rien n'y est trouvé qui correspond à la requête, alors le "premier segment" est utilisé comme nom de module et recherché dans tous les emplacements de votre module pour voir si quelque chose correspond. Dans mon cas, j'ai de nombreux emplacements de modules pour agir en tant que catégories, afin de rendre mes modules plus organisés en fonction de leur but. Dans ce cas, il recherche un module nommé "premier segment" dans chaque emplacement de module. Si aucun nom de module ne correspond à "premier segment" dans tous ces emplacements, alors le contrôleur par défaut ou le contrôleur 404 est chargé ensuite et il se termine ici.
Ce que je voulais, c'était que ces deux premières étapes se déroulent de manière similaire, mais dans le cas où rien n'est trouvé après les deux premières étapes, je voulais que tous les fichiers routes.php soient traités à partir de tous les modules dans tous les emplacements de modules pour voir s'il existe quelque chose qui correspond à l'URI demandé. Si l'une des routes correspond, alors le contrôleur et la méthode du module approprié seraient utilisés, sinon il procéderait normalement au contrôleur par défaut ou au contrôleur 404. En d'autres termes, si vous utilisez cette fourchette et que vous continuez à configurer vos modules et URI de la même manière qu'auparavant, où le premier segment correspond au nom du module, alors la troisième étape ne sera jamais traitée puisqu'elle se trouve à l'étape 2. Pour ce cas, la seule fois que la troisième méthode serait traitée est si quelqu'un vient à votre site Web à une route qui n'existe pas car il va essayer de chercher à travers tout à ce point en analysant tous les fichiers de routage.
Il y aura un succès négligeable à peformance si vous utilisez des routes qui ne commencent pas avec le nom de votre module puisque plus d'emplacements sont recherchés et plus de fichiers routes.php sont chargés, cependant le coût est petit et pour moi vaut la peine de garder tout ce qui est contenu dans les modules si vous aimez garder le routage personnalisé de module là pour le rendre plus facile à utiliser dans d'autres applications. Il suffit de le déposer dans un emplacement de module et c'est tout, sans s'inquiéter d'ajouter des routes si le nom du module ne correspond pas à la route que vous voulez utiliser pour charger un contrôleur. J'ai fait quelques tests avant et après en utilisant la version originale par rapport à cette version fourchue et cela a pris environ 0,005 secondes de plus avec la version fourchue. Le succès de la performance semble donc très faible si c'est important pour vous.
Cependant, avec les changements ci-dessus qui diminuent légèrement la performance, nous avons aussi fait de nombreux autres changements qui accélèrent la performance. L'extension originale de wiredesignz était inefficace lorsqu'il

s'agissait de localiser les modules et les contrôleurs, car pour chaque module qu'il essayait de trouver, il cherchait à nouveau chaque emplacement de module pour chaque module afin de voir ce qu'il pouvait trouver. Ceci est redondant puisque les mêmes chemins exacts ont déjà été recherchés auparavant si vous aviez plus d'un module. Cette version cartographie en fait tous les emplacements de chaque module et met ensuite en cache le résultat de sorte que lorsqu'il est nécessaire de charger le contrôleur, la bibliothèque, etc. d'un module, il n'a pas besoin de chercher partout, car il sait déjà exactement où chercher. Donc, si vous avez un certain nombre de modules, ces changements devraient aider à compenser certains des coûts de performance d'avoir ces fonctionnalités supplémentaires.
Enfin, avant par défaut, l'emplacement de votre module sera automatiquement considéré comme une route si le nom du contrôleur correspond au nom du module, sauf indication contraire. Je n'aimais pas cela car je pouvais facilement voir l'oubli de rendre un module inaccessible et ainsi créer un accès non désiré via une URL directe si seulement d'autres contrôleurs devaient avoir accès à un module. Par exemple, si vous avez quelque chose à votre module/yourcontroller/yourmethod où votre contrôleur a le même nom que votre module, alors cet URI aussi bien que votre module ou votre module/votre module fonctionnerait aussi à moins que vous n'ajoutiez quelque chose comme ceci contenu dans votre fichier module/config/routes.php :

```
/*
|--------------------------------------------------------------------------
Supprimer la route par défaut définie par les extensions de module.
|--------------------------------------------------------------------------
|
Normalement, par défaut, cette route est acceptée :
|
module/contrôleur/méthode
|
Si vous ne voulez pas autoriser l'accès par cet itinéraire, vous devez ajouter :
|
$route['module'] = "";
$route['module/(:any)'] = "";
|
*/
route['yourmodule'] = "" ;
route['yourmodule/(:any)'] = "";

/*
|--------------------------------------------------------------------------
Itinéraires à accepter
|--------------------------------------------------------------------------
|
Cartographier ici tous vos itinéraires de modules valides tels que :
|
Votre itinéraire['your/custom/route'] = "controller/method" ;
|
*/
route['good-route'] = "yourcontroller/yourmethod" ;
```

La version originale devrait avoir votre module au début de l'itinéraire pour que le fichier routes.php soit lu.
route['yourmodule/good-route'] = "yourcontroller/yourmethod" ;
Donc, c'est le travail autour, malheureusement, j'ai tendance à négliger les choses et pourrait facilement oublier de configurer cette règle au début pour supprimer la route par défaut pour le module et ses contrôleurs. Ainsi, avec cette fourchette par défaut, toutes ces routes sont supprimées et seules les routes que vous spécifiez fonctionneront. Vous pouvez changer ce comportement si vous le souhaitez en éditant le fichier Router.php et en changeant :
protected static $remove_default_routes_default_routes = TRUE ;
à

protected static $remove_default_routes_default_routes = FALSE ;

Derniers mots sur ce fork
Gardez à l'esprit que de nouveaux bugs peuvent être introduits à partir de cette fourchette. Si vous trouvez des bogues ou si vous avez des correctifs, vous aimeriez avoir de vos nouvelles pour qu'ils puissent être corrigés. Utilisez-le à vos propres risques et avec beaucoup de tests.
Mise à jour 6 août 2013
Les fichiers ont été mis à jour pour prendre en charge les fichiers d'application qui prennent le pas sur les fichiers du module. Pour les remplacer, ils doivent être placés dans les répertoires appropriés avec des noms identiques.
Le support a également été ajouté pour être strict avec les barres obliques. Je suis en particulier sur la façon dont mes URLs sont formés, et je veux que les pages régulières ou les points finaux fonctionnent de la même manière que CodeIgniter sans barre oblique. Toutefois, les URL qui représentent réellement un répertoire ou ont des URL supplémentaires sous lui, je veux avoir une barre oblique finale pour indiquer que c'est un parent d'autres ressources. À titre d'exemple, je préférerais ce qui suit :
http://www.example.com/
http://www.example.com/page
http://www.example.com/directory/
http://www.example.com/directory/page
http://www.example.com/pagewithextensions.html
Codeigniter par défaut ne vous permet pas de choisir une structure comme celle-ci. Il acceptera simplement les deux façons, ce qui est possible de provoquer des duplications de contenu.
Bien qu'il s'agisse d'un changement subtil, de nombreux changements ont été nécessaires pour qu'un site Web fonctionne strictement de cette façon. Si un répertoire est chargé sans la barre oblique, le résultat sera une page 404 à moins que vous ne fournissiez des routes pour honorer les deux situations. Le but est de pouvoir vous faire spécifier dans votre configuration de routage si l'URL doit ou non avoir une barre oblique finale, de cette façon vous avez la liberté d'utiliser vos préférences, et d'être strict en même temps en ne permettant que ce que vous spécifiez quand il s'agit de barres obliques. Si vous voulez qu'une barre oblique de suivi soit autorisée, ajoutez simplement la barre oblique de suivi directement dans le fichier de routage :
route['mydirectory/'] = "directory_controller/index" ;
Ceci n'autorisera qu'une URL comme la suivante :
http://www.example.com/mydirectory/
Si vous avez besoin à la fois de la version slash trailing et de la version slash non-trailing pour fonctionner, spécifiez simplement les deux :
route['mydirectory/'] = "directory_controller/index" ;
route['mydirectory'] = "directory_controller/index" ;
Si vous voulez seulement utiliser les URI standards tels que générés par CodeIgniter, ces changements ne devraient pas vous affecter puisque CodeIgniter supprime toutes les barres obliques pour les routes et liens par défaut. Ces changements vous obligeront toujours à formater correctement vos URLs pour créer des liens vers les pages appropriées, que ce soit avec ou sans barre oblique.
Voici la documentation originale fournie par wiredesignz pour référence :
Soutenir le développement d'extensions modulaires - HMVC

Extensions modulaires - HMVC
Modular Extensions rend le framework PHP CodeIgniter modulaire. Les modules sont des groupes de composants indépendants, généralement le modèle, le contrôleur et la vue, disposés dans un sous-répertoire de modules d'application qui peuvent être déposés dans d'autres applications CodeIgniter.
HMVC est l'abréviation de Hierarchical Model View Controller.
Les contrôleurs de module peuvent être utilisés comme des contrôleurs normaux ou des contrôleurs HMVC et ils peuvent être utilisés comme des widgets pour vous aider à construire des partiels de vue.
Caractéristiques :
Tous les contrôleurs peuvent contenir une variable de classe $autoload, qui contient un tableau d'éléments à charger avant d'exécuter le constructeur. Ceci

peut être utilisé avec module/config/autoload.php, mais l'utilisation de la variable $autoload ne fonctionne que pour ce contrôleur spécifique.

```
::::php
<?php
classe Xyz étend MX_Controller MX_Controller
{
    autoload = tableau(
        helpers' => array('url','form'),
        libraries' => array('email'),
    ) ;
}
```

Le tableau Modules::$locations peut être défini dans le fichier application/config.php. ie :

```
::::php
<?php
$config['modules_locations'] = array(
    APPPATH.'modules/' => '.../modules/',
) ;
```

Modules::run() la sortie est mise en mémoire tampon, donc toute donnée retournée ou sortie directement par le contrôleur est capturée et renvoyée à l'appelant.
En particulier, $this->load->view() peut être utilisé comme vous le feriez dans un contrôleur normal, sans avoir besoin de retour.
Les contrôleurs peuvent être chargés comme variables de classe d'autres contrôleurs en utilisant $this->load->module('module/controller') ; ou simplement $this->load->module->module('module') ; si le nom du contrôleur correspond au nom du module.
Tout contrôleur de module chargé peut alors être utilisé comme une bibliothèque, c'est-à-dire : $this->controller->method(), mais il a accès à ses propres modèles et bibliothèques indépendamment de l'appelant.
Tous les contrôleurs de modules sont accessibles à partir de l'URL via module/contrôleur/méthode ou simplement module/méthode si les noms de module et de contrôleur correspondent. Si vous ajoutez la méthode _remap() à vos contrôleurs, vous pouvez empêcher tout accès indésirable à partir de l'URL et rediriger ou signaler une erreur comme vous le souhaitez.
Notes :
Pour utiliser la fonctionnalité HMVC, comme Modules::run(), les contrôleurs doivent étendre la classe MX_Controller.
Pour utiliser uniquement la séparation modulaire, sans HMVC, les contrôleurs étendront la classe CodeIgniter Controller.
Vous devez utiliser des constructeurs de style PHP5 dans vos contrôleurs, c'est à dire :

```
::::php
<?php
classe Xyz étend MX_Controller MX_Controller
{
    fonction __construct()
    {
        parent::__construct() ;
    }
}
```

Les constructeurs ne sont pas nécessaires à moins que vous n'ayez besoin de charger ou de traiter quelque chose lorsque le contrôleur est créé pour la première fois.
Toutes les bibliothèques d'extension MY_ doivent inclure (exiger) leur fichier de bibliothèque MX équivalent et étendre leur classe MX_ équivalente.
Chaque module peut contenir un fichier config/routes.php dans lequel le routage et un contrôleur par défaut peuvent être définis pour ce module en utilisant :

```
::::php
<?php
route['nom_du_module'] ='nom_du_contrôleur' ;
```

Les contrôleurs peuvent être chargés à partir des sous-répertoires de l'application/contrôleurs.

Les contrôleurs peuvent également être chargés à partir des sous-répertoires module/contrôleurs.
Les ressources peuvent être chargées transversalement entre les modules. ie : $this->load->load->model('module/model') ;
Modules::run() est conçu pour retourner les partiels de vue, et il retournera la sortie tamponnée (une vue) d'un contrôleur. La syntaxe pour l'utilisation des modules::run est une chaîne segmentée de style URI et des variables illimitées.
::::php
<?php
Les noms de module et de contrôleur sont différents, vous devez également inclure le nom de la méthode, y compris'index' **//.
modules::run('module/controller/method', $params, $params, $....) ;

les noms des modules et des contrôleurs sont identiques, mais la méthode n'est pas'index' **//.
modules::run('module/method', $params, $params, $....) ;

Les noms des modules et des contrôleurs sont identiques et la méthode est'index' **//.
modules::run('module', $params, $params, $....) ;

Les paramètres sont optionnels, vous pouvez passer n'importe quel nombre de paramètres. **/
Pour appeler un contrôleur de module à partir d'un contrôleur, vous pouvez utiliser $this->load->module() ou Modules::load() et la méthode PHP5 est disponible pour tout objet chargé par MX. ie : $this->load->library('validation')->run().
Pour charger les langues des modules, il est recommandé d'utiliser la méthode Loader qui transmettra le nom du module actif à l'instance Lang ; par exemple : $this->load->load->language('language_file') ;
La fonction PHP5 spl_autoload vous permet d'étendre librement vos contrôleurs, modèles et bibliothèques à partir de classes de base application/cœur ou application/bibliothèques sans avoir besoin de les inclure ou de les exiger spécifiquement.
Le chargeur de bibliothèque a également été mis à jour pour s'adapter à certaines fonctionnalités de CI 1.7 : les alias de bibliothèque sont acceptés de la même manière que les alias de modèle, et le chargement des fichiers de configuration à partir du répertoire de configuration du module comme paramètres de bibliothèque (re : form_validation.php) ont été ajoutés.
$config = $this->load->load->config('config_file'), Retourne le tableau de configuration chargé à votre variable.
Les modèles et les bibliothèques peuvent également être chargés à partir de sous-répertoires dans leurs répertoires d'application respectifs.
Lorsque vous utilisez la validation de formulaire avec MX, vous devrez étendre la classe CI_Form_validation comme indiqué ci-dessous,
::::php
<?php
application/bibliothèques/MY_Form_Form_validation **//
classe MY_Form_validation_validation étend CI_Form_validation_Form_validation
{
    public $CI ;
}
avant d'assigner le contrôleur courant comme variable $CI à la bibliothèque form_validation. Cela permettra à vos méthodes de rappel de fonctionner correctement. (Cette question a également été discutée sur les forums CI).
::::php
<?php
classe Xyz étend MX_Controller MX_Controller
{
    fonction __construct()
    {
        parent::__construct() ;

        Ceci->charger->library('form_validation') ;

```
        Ceci->form_validation->CI =& $this ;
    }
}
```

Afficher les partiels

L'utilisation d'un module comme vue partielle à partir d'une vue est aussi simple que l'écriture :

::::php
<?php echo Modules::run('module/controller/method', $param, $....) ; ?> >

Les paramètres sont optionnels, vous pouvez passer n'importe quel nombre de paramètres.

Installation d'extensions modulaires

1. Commencez par une installation CI propre.
2. Définissez $config['base_url'] correctement pour votre installation.
3. Accédez à l'URL /index.php/welcome => montre Bienvenue à CodeIgniter.
4. Drop Modular Extensions third_party files into the CI 2.0 application/third_party directory
5. Drop Modular Extensions core files into application/core, le fichier MY_Controller.php n'est pas nécessaire à moins que vous ne souhaitiez créer votre propre extension de contrôleur.
6. Accédez à l'URL /index.php/welcome => montre Bienvenue à CodeIgniter.
7. Créer une structure de répertoire de module application/modules/welcome/contrôleurs
8. Déplacer le contrôleur application/controllers/welcome.php vers application/modules/welcome/controllers/welcome.php.
9. Accédez à l'URL /index.php/welcome => montre Bienvenue à CodeIgniter.
10. Créer un répertoire application/modules/welcome/views.
11. Déplacer la vue application/modules/welcome_message.php vers application/modules/welcome/views/welcome_message.php
12. Accédez à l'URL /index.php/welcome => montre Bienvenue à CodeIgniter.

Vous devriez maintenant avoir une installation d'Extensions Modulaires en cours d'exécution.

Guide d'installation :

Les étapes 1 à 3 vous indiquent comment faire fonctionner une installation CI standard - si vous avez une installation CI propre/testée, passez à l'étape 4.

Les étapes 4-5 montrent que le CI normal fonctionne toujours après l'installation de MX - il ne devrait pas interférer avec la configuration normale du CI.

Les étapes 6-8 montrent que MX travaille aux côtés de CI - le contrôleur est déplacé dans le module "bienvenue", le fichier de vue reste dans le répertoire CI application/vues - MX peut trouver les ressources du module à plusieurs endroits, y compris dans le répertoire de l'application.

Les étapes 9-11 montrent que le MX fonctionne à la fois avec le contrôleur et la vue dans le module "bienvenue" - il ne devrait pas y avoir de fichiers dans les répertoires application/contrôleurs ou application/vues.

FAQ

Q. Que sont les modules, pourquoi devrais-je les utiliser ?

A. (http://en.wikipedia.org/wiki/Module) (http://en.wikipedia.org/wiki/Modular_programming) (http://blog.fedecarg.com/2008/06/28/a-modular-approach-to-web-development)

Q. Qu'est-ce que le HMVC Modulaire, pourquoi devrais-je l'utiliser ?

A. Modulaire HMVC = Multiples triades MVC

Ceci est particulièrement utile lorsque vous avez besoin de charger une vue et ses données dans une vue. Pensez à ajouter un panier d'achat à une page. Le panier d'achat a besoin de son propre contrôleur qui peut appeler un modèle pour obtenir les données du panier. Ensuite, le contrôleur doit charger les données dans une vue. Ainsi, au lieu du contrôleur principal qui gère la page et le panier d'achat, le panier d'achat MVC peut être chargé directement dans la page. Le contrôleur principal n'a pas besoin d'en être informé et est totalement isolé.

Dans CI, nous ne pouvons pas appeler plus d'un contrôleur par demande. Par conséquent, pour atteindre HMVC, nous devons simuler des contrôleurs. Cela peut se faire avec des bibliothèques, ou avec cette contribution "Modular Extensions HMVC".

La différence entre l'utilisation d'une bibliothèque et d'une classe HMVC "Modular HMVC" est : 1. Pas besoin d'obtenir et d'utiliser l'instance CI au sein d'une classe HMVC 2. Les classes HMVC sont stockées dans un répertoire de modules par opposition au répertoire des bibliothèques.
Q. Est-ce que les extensions modulaires HMVC sont les mêmes que la séparation modulaire ?
A. Tout comme la séparation modulaire, les extensions modulaires rendent les modules "portables" vers d'autres installations. Par exemple, si vous créez un bel ensemble modèle-contrôleur-vue de modèle autonome de fichiers, vous pouvez amener ce MVC dans un autre projet en copiant un seul dossier - tout est dans un seul endroit au lieu d'être réparti autour des dossiers modèle, vue et contrôleur.
Modulaire HMVC signifie triades modulaires MVC. Modular Separation and Modular Extensions permet de regrouper les contrôleurs, modèles, bibliothèques, vues, etc. dans des répertoires de modules et de les utiliser comme une mini application. Mais, Modular Extensions va encore plus loin et permet à ces modules de "parler" entre eux. Vous pouvez obtenir la sortie du contrôleur sans avoir à passer par l'interface http à nouveau.

Traduit avec https://www.deepl.com/translator