Tutorials for better PHP

# [Scaling Web Applications with HMVC](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/)

by [Sam de Freyssinet](http://techportal.inviqa.com/author/sclark/) |  [44 comments](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comments) | February 22, 2010

The last decade has been witness to the second iteration of web design and development. Web sites have transformed into web applications and rarely are new projects commissioned that do not involve some element of interactivity. The increasing complexity of the software being developed for the internet fuelled a requirement for structured and considered application design.

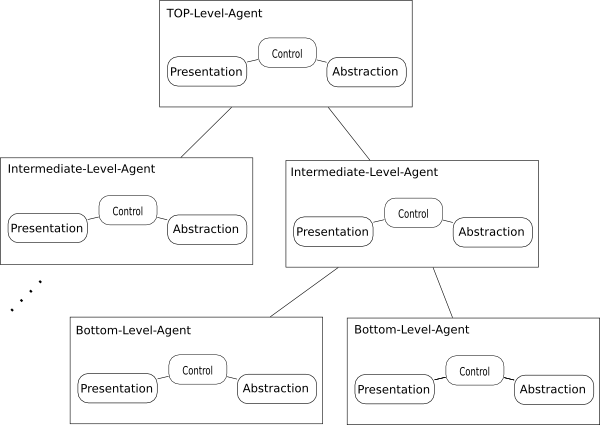
Today the most common design pattern for web software is the Model-View-Controller (MVC) pattern. The widespread adoption of the MVC design pattern was supported, in part, by the success and popularity of the Ruby on Rails framework. MVC is now synonymous with web application development across all platforms.

With the rising complexity of projects developed for the web, modern software for the web increasingly relies on dedicated services to perform processor intensive tasks. This has been encouraged further by the introduction of cloud services from Amazon, Google and several others enabling developers to considerably reduce the processor load on their servers. Each service is usually designed as a separate piece of software that runs in its own domain using its own resources.

When working with small budgets, it is generally much harder to convince clients of the benefits of funding more than one complete piece of software. In these situations I have found that many clients conclude that scalability is not a concern. They "look forward to the day when they will have to worry about scaling".

To reduce the initial investment, usually it is decided that the application should designed to be one holistic piece of software containing all the required features. This represents a potential point of failure if the software becomes very popular in a short timeframe. I have painful memories of refactoring existing codebases that have not scaled well. It can also be very costly in time and resources to re-architect software that not scaled well. Ideally applications should grow organically as required and without large sums of money being exchanged in the process.

## Hierarchical-Model-View-Controller pattern

[](http://en.wikipedia.org/wiki/File:Pac-schema.png)

The Hierarchical-Model-View-Controller (HMVC) pattern is a direct extension to the MVC pattern that manages to solve many of the scalability issues already mentioned. HMVC was first described in a blog post entitled [*HMVC: The layered pattern for developing strong client tiers*](http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html) on the JavaWorld web site in July 2000. Much of the article concentrates on the benefits of using HMVC with graphical user interfaces. There has been [some suggestion](http://web.archive.org/web/20050205080537/http://www.javaworld.com/javaworld/jw-09-2000/jw-0908-letters.html) that the authors where actually re-interpreting another pattern called Presentation-Abstraction-Control (PAC) described in 1987. The article in JavaWorld provides a detailed explanation of how HMVC can aid in the design of desktop applications with GUIs. The focus of this article is to demonstrate how HMVC can be used to create scalable web applications.

HMVC is a collection of traditional MVC triads operating as one application. Each triad is completely independent and can execute without the presence of any other. All requests made to triads must use the controller interface, never loading models or libraries outside of their own domain. The triads physical location within the hosting environment is not important, as long as it is accessible from all other parts of the system. The distinct features of HMVC encourages the reuse of existing code, simplifies testing of disparate parts of the system and ensures that the application is easily enhanced or extended.

To successfully design applications that implement the HMVC pattern, it is critical that all of the application features are broken down into systems. Each system is one MVC triad within the larger HMVC application, independently managing presentation and persistent storage methods. Presently few frameworks are available that support HMVC without additional extensions, or use inefficient Front Controllers and dispatching. [Kohana PHP version 3](http://www.kohanaphp.com) is a framework that was designed from the ground up with HMVC at the core. I will be using Kohana PHP 3 for all of the code examples in this document.

Kohana 3 uses the core request object to call other controllers. Requests can be made internally to application controllers or externally to web services transparently using the same request class[[1](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/" \l "footnote-1)]. If a MVC triad is scaled out, the request only requires modification of one parameter.

**<?php**

**class** Controller\_Default **extends** Controller {

**public** **function** action\_index()

{

*// Internal request example*

$internal\_request = Request::factory('controller/action/param')

->execute();

*// External request example*

$external\_request = Request::factory('http://www.ibuildings.com/controller/action/param')

->execute();

}

}

Requesting internally requires a valid route path targeting a controller and action. Creating a request to an external resource is as simple as supplying the full URL. This feature makes internal and external requests quickly interchangeable, ensuring that scaling out triads is a relatively simple task.

Using the Kohana Request class to provide data from internal controllers may seem similar to action forwarding in other frameworks, such as the Zend Framework. In reality the two methods are quite different. Kohana Requests have the ability to operate as unique requests in isolation. Forwarding actions do not operate in this way, each invoked controller action exists within the originating request. To demonstrate this, consider the example below.

Default Controller – /application/controllers/default.php

**<?php**

*// First request controller*

**class** Controller\_Default **extends** Controller

{

**public** **function** action\_index()

{

*// If the Request was a GET request*

if ($this->request->method === 'GET')

{

*// Output POST, will print array (0) { empty }*

var\_dump($\_POST);

*// Create a new request for another resource*

$log = Request::factory('/log/access/'.$page\_id);

*// Set the request method to POST*

$log->method = 'POST';

*// Apply some data to send*

$log->post = array(

'uid' => $this->user->id,

'ua' => Request::user\_agent('browser'),

'protocol' => Request::$protocol,

);

*// Log the access*

$log->execute();

*// Output POST, will still print array (0) { empty }*

var\_dump($\_POST);

}

}

}

Log Controller – /application/controllers/log.php

**<?php**

*// Second request controller*

**class** Controller\_Log **extends** Controller

{

**public** **function** action\_access($page\_id)

{

*// When requested from the index action above*

*// will print the posted vars from the second request*

*// array (3) {string (3) 'uid' => int (1) 1, string (2) 'ua' => string(10) 'Mozilla ... ...*

var\_dump($\_POST);

*// Create a new log model*

$log = **new** Log\_Model;

*// Set the values and save*

$log->set\_values($\_POST)

->save();

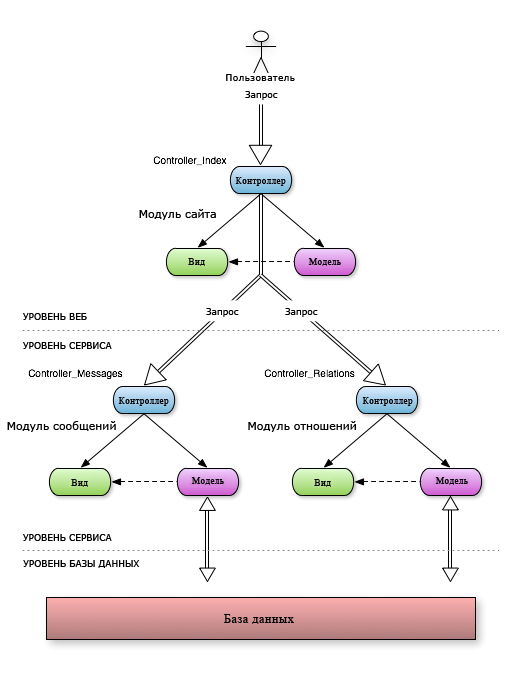
}

}

The example above demonstrates the independence afforded to the Request object. The initial request invokes the Default controller index action from a GET request, which in turn invokes a POST request to the Log controller access action. The index action sets three post variables which are not available to the global $\_POST variable from the first controller. When the second request executes, $\_POST has the post variables we made available to that request. Notice how after $log->execute(); has finished within the index controller, the $\_POST data is not there. To do dynamic interaction of this kind within other frameworks requires creating a new request using a tool like Curl.

## Gazouillement, the status service with a continental twist

To demonstrate the power of Hierarchical-MVC lets look at an example of a hypothetical status update service called Gazouillement, which works in a similar way to Twitter. Gazouillement has been designed around a service-oriented-architecture (SOA) to ensure the message and relationship engines are disparate from the web interface.



Traffic to the server will be relatively light initially. The service is new with an audience completely unaware of its presence. So it is safe to allow all of the application logic to execute on the same server for the time being.

Lets implement the controller to display a user’s homepage. A user homepage will show their most recent status messages, plus a list of people the user is following.

Index Controller – /application/controllers/index.php

**<?php**

*// Handles a request to http://gazouillement.com/samsoir/*

**class** Controller\_Index **extends** Controller {

**public** **function** action\_index()

{

*// Load the user (samsoir) into a model*

$user = **new** Model\_User($this->request->param('user'));

*// If user is not loaded, then throw a 404 exception*

if ( ! $user->loaded)

throw **new** Controller\_Exception\_404('Unable to load user :user', array(':user' => $this->request->param('user')));

*// Load messages for user in xhtml format*

$messages = Request::factory('messages/find/'.$user->name.'.xhtml')

->execute()

->response;

*// Load relationships for user in xhtml format*

$relations = Request::factory($user->name.'/following.xhtml')

->execute()

->response;

*// Apply the user index page view to the response*

*// and set the user, messages and relations to the view*

$this->request->response = View::factory('user/home', array(

'user' => $user,

'messages' => $messages,

'relations' => $relations,

));

}

}

Now a review of what Controller\_Index::action\_index() is doing. Initially the action attempts to load a user based on the user parameter of the url. If the user fails to load, a 404 page is displayed. A new request for messages is created using the users name property as a parameter within the request uri, asking for a response in xhtml format. Another request for the users relations is made in a similar manner, again in xhtml format. Finally a view object is created with the user, messages and relations responses set to it.

As the existing messages and relations are loaded using a new request, the entire application logic for each service remains abstracted from the web site. This architecture provides two significant advantages over traditional controller execution;

1. The controller is not concerned with any part of the messages service execution logic. The controller only requires that the result is xhtml formatted. No additional libraries or extensions were loaded within the controller execution.
2. Each controller is only responsible for one particular task, ensuring that writing unit tests for controllers is significantly less complicated.

Due to the abstraction currently demonstrated, it is impossible to see what the services are doing. So lets look at the messages service controller, starting with the route defined in the bootstrap to internally handle requests for messages. The [Kohana Route class](http://v3.kohanaphp.com/guide/tutorials.urls) handles internal url parsing, mapping supplied uri elements to controllers, actions and parameters.

Route setup – /application/bootstrap.php

Route::set('messages', 'messages/<action>/<user>(<format>)', array('format' => '.w+'))

->defaults(array(

'format' => '.json',

'controller' => 'messages',

));

This sets a route for the messages service, presently located within the main application domain. The url request for messages/find/samsoir.xhtml will be routed to the messages controller, calling the find() action and passing 'user' => 'samsoir', plus 'format => '.json' as parameters.

Messages Controller – /application/controllers/messages.php

**<?php**

**class** Controller\_Messages **extends** Controller {

*// Output formats supported by this controller*

**protected** $supported\_formats = array(

'.xhtml',

'.json',

'.xml',

'.rss',

);

*// The user context of this request*

**protected** $user;

*// this code will be executed before the action.*

*// we check for valid format and user*

**public** **function** before()

{

*// Test to ensure the format requested is supported*

if ( ! in\_array($this->request->param('format'), $this->supported\_formats))

throw **new** Controller\_Exception\_404('File not found');

*// Next test the username is valid*

$this->user = **new** Model\_User($this->request->param('user');

if ( ! $this->user->loaded())

throw **new** Controller\_Exception\_404('File not found');

return parent::before();

}

*// This finds the users messages*

**public** **function** find()

{

*// Load messages user 1:M messages relation*

$messages = $this->user->messages;

*// Set the response, using a prepare method for correct output*

$this->request->response = $this->\_prepare\_response($messages);

}

*// Method to prepare the output*

**protected** **function** \_prepare\_response(Model\_Iterator $messages)

{

*// Return messages formatted correctly to format*

switch ($this->request->param('format') {

case '.json' : {

$this->request->headers['Content-Type'] = 'application/json';

$messages = $messages->as\_array();

return json\_encode($messages);

}

case '.xhtml' : {

return View::factory('messages/xhtml', $messages);

}

default : {

throw **new** Controller\_Exception\_404('File not found!');

}

}

}

}

The detail of how user messages are retrieved is demonstrated within the Controller\_Messages controller. All of the methods and properties are exclusively related to the messages context, including the relationship to users. Lets step through the messages controller code to understand what is happening.

The request object initially invokes the before() method ahead of the defined action. This allows code to execute ahead of any action, normalising common controller tasks. The before() method first tests the file format requested is supported, followed by a test to ensure the user is valid. Assuming before() completed without exception, the request object will invoke the find() action. find() loads the users messages as a Model\_Iterator object. It is important to note that the iterator will be empty if no messages in the relationship were found. Finally the messages iterator is passed to a parser method \_prepare\_response() that correctly formats the data for output and sets any headers that are required.

The two controllers; Controller\_Index and Controller\_Messages; were both executed by a single request to the application. However each controller was unaware of the others presence. Any developer could read the current codebase and understand what is executing and where, providing another great feature of HMVC—maintainability.

After completing the code for the other services, the company directors are happy with the first iteration of development and green-light deployment to a live server for a limited beta trial. After a couple of weeks, the application is ready for open use. The following months see further enhancements and optimisations across the entire architecture as the customer base steadily grows.

## The Stephen Fry Effect

Stephen Fry ([@stephenfry](http://twitter.com/stephenfry)) is currently one of the most famous users of Twitter, boasting a formidable 1.3 million (and counting) followers. In the past Stephen has been capable of disabling web sites just by tweeting a url to the world, in essence creating a [Distributed Denial-Of-Service (DDOS) attack](http://en.wikipedia.org/wiki/Denial-of-service_attack) on the server residing at the end of the link.

Gazouillement has seen steady growth consistently for the past few months. Although the response time for most page requests has increased marginally, they are currently well within the acceptable standards. Then a twitter user with a follower count similar to Stephen Fry's, suddenly posts a message containing the url for Gazouillement. The application is in real trouble.

The first hurdle to overcome would be the shear volume of traffic. The application would suddenly have to handle thousands of requests per second, from only a few hundred per second previously. In this scenario Gazouillement would most likely overload and potentially crash the server. It is clear the application requires some optimisation and enhancement to handle considerably more traffic.

Analysing code to enhance performance is far from a simple task. [Previous articles](http://techportal.ibuildings.com/2009/12/01/profiling-with-xhprof/) on TechPortal have demonstrated the use of tools such as [XHProf](http://mirror.facebook.net/facebook/xhprof/) to provide deep memory and CPU analysis during code execution. Kohana PHP comes with an internal [benchmarking](http://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8) tool called the Profiler, which is supplied in the core. The Kohana profiler is complimentary to XHProf or similar tools, not a replacement. It helps developers identify slow parts of Kohana's execution that can subsequently be examined in greater detail using XHProf, or an internal module called Codebench that is supplied with the framework.

Enabling the profiler in Kohana requires the modification of a single parameter within the bootstrap.

Bootstrap – /application/bootstrap.php

*//-- Environment setup --------------------------------------------------------*

Kohana::$profiling = **TRUE**;

Finally, add the Profiler view to the end of the controller response. This is best done by adding an after() method to your controller. This will ensure the profiler is displayed for all actions.

Index Controller – /application/controllers/index.php

**public** **function** after()

{

*// Add the profiler output to the controller*

$this->request->response .= View::factory('profiler/stats');

}

Once the profiler is enabled, the output will appear at the foot of every action within that controller. Best practice is to extend the Kohana Controller and apply the profiler view within the extended classes after() method. Now, all the system controllers are outputting profiling data during development.

The profiler reports on the execution of the framework, grouping various contexts together, including; initialisation; requests; database queries; and other operations juxtaposed with memory allocation and cpu time. As each request created during execution is reported, spotting requests that are taking a long time to execute is much easier. Once slow requests have been isolated, tools such as XHProf can analyse the execution of slow actions in greater detail.

## Scaling out Gazouillement

Performance analysis of the whole of the Gazouillement application reveals that the retrieval of messages is causing massive bottlenecks. The development team refactors and optimises the messages MVC triad as much as possible, but the required performance gains are not met. After exhausting all the vertical scaling options including upgrading the server memory and processors, the company directors agree to scaling out the application starting with the messages system.

In traditional MVC applications, a new service has to be designed, developed, acceptance tested and then deployed. This process can take weeks and months, requiring significant investment. Essentially it is new piece of software that will be integrated into the Gazouillement app.

Hierarchical-MVC applications can create new services from the existing codebase in a fraction of the time. All interactions with the messages service were through the main controller, so the only modification to the current codebase will be to the requests for messages. The messages service is migrated to another server and optimised for database interactions. This new server only executes actions relating to messages, increasing the performance of all message related operations considerably.

Index Controller – /application/controllers/index.php

**<?php**

*// Handles a request to http://gazouillement.com/samsoir/*

**class** Controller\_Index **extends** Controller {

**protected** $\_messages\_uri = 'http://messages.gazouillement.com';

**public** **function** action\_index()

{

*// Load the user (samsoir) into a model*

$user = **new** Model\_User($this->request->param('user'));

*// If user is not loaded, then throw a 404 exception*

if ( ! $user->loaded)

throw **new** Controller\_Exception\_404('Unable to load user :user', array(':user' => $this->request->param('user')));

*// -- START REVISED CODE --*

*// New messages URI to external server*

$messages\_uri = $this->\_messages\_uri.'/find/'.$user->name'.xhtml';

*// Load messages for user in xhtml format*

$messages = Request::factory($messages\_uri)

->execute()

->response;

*// -- END REVISED CODE --*

*// Load relationships for user in xhtml format*

$relations = Request::factory($user->name.'/following.xhtml')

->execute()

->response;

*// Apply the user index page view to the response*

*// and set the user, messages and relations to the view*

$this->request->response = View::factory('user/home', array(

'user' => $user,

'messages' => $messages,

'relations' => $relations,

));

}

}

The code above highlights the tiny change required in the Controller\_Index::action\_index() controller serving the public website. Rather than the messages request referencing an internal controller action, the request now points to the messages service running on a subdomain called http://messages.gazouillement.com/. What would usually be a projected as a major architecture overhaul has become a minor alteration. The associated testing and acceptance overhead is considerably smaller as not much code has changed.

This is just one example of how the Hierarchical-Model-View-Controller pattern enables software engineers to design web applications, which can scale both vertically and horizontally from the day one. We saw how the request object can be interrogated allowing controllers to return correct data for each context using a simple interface. Finally we have seen how traditional MVC triads can be scaled, and witnessed how the request object in Kohana makes scaling out a simple task. The costs involved scaling the application have been relatively small due to minimal alterations, keeping the company directors very happy.

For more information about Kohana PHP, please visit the [Kohana website](http://www.kohanaphp.com). Kohana PHP 3 can be downloaded from Github and there is also api documentation available at <http://v3.kohanaphp.com/guide>.

1. The request class used in this example is currently available as part of a Kohana Core development branch within my personal github account, which can be obtained from <http://github.com/samsoir/core>. If using the official Kohana PHP 3.0 download, [a custom extension of the request class](http://github.com/samsoir/core/commit/3159a51ae46299572da960aef6b85853d2a0750e) is required.

* [Optimising HMVC Web Applications for Performance](http://techportal.inviqa.com/2010/11/16/optimising-hmvc-web-applications-for-performance/)
* [DPC Radio: Profiling PHP Applications](http://techportal.inviqa.com/2011/12/14/dpc-radio-profiling-php-applications/)
* [DPCRadio: The Art of Scalability](http://techportal.inviqa.com/2010/11/23/dpcradio-the-art-of-scalability/)
* [Habits of Highly Scalable Web Applications](http://techportal.inviqa.com/2010/03/02/habits-of-highly-scalable-web-applications/)
* [Surviving a Plane Crash](http://techportal.inviqa.com/2010/02/16/surviving-a-plane-crash-2/)
* [Profiling with XHProf](http://techportal.inviqa.com/2009/12/01/profiling-with-xhprof/)
* [Introduction to JavaScriptMVC](http://techportal.inviqa.com/2009/09/23/introduction-to-javascriptmvc/)

44 Responses

Stay in touch with the conversation, subscribe to the [RSS feed for comments on this post](http://techportal.inviqa.com/feed/).

1. [Damian Jakusz-Gostomski](http://www.gostomski.co.uk/) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-413)

Very interesting read. I'm currently more of a CodeIgniter guy but stuff like this might be that final push which finally makes me get round to taking a serious look at Kohana

1. Carsten said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-414)

Thanks a lot! Coincidentally I'm just starting a complete rewrite of my community project with Kohana 3 and this tutorial comes at the exact right moment!

1. [Marco Barbosa](http://marcobarbosa.com) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-415)

Wow great post.

Thanks for sharing!

1. [Mats Lindh](http://e-mats.org/) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-416)

Very interesting indeed and a good introduction to HMVC. Seems like I'll have to take a closer look on Kohana 3!

1. Ren said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-418)

Nice, but seemingly could be improved.

It seems all requests are performed in series, whereas some could be combined in parallel.

Two or more external requests could be dispatched in parallel with curl\_multi/stream\_select.

Even two or more internal requests which are cached (in memcached for example) could be combined and fetched by MemCached::getMulti() or MemCached::getDelayed() if have other stuff to be getting on with.

1. Sam de Freyssinet said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-419)

@Ren

Thanks for your comment. The request class distributed with Kohana is designed for single requests. However, as Kohana is highly extendable there is no reason why parellel requests could not (and should not) be implemented.

The scope of this article was to introduce new concepts afforded by the HMVC design pattern. Related subjects such as parallel requests, reverse routing, caching and other optimisations are all possible and encouraged, however go beyond the remit I gave myself.

1. Michael Stillwell said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-420)

Nice! So, essentially, the "Kohana way" is to use the same HTTP-based calling convention for both internal and external resources? (Is there any non-trivial service/data source that you \*wouldn't\* want to access via a Kohana request object?) I do like the idea of HTTP everywhere ...

1. [Adrien M.](http://www.ze-technology.com) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-424)

Hi ! Sorry for that question, but program did you use to draw suck graphics ? :)

Excellent article by the way !!

1. Sam de Freyssinet said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-425)

@Michael

As with most things in life, the use of the request object internally is subjective. Dataproviders that are directly related to the triad in execution probably do not require a unique request.

When compared to invoking a library method traditionally, there is an overhead to the execution time that must be considered for internal requests. However much of the deficit can be mitigated against if caching is used sensibly.

1. Mamsaac said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-427)

I would love to see more of this (as in a continuation of this post). I will check for more information of HMVC, but this works as a nice and simple (and very practical) introduction. Thanks.

1. [Gerard Sychay](http://www.straylightrun.net) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-428)

To do dynamic interaction of this kind within other frameworks requires creating a new request using a tool like Curl.

So does the Request::factory() code actually execute another HTTP request underneath, or is it some PHP trick?

Also, would you say this is generally-speaking a SOA?

Nice article.

1. [Oliver](http://broddlit.wordpress.com/) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-429)

Nice post, thanks for the insight into this, I actually missed HMVC completely so far! And its pretty simple and obvious to do it that way.

1. Felix Kitaka said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-430)

Nice. First got know about Kohanna in some open source project (Ushahidi). I think HMVC is the best reason for me to make a switch from Zend.

1. John said

on [February 24, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-433)

It's remarkable how close this example appears to the foundation for a Rails 2 app; I am very impressed by the Kohana framework and will definitely be investigating it further.

1. [Jonathan Nieto](http://www.jnieto.org) said

on [February 25, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-438)

Indeed, very interesting, HMVC seems really useful!, I already use symfony but surely I'll take a look to Kohana 3.

Just one question, how do you think it would be the best way to secure the calls to this services external services.

1. Sam de Freyssinet said

on [February 25, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-439)

@Garard  
Request factory does not use curl if handling a request to an internal resource. This would be inefficient. Instead the Request object isolates its environment internally, a quick look at the source code will reveal all.

I would agree that the HMVC pattern is a close match to the SOA pattern - kind of micro-SOA.

@Jonathan

Securing the request calls is another subject I chose to avoid to try and keep this article concise. There are multiple ways to secure these requests, from a custom SSO system to industry standard OpenID/OAuth combinations. It is entirely up to the implementor.

1. [HitMan](http://www.netfreaks.gr) said

on [February 25, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-440)

FYI, there is a library for CodeIgniter to make your app modular, called "Modular Extensions - HMVC" ( cur version 5.2! ) and it's working good!

I used it at my last project and encountered almost zero errors or so...

Kohana is good, I started with it, but for some reasons i wouldn't like to disclose atm, i switched to CI and I'm really happy so far with my choice. But i still want to see Kohana proceed and get better.

Conclusion: Really good introductory article about HMVC! Thumbs up :-)

1. Guido said

on [February 27, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-441)

Seriously. It's about time people started talking HMVC. MVC is HMVC's little sister.

1. [Alex](http://www.alexanderdickson.com) said

on [April 13, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-445)

This was a great read, cheers Sam!

Just curious, in your Controller\_Messages, would you keep the controller limited to internal requests only, perhaps in the before() method?

I've seen something like this on Stack Overflow, would this be similar to what you use?

if (Request::instance() !== $this->request) {  
throw new Controller\_Exception\_404('External request');  
}

<http://stackoverflow.com/questions/2393051/favourite-kohana-tips-features/2393085#2393085>

1. [alarabistar](http://www.alarabistar.com/) said

on [May 9, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-447)

this is wonderful tutorial .. i read it 3 times and get a fantastic results and sure i put a  
copy of this lesson on my site here

1. [alarabistar](http://www.alarabistar.com) said

on [May 19, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-450)

One of the best articles i’ve ever read!

### Continuing the Discussion

1. [*Scaling Web Applications with HMVC*](http://w3digg.com/2010/02/22/scaling-web-applications-with-hmvc/) *linked to this post* on February 22, 2010

[...] The last decade has been witness to the second iteration of web design and development. More here: Click Here…Read Other Interesting Posts in FacebookThe Future Of The Internet: Search Looks BrightFacebook [...]

1. [*abcphp.com*](http://abcphp.com/story/2528/) *linked to this post* on February 22, 2010

**Scaling Web Applications with HMVC – techPortal...**

The last decade has been witness to the second iteration of web design and development. Web sites have transformed into web applications and rarely are new projects commissioned that do not involve some element of interactivity. The increasing complexi...

1. [jaredrichardson (jared richardson)](http://twitter.com/jaredrichardson/statuses/9484138441) said

on [February 22, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-422)

HMVC <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [*php-html.net*](http://php-html.net/story/705/) *linked to this post* on February 22, 2010

**Scaling Web Applications with HMVC...**

The Hierarchical-Model-View-Controller (HMVC) pattern is a direct extension to the MVC pattern that manages to solve many of the scalability issues already mentioned. HMVC was first described in a blog post entitled HMVC: The layered pattern for develo...

1. [*Ibuildings techPortal: Scaling Web Applications with HMVC | Webs Developer*](http://www.websdeveloper.com/ibuildings-techportal-scaling-web-applications-with-hmvc/) *linked to this post* on February 22, 2010

[...] about a slight modification to a well-known pattern (MVC) by adding some scalability - the Hierarchical-Model-View-Controller pattern. It can also be very costly in time and resources to re-architect software that not scaled well. [...]

1. [*Ibuildings techPortal: Scaling Web Applications with HMVC | Development Blog With Code Updates : Developercast.com*](http://www.developercast.com/2010/02/22/ibuildings-techportal-scaling-web-applications-with-hmvc/) *linked to this post* on February 22, 2010

[...] about a slight modification to a well-known pattern (MVC) by adding some scalability - the Hierarchical-Model-View-Controller pattern. It can also be very costly in time and resources to re-architect software that not scaled well. [...]

1. [*Escalar aplicaciones web usando HMVC | Sentido Web*](http://sentidoweb.com/2010/02/22/escalar-aplicaciones-web-usando-hmvc.php) *linked to this post* on February 22, 2010

[...] Scaling Web Applications with HMVC [...]

1. [*Le Web avec HMVC | traffic-internet.net*](http://www.traffic-internet.net/?p=10011) *linked to this post* on February 23, 2010

[...]  Scaling Web Applications with HMVC (0 visite) [...]

1. [*links for 2010-02-23 | .:: a few thoughts on the subject by rob wright ::.*](http://f1fe.com/blog/2010/02/23/links-for-2010-02-23/) *linked to this post* on February 24, 2010

[...] Scaling Web Applications with HMVC – techPortal (tags: architecture designpatterns hmvc mvc scaling development) February 23rd 2010 Posted to Links [...]

1. [alexdickson (alex dickson)](http://twitter.com/alexdickson/statuses/9563919328) said

on [February 24, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-435)

Scaling Web Applications with HMVC by Sam de Freyssinet <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [*Ibuildings techPortal: Scaling Web Applications with HMVC « dot nikis*](http://nikis.wisdom.bg/2010/02/23/ibuildings-techportal-scaling-web-applications-with-hmvc/) *linked to this post* on February 24, 2010

[...] about a slight modification to a well-known pattern (MVC) by adding some scalability – the Hierarchical-Model-View-Controller pattern. It can also be very costly in time and resources to re-architect software that not scaled well. [...]

1. [leolower (Leonardo)](http://twitter.com/leolower/statuses/9592890806) said

on [February 24, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-437)

Scaling Web Applications with HMVC – techPortal <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [*Links for 2010-02-27 | Stéphane Thibault*](http://xn--stphanethibault-cnb.com/2010/02/28/links-for-2010-02-27/) *linked to this post* on February 28, 2010

[...] Scaling Web Applications with HMVC – techPortal (tags: WebDev PHP DesignPatterns MVC HMVC SOA Software Architecture Scaling) [...]

1. [*Scaling Web Applications with HMVC | Guido Chiappucci's Blog*](http://www.chiappucci.it/blog/2010/02/uncategorized/scaling-web-applications-with-hmvc.html) *linked to this post* on March 1, 2010

[...] <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/> [...]

1. [AnwarSumawang (Anwar Sumawang)](http://twitter.com/AnwarSumawang/statuses/12044509518) said

on [April 12, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-444)

HMVC (<http://bit.ly/cKttqu>), the answer to Qs like: "How do I pass this array to the controller I'm redirecting to?" Is it good practice?

1. [*Kohana PHP framework*](http://www.geekality.net/2010/04/19/kohana-php-framework/) *linked to this post* on April 19, 2010

[...] I finally stumbled over a framework called Kohana. Kohana is an open source, object oriented H MVC web framework built using PHP5 by a team of volunteers that aims to be swift, secure, and [...]

1. [*» External Requests in Kohana sudo rm -rf /\**](http://o1iver.net/blog/?p=46) *linked to this post* on May 10, 2010

[...] Here is how I handle external requests. This method is really copied from this site: <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/> [...]

1. [*New better version « sudo rm -rf /\**](http://o1iver.net/blog/?p=52) *linked to this post* on May 10, 2010

[...] Here is how I handle external requests. This method is really copied from this site: <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/> [...]

1. [igorgurski (Igor Bueno Gurski)](http://twitter.com/igorgurski/statuses/28912330158) said

on [October 27, 2010](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-451)

Scaling Web Applications with HMVC - <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [xavividal (Xavier Vidal)](http://twitter.com/xavividal/statuses/22298697681338369) said

on [January 4, 2011](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-452)

Spread the HMVC <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [rabidmachine9 (Konstantinos P.0)](http://twitter.com/rabidmachine9/statuses/39733472826372096) said

on [February 21, 2011](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-453)

nice article on HMVC <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/>

1. [senthilgurue (senthilgurue)](http://twitter.com/senthilgurue/statuses/40287131897307136) said

on [February 23, 2011](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-454)

Scaling Web Applications with HMVC : <http://tinyurl.com/yktsl2g>

1. [victhor\_mendes (wesley victhor )](http://twitter.com/victhor_mendes/statuses/62641700425826305) said

on [April 25, 2011](http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/#comment-455)

Scaling Web Applications with HMVC - <http://techportal.ibuildings.com/2010/02/22/scaling-web-applications-with-hmvc/> #dev

Начало формы

TechPortal is a trademark of Inviqa.