

Monday, February 27, 2006

The no-framework PHP MVC framework

March 1 - Disclaimer: Since a lot of people seem to me misunderstanding this article. It isn't about OOP vs. Procedural programming styles. I happen to lean more towards procedural, but could easily have gone more OOP. I simplified the code a bit for brevity, but have added a light OO layer back in the model now. Not that it makes a difference. What I was hoping to get across here is a simple example of how you can use PHP as-is, without additional complex external layers, to apply an MVC approach with clean and simple views and still have all the goodness of fancy Web 2.0 features. If you think I am out to personally offend you and your favourite framework, then you have the wrong idea. I just happen find most of them too complex for my needs and this is a proposed alternative. If you have found a framework that works for you, great.

So you want to build the next fancy Web 2.0 site? You'll need some gear. Most likely in the form of a big complex MVC framework with plenty of layers that abstracts away your database, your HTML, your Javascript and in the end your application itself. If it is a really good framework it will provide a dozen things you'll never need.

I am obviously not a fan of such frameworks. I like stuff I can understand in an instant. Both because it lets me be productive right away and because 6 months from now when I come back to fix something, again I will only need an instant to figure out what is going on. So, here is my current approach to building rich web applications. The main pieces are:

PHP 5
Yahoo! User Interface Library
JSON

MVC?

I don't have much of a problem with MVC itself. It's the framework baggage that usually comes along with it that I avoid. Parts of frameworks can be useful as long as you can separate the parts out that you need. As for MVC, if you use it carefully, it can be useful in a web application. Just make sure you avoid the temptation of creating a single monolithic controller. A web application by its very nature is a series of small discrete requests. If you send all of your requests through a single controller on a single machine you have just defeated this very important architecture. Discreteness gives you scalability and modularity. You can break large problems up into a series of very small and modular solutions and you can deploy these across as many servers as you like. You need to tie them together to some extent most likely through some backend datastore, but keep them as separate as possible. This means you want your views and controllers very close to each other and you want to keep your controllers as small as possible.

Goals for this approach

Clean and simple design

HTML should look like HTML

Keep the PHP code in the views extremely simple: function calls, simple loops and variable substitutions should be all you need

Secure

Input validation using `pecl/filter` as a data firewall

When possible, avoid layers and other complexities to make code easier to audit

Fast

Avoid `include_once` and `require_once`

Use APC and `apc_store/apc_fetch` for caching data that rarely changes

Stay with procedural style unless something is truly an object

Avoid locks at all costs

Example Application

Here is the example application I will be describing.

It is a form entry page with a bit of Javascript magic along with an sqlite backend. Click around a bit. Try to add an entry, then modify it. You will see the server->client JSON traffic displayed at the bottom for debug purposes.

The Code

This is the code layout. It uses AJAX (with JSON instead of XML over the wire) for data validation. It also uses a couple of components from the Yahoo! user interface library and PHP's PDO mechanism in the model.

The presentation layer is above the line and the business logic below. In this simple example I have just one view, represented by the add.html file. It is actually called add.php on the live server, but I was too lazy to update the diagram and it really doesn't matter. The controller for that view is called add_c.inc. I tend to name files that the user loads directly as something.html or something.php and included files as something.inc. The rest of the files in the presentation layer are common files that all views in my application would share.

ui.inc has the common user interface components, common.js contains Javascript helper functions that mostly call into the presentation platform libraries, and styles.css provides the stylesheet.

A common db.inc file implements the model. I tend to use separate include files for each table in my database. In this case there is a just single table called "items", so I have a single items.inc file.

Input Filtering

You will notice a distinct lack of input filtering yet if you try to inject any sort of XSS it won't work. This is because I am using the pecl/filter extension to automatically sanitize all user data for me.

View - add.html

Let's start with the View in add.html:

The main thing to note here is that the majority of this file is very basic HTML. No styles, or javascript and no complicated PHP. It contains only simple presentation-level PHP logic. A modulus operation toggles the colours for the rows of items, and a loop around a heredoc (

Posted by Rasmus in PHP at 14:39

Sunday, February 26. 2006

Apache 1.3.34 Debian Package w/ mod_deflate and no pthreads

Debian's Apache1 package doesn't quite do what I need. I have been building my own and overwriting the files from the Debian package, but that can get annoying. So I hacked my changes into the Debian source package and built real .debs. I figure they might be useful to others.

The main changes are to get rid of -lpthread (needed by mod_perl) and -lexpat and to add mod_deflate. To enable mod_deflate make sure your /etc/apache/modules.conf file has:

```
AddModule mod_deflate.c
```

And in your httpd.conf add:

```
DeflateEnable      on
DeflateMinLength   1024
DeflateCompLevel   8
DeflateProxied     on
DeflateDisableRange "MSIE 4."
DeflateVary        on
DeflateTypes       text/css
DeflateTypes       text/plain
DeflateTypes       text/rtf
DeflateTypes       text/xml
DeflateTypes       text/javascript
DeflateTypes       image/vnd.dwg
DeflateTypes       image/vnd.dxf
DeflateTypes       application/msword
DeflateTypes       application/vnd.hp-HPGL
DeflateTypes       application/vnd.ms-access
DeflateTypes       application/vnd.ms-excel
DeflateTypes       application/vnd.ms-powerpoint
DeflateTypes       application/vnd.ms-project
DeflateTypes       application/vnd.visio
DeflateTypes       application/x-javascript
```

Posted by Rasmus in Software at 23:06