PHP MySQL Website Programming

Problem – Design – Solution

Chris Lea
Mike Buzzard
Jessey White-Cinis
Dilip Thomas

Apress™

PHP MySQL Website Programming

Problem – Design – Solution

ISBN (pbk): 1-59059-150-X

Printed and bound in the United States of America 2345678910

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.
In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

14

The Road Ahead

Having deployed the site on our ISP or local machine, and having tinkered with it and browsed the code, we are done with the task that had been set for us at the beginning of this book – building an application-driven PHP MySQL web site.

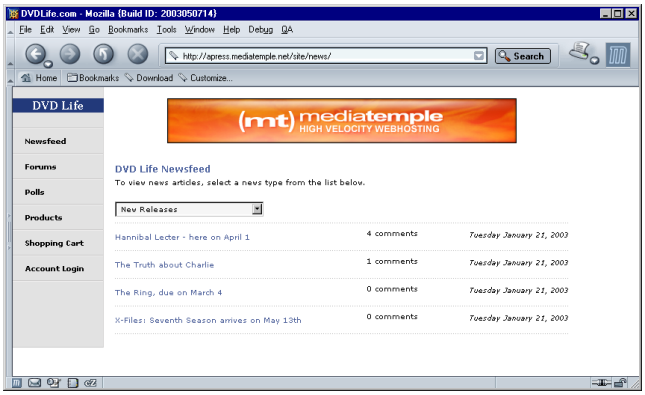
Now let's review what we have done.

Our Finished Site

After laying out the basic plan for the site, in Chapters 1 and 2, we looked at the creation of a solid foundation for the UI that could be employed throughout the design of our application. In Chapter 3 we created a file that uniformly creates the header, navigation, and footer HTML for any given page.

Then in Chapters 4 and 5 we looked at laying the foundation for our application by implementing a user management system for the Content Management System (CMS) as well as the web site's visitor accounts.

Finally, it was time to build on to the core framework by implementing vertical applications.

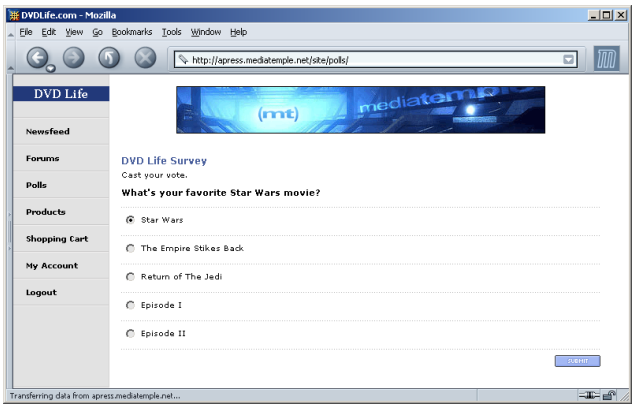


Browse Through the Latest News Items

Chapter 6 built a simple news management and delivery system, and also laid the groundwork for ou RSS news syndication, which we then detailed in Chapter 7.

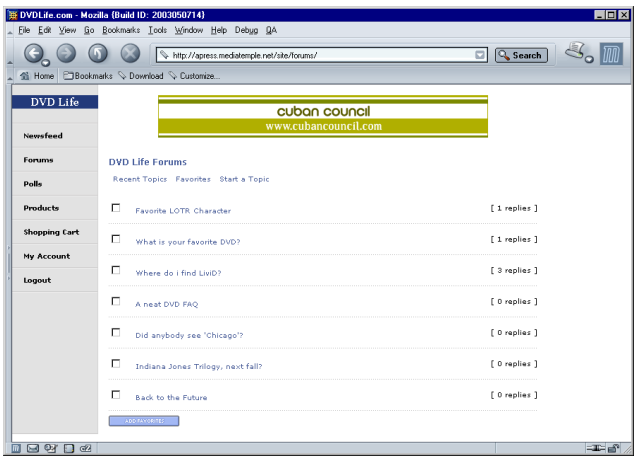
To encourage community presence we made a provision for users to post opinions about news articles as well.

We also discussed different options to implement reliable, practical, and effective advertising systems into the web site to supplement our revenue model, in Chapter 8.



Cast Your Vote in the Polls

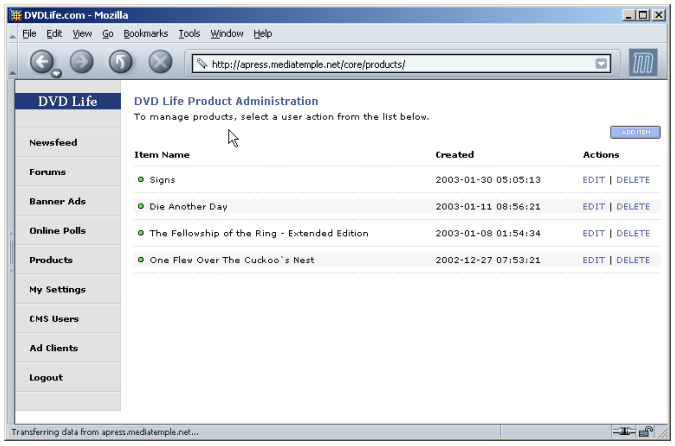
Chapter 9 built the polling system, attempting to make sure it was close to one-person, one-vote, while not discouraging visitors from participating i the poll.



Browse Through the Forums

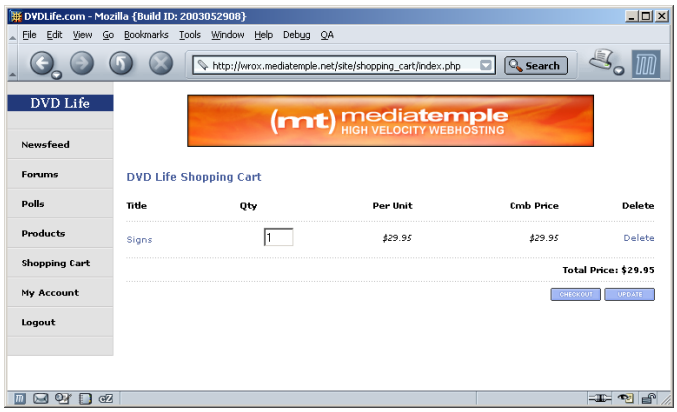
Chapter 10 built the discussion forums, where visitors could browse the message in the forums, ask questions and reply to those of others, and share ideas and tips.

e-mail is an effective direct marketing tool, and Chapter 11 added newsletter functionality to the web site to get information across to our users in a simple and controlled fashion.



Look Through the Shop and Pick Some Products

Chapter 12 built a products list and the item selection process.



Buy Some Products

Chapter 13 finished the shopping process by building a product ordering and fulfillment system, and utilized VeriSign's PayFlow Pro API to implement the financial transactions.

Now Create an Account and Start Making Posts to the Site

Built early on in Chapter 5, we showed how to authenticate and store user accounts.

Over the course of our investigation into building dynamic, interactive web sites using PHP and MySQL, we've discussed a number of technologies that can help us to accomplish this. We started with building libraries of useful components, like our basic header and footer templates. Then we introduced how classes can be used to modularize code relating to database access, making it easier to reuse, rather than rebuilding it from scratch each time we wish to implement similar functionality. These are core programming concepts that are useful no matter what language we're programming with.

Now that we've created a working site that meets all of our functional requirements, this is an excellent time to reflect on how well we've done and whether we could have done better. In most projects this is called **refactoring**, and is done continuously during the development process. However since our aim is not to throw you into the deep end, we have left this stage to the very end of the book.

A lot of the concepts that we will explore in the chapter on refactoring will be used more in the next book in this series, in which we'll examine building e-commerce sites with PHP and MySQL.

Refactoring

Like many other aspects of programming, refactoring is not an exact science; suggestions and ideas that are introduced during refactoring have both advantages and disadvantages. It's up to you as the programmer to decide if any of the refactoring ideas would be beneficial in your application.

So let's dive in and take a look at what we have done.

Our Base Structure

In Chapter 2, we laid out our files and grouped them according to function or purpose, either as modules or shared resources. This is a common method in many open source projects; you may have seen similar layouts before if you've browsed around existing PHP applications.

As you've read this book, you may have noticed that we introduced early on the idea of using functions like:

```
openPage();
```

to accomplish tasks that are often repeated.

As the application codebase grew, remembering where this was defined (in Chapter 3) became more difficult. In most projects it is also important that when somebody looks at the code you have written, that person will be able to find function definitions and change them quickly. One of the more common methods of facilitating this is by writing classes that act as wrappers for often-reused functions.

```
Site_Elements::openPage();
```

By defining the function as a method of the `Site_Elements` class, we can help the casual reader of the code to have a reasonable chance of working out where `openPage()` is actually defined.

It's a good idea to place class definitions in include files with appropriate names such as `Site_Element.class.php`*. However, even wrapper classes/methods shouldn't be arbitrary – they should model a logical component of the application.*

You may also notice here that we have named our class very similarly to the directory in which it resides. As you explore PEAR, you will realize this is their standard, and makes locating classes and code very easy.

Real Life Annoyances

Like all perfect designs, real life has a horrible habit of coming back and biting us. You may have noticed early on that we stored some user information in cookies.

```
if( isset($_COOKIE["cUSER"]) ) {
    ....
```

This offers a very easy way to deal with storing variables and passing them from page to page. But what happens if the end user has read some horror story about Internet Explorer and viruses and thought that turning off cookies was a good idea? How do we deal with such a case?

Like most advanced server application languages, PHP has a built-in feature known as sessions, which can be configured to use automated URL rewriting if the user has turned off cookies in his/her browser.

In order to enable a user session, all we have to do is add this code to the start of any page.

```
session_start();
```

Once you've done this, you can access any session variable using the PHP superglobal `$_SESSION` array. For example:

```
if( isset($_SESSION["cUSER"]) ) {
    // .....
}
```

You can also store object instances, like the data objects we created in Chapter 4, in the session, making it easy to manage large amounts of data about a user.

For more information about sessions in PHP, see *Professional PHP4 Programming* from *Apress* (*ISBN 1-861006-91-8*). It would also be good to refer to the session handling section in the PHP Manual at http://www.php.net/manual/en/ref.session.php.

Looking for More Flexibility in Look and Feel

In Chapter 3 we used Cascading Style Sheets (CSS) in our header and footer code. This served us very well throughout the book, and is likely to work well in most simple situations. But what if you wanted to provide a different look and feel for affiliate DVD shops, so they could use your online shop, but with their own logos, layout, and color scheme?

Well, you could go down the road of writing an individual template file for each affiliate, but over time this broken up HTML and PHP file is very difficult to maintain, since you cannot instantly get an idea of what the site will look like when editing the HTML.

This is where **templating** in PHP comes in – although PHP really started off as a way to add 'variables' to HTML, it's now far more than that, as you can see from the site we have developed. Templating solutions were developed to try and solve the issue of mixing PHP and HTML. The great advantage of this is that you can design your web pages in WYSISYG tools like Macromedia Dreamweaver, which also allows you preview PHP pages live on a development server. This static HTML can then be easily integrated with the web site's PHP code.

While there is not enough space in this book to cover all of the available packages, or even tell you about the detailed benefits of each, we will provide a general overview of the two main types and their advantages and disadvantages.

For a more detailed discussion of PHP templating, look out for the *PHP Templating Handbook (ISBN 1-861008-96-1)* currently under development by this publisher.

HTML Tag Replacers

In the HTML file, we can put in comments and tags indicating the start and end of blocks or where to show variables:

```
<!-- BLOCK: START someloop -->
<td>{somevariable}</td>
```

These engines are usually very easy to write templates for in WYSISYG editors, and the person designing the HTML page does not require any real knowledge of writing PHP applications.

A lot of the work is done by the PHP program when displaying the page by replacing the tags with PHP variables or values, often using the PHP's `str_replace` function. For simple tasks this type of template is very easy to use; however, it can become difficult to work with if you have many complex loops and other programming blocks. You should also be aware that if performance is a concern, some packages that use this method can be very slow.

PHP Code Replacers

The second approach for templates is to take a template (similar to the one we saw earlier) and replace it with PHP code:

```
<?php foreach($someblock as $somevariable) { ?>

    <td><?php echo $somevariable ?></td>

<?php } ?>
```

This method is significant in terms of performance, since the file is converted only when it's modified, and can allow template engines to have considerably more features.

Hopefully this gives you a start on understanding templates and how they could be implemented with our DVD site; you will be seeing more of this in the next book on e-commerce.

Other than these resources, you may want to explore XSLT in the *PHP4 XML* book from *Apress* (*ISBN 1-861007-21-3*). As is often the case, there is no single 'right' way to do this, just a lot of excellent alternatives for you to make your own choice based on your circumstances and requirements:

- PEAR (http://pear.php.net/) includes a number of template packages
- Smarty is a well developed engine (http://smarty.php.net/)
- The PHP Classes site has a mix of various template engines (http://www.phpclasses.org/)

More PEAR

Throughout this book we have used the PEAR::DB abstraction layer rather than PHP's own MySQL functions. From a programming perspective this "reduce each database's API feature set to the lowest common denominator" feature is extremely advantageous. Since PEAR is closely tied to PHP, there are a considerable number of users supporting, improving, and fixing the packages. With the release of PHP 4.3, PEAR is being more closely integrated with PHP itself, and the forthcoming *Professional PHP5* book from *Apress (ISBN 1-861007-85-X)* will contain a lot more information on the other packages available.

The purpose of this book was to learn how to design a dynamic web site, rather than a PEAR tutorial. But now that we are considering refactoring, we should take a look at some of the packages in PEAR that might help to improve our application.

The first of these is the PEAR error handler, part of the base PEAR system. `PEAR_ERROR` or an extended version is really the best method to store and retrieve error messages. In our application, we do use `PEAR_ERROR` everywhere we can, and, once we get the error messages, trap them in our own way so that we can display them in reasonably friendly ways.

In the application we created for this book, we implemented our own error handler; when something went wrong, we called our 'print out an error page'. Using PEAR, we normally check to see if there was a PEAR error. Furthermore, PEAR has the ability to set a callback function for errors, and PEAR's `raiseError()` method can also emit our own application's error messages.

At the time of writing there were over 100 PEAR packages, offering a multitude of tools for databases like data objects and query builders, for XML parsing, for sending e-mail, and many more. It's well worth a look to see if any of those packages could be used to replace or extend our existing code:

- PEAR home page (http://pear.php.net/)
- The CVS repository of PEAR code (http://cvs.php.net/cvs.php/pear)
- PEAR tutorials listing (http://php.weblogs.com/php_pear_tutorials/)

Classes Everywhere

We've used classes heavily for implementing our data objects, and our page output remained as straight-through output. You may have noticed a lot of similarity in the code on different pages; a lot of it took input, and called a related object to create, update, list, or delete it. This commonality is what the classes thrive on.

Examining our page output code, you can see that a base set of methods, say for redirecting to the add or update functionality, would save considerable code. While you could create a function library to do this, using classes will make the code considerably easier to read later, and just by extending this base page library class, you can very quickly provide the functionality in each module.

Going on a lot further here, we might be able to offer a different solution to browser redirection on submitting data. If we became less dependent on global and context variables, we could just pass control of our application from one object to another.

As mentioned early on, we have assumed that `register_globals` is turned off (as it is by default beginning with PHP version 4.2.0). Sometimes, however, people may end up turning it back on so they can run that 'cool' webmail application they really like – so it can be a very dangerous assumption to make. We could program defensively by using classes, which don't inherit the global variable scope or take precautions against this.

The following is a list of some excellent online resources exploring the many facets of PHP' s object-oriented (OO) coding:

- PHP Patterns (http://www.phppatterns.com/)
  A site devoted to helping PHP developers learn and employ object-oriented programming (OOP) patterns in their applications
- Object-Oriented PHP (http://www.phpbuilder.com/columns/luis20000420.php3 and http://www.phpbuilder.com/columns/mark20000727.php3)
  A pair of articles from PHPBuilder – the first is intended for developers with OOP experience but who might be new to PHP, and the second is for PHP users who are new to OOP
- PHP Classes (http://www.phpclasses.org/)
  A more general collection of PHP programming classes, of varying quality, often a useful to base for your own classes

Also Apress's forthcoming *Professional PHP 5* book (*ISBN 1-861007-85-X)* is expected to cover the enhanced OOP functionality of PHP 5.

The Little Details

If you have ever written a document and given it to somebody to check, looking for deep and meaningful analysis, but ended up with them dissecting on your spelling and grammar, you are going to love this.

No refactoring is complete without going through your code line by line. So let's take a look at some of ours and see if we could have done better.

We used the following code bit throughout to send queries to the database. Since MySQL and most other databases use persistent connections, there is no real overhead in connecting; in fact, if you have connected already and try to do it again, nothing actually happens. So we don't really need to store that connection object; we could just create a `query()` method in the base data object and keep calling it:

```
$this->_oConn = &DB::connect(DSN)
....
$this->_oConn->query(...);
```

Note that this is true in our own context only if PHP and MySQL are configured to do so. However, it is true that connections last for the lifetime of the script in which they're created.

You may have noticed in Chapter 4 that we used the `$POSTED` flag to indicate if anything had been posted:

```
if (_$POST) $POSTED = TRUE;

if ($POSTED) { ...
```

This is set in the `handlers.php` file. However, as with some of the other functions we've created, it's easy to forget after a while where we defined this variable. This becomes evident when you return to the project after six months. Could we perhaps have done without it?

As we progressed through the steps required to build our application, we noticed that we were creating a potential for bugs to creep in by employing user-defined arrays, rather than just using the column names from the database. In the later stages, you will have seen that we changed this, and the code became a lot clearer and required less documentation. Also, it was less susceptible to nothing appearing on the page when someone accidentally typed 'Name' in the wrong case:

```
while(list($a,$b,$c) = $rsTmp->fetchRow(DB_FETCHMODE_ORDERED)) {
    $return[$i]['App Name'] = $a;
    $return[$i]['App Path'] = $b;
}
```

In the following example, you can see that we have created a temporary `$ret` variable, yet we do not do anything with it except return it:

```
if ($something) {
    $ret = TRUE;
} else {
    $ret = FALSE;
}
return $ret;
```

When you consider the performance of the application, one of the aims is to reduce both code and temporary variables. If this code is called a lot, then simply returning directly could double the speed of the operation.

A considerable amount of our code is concerned with SQL statements, and returning the results. It is likely that some form of query-building tool may reduce our code size considerably and hence reduce the time to develop:

```
$sql = "SELECT
            account_id,
            order_total_price,
            order_cc_number,
            order_cc_exp_dt,
            order_ship_dt,
            order_pfp_confirm,
            status,
            created_dt
        FROM
            ".PREFIX."_orders
        WHERE
            order_id = '".$this->_iOrderId."'
        ";
```

In the following example:

```
header("Location: ".SELF);
```

our page redirects to itself; however, you cannot be certain where the constant `SELF` is actually defined. Our application uses constants as configuration items. To change the settings of the program, we must edit the file where all these are defined. Are there other alternatives like XML configuration or `.ini` files that could be used?

The above example is also good for demonstrating that at times using a constant is unnecessary and it's better just to use something that's already supplied by the environment (`$PHP_SELF`, for example).

As mentioned at the beginning of the section, refactoring is not an exact science; the choices you make as a programmer to use specific methods like `strcmp()`:

```
if (!strcmp($op, "deact")) {
```

where an '==' or a 'switch/case' might work are really a mix of personal preference, experience, and style.

Hopefully we have given you an idea of what could be done to the PHP code here to improve it. But at the end of the day, when you write your application, you will end up with the same situation. It's always possible to improve what you've written, but at some point you just have to stop and say 'this works', and realize that making more changes has diminishing returns.

Strike a balance between elegance, efficiency, reusability, and simply getting the job done.

Pushing MySQL to the Edge

As you can see from the web site we have developed, we are making great use of MySQL, in almost every aspect of the site. However we have only scratched the surface of what MySQL can do.

In the early chapters we started using table locking to ensure that when we added a record, other processes were temporarily prevented from writing. However, the latest versions of MySQL now support full rollback and transactions. Could we have made use of this in the shopping cart system?

Using this in conjunction with replication, we can start to think about preventing an avalanche of e-mails in the morning from DVD enthusiasts wondering why they can't get to the site, or worse still, wondering if you are still in business and will be delivering their orders.

We have also skipped over the more precise details of database optimization; in some instances we have deliberately done a few more queries than are necessary just to make the code more readable.

A lot of this is explored in the *MySQL Transactions and Replication Handbook* from *Apress* (*ISBN 1-861008-38-4*). Although we focus on MySQL, most of the code in this book will work just as well on other databases, like PostgreSQL, Oracle, or Microsoft SQL server.

Getting Too Busy

Once the web site is set up, and if it's fairly user-friendly, people start visiting and joining in. Membership and traffic grow, and we spend more time adding lots of cool new features to the site. At some point, you'll consider listing on NASDAQ, but that's for another book. Meanwhile, you have to work out how to deal with this increasing load. Your server is overheating and you are losing potential sales.

Apart from using MySQL, you'll need to look at page caching, usually by working out which pages are really dynamic, and which ones can be stored and updated only upon the occurrence of a relevant event occurring, like somebody adding a comment to a forum. Again have a look around PEAR and you will see a few examples.

Apache also offers features to help us out. Reverse proxies enable us to use multiple servers to do the CPU-intensive work, and to use one machine relay the output to the user. The *Professional Apache 2* book from *Apress* (*ISBN 1-861007-22-1*) covers a lot of the tricks on how to make the most out of Apache.

There are also a number of PHP accelerators available, and we can take our pick from Zend, Ioncube, or APC. All these tools turn the PHP code into bytecode (like Java), and some do optimizations to make your code faster still:

- Zend Performance Suite (http://www.zend.com/)
- IonCube PHP Accelerator (http://www.ioncube.com/)
- APC Alternative PHP Cache (http://apc.communityconnect.com/)

More XML

We've illustrated the use of XML for syndicating news content and employing it in conjunction with XSLT for creating templates, but this only touches the surface, and there are many more ways in which we could have used XML. Here are just a few ideas:

- Configuration
  It's becoming quite common to see XML configuration files offering a very flexible method to store and retrieve settings for your site. There are PEAR libraries to support these as well.
- Importing
  It's time-consuming to type in all the new DVD titles available into your database. Your DVD wholesale supplier will be only too happy to provide you with an inventory for you to offer on your site, and these days the best way to get the data is in XML format.
- Exporting
  In most places, you will need to pay taxes on all the income generated from your DVD site. Usually this involves working with an accountant or some kind of accounting package. These days importing of XML-based accounting data is becoming a common feature of modern accounting packages.
- Point of Sale (POS) systems
  Perhaps you run a small 'real shop' alongside your DVD web site. If you don't want to have two inventory systems, one for the Web and one for the shop, you could use XML RPC or SOAP so the POS system can talk to your DVD site. You may even want to use PHP-GTK on the POS.

As you can see, there are a multitude of uses for XML in our site that we have yet to explore, and the following references may prove worthwhile:

- XML @ W3C (http://www.w3.org/XML/)
- XMLHack (http://www.xmlhack.com/)
- XML-RPC home page (http://www.xmlrpc.com/)
- *PHP 4 XML* from *Apress Press* (*ISBN 1-861007-21-3*)
- ActiveState's Simple Web Services API (SWAPI) for PHP (http://aspn.activestate.com/ASPN/WebServices/SWSAPI/phptut)
- UDDI.org (http://www.uddi.org)
  Universal Description Discovery and Integration – for Web Services publication and discovery
- phpUDDI (http://phpuddi.sourceforge.net/)
  PHP classes for Web Services Discovery
- W3C's Web Services Activity Page (http://www.w3.org/2002/ws/)
- *PHP Web Services* from *Apress Press* (*ISBN 1-861008-07-4*)
- Resource Definition Format @ W3C (http://www.w3.org/RDF/)
- RSS Specs (http://web.resource.org/rss/1.0/ and http://backend.userland.com/rss/)

Get Building

We have come out of this book with a basic methodology for creating a dynamic web site using PHP and MySQL, along with a knowledge of why we built the site the way we did. We can easily borrow these techniques to create a PHP web site from design to deployment. However, it is by no means the end of the road. On the other hand, it is a milestone for the road ahead.

The next step is to build your own site. This book will have given you a framework and some modules to use or modify. Now you just need to tailor the modules and templates that we've provided to fit your own needs.

We also hope that you will build your own modules in the framework we created here. A lot of our design work went into providing new modules that are easy to add and modify – and you will be able to link your modules to our central accounts system, modify our header and footer controls, or add some of the ideas discussed in refactoring. This book will thus serve as a reference of how similar projects are built, so you can have a better understanding of how to contribute to this and many other projects, with the aim of rapidly delivering your own web site.

Join the Community

No developer works in isolation, nor would we expect you to. There are many online communities where you can get assistance with obtaining, learning, and improving your skills with the Web technologies discussed in this book. We provide one in conjunction with this book.

Our DVD Life application includes a discussion forum. Although this forum was built for the purpose of discussing DVD-related news, we could have book-specific discussions emerging here in the form of posts and replies from readers and the authors themselves (http://apress.mediatemple.net/site/forums/).

You'll be able to find solutions there to your problems and to share ideas. Also, if you're looking for modules, adapted from those in the book or using our framework, you can find out if someone else hasn't already written the one you need, or can help you do so yourself. You could also tell everybody about the sites that you've developed.

Through the forum at DVD Life we hope to build up a list of the best web sites built with the help of this book. If you do something really impressive, we might even ask you to write for us about it!

Mailing Lists

PHP already has a huge following, as you can see by browsing the various PHP mailing lists. Sign up for the php-general and the pear-general lists at php.net, and the PEAR developer's list to get involved in the development of PEAR itself.

These groups are full of helpful people with similar interests and extensive experience:

- Sign up to a PHP Mailing list (http://www.php.net/mailing-lists.php)
- Search the archives (http://marc.theaimsgroup.com/)

Is There More to Come?

Yes! We are already planning a sequel to this book, which we're tentatively calling 'PHP E-Commerce Problem – Design – Solution', that will be written with the intent of showcasing the key design and development issues for applying PHP to e-commerce solutions. By using an ongoing case study we will cover all the common issues relating to e-commerce and impart developer know-how specific to solving problems in this area of application development. We encourage your valuable feedback and inputs into this sequel, by making relevant posts at DVD Life.