Model View Controller(MVC) in PHP

By admin in Patterns 149 Comments

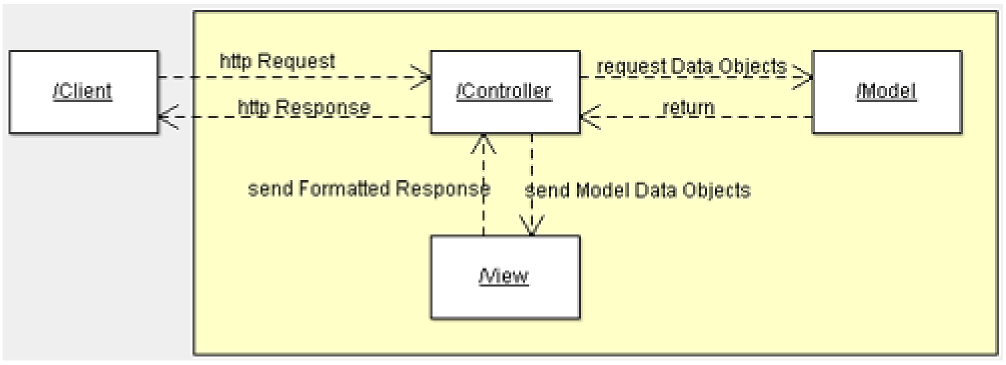
The model view controller pattern is the most used pattern for today's world web applications. It has been used for the first time in Smalltalk and then adopted and popularized by Java. At present there are more than a dozen PHP web frameworks based on MVC pattern.

Despite the fact that the MVC pattern is very popular in PHP, is hard to find a proper tutorial accompanied by a simple source code example. That is the purpose of this tutorial.

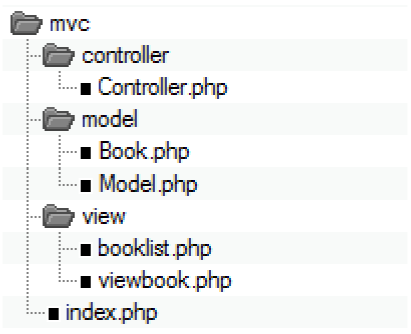
The MVC pattern separates an application in 3 modules: Model, View and Controller:

- The **model** is responsible to manage the data; it stores and retrieves entities used by an application, usually from a database, and contains the logic implemented by the application.
- The **view (presentation)** is responsible to display the data provided by the model in a specific format. It has a similar usage with the template modules present in some popular web applications, like wordpress, joomla, …
- The **controller** handles the model and view layers to work together. The controller receives a request from the client, invokes the model to perform the requested operations and sends the data to the View. The view formats the data to be presented to the user, in a web application as an html output.

The above figure contains the MVC Collaboration Diagram, where the links and dependencies between figures can be observed:



Our short php example has a simple structure, putting each MVC module in one folder:



```
mvc
  controller
    Controller.php
  model
    Book.php
    Model.php
  view
    booklist.php
    viewbook.php
  index.php
```

Controller

The controller is the first thing which takes a request, parses it, initializes and invoke the model and takes the model response and sends it to the presentation layer. It’s practically the liant between the Model and the View, a small framework where Model and View are plugged in. In our naive php implementation the controller is implemented by only one class, named unexpectedly controller. The application entry point will be index.php. The index php file will delegate all the requests to the controller:

view plaincopy to clipboardprint?

```
// index.php file
include_once("controller/Controller.php");

$controller = new Controller();
$controller->invoke();
```

Our Controller class has only one function and the constructor. The constructor instantiates a model class and when a request is done, the controller decides which data is required from the model. Then it calls the model class to retrieve the data. After that it calls the corresponding passing the data coming from the model. The code is extremely simple. Note that the controller does not know anything about the database or about how the page is generated.

view plaincopy to clipboardprint?

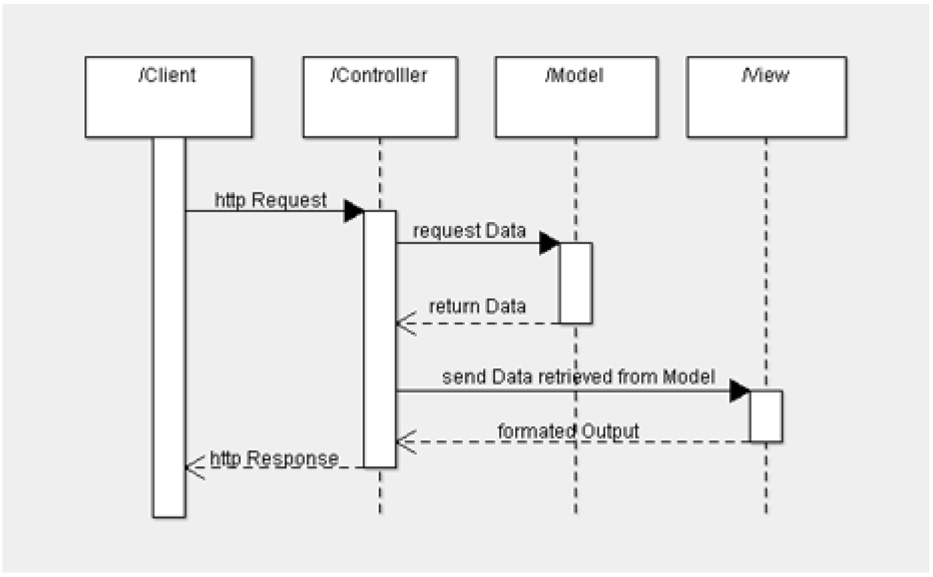
```
include_once("model/Model.php");

class Controller {
    public $model;

    public function __construct()
    {
        $this->model = new Model();
    }

    public function invoke()
    {
        if (!isset($_GET['book']))
        {
            // no special book is requested, we'll show a list of all available books
            $books = $this->model->getBookList();
            include 'view/booklist.php';
        }
        else
        {
            // show the requested book
            $book = $this->model->getBook($_GET['book']);
            include 'view/viewbook.php';
        }
    }
}
```

In the following MVC Sequence Diagram you can observe the flow during a http request:



Model and Entity Classes

The Model represents the data and the logic of an application, what many calls business logic. Usually, it’s responsible for:

- storing, deleting, updating the application data. Generally it includes the database operations, but implementing the same operations invoking external web services or APIs is not an unusual at all.
- encapsulating the application logic. This is the layer that should implement all the logic of the application. The most common mistakes are to implement application logic operations inside the controller or the view(presentation) layer.

In our example the model is represented by 2 classes: the “Model” class and a “Book” class. The model doesn’t need any other presentation. The “Book” class is an entity class. This class should be exposed to the View layer and represents the format exported by the Model view. In a good implementation of the MVC pattern only entity classes should be exposed by the model and they should not encapsulate any business logic. Their solely purpose is to keep data. Depending on implementation Entity objects can be replaced by xml or json chunk of data. In the above snippet you can notice how Model is returning a specific book, or a list of all available books:

view plaincopy to clipboardprint?

```
include_once("model/Book.php");

class Model {
    public function getBookList()
    {
        // here goes some hardcoded values to simulate the database
        return array(
            "Jungle Book" => new Book("Jungle Book", "R. Kipling", "A classic book."),
            "Moonwalker" => new Book("Moonwalker", "J. Walker", ""),
            "PHP for Dummies" => new Book("PHP for Dummies", "Some Smart Guy", "")
        );
    }

    public function getBook($title)
    {
        // we use the previous function to get all the books and then we return the requested one.
        // in a real life scenario this will be done through a db select command
        $allBooks = $this->getBookList();
        return $allBooks[$title];
    }


}
```

In our example the model layer includes the Book class. In a real scenario, the model will include all the entities and the classes to persist data into the database, and the classes encapsulating the business logic.

view plaincopy to clipboardprint?

```
class Book {
    public $title;
    public $author;
    public $description;

    public function __construct($title, $author, $description)
    {
        $this->title = $title;
        $this->author = $author;
        $this->description = $description;
    }
}
```

View (Presentation)

The view(presentation layer)is responsible for formating the data received from the model in a form accessible to the user. The data can come in different formats from the model: simple objects(sometimes called Value Objects), xml structures, json, …

The view should not be confused to the template mechanism sometimes they work in the same manner and address similar issues. Both will reduce the dependency of the presentation layer of from rest of the system and separates the presentation elements(html) from the code. The controller delegates the data from the model to a specific view element, usually associated to the main entity in the model. For example the operation “display account” will be associated to a “display account” view. The view layer can use a template system to render the html pages. The template mechanism can reuse specific parts of the page: header, menus, footer, lists and tables, …. Speaking in the context of the MVC pattern

In our example the view contains only 2 files one for displaying one book and the other one for displaying a list of books.

viewbook.php

view plaincopy to clipboardprint?

```
<html>
<head></head>

<body>

   <?php

       echo 'Title:' . $book->title . '<br/>';
       echo 'Author:' . $book->author . '<br/>';
       echo 'Description:' . $book->description . '<br/>';

   ?>

</body>
</html>
```

booklist.php

view plaincopy to clipboardprint?

```
<html>
<head></head>

<body>

   <table>
       <tbody><tr><td>Title</td><td>Author</td><td>Description</td></tr></tbody>
       <?php

           foreach ($books as $title => $book)
           {
               echo '<tr><td><a href="index.php?book='.$book->title.'">'.$book->title.'</a></td><td>'.$book->author.'</td><td>'.$book->description.'</td></tr>';
           }

       ?>
   </table>

</body>
</html>
```

The above example is a simplified implementation in PHP. Most of the PHP web frameworks based on MVC have similar implementations, in a much better shape. However, the possibility of MVC pattern are endless. For example different layers can be implemented in different languages or distributed on different machines. AJAX applications can implements the View layer directly in Javascript in the browser, invoking JSON services. The controller can be partially implemented on client, partially on server…

This post should not be ended before enumerating the advantages of Model View Controller pattern:

- the Model and View are separated, making the application more flexible.
- the Model and view can be changed separately, or replaced. For example a web application can be transformed in a smart client application just by writing a new View module, or an application can use web services in the backend instead of a database, just replacing the model module.
- each module can be tested and debugged separately.

The files are available for download as a zip from http://sourceforge.net/projects/mvc-php/files/mvc.zip/download