Annals of "Dunarea de Jos" University of Galati
Fascicle I. Economics and Applied Informatics
Years XXI – nº3/2015
ISSN-L 1584-0409 ISSN-Online 2344-441X
www.eia.feaa.ugal.ro

Web Application Frameworks

Maria Cristina ENACHE*

ARTICLE INFO

Article history:
Accepted December 2015
Available online December 2015
JEL Classification
M12

Keywords:
Frameworks, e-commerce, Model, View

ABSTRACT

As modern browsers become more powerful with rich features, building full-blown web applications in JavaScript is not only feasible, but increasingly popular. Based on trends on HTTP Archive, deployed JavaScript code size has grown 45% over the course of the year. MVC offers architectural benefits over standard JavaScript — it helps you write better organized, and therefore more maintainable code. This pattern has been used and extensively tested over multiple languages and generations of programmers. It's no coincidence that many of the most popular web programming frameworks also encapsulate MVC principles: Django, Ruby on Rails, CakePHP, Struts, or Laravell.

1. Introduction

MVC is a model of development (design pattern) which proposed decomposition and demarcation application in three interdependent components: Model, View, controller. The model of development MVC is applicable in any field but it is used mainly in the sphere web applications.

As the name suggests, a controller is the component that controls how a request is handled, what components do what tasks, and what response is sent back. That's its job. Querying the DB, processing raw result sets and deleting data from certain tables is not part of a controllers' job description.

The model is provided with all logic of the business of the entity which it reflects. The model makes the connection with the environment by the persistence of the system in the collection and storage of data, which it receives/supplies back to controller. It is preferable that the model to process and to provide the data controller in the form of specific data structures programming language and not to add unnecessary complexity information.

View is part of the presentation of information in a format user specific, easy-to-understand/processed by it. It receives from controller all the data it needs so that it is no longer any need for any other calls functionality from the model to the other components. Suite of entities that they are working with view are only specific objects GUI interface of type: Edit fields, buttons, links URL, etc.

A perfect demarcation between the three components is not always possible, and deployment in various framework-changer which varied significantly.

The MVC concepts are a little abstract, it's true, but it's an incredibly common pattern. It is literally all around you. In fact, you're looking at MVC right now.

Model = HTML

```
<div id="logo" class="png" style="height:125px;width:50%;">
<h1><a href="http://www.feaa.ugal.ro/" style="height:125px;" title="Site-ul Facultăţii de Economie şi Administrarea Afacerilor">Facultatea de Economie şi Administrarea Afacerilor</a>
</h1>
</div>
```

View = CSS

```
<style type="text/css">
#logo {align:center;
background: rgba(255, 255, 255, 0)
}
.png {background:url ("img/logo.png")}
</style>
```

*"Dunarea de Jos" University of Galati, Romania. E-mail addresse: mpodoleanu@ugal.ro (M. C. Enache)

Controller = Browser

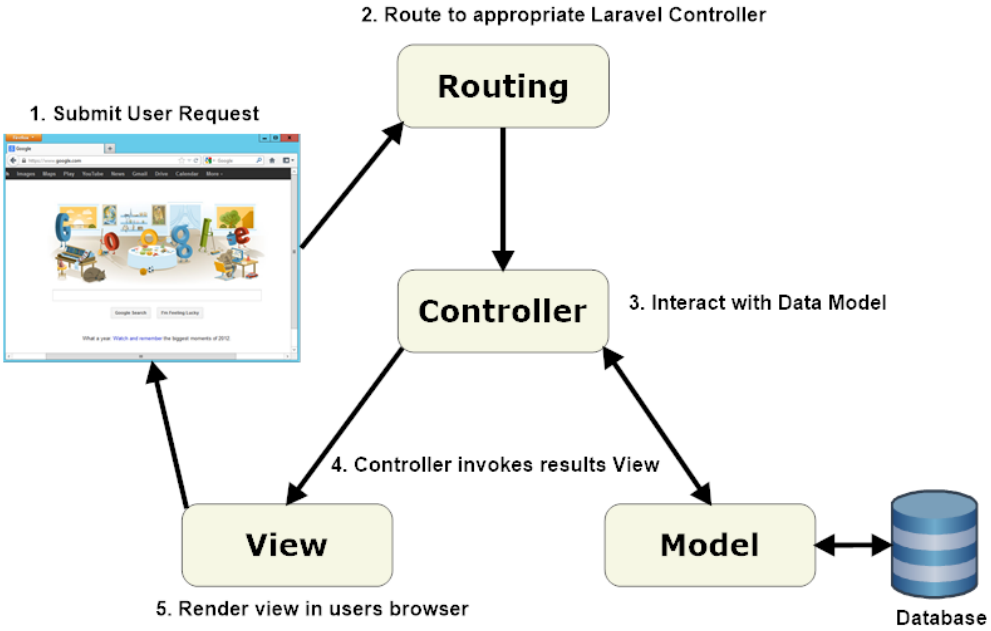
Model - The HTML is the "skeleton" of bedrock content and text that communicates information to the reader.

View - The CSS adds visual style to the content. It is the "skin" that we use to flesh out our skeleton and give it a particular look. We can swap in different skins via CSS without altering the original content in any way. They are relatively, but not completely, independent.

Controller - The browser is responsible for combining and rendering the CSS and HTML into a set of final pixels on the screen. It gathers input from the user and marshals it to any JavaScript code necessary for the page to function. But here, too, we have flexibility: we can plug in a different browser and get comparable results. Some browsers might render it faster, or with more fidelity, or with more bells and whistles.

So if you believe the web has been at all successful - most signs I've seen point to yes - then you also have to acknowledge the incredible power of Model-View-Controller.

Interacting with the database is the job of the Model



2. The attributes of a web MVC framework

The modularity - each component a member of framework-independent is an element that can be used as required by the application and which can be linked to other modules of our application or with other applications and framework.

Data security - most of the times the purpose of a web applications is to provide access to certain products and information in a manner based on access rights. So not all users have access to all the information of the system, or not all the information about a product is readily available to all users. For example, a system administrator has access to all the information for a product such as stock, while a purchaser of products

from online store can view the price but not the stock. A framework provides mechanisms for user authentication in the system and to restrict access according to the type which user accesses the system.

Expandability - Although are designed and developed with the aim of serving as fully a certain functionality, framework to solve all the cases and the specific problems that may arise in a specific application, in a given business. That's why engineers who develop such a framework airbags deploy in system structure mechanisms by means of which modules can be extended or replaced completely without speculations of the structure and manner of programming embedded in framework.

Performance - time of execution of the application, the processing time and preparation of the information in the format of your choice to your browser, it is essential for the success of web applications. In the context of 2015 when computing technologies and the transfer of data via the Internet are to unprecedented levels, we think that loading speed of web pages to no longer be a problem. But the volume of data, complexity of the information processed and transferred is increasing, with the increasing internet, with an increase of activities carried out by each of us via the Internet and the ever-increasing expectations to the shape of the presentation of web content.

3. The templating System

A very important of all framework is the templating system. In the case of Laravel this system is called blade and is as simple as strong. Unlike other engines of templating, blade does not impose any restrictions relating to writing PHP code in the views (view). In connection with the templating systems always call into question the appearance performance. In the case of Blade all views are compiled in the form of a code PHP and are saved in the system cache. Cache is refreshed when changes are made a certain views:

- definition of a layout which contains the HTML;
- Define sections using the command @section which can be overcrowded in sub-views(partials);
- Extension layout - is done by command @extends at the beginning of a partial views;
- inclusion of partial views: using command @includes.

```
<div>
  @include('errors')
  <form>
    <!-- conţinutul formularului -->
  </form>
</div>
```

- Display tags **{{<expesie >}}**;
- Control structures **@if**, **@for**, **@foreach**, **@while**:

```
@foreach ($users as $user)
  <p>Nume utilizator: {{ $user->name }}</p>
@endforeach
```

Developing an application very simple, requires a certain amount of time of study framework, especially understanding concepts, mechanisms and paradigm concerned in general. While a knowledge of language remains a fundamental requirement, knowledge of a framework such as Laravel, Symfony or Zend, which are the leaders framework-PHP at this time, he picked up COPS to a whole new level.

Concept of "Dependency Management" becomes more and more an indispensable tool for the community developers of web applications. Laravel uses a multitude of components, modules, libraries, installed and hot-pluggable repository of the Composer. All framework's market leader in this moment I use Composer and can be installed themselves as a dependent of Composer. This system of dependencies provides a well-defined structure and modularity, which are essential for the development under a RAD (Rapid Application Development).

4. Passing data to views

In most applications, not all pages are static. There is often a need to retrieve data from a database or other data source and embed it into the resulting HTML that will be presented to the user. Not so long ago, developers would embed SQL queries directly into the application code that would result in giant unmaintainable mess. For example a page that shows a table containing all order inquiries in a shop might look like this:

```
<?
$catId = (isset($_GET['catId']) && $_GET['catId'] > 0)? $_GET['catId'] : 0;
$sql = "SELECT * FROM tbl_orders, tbl_product, tbl_category
   WHERE tbl_orders.inquiry_pdid = tbl_product.pd_id AND tbl_product.cat_id = tbl_category.cat_id
   AND tbl_category.cat_parent_id = $catId
   ORDER BY tbl_orders.inquiry_date DESC, tbl_category.cat_parent_id ASC";
$result = dbQuery($sql);
?>

<table width="100%" border="0" cellpadding="5" cellspacing="0" class="text">
 <tr id="listTableHeader">
  <td>Inquiry ID </td>
  <td>Batch ID</td>
  <td>Customer</td>
  <td>Brand</td>
  <td>Product</td>
  <td>Date</td>
 </tr>

 <?php if (dbNumRows($result) > 0) {
  while($row = dbFetchAssoc($result)) {
  extract($row);
 ?>
 <tr>
  <td><?php echo $inquiry_id; ?></td>
  <td><?php echo $inquiry_odid; ?></td>
  <td><?php echo $inquiry_batchall; ?></td>
  <td>

  <?php
   $sqlname = "SELECT cat_name FROM tbl_category WHERE cat_id = $cat_parent_id";
   $resultname = dbQuery($sqlname);
   $rowname = dbFetchAssoc($resultname);
   echo $rowname['cat_name'];
  ?>

  </td>
  <td>

<?php echo $cat_name; ?></td>
  <td><?php echo $pd_name; ?></td>
  <td><?php echo formatDate($order_date); ?></td>
 </tr>
<?php } } ?>

</table>
```

Notice how the SQL queries, query parameters and HTML output are all inside of single PHP file making debugging and modification of this code practically impossible.

MVC frameworks like solve these problems by separating the application's logic from the data output (visual representation). MVC frameworks provides a few ways of passing the data retrieved inside of the application to the application's views:

- By using the arguments of "View::make" method
- By using a method "with" appended to "View::make" method

Both of these methods lead to the same result – providing the view template with data that is passed from the application. Let's look at these two methods in more detail.

```
// routes.php
Route::get('login', function()
{
  // Create an array of data that will be used in the view template
  $data = array('currentDateTime' => date('Y-m-d H:i:s'));
  // Pass the data array as the second argument of View::make
  return View::make('login', $data);
});

// File app/views/login.php
Current date and time: <br>

<?php
  // Display the data in the view
  echo($currentDateTime);
?>

<form action="login" method="post">
  User: <input type="text" name="username"><br>
  Parola: <input type="password" name="password">
  <input type="submit" value="Login">
</form>
```

5. Conclusions

The programmer no longer means simply knowledge in one or more languages and both. The developer of web applications at the level of 2015 requires a better knowledge of the concepts, systems and instruments concerned that an application to a level at least medium ad: authentication, authorization, cache, many types of databases, encryption, location, API, etc. many of these are already provided at least by framework's referred to.

Applications interact with users and present data. The separation of interaction, presentation and data is one that comes fairly naturally to an experienced programmer - this doesn't need to attribute itself to the idea of MVC though.

In web applications the term MVC is particularly meaningless and overused. Django was created without MVC in mind, but it's a well designed framework, so it's easy to say, "the views are the controllers and the templates are the views". Rails Controllers have a render method and provide data to the Views, which goes against the Smalltalk MVC of "[the view] gets the data necessary for the presentation from the model by asking questions." If Smalltalk MVC and Rails MVC are fairly different does this mean one is "wrong" and the other is "right"? No, of course not. But adherence to the idea of MVC encourages people to try and force everything into only three types of components. Do permissions get handled by the controller or the model? What about observers/subscribers? There are lots of parts in a web application that don't fit cleanly into MVC.

Framework charts top supplies starting systems which enclose a use optimal Pattern design-known where appropriate, thus ensuring minimum redundancy, disengage, the outline responsibilities which determines scalability, maintainability of application. Last of thus becomes easier to achieve: reduce costs and predictability of the development and maintenance of a web applications.

References

1. *Bower.io. (2015). Bower. Retrieved from Bower Website: http://bower.io/*
2. *c2.com. (2013). Gang of Four. Retrieved from c2.com: http://c2.com/cgi/wiki?GangOfFour*
3. *Composer Website. (fără an). Composer Documentation. Retrieved from Composer Website: https://getcomposer.org/doc/*
4. *DocForge. (2014, June 20). Web application framework. Retrieved from DocForge: http://docforge.com/wiki/Web_application_framework*
5. *Laravel Website. (2015). Retrieved from Laravel Website: http://laravel.com/*
6. *Node.js. (2015). Node.js. Retrieved from Node.js Website: https://nodejs.org/*
7. *Wikipedia. (2015, June 21). Aplication Programming Interface. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Application_programming_interface*
8. *Wikipedia. (2015, April 14). Design Patterns. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Design_Patterns*
9. *Wikipedia. (2015, June 23). Model-View-Controller. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Model-view-controller*
10. *Wikipedia. (2015, June 23). Open Source. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Open_source*
11. *Wikipedia. (2015, May 7). Single responsibility principle. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/ Single _responsibility_principle*
12. *Wikipedia. (2015, Iunie 23). Wikipedia. Retrieved from Representational state transfer: https://en.wikipedia.org/wiki/Representational_state_transfer*
13. *Zend Framework. (2015). Zend Framework 2 Documentation. Retrieved from Zend Framework: http://framework.zend.com/learn/*