Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 69 (2014) 1172 – 1179

Procedia Engineering

www.elsevier.com/locate/procedia

24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013

Designing an MVC Model for Rapid Web Application Development

Dragos-Paul Pop*, Adam Altar

Romanian-American University, 1b Expozitiei Blvd., Bucharest, 012101, Romania

Abstract

In this paper, we present a model for rapid web application development. This model is based on the Model-View-Controller architecture (MVC) and has several other useful components like security, form generation and validation, database access and routing. This model was implemented using the PHP programming language, but it can be implemented in other development languages and environments using the same concepts. Improvements in both development and maintenance time have been the main objectives of this research, with the added benefit of correct and maintainable code.

Keywords: model; view; controller; MVC; web application; design pattern

1. Introduction

Web application development has come a long way since the beginning of the World Wide Web. A myriad of technologies and programming languages are now used to build web applications, but because of the way the World Wide Web evolved and because of the speed at which it did so these technologies didn't really have time to evolve and cope with the pace. Many players tried to come up with different and exotic technologies to mainly improve the user experience and help developers build faster and more powerful web applications. Some of these technologies have played an important role in web development but have seen an important drop in usage over the last years, like Java Applets and Microsoft Silverlight. On the other hand many technologies have evolved from simple toys to powerful and important parts of todays' web ecosystem, like JavaScript, Flash and XML.

* Corresponding author. Tel.: +40-724-261-805;
E-mail address:pop.dragos.paul@profesor.rau.ro

doi:10.1016/j.proeng.2014.03.106

The web environment today uses HTML and CSS to present data to users and interaction is accomplished via JavaScript. These technologies are called “front-end” or “client-side” technologies. On the other hand, “back-end” or “server-side” technologies refer to data storage and processing technologies.

2. Problem formulation

Front end and back end technologies come together to build web applications, but because the World Wide Web has evolved at such a fast pace and because developers need to use a rather large number of technologies to build just a single web application, the result of their work is oftentimes difficult to maintain and fix.

Developers combine HTML code with server side programming languages to create dynamic web pages and applications and this leads to highly entangled and unmaintainable code.

Another problem has grown out of the fact that web technologies are increasingly used to build all sorts of complex applications. Microsoft embraces web technologies to encourage developers to build applications for its’ latest operating system, Windows 8. Also, a lot of frameworks exist that help web developers write applications for mobile devices, like PhoneGap and Appcelerator Titanium. Even more, a mobile operating system is in the works and devices are expect later this year that uses web technologies entirely for the developer API (Mozilla Firefox OS). For these reasons a web application is generally created by a whole team of specialized developers, each working with their favorite technology, like HTML and CSS for the presentation layer, JavaScript for client-side interaction, PHP (or ASP, Java, Python, Pearl, Ruby, etc.) for server-side logic and MySQL (or Oracle Database, Microsoft SQL Server, etc.) for data storage and management.

Each of these specialist needs to work with their colleagues in such a way that their code pieces fit inside the overall design of the application. For example, the client-side (data presentation) developer needs to alter the HTML and CSS code in such a way that he doesn’t break the server-side developers’ code that resides in the same file. Also, when a database developer alters the schema for an application the server-side developer may need to change a lot of code to make the application work.

The important thing to note here is that there is an acute need to separate presentation from logic and data storage in an application.

There are some application design paradigms and patterns that offer solutions to this problem, but the focus nowadays is on the MVC pattern.

In conducting the research we have used our experience is building web applications with various systems and frameworks and we have tried to identify both strengths and weaknesses of these systems while providing our own view on how these practices could be improved. Studied frameworks and systems include: Symfony, CakePHP, CodeIgniter Zend Framework, Laravel, Fuel PHP, Ruby on Rails and ASP.NET MVC.

3. The MVC pattern and literature overview

In this section we will review the current status of the research in this field and take a look at the literature behind the MVC pattern, describing the main functional parts of the pattern.

The MVC design pattern was first envisioned by Trygve Reenskaug in the 1970s at the Xerox Parc. According to him, “the essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer”. [1]

Later on, in 1988, the MVC paradigm was described in detail by Krasner and Pope in their article “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”, published in the Journal of Object-Oriented Programming.

They stress out that there are enormous benefits to be had if one builds applications with modularity in mind. “Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit without having to know everything about the other units.” [2]

An application is divided into three main categories: the model of the main application domain, the presentation of data in that model and user interaction. [2]

The MVC pattern splits responsibilities into three main roles thus allowing for more efficient collaboration. [3] These main roles are development, design and integration.

The development role is taken on by experienced programmers that are responsible for the logic of the application. They take care of data querying, validation, processing and more.

The design role is for the developers that are responsible for the application look and feel. They display data that is fed from the developers working on the first role.

The integration role gathers developers with the responsibility to glue the work of the previous two roles. [3]

The MVC design pattern is such a good fit for web application development because they combine several technologies usually split into a set of layers. Also, MVC specific behavior could be to send specific views to different types of user-agents. [13]

"User interaction with an MVC application follows a natural cycle: the user takes an action, and in response the application changes its data model and delivers an updated view to the user. And then the cycle repeats. This is a very convenient fit for web applications delivered as a series of HTTP requests and responses." [4]

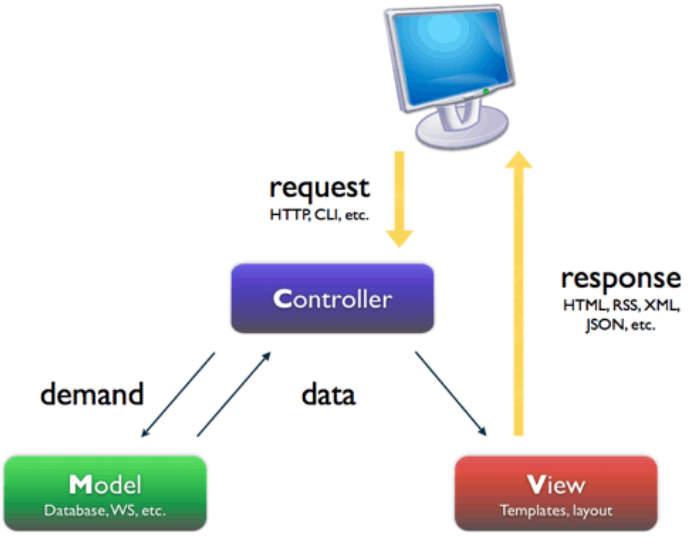

Fig. 1 The MVC pattern.

Source: http://stackoverflow.com/questions/5966905/which-mvc-diagram-is-correct-web-app

3.1. The Model

The Model is the part of the system that manages all tasks related to data: validation, session state and control, data source structure (database). The Model greatly reduces the complexity of the code the developer needs to write. [5]

The Model layer is responsible with the business logic of an application. It will encapsulate methods to access data (databases, files, etc.) and will make a reusable class library available. Usually a Model is built with data abstraction in mind, validation and authentication. [4]

Moreover, the Model is made up of classes that define the domain of interest. These objects that belong to the domain often times encapsulate data that is stored in databases, but also include code that is used to manipulate this data and enforce business rules. [6]

As a conclusion, the Model mainly handles data access abstraction and validation. The Model holds methods for interaction with different data sources.

We believe in the Thin Model approach, which says that a model should be kept as simple as possible, only encompassing data processing that is strictly tied to the real life object that is modeled. The Thin Model is coupled with a Fat Controller, which houses most of the data processing required by the application. This way, the models become highly reusable between applications and most of a developer's work is kept inside controllers.

The novelty our Model system brings to the MVC world is a system for versioning control, based on the idea of migrations, that doesn't only keep track of data structure but of data itself. The system uses xml files to store data between migrations, making the versioning process for databases an easily accomplishable thing.

3.2. The View

The View is responsible with graphical user interface management. This means all forms, buttons, graphic elements and all other HTML elements that are inside the application. Views can also be used to generate RSS content for aggregators or Flash presentations. By separating the design of the application from the logic of the application we greatly reduce the risk of errors showing up when the designer decides to alter the interface of that application by changing a logo or a table. In the same time, the developers' job is greatly reduced because he no longer needs to see HTML code elements, design elements and graphical elements. [5]

The View layer is what can normally be called web design or templates. It controls the way data is displayed and how the user interacts with it. It also provides ways for data gathering from the users. The technologies that are mainly used in views are HTML, CSS and JavaScript. [4]

As a general rule, a view should never contain elements that belong to application logic, in order to make it easier for the designer to work with it. This means logical blocks should be kept at a minimum.

Most web application frameworks today use some sort of template engine that takes advantage of generator elements in order to keep HTML code at a minimum and reduce the risk of typing mistakes. These generators are usually used to make complex web partials, like forms, tables, lists, menus and so on. The problem we identified in this case is that all implementations of this idea use behind the scenes or opaque generation of partials. This way, a front-end developer can only see the resulting code after it's been generated, having little to no way of modifying the template. Our system uses special HTML comments to insert and generate partials based on templates that are also HTML-only files. The pre-processing system uses the comments to interpret special commands in order to insert data into the templates. This makes the whole process very transparent for the front-end developer, who can see the whole markup before rendering the view.

3.3. The Controller

The Controller is responsible for event handling. These events can be triggered by either a user interacting with the application or by a system process. A controller accepts requests and prepares the data for a response. It is also responsible with establishing the format of that response. The Controller interacts with the Model in order to retrieve the needed data and generates the View. This process is also known as an action or a verb. [5] When a request arrives at the server, the MVC framework dispatches it to a method in a controller based on the URL. [14]

The Controller binds all application logic and combines the display in the View with the functionality in the Model. It is responsible with data retrieval from the View and with establishing the execution path for the application. The Controller will access the Model functionality and it will interpret the data received so that it can be displayed by the View. It is also responsible with error handling. [4]

A Controller manages the relationship between a View and a Model. It responds to user requests, interacts with the Model and decides which View should be generated and displayed. [6]

As mentioned above, we embrace the Fat Controller approach, believing that all application-specific data processing should be handled at the controller level.

Our controller system is RESTful and supports JSON(P), text and XML data formats for requests and responses.

4. Database abstraction

In object oriented programming applications are built using objects. These objects reflect real life objects and they contain both data and behavior. The relational model that is widely used for data storage in applications uses tables to store data and data manipulation languages to interact with that data. Some database management systems have object oriented features, but they are not fully compatible. It is very clear that these two architectures are widely used and will be for a long time from now. Moreover, the relational model and object oriented programming

are used together most of the time to build applications of every scale. But the way the two technologies combine is far from perfect. The term used for this mismatch is "object-relational impedance mismatch". [7]

The reason behind this mismatch is that the two technologies rely on different concepts. Object oriented programming relies on proved programming concepts, while the relational models follows mathematical principles.

Object-relational mapping systems were developed solely to address this issue.

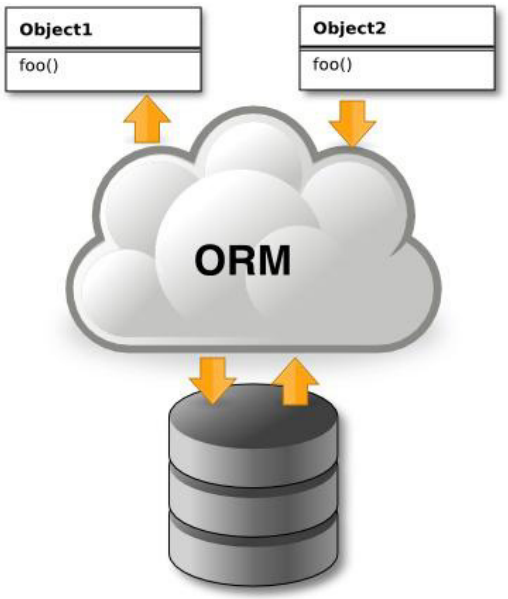

Fig. 2 ORM System.

Source:http://danhartshorn.com/2011/12/object-relational-mapping-wikipedia-the-free-encyclopedia

An object-relational mapping system (ORM) is defined as being a tool that provides a methodology and a mechanism for object oriented systems to store data in safe way and for a long period of time in a database, having transactional control over them, but being expressed, if needed, as objects inside the application. [8]

An ORM system frees the developer from the worry of knowing the database structure or schema. Data access is done using a conceptual model that reflects the developers' business objects. [9]

In conclusion, we can say that an object-relational mapping system is a way to correlate data from a relational system with classes in a programming language, allowing the developer to work with well-known concepts from object oriented programming to manage data from that relational system.

An MVC platform must provide developers with a way to interact with a database management system to store and retrieve data. Most of the times, this is an integrated ORM system.

Because in the MVC architecture the Model is the layer that interacts with data, the ORM system is hidden behind it. Basically, each Model is connected to the ORM systems and it uses the system to interact with data. But developers may access the resources of the ORM system without the need to use a Model.

Our ORM system supports many-to-many relations between objects and has a setting for maximum depth level for object linkage.

Also, we provide an option to keep certain data sets in memory and skip database querying, reducing response times at a minimum.

As far as NoSQL data stores are concerned, we support an interface to access a number of such systems. This interface is part of the ORM system.

5. Security

Web application vulnerabilities are the main cause of attack on users from the business environment. [10]
There are a lot of things developers need to be aware when building web applications.
Security risks can be divided into several categories: [11]

- User input validation: buffer overflow, cross-site scripting, SQL injection, canonicalization;
- Authentication: network sniffing, brute-force attacks, dictionary attacks, cookie faking, identity theft;
- Authorization: access privilege violation, private data display, data altering, stalking attacks;
- Configuration management: unauthorized access to the management interface, unauthorized access to configuration zones, access to configuration data stored as text, lack of action logging;
- Sensible information: access to sensible data in the database, network sniffing, data altering;
- Session management: session theft, session altering, man in the middle attacks;
- Encryption: weak security keys, weak encryption;
- Parameter manipulation: query string, form field, cookie and HTTP header manipulation;
- Exception management: DOS attacks;

Developers need to defend against all of these types of attacks and security needs to be handled at every layer in the MVC architecture.

Our system proposal has built in CSRF attack prevention, a simple authentication schema, a complex role based security system and highly customizable access control lists for resources.

6. Routing

The way the HTTP protocol handles URL writing is very similar to the way resource paths are written in the Unix environment. [12]

Web servers bear this in mind and implicitly look for data in a hierarchical file system.

This way of accessing resources is relatively easy and intuitive, but as web application complexity grows, the need for a better system arises. To solve this, we can configure the web server and the development framework in such a way that it will interpret results in a unique manner and access the resources.

These systems are called URL Mapping or URL Routing systems. The technique is also known as URL Beautifying and helps developers build ordered and nice URLs data both users and search engines can handle better.

Our system is built on the “controller/action/params” paradigm, but it also allows developers to configure their own static routes.

Access control is done at the routing level in order to reduce unneeded processing time.

7. Rapid prototyping

In the world of web application development a lot of application parts seem to be used multiple times. For example web forms are a tool that is used really often. So are tables and lists. But building such elements from scratch is rather difficult many times, because the involve writing a lot of complex markup.

Each system can benefit from a rapid prototyping module that not only generates these tools quickly, but also provides ways to validate data (for forms).

Such tools are really useful when combined with ORM systems to build CRUD systems from the ground up with very few lines of code.

As stated in the section on Views above, our system is based on a flexible and fully transparent template engine, that comes to benefit both back-end and front-end developers alike.

8. System proposal overview

Our platform proposal unifies all of the above aspects in one single package that is robust and easy to use and maintain.

The structure above can be described as follows. The data source can be any type of database. The ORM system is a set of classes built specifically for most well-known database management systems. One main class provides basic mapping, relations and management for a general type of table in the desired DBMS. Relations should be mapped as to allow fetching of related data in an object oriented way, without the need for complex queries written by the developer. Both eager and lazy loading of data should be considered.

Models are built as children of the main ORM class inheriting functionality from that main class. In addition, Models are enriched with unique capabilities that describe the real life object that should be modeled.

The Controller is a base class that handles Model querying, access security and route resolving. Controllers are built as children of the base Controller and inherit the functionality and are enriched with application logic corresponding with the desired actions.

The Rapid Prototyping system is as set of classes that allow developers to build complex HTML objects like forms and tables without the need to write HTML code. Forms should provide the possibility of data validation and tables that of sorting and nesting. The Rapid Prototyping system should provide DOM-like manipulation capabilities in order to access the data.

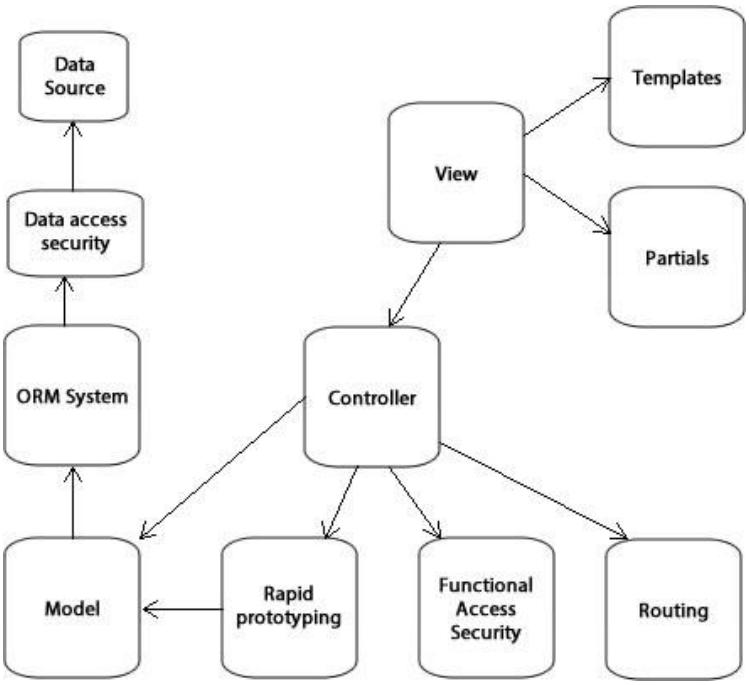

Fig. 3. MVC structure proposal.

The View is a base class containing all display logic. Views are built as children of this main class and should be tied to a specific action and controller. Views should be pre-rendered for performance reasons and should be capable of caching. The View system can depend of a Template system and a Partials system for code reuse.

9. Conclusion

Based on the proposal formulated above, we have built a solid framework that is capable of reducing web application development times drastically, allowing developers to focus on application specific tasks, rather than wasting time trying to implement well-known patterns and practices.

Web technologies have come a long way in little time and the market for web applications is ever growing. Our system helps developers improve their work speed and quality and provides a nice working framework and platform for both beginners and experienced users.

We have built a number of applications using components form the platform proposition and these applications showthat the system we are building is on the right track, while giving us valuable input on things that need to be improved.

10. Further research

The focus of our future research in this area is improving support for the various NoSQL systems that are available, because we think this area is of big future interest for the IT community. Also, we aim at improving the speed of the rendering engine by incorporating caching techniques for data and views.

Another important step is launching the project in the open source community in order to build more test case applications that generate valid results. These results will be used to further optimize the platform. Also, we aim to support packages built by the community in order to further extend the platform.

Acknowledgements

This work was co-financed from the European Social Fund through Sectorial Operational Program Human Resources Development 2007-2013, projects POSDRU/107/1.5/S/77213 and POSDRU/88/1.2/S/55287 „Ph.D. for a career in interdisciplinary economic research at the European standards”.

References

[1] Trygve Reenska, http:// heim.ifi.uio.no / ~trygver/themes/mvc/mvc-index.html.
[2] Glenn E. Krasner, Stephen T. Pope, “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”, Journal of Object-Oriented Programming, vol. 1, no. 3, 1988, pp. 26-49.
[3] Kevin McArthur, Pro PHP: Patterns, Frameworks, Testing and More, Apress, 2008.
[4] A Freeman, S Sanderson, Pro ASP.NET MVC 3 Framework, Apress, 2011.
[5] W. J. Gilmore, Easy PHP Websites, Columbus, Ohio: W.J. Gilmore, LLC, 2009.
[6] J. Galloway, P. Haack, B. Wilson şi K. S. Allen, Professional ASP.NET MVC 3, John Wiley & Sons, Inc., 2011.
[7] http://www.agiledata.org/essays/impedanceMismatch.html
[8] E. J. O'Neil, Object/relational mapping 2008: hibernate and the entity data model (edm), Proceedings of the 2008 ACM SIGMOD international conference on Management of data
[9] J. Lerman, Programming Entity Framework, O’Reilly, 2010.
[10] http://www.rapid7.com/solutions/technology/web-application-security.jsp
[11] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan, Improving Web Application Security: Threats and Countermeasures, Microsoft Press, Iunie 2003.
[12] http://www.w3.org/People/Berners-Lee/FAQ.html#etc
[13] Badurowicz, M, “Mvc Architectural Pattern In Mobile Web-Applications”, Actual Problems Of Economics, 2011, pp.305-309.
[14] Stratmann, E., & Ousterhout, J.,“Integrating Long Polling with an MVC Web Framework”, 2nd USENIX Conference on Web Application Development, 2011, pp113-124.