Aku Kolu

MVC FRAMEWORKS IN WEB DEVELOPMENT

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2012

ABSTRACT

Kolu, Aku
MVC Frameworks in Web Development
Jyväskylä: University of Jyväskylä, 2012, 28 s.
Information Systems, Bachelor's Thesis
Supervisor(s): Hirvonen, Pertti

With the increased demand of complex, well-scalable and maintainable web applications, the MVC architecture is increasing in popularity and frameworks (whether they utilize the MVC architecture or not) are quickly becoming de facto –standard in web development. This Bachelor's Thesis introduces the use of MVC architecture in web development and how several web application frameworks make use of it. This research introduces the concepts of both the MVC architecture and web application frameworks but does not go into the basics of web development.

Three examples of web application frameworks that utilize the MVC architecture are introduced. Due to the constraints provided by the size of this research, this paper does not have a thorough comparison between the three frameworks. Instead, it focuses on finding out how each framework provides answer to certain complexity often required in today's modern Web 2.0 applications.

Keywords: MVC architecture, MVC model, web application frameworks, web development

TIIVISTELMÄ

Kolu, Aku
MVC Frameworks in Web Development
Jyväskylä: Jyväskylän yliopisto, 2010, 28 s.
Tietojärjestelmätiede, kandidaatin tutkielma
Ohjaaja(t): Hirvonen, Pertti

Monimutkaisten web-sovellusten lisääntyessä MVC-arkkitehtuuri on kasvattanut räjähdysmäisesti suosiotaan ja sovelluskehyksistä on tullut kiinteä osa web-sovelluskehitystä. Tämä kandidaatintutkielma käsittelee MVC arkkitehtuuria osana web-sovelluskehitystä ja sitä miten tämä sitä on hyödynnetty erilaisissa sovelluskehyksissä. Käsitteinä määritellään MVC arkkitehtuuri sekä web-sovelluskehykset, mutta ei keskitytä web-ohjelmoinnin perusteisiin.

Tämä tutkielma esittelee kolme käytännön esimerkkiä web-sovelluskehyksistä, joissa MVC arkkitehtuuria on hyödynnetty. Kandidaatintutkielman koko huomioonottaen, tässä tutkielmassa ei tehdä perustavanlaatuista vertailua näiden sovelluskehysten välillä, vaan keskitytään vastaamaan kysymyksiin miten kukin sovelluskehys vastaa nykypäivän vaatimuksiin ja haasteisiin.

Asiasanat: MVC-arkkitehtuuri, web-sovelluskehykset, web-sovelluskehitys

FIGURES

TABLE OF CONTENTS

1 INTRODUCTION

Web development has progressed greatly during the last decade. As the demand for high quality web applications has increased, certain patterns and architectures have emerged to help with the increased complexity of the applications. MVC – short of Model-View-Controller – is one of these design architectures and while not necessarily being de facto standard at the time of writing this thesis, it is indeed becoming a strong player in the field of web development.

This research will give an overview of what is the MVC model, what are its advantages and disadvantages and how it differs from preceding design patterns. As the MVC in web development is utilized in the form of web frameworks, this research will also find out what are the intrinsic features of a good framework; why are they used and what differentiates them from each other. The three research problems this paper will answer are as follows: "What are the advantages and disadvantages of the MVC architecture" will be discussed in chapter 2. "What makes a good web application framework" will be covered in chapter 3 and finally, "How is the MVC implemented in practice" will be answered in chapter 4.

To fully understand the contents of this paper, some basic understanding about web development in general is required. As the basics of web development will not be discussed, knowledge about the stateless nature of the web, the separation between client-side and server-side code and the basic principles of (object-oriented) software design are essential. The research itself is done as a literature review. Pack publishing is one of the most prolific IT book publishers out there and it offers a lot of credible, up-to-date information about the latest phenomena in the field of web development. However, to rule out the possibility of any biases, at least the same amount of references from other publishers will be used. Finally, academic articles from various libraries will be used to back up the evidence provided by the books.

The following chapter introduces the reader to the architectural model of the MVC. Illuminating the way each component works in conjunction with the others, the chapter explains in detail why the MVC model is so widely-used

today. As well as pointing out its benefits, this chapter also takes in consideration the controversy MVC has received and shortly introduces several alternative architectural patterns. Throughout the paper, MVC will be referred to as *design model, architectural model* or *MVC pattern*. The third chapter, MVC frameworks, links the design model to the frameworks in which the MVC can be utilized. As well as describing the usefulness of the web frameworks, it also points out the most important aspects a framework needs to take into consideration in order to fulfill the requirements of modern web applications. Finally, MVC utilization compares three very popular frameworks where the architectural model is used. As each framework introduced takes a bit different approach to tackle the same problems, the way they are evaluated is based on how well they fulfill the modern framework requirements introduced in the preceding chapter.

2 MVC ARCHITECTURE

This chapter will introduce the MVC architecture, its benefits and drawbacks. By breaking down the architectural pattern into three parts, each part (Model, View and Controller, respectively) will be covered in detail as well as providing some insight about the variations of the pattern. Throughout this research, MVC architecture will also be referred to as MVC pattern or MVC design model.

2.1 Introduction

Several years ago, in the early 2000's, the model-view-controller pattern was introduced to web development (Hourieh, 2008, p.6). However, according to Freeman et al. (2011, p.10), MVC pattern was originally invented as early as in the 1970's as a part of a Smalltalk project to better organize some early GUI applications. While some of its original design was very Smalltalk specific, the broad concept is applicable and very well suited to modern web applications today. Some people also consider modern MVC being the evolution of IPO (Input, Processing and Output), which was the best-practice functional model of the past era (Lecky-Thompson et al. 2004, p.243).

MVC is short of Model, View and Controller. As described by Samisa (2009, p.31), MVC pattern enables us to break down the system into three parts, model, view and controller, which are responsible of business logic, presentation logic and programming logic, respectively. There's actually the fourth component as well – the infrastructure that glues the MVC components together – but rather than being part of the actual design model its implementation can vary greatly depending on the web application framework, which will be covered in the following chapter (Lecky-Thompson et al. 2004, p.243).

Although the basic principle of the pattern is relatively easy to understand, the details of its implementation can sometimes get very complex. Figure 1 highlights the process flow of a typical request in MVC-based system: first, the user makes an action, which is managed by the controller. Controller handles

the required data manipulation with the model and passes it on to the view (resulting in indirect relationship between Model and View), which builds the markup based on the delivered data and shows it to the user.

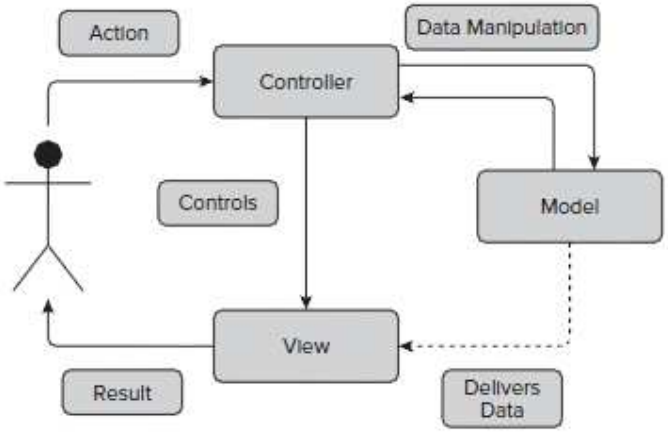

FIGURE 1: Model-View-Controller pattern (Porebski et al. 2011, p.15)

2.1.1 Model

Model encapsulates the business logic of the application. However, there's more than just pulling out the data from the database. Ideally, it should contain information about the database structure and all the dependencies and relationships of the database tables. (Porebski et al. 2011, p.15) Although it is not necessarily so, often a model classes in MVC frameworks represent a single database table and holds all the relevant information of the table it is representing. Thus, all the database validation rules should be contained in the model class and all the data validated before executing the actual database operations. If every database table is represented by a model class, database operations like modifying, adding or deleting the data become much easier and better organized. (Ahsanul et al. 2008, p.8)

2.1.2 View

View is responsible for showing the data to visitors and it's often built using HTML with little embedded code. No database operations or programming logic should be executed here - frameworks usually have a way of passing the relevant data to the views, which then simply present it. This should make views interchangeable at any time, without modifying the other layers. The general principle is that all the programming operations more sophisticated than conditional statements or loops should be done either in the controller or in the model. (Winesett 2010, p.10. Porebski et al. 2011, p.15).

2.1.3 Controller

Controller is the programming logic and the link between the view and the model. Typically it handles the user interactions, such as GET and POST – requests, fetching the data from the appropriate model, processing it and finally passing it on to the right view to be displayed. It's generally suggested to keep controllers simple and light – complex data processing should happen in the model. As stated in the book by Ahsanul et al. (2008, p.8), "this design philosophy is sometimes referred to as 'fat models, thin controllers' ".

2.2 Benefits

By forcing the separation of concerns, the benefits of using the MVC pattern are clear: the designers can work on the user interface without having to worry about the underlying data management and the developers can focus on the data logic without getting into the details of the presentation. As it is possible to have multiple views use the same model, the structure also helps the developers avoid code repetition. (Hourieh 2008, p.6. Pope 2009, p.10)

Another benefit of the architecture is its scalability. In classic software design patterns where the UI elements, data logic and the programming logic were all in the same place, the maintenance quickly became cumbersome as the application grew in size. As all the pieces in MVC are well-defined and self-contained, it doesn't really matter how large the application becomes as long as the principles of the MVC are followed rigorously. However, most frameworks only encourage this as it would be difficult to enforce. (Freeman 2011, p.64)

2.3 Controversy

Even though MVC architecture has numerous advantages from the software engineering point of view and it handles some of the traditional choking points of creating and maintaining a web application, it's not suitable for every project. Wang (2011) criticizes that even though MVC pattern seems to be superior in comparison due to the convenience of managing the project, it comes with the expense of cost-efficiency. For small and medium-sized applications which tend to emphasize loading speed and runtime efficiency, MVC might not be the best solution as it creates additional workload and makes the application more complicated. Every time the page sends a request, it goes through the MVC engine (controller class), resulting in marginally slower loading speeds. Another disadvantage is that the expenditure of time is large in the early stages of the project (Wang, 2011). Thus, if the project life cycle is relatively small and there is little need for maintenance or further development, there's no point in wasting a lot of time setting up a large MVC engine.

Crosscutting concerns are not fully addressed in the implementation of the MVC framework either. As per the article written by Kojarski & Lorenz (2003), crosscutting concerns can be divided in two patterns, Inter-crosscutting and Intra-crosscutting. "The tangling of functionality, presentation, and control within a single dynamic page is a result of *intra*-crosscutting of the application's concerns" (Kojarski et al. 2003). Basically this means mixing up the markup, programming logic and data logic. These problems contributed to MVC being introduced to web development as the MVC architecture handles those problems efficiently. It's the inter-crosscutting concerns, the aspects of the program which rely or affect other parts of the program, which the MVC architecture doesn't offer a solution. According to Kojarski et al. (2011), it poses much more severe problem. As many separate parts of the web application depend on the underlying data structure (tables in a SQL database, for instance), the result is code being scattered in many different files. If the data structure is updated or modified, all the code within the application which are dependent on that particular data model will need to be updated in their respective files as well (Kojarski et al. 2003). This can be tedious and difficult if the system is large enough. Aspect Oriented Programming (AOP) is introduced as one of the possible solutions to this problem, but it is not covered in this paper.

While not directly being an architectural problem per se, many MVC frameworks also pose challenges to the programmer in the form of often having to extensively study the framework documentation to be able to use it. While this holds true to all kinds of development tools, the switching costs in time and money might be so steep that the benefits of switching over to a framework need to be tangible. When the programmer new to a framework is hit by a bug, it can take excessive amounts of time to figure out whether it's really in the code or in the framework itself (Abeysinghe 2009, p.48). In the latter case, it might be really hard to find a solution if it's possible at all, due to the way many frameworks are designed. Often complex systems in which the frameworks are used require very fine-grained operations and if the framework doesn't seem to flex to the programmer's needs, finding a solution around it can be a challenging task without a thorough knowledge of the framework. This is where the Internet community and the other developers using the framework come in handy as it is much likely that someone else has run into a similar problem already. An in-depth look into the frameworks, including the introduction of some of the popular frameworks today, will be covered in the next two chapters.

2.4 Variations

2.4.1 MVP

Model View Presenter (MVP) is a variation of MVC created to solve the problems of MVC where the structure of the model-view coupling is too complex

(Zhang et al. 2010). In traditional MVC, models notify views when the data is being modified. However, MVP takes more direct approach to a stateful view as it forces all data transfer to go via Controller, which in this context is called Presenter (Freeman 2011, p.68).

Because MVP pattern seems to be even better for web development than the traditional MVC, most so-called MVC frameworks in web development actually follow MVP, despite of what they are called, because usually the Controller does the majority of the data transfer between the model and the view (Porebski 2011, p.15).

2.4.2 MVVM

Model View ViewModel (MVVM) pattern originated from Microsoft in 2005 as a modification of the MVP pattern to support event-driven programming and to leverage benefits of the WPF data binding and Silverlight (Jarnjak, 2010). Models and views have largely the same roles in MVVM as they do in MVP. The most notable difference in this pattern is the view model, which is an abstract representation of the UI (Freeman 2011, p.68). Model contains the data and does not know about the view or the view model, view model is an abstraction of the view and contains all of its data, making use of a data-binding mechanism. It doesn't have a reference to the view (it's the other way around), it simply contains all the data view makes use of, and updates it accordingly when the properties of the elements are exposed by the view. Finally, view has all the UI elements which are bound to view model properties (Freeman 2011, p.68, Jarnjak 2010). Although MVVM originates from Microsoft and most of its implementations are associated with .NET platform, frameworks for other languages exist as well. Outside of the Microsoft stack, MVVM is called MVB (Model View Binder).

3 MVC FRAMEWORKS

This chapter defines web application frameworks and discusses their benefits and drawbacks. The answer for the question "What makes a good web application framework" will be provided in the following subsections of this chapter. Finally, certain complexity issues of the modern web development will be looked into and they will be used as a reference when the implementations of the MVC-model are being assessed in the next chapter.

3.1 What defines a web framework

Frameworks are essentially chunks of code wrapped up using a certain architecture or design pattern that (in theory) should speed up the development process of the web applications. Frameworks can be seen as a set of abstractions, sort of half-ready applications or application skeletons, which you can then extend and modify to suit your needs. However, it's all about striking a balance between bad flexibility and a steep learning curve, for the benefits of using a framework come with a price and their greatest benefit can also be seen as their weakness. To be able to benefit from using a framework, one does have to comply with its regulations, coding methodology and conventions. If conventions are heavily enforced and the configuration required to set up the framework is at minimum, the flexibility can suffer. On the other hand, if the framework requires heavy configuration and knowledge to start with, it makes the learning curve significantly steeper. (Porebski et al. 2011, p.2).

Writing code is a constant struggle between the myriad solutions to a single problem. While not necessarily being the most optimal solutions, many frameworks offer a set of choices for the user (in this case, the programmer). If they are sensible choices, they make writing code much simpler (Upton 2009, p.15). After all, in the field of web development the chances that some other software professionals have tackled the same problem are very high so there's no real point in trying to reinvent the wheel (Abeysinghe 2009, p.54).

Large systems tend to have complex problems and the best way to solve it is to break it down into separate, more manageable parts and it is in these smaller parts where we find similarities to the ones already solved by other developers, even though the problems as a whole are often unique (Abeysinghe 2009, p.59). Frameworks, much like libraries, aim to solve that kind of common problems. However, frameworks go much beyond that. According to Porebski et al. (2011, p.2), there's one fundamental difference between a framework and a library; where libraries are *called* from your code, frameworks *call* your code. Basically this means that framework is a skeleton on top of which the developer builds the application, whereas library is a set of attachable modules which can be used on top of the application itself.

3.2 Benefits of using a framework

Even though MVC design architecture offers general benefits already mentioned above, the real difference of using an MVC framework greatly depend on the server-side language and the actual implementation of the framework in question. It really boils down to how using the framework compensates the architectural flaws of the scripting languages the developer is using. In PHP in particular, one of the most fundamental disadvantages is the excessive liberty of code, which often results in difficult, even indecipherable coding structure in terms of reading and management (Wang, 2011). In many other programming languages, such as C# or Java, good programming practices are enforced through their design, but PHP requires self-discipline, which is the main difference between an experienced and inexperienced developer. The lack of self-discipline in rapid cycle PHP development is likely to create complex and tangled, `spaghetti`-style code, which is very difficult to maintain and reuse (Lecky-Thompson 2011, p.241). By introducing a set of rules the programmer has to follow, the MVC addresses the problems above by separating backstage PHP code and HTML pages, thus making the program more convenient to debug. Emphasis of data-abstraction promotes code reusability and the separation of data classes make the project easier to maintain. (Wang, 2011)

Many projects where frameworks are used today require multiple programmers working in conjunction and this is where a strict set of rules and conventions comes in handy. Especially in API design, the programmers will have to find a common ground and make sure each team member follows the set of rules and conventions used – 'I did it part my way' kind of mindset is not going to work in a large scale project (Abeysinghe 2009, p.15). Even then, not everyone should be working on the same thing. Different parts of the system need to be divided to the programmers evenly and that's often referred to as 'separation of concerns'. Frameworks following the MVC architecture promote so-called *horizontal* division where different groups take care of different aspects of the MVC – one group for models, one for controllers and one for views (Abeysinghe 2009, p.15). Since web development usually involves two different

types of professionals, UI developers (so-called *web designers*) and functionality experts (*web programmers*), the horizontal separation of concerns where designers can be allocated to the views and programmers to the models & controllers offers substantial benefits as long as people working in the team do not cross the borders of their area of responsibility (Kojarski et al. 2003).

3.3 What makes a good framework

There are lots of good frameworks around and most criteria for evaluating a good framework apply not only to frameworks but any software projects in general (Merkel 2010, p.252). Such criteria are e.g. documentation, code quality, performance and scalability. However, not all frameworks are good for every project. There are small, lightweight frameworks suitable for small applications and then there are large, feature-packed frameworks which have solutions to very specific problems at the cost of performance. Comprehensive web frameworks often implement tools for specific tasks in order to save time or simplify the process, such as ORM tools, template engines, automated unit tests etc.

3.3.1 Simplicity

Frameworks simplify the software development as they handle some of the application complexity on developers' behalf. However, if the framework requires a lot of knowledge or configuration, it doesn't really *simplify* development anymore. Thus, the design philosophy of many successful frameworks is *convention over configuration.* This means that if the file name or locations are set according to the framework's conventions, the framework can find and use them automatically, without having to set configurations to everything. (Abeysinghe 2009, p.55)

3.3.2 Documentation

Good documentation is arguably the most important aspect of a good framework. If the documentation is lacking, the developer might spend hours figuring out a simple framework-related issue, negating the reason why frameworks are used in the first place – to speed up the rapid development process (Merkel 2010, p.254). Most frameworks are documented but the level of documentation can differ greatly. Some may be designed in a way that the API is all that experienced programmers need to get the required information while others may have central document library with comprehensive tutorials, code snippets, FAQs etc. Either way, most of the popular frameworks out there have two types of documentation that should be taken into consideration upon choosing the framework. Formal documentation includes the API and all its related documents put forward by the framework developers while informal documentation,

such as mailing list logs or forum archives, are solutions to problems asked (and answered by) the community members and fellow framework users. (Abeysinghe 2009, p.58)

3.3.3 Performance

Performance is an important factor in modern web applications. As discussed in the preceding chapter, using MVC framework usually results in marginally slower loading times. The amount of memory and cache consumed by the framework is called *footprint,* which largely depends on the framework size. It's often a compromise between choosing a bulky but feature-rich and light but hardly scalable framework (Abeysinghe 2009, p.50). This is where frameworks like Zend shine, as the user can pick the most favorable parts of the framework to adjust the footprint to his/her needs. Zend Framework will be covered in the following chapter.

3.3.4 Community

Documentation, no matter how thorough, can never cover all the possible scenarios. There can be situations where the use case is so unique that only a handful of other people have done that or situations where the developer's expertise just isn't enough. When that occurs, it is luxury to have someone to tell you what to do (Abeysinghe 2009, p.57). The community around most frameworks is divided in two; those who contribute to the framework development itself (this is the case especially in open-source frameworks) and those who use the framework and help others tackling the most common problems and avoiding the pitfalls. Naturally the quality and experience of the framework developers is an important factor as well but this subsection will focus on the community from the framework user's point of view. A general rule of thumb is "the more users who have invested time and resources into learning and using the framework, the easier it will be to get support if something goes wrong" (Merkel, 2010). Many open source frameworks have forums, mailing lists or IRC channels to provide community support and answers to the most common questions regarding the popular frameworks (such as the ones introduced in the following chapter) can easily be found in the relevant documentation, forums or at the programming community Stackoverflow. The communities around frameworks vary in activeness and friendliness so it's an important factor to be considered when selecting the framework (Abeysinghe 2009, p.57-58).

3.4 Dealing with complexity

The demands of modern web applications include, but are not limited to the following aspects of web development. This subsection will introduce the most

common areas that need to be covered in well-scalable, demanding web applications. In chapter 3 it will then be discussed how the three frameworks introduced in that chapter actually deal with the following areas and their complexity.

3.4.1 Object-relational mapping

Object-oriented design is the most widely-used approach in web development today, even though many of these systems have relational back-end databases. Object-relational mapping (ORM) is a technique that performs a mapping of the relational database tables into programming language objects. However, this is not possible without proper tools, for there are a few inconsistencies and differences between object-oriented and relational models. Data types, for instance, are not fully compatible. In SQL-databases the length of the field is often declared whereas many programming language simply deal with different types of variables (e.g. strings that can be of any length). That's why ORM tools are included in many frameworks – to provide a simple data access that can be used from the within the programming language.

3.4.2 Template engines

Template engines are tools that were created to help with the development of dynamically-created pages. A typical 'view' (or HTML page) created with a template engine has references to some server-side functions or variables. The original idea behind templates was to separate the UI from the business logic (similar to two-tier architecture) and make markup code look clean. However, rather than enforcing the principle of separation, most template engines only encourage it. In the world of strict deadlines, it's often easier to cut the corners and slip some business logic into the template, resulting in tangled and messy code. (Parr, 2004)

Template engines were very popular back in the days when web application frameworks were seldom used. However, they are not nearly always used in MVC frameworks as they tend to divide opinions. They are of limited usefulness to the developers and tend to have their own set of tags, in some cases even their own language which steepens the learning curve. Still, a correctly used template engine can be a valuable tool and whilst most MVC frameworks don't ship with one, they usually have the option to include it as a plugin. (Porebski 2011, p.249).

3.4.3 Unit testing

There are myriad types of testing in software development cycle. Unit testing are precise and only cover small bits of code related to algorithms, business logic or other back-end infrastructure. Integration testing covers broader area, how

different classes or entities in the project work with each other. Finally, user acceptance testing happens when the customer starts using the final product. (Freeman et al. 2011, p.78, Merkel 2010, p.291)

While tests covering broad areas of the project can seldom be automated, unit test automation is often supported out-of-the-box by the frameworks. Even though unit testing is always optional, it can provide tremendous support for the application development. The goal of testing is to catch bugs within the software but the cost and simplicity of fixing the bug often depend on *when* is the testing done. As the unit tests are usually automatic once they have been set up, the developers can catch bugs at the early stages and fixing them will be relatively easy and cheap. Bugs that are discovered later during integration or user acceptance testing will often require a professional to discover the problem and fix it (Merkel 2010, p.294). That's why automated unit tests are considered an important part of a successful web framework.

Successful unit testing is often defined by several guidelines: only test small parts of the code at a time, nothing larger than a class; isolate the testing from the production code; and finally, running the test must yield automatic pass/fail results (Galloway et al. 2011, p.292). In agile software development, and XP (Extreme Programming), unit testing is the basis of writing an application. Unlike in traditional software development, the tests in XP are defined up front. This approach is called TDD (Test-Driven-Development). As the agile software development has grown in popularity during the last few years, many frameworks today provide support for easy and fast unit testing.

3.4.4 URL routing

Before the MVC architecture was used, URLs usually represented the actual files on the server hard disk. However, with MVC this doesn't make sense as the user requests are processed by controller classes. URL routing has two functions: examine the incoming URLs to allocate the request to the correct controller and action and generate outgoing URLs that correspond to controller actions to be rendered in HTML. This allows the developer to create user-friendly URLs that are short, easy to type and reflect the site structure. Even though most modern MVC frameworks have URL routing implemented automatically, it is definitely worth mentioning as a core feature. After all, the URL is a user interface for the Web like any other, even though it seldom gets much attention. (Freeman et al. 2011, p.325, Galloway et al. 2011, p.213)

3.4.5 Security

Security is an important matter as disregarding security issues can lead to serious problems. Because its effects might not be visible until someone exploits the vulnerabilities, the security enforcement might be one of the best reasons to use a good framework. Security issues can also be fairly complex; the inexperienced programmers benefit the most as they might otherwise leave security vulnera-

bilities or pigeon holes to their applications but it is also valuable for the seasoned veterans as it saves time not having to write them from scratch. (Porebski 2011, p.229)

While most developers know how to prevent SQL injection attacks (many programming languages actually offer specific functions for the job), preventing XSS (cross site scripting) or CSRF (cross-site request forgery) attacks is a bit more complicated. XSS attacks can be done by exploiting a vulnerability which allows the execution of JavaScript code on the webpage. Escaping, filtering or sanitizing the HTML by validation can eliminate some XSS vulnerabilities, but writing the sanitization functions can be tedious. CSRF attacks can be carried out by manipulating forms and many developers have problems defending against them as it takes so much knowledge how to do it (Porebski 2011, p.244). Fortunately, many frameworks have the tools to prevent against both XSS and CSRF attacks so the developers can focus on writing the content of their applications instead of reinventing the wheel in the security matters.

3.4.6 AJAX

AJAX (Asynchronous JavaScript and XML) is a combination of two technologies and it's used to communicate with the server without having to refresh the whole webpage. JavaScript's XMLHttpRequest object, provided by modern browsers, can be sent back to the server asynchronously, meaning that the webpage doesn't stall the functionality while waiting for the response (Rajsekhar 2008, p.178). This is the basis for all modern web applications as it enhances the user experience when only the required parts of the page can be loaded separately without resetting text fields, cursor position etc. The benefits are not limited to better user experience as it also increases performance by saving bandwidth. (Hourieh 2008, p.94). Most modern web application frameworks support AJAX out-of-the-box to some extent, by providing helpers and widgets built around the one or more AJAX libraries.

4 MVC UTILIZATION

This chapter will focus on three very popular frameworks that utilize the MVC pattern, Ruby on Rails, .NET MVC 3 and Zend Framework. Rather than comparing them with each other in detail, the following sections will find out how certain complexity (introduced in the previous chapter) is met. It should be noted that at the time of writing this paper, new versions from both .NET MVC and Zend Framework have just been released (MVC 4 and ZF2, respectively) but this research will focus on the prevalent versions of the respective frameworks.

4.1 Ruby on Rails

Ruby on Rails (RoR), a full-stack web application framework written in Ruby was developed in 2004 by David Heinemeier Hansson of 37signals (Langley, 2008). It follows MVC architecture and its features include full object-oriented design, support for Web 2.0 features, RESTful web services and AJAX, making it well-suited for projects that require scalability (Viswanathan, 2008).

Like many other web frameworks, RoR favors convention over configuration. In this case, not only does it create a skeleton application but it also stocks them with defaults so the developer doesn't have to deal with messy XML configuration files (Holzner 2007, p.89). It doesn't stop there, Rails is as much a set of guidelines and a way of writing an application as it is a framework. In addition to MVC architecture, RoR embodies the Agile Manifesto and is designed with immediate feedback in development cycle, REST-based design and test-driven development in mind (Benson 2008, p.22). RoR also implements a set of high-level modules, so-called *subframeworks* which the developer is using most of the time. ActiveRecord, ActionView and ActionController are such modules for the M, V and C, respectively (Thomas et al. 2007, p.229).

ActiveRecord is an ORM-tool which establishes the connection between the domain objects and the database. AR classes receive their attributes directly

from the databases and Rails handles the generation of classes automatically. If the developer follows Rails naming conventions with the tables, no configuration for running the ORM tool is required as would be the case with traditional ORM tools (Bächle et al. 2007).

ActionView encapsulates the functionality to render templates, usually HTML, XML or JavaScript back to the user. Comparable to other template engines such as .NET Razor, ActionView separates the code with <%...%> -tags inside of which the code fragments are placed. To prevent putting too much code in template files, Rails provides *helpers*, small modules containing methods that assist the view. By isolating these methods from the view, the code can be tested as individual units and the same methods can be used in different views.

Finally, ActionController is where the programming logic in Rails takes place and it handles the URL routing and other standard controller actions. However, Rails differs from many other frameworks by providing *resources,* a macro-route facility for a RESTful approach. It makes use of HTTPs request and response-codes, GET, POST, PUT and DELETE, promoting application scalability. (Thomas et al. 2007, p.409-410)

The best available RoR documentation is available at Ruby on Rails website, *api.rubyonrails.org*. It features the full API but also getting started –guides and several tutorials for building RoR web applications. There is also a Google group, Ruby on Rails Talk, where Rails users seek help and discuss the framework-related matters. Blog posts about RoR are gathered under one roof, called Planet Ruby on Rails and there are also IRC channels for real-time conversation. (rubyonrails.org, 2012)

4.2 ASP.NET MVC 3

Scott Guhrie of Microsoft designed the prototype of ASP.NET MVC back in 2007 to create an application that supports the MVC-design. The prevalent web application framework, ASP.NET Web Forms supported two layers of abstraction and attempted to hide both HTTP and HTML by modeling the UI as a hierarchy of server-side control objects in order to make web development feel exactly the same as Windows Forms development (Freeman et al. 2011, p.5). In theory it was possible to separate concerns following the MVC design pattern but to do that, each developer had to build an implementation of their own.

After its initial release, ASP.NET MVC became so popular that it was developed even further, resulting in MVC 2 and MVC 3 being released within two years of the first version. Although the previous versions of ASP.NET MVC implemented the design model efficiently, MVC 3 makes learning it even easier. Its other improvements over previous versions include extending the attribute-driven validation system, annotation support, native support for jQuery AJAX and JSON bindings. (Galloway et al. 2011, p.4-5)

ASP.NET MVC works in tune with HTTP, just like Ruby on Rails, giving the developer total control over the requests passing between the browser and

the server and URL-routing is implemented automatically. This means that clean URLs are provided by default, without the need of extra configuration. However, unlike in Ruby on Rails, ASP.NET MVC does not have ORM-tools or built-in automatic testing tools included. This is because .NET platform already offers a wide range of tools for both jobs: object-relational mapping can be handled with NHibernate, Subsonic or Microsoft's Entity Framework and for unit testing there are three different methods the developer can choose from: NUnit and XUnit are open source test tools and Microsoft also has its own MSTest. With ASP.NET MVC it's also possible to create UI automation tests by writing test scripts that simulate user interactions in a web page.

One strongpoint of MVC 3 is its template engine, called Razor. It ships out-of-the-box, being part of the framework and it simplifies the generation of views greatly. Previous versions of the Microsoft's MVC-model had Web Forms View Engine which used the same files and syntax as Microsoft's previous design architecture, Web Forms. However, the syntax was heavily XML-like and often difficult and awkward to read. Razor simplifies this syntax and uses @-symbol to separate code snippets from the HTML markup, thus making the syntax easier to type and read. (Galloway et al. 2011, p.52).

Where ASP.NET MVC 3 really stands out is the approach to Asynchronous Programming Model (APM). With the help of asynchronous controllers the developer can speed-up performance if multiple simultaneous I/O bound requests are required. For instance, if multiple non-interdependent web service requests are required, asynchronous controllers can greatly improve performance. However, this comes at the cost of greatly increased complexity, for parallel asynchronous operations tend to yield all kinds of problems, not least being extremely difficult to test and debug properly. Still, albeit they address a niche problem most MVC applications don't suffer from, it may well be a deciding factor for the platform & programming language if the application-to-be is known to be large and complex. (Freeman et al. 2011, p.490)

The community and documentation of ASP.NET MVC 3 are, due to its popularity, large and comprehensive. The Microsoft Developer Network (MSDN) is a good general resource for ASP.NET and its MVC framework. Because ASP.NET MVC is open source, its source code can be seen (and downloaded) at CodePlex. (Freeman 2011, p.13)

4.3 Zend Framework

Zend Framework (ZF) is a PHP framework created by two core developers of PHP, Andi Gutmans and Zeev Suraski. ZF follows solid object oriented design principle and its key features are simplicity, component-based and loosely-coupled architecture (Porebski 2011, p.9). Loosely coupled means that the components in ZF have few (if any) dependencies on other components and the developers can cherry-pick the most suitable ones for their projects, thus enhancing flexibility. Following this design principle, the use of MVC architecture is

also optional. Other notable out-of-the-box features include authentication/authorization, database abstraction, session management and web services. (Pope 2009, p.4)

ZF doesn't have a fully functional ORM tools but it does provide a lightweight wrapper for the PDO (PHP Data Objects), an extension that provides a data-access interface. Zend_Db is a family of classes which manages the tables and their relations allowing the execution of SQL queries in an object-oriented way but if that's not enough for the developer's needs, Propel and Doctrine are notable ORM solutions, which can easily be installed as plugins (PHP Documentation, 2012; Porebski 2011, p.58).

The use of template engines is entirely optional in ZF. It's not included as a core feature, but ZF provides out-of-the-box support for Smarty, one of the most popular template engines written in PHP. If the developer can cope with having to learn additional (Smarty) syntax, using it may simplify the views. However, Smarty (amongst other PHP template engines) has received some criticism as it replicates the features already offered by PHP by replacing the smarty code with regular PHP code at runtime, which causes additional processing overhead when each view is practically rendered twice. (Porebski 2011, p.263)

In addition, Zend Technologies provides two products to go with the framework, Zend Server and Zend Studio. Zend Server is a PHP application server which includes PHP, code accelerator and monitoring & problem diagnostics tool and support for the most popular databases. It also automates software updates, hotfixes and security patches for all platforms (Business Wire, 2009). Zend Server is not required to run ZF but it offers substantial benefits to the developers. For instance, Zend_Log and Zend_Cache –tools help with logging and caching respectively, and its Plaform as a Service –solution helps streamlining the rapid deployment of web applications (Grehan, 2011). Optimized for running ZF, using Zend Server can improve performance as much as 200%, according to PR Newswire's article. Zend Studio is a commercial IDE designed for coding PHP. It has all the standard functionalities of an advanced IDE tool, from code completion to smart bracket matching but in addition, it offers CVS (code versioning system) integration for both Git and Subversion and unit test support (Gibbs, 2003). According to the article published in PR Newswire (2011), Zend studio can provide 67% faster default startup on average, 40% smaller default disk footprint and a lower memory footprint.

Evans (2008, p.5) describes the active community of ZF to be its strongest asset. Not only does ZF website (*framework.zend.com*) house the full documentation and API but also a bug-tracking system, tutorials and pointers to ZF user groups, IRC channels, mailing lists etc. Finally, there's a developer wiki for the contributors of the framework – the framework being open-ended, anyone can suggest features and start contributing. (Evans 2008, p.5, framework.zend.com, 2012)

5 CONCLUSION

This research discussed the MVC architecture as a concept, providing insight to its benefits and drawbacks. Several variations of the pattern were also introduced and it was noted that many web application frameworks which utilize the MVC architecture actually use some of the variations of the original model. As provided by the section 2.2., it is evident why the MVC is a very popular phenomenon today. Even though the architectural model doesn't provide a clean solution to crosscutting concerns, the advantages far outweigh the disadvantages in web applications that meet the certain level of complexity and need to be well scalable and easily maintained.

Another area of research, web application frameworks, was also discussed in this paper. Even though they mostly utilize the MVC pattern, the real benefit comes from dealing with complexity often required by modern web applications so that the programmer does not have to reinvent the wheel. Such issues are object-relational mapping, template engines, URL routing, AJAX and security, for instance.

Finally, three examples were provided of some popular frameworks that utilize the MVC pattern. The literature reviewed in this research did not have critical comparison between the frameworks – they rather focused on each respective framework they were written about. Thus, this research did not go into details of comparing each part of the frameworks but rather tried to provide information how the three frameworks provided answers to aforementioned complexity.

In addition to reviewing the benefits and controversy of the MVC architecture and web application frameworks, the conclusion of this paper is that when selecting a framework, it mostly boils down to the selection of server-side scripting language. There are myriad frameworks to each programming language that can be used on the back-end and while this research only reviewed three of them, it was discovered that none of the examples were really lacking in comparison. The open and community-driven nature of the frameworks ensures that high-demand features get implemented efficiently.

Pursuing the same subject, a more thorough follow-up about the web design architectures and web application frameworks could be used in a Pro Gradu –research. Because the use of the MVC architecture is still rather new in web development, academic research about the subject is nearly nonexistent. For further research, the following topics are suggested:

- Comparison between MVC architecture and Java servlets in web development
- The cost-efficiency and learning curve of the MVC frameworks
- Empirical study: how open-source benefits the MVC frameworks and its users

REFERENCES

Abeysinghe, S. (2009). *PHP Team Development.* Packt Publishing.

Ahsanul, B., Anupom S. (2008). *CakePHP Application Development.* Packt Publishing.

Benson, E. (2008). *The Art of Rails*. Wiley Publishing.

Bächle, M., Kirchberg, P. (2007). *Ruby on Rails.* Software, IEEE. 24:6.

Evans, C. (2008). *php|architect's Guide to Programming with Zend Framework.* Marco Tabini & Associates.

Faster and leaner zend studio 9.0 PHP IDE paves the way to develop and debug web apps more productively in cloud and on-premise. (2011, Nov 15). PR Newswire.

Freeman, A., Sanderson S. (2011). *Pro ASP.NET MVC 3 Framework, Third Edition.* Apress.

Galloway, J., Haack, P., Wilson, B., Allen, K. Scott. (2011). *Professional ASP.NET MVC 3*. p.5-6. Wiley Publishing.

Gibbs, M. (2003). The zen of zend studio. Network World, 20(13), 38-38.

Grehan, R. (2011). Fabulous PHP Frameworks : Zend Framework. Inforworld.Com.

Hourieh, A. (2008). *Learning Website Development with Django*. Packt Publishing.

Holzner, S. (2007). *Beginning Ruby on Rails.* Wiley Publishing.

Jarnjak, F. (2010). *Flexible GUI in robotics applications using Windows Presentation Foundation framework and Model View ViewModel pattern.* New Trends in Information Science and Service Science (NISS).

Kojarski, S., Lorenz, D. (2003). *Domain Driven Web Development With WebJinn.*

Langley, N. (2008). *Ruby on Rails takes Rest from Soap.* Computer Weekly (Feb 12, 2008) : 34.

Lecky-Thompson, E., Eide-Goodman, H., Nowicki, S., Cove, A. (2004) *Professional PHP5*. Wiley Publishing.

Merkel, D. (2010). *Expert PHP 5 Tools.* Packt Publishing.

Parr, T. (2004). *World Wide Web: Proceedings of the 13th international conference*, WWW '04, 2004.

Pope, K. (2009). Zend Framework 1.8 Web Application Development. Packt Publishing.

PHP Documentation (2012). http://php.net 04.10.2012.

Porebski, B., Przytalski, K., Nowak, L. (2011). *Building PHP Applications with Symfony, CakePHP, and Zend Framework*. Wiley Publishing.

Rajshekar, A. P. (2008). *Building Dynamic Web 2.0 Websites with Ruby on Rails.* Packt Publishing.

Ruby on Rails documentation. (2012). http://rubyonrails.org 14.10.2012.

Thomas, D., Hansson, D. (2007). *Agile Web Development with Rails Second Edition.* The Pragmatic Programmers.

Upton, D. (2009). *Codeigniter 1.7.* Packt Publishing.

Wang, G. (2011). *Application of lightweight MVC-like structure in PHP.*

Viswanathan, V. (2008). *Rapid Web Application Development: A Ruby On Rails Tutorial*. Software, IEEE. 25:6.

Winesett, J. (2010). *Agile Web Application Development With Yii 1.1 and PHP5.* Packt Publishing.

Zend Framework documentation. (2012). http://framework.zend.com 18.10.2012.

Zend server 5.6: Mac developers get a new tool for rapid web application development using PHP. (2012, Jan 19). PR Newswire.

Zhang, Y., Yanjing, L. (2010). *An architecture and implement model for Model-View-Presenter pattern.* Computer Science and Information Technology (ICCSIT).