

# khawkins\_p145441\_assignmentT woReport

*by* Kieran Hawkins

---

|                |                                                                                    |                 |       |
|----------------|------------------------------------------------------------------------------------|-----------------|-------|
| FILE           | 2324_KIERAN_HAWKINS_KHAWKINS_P145441_ASSIGNMENTTWOREPO<br>RT_93284_1715244568.DOCX |                 |       |
| TIME SUBMITTED | 23-MAR-2018 07:13PM (UTC+0000)                                                     | WORD COUNT      | 4826  |
| SUBMISSION ID  | 85481927                                                                           | CHARACTER COUNT | 37341 |

---

# Investigating The Use Of MVC Within Enterprise Applications

---

---

**Name :** Kieran Hawkins

**P Number :** P145441

---

**Module Title :** Advanced Web Development

**Assignment Title :** Investigating The Use Of MVC Within  
Enterprise Applications

**Lecturer :** Michael Dacey

**Deadline Date :** Friday 23rd March 2018

## Table of Contents

|     |                                                |    |
|-----|------------------------------------------------|----|
| 1.0 | Introduction .....                             | 4  |
| 2.0 | Using MVC within Enterprise Applications ..... | 4  |
| 3.0 | MVC Controllers .....                          | 5  |
| 4.0 | Presenting Data .....                          | 7  |
| 5.0 | Common MVC Frameworks .....                    | 7  |
| 6.0 | Conclusions and Recommendations .....          | 9  |
| 7.0 | Reference List .....                           | 10 |
| 8.0 | Implementation Code .....                      | 12 |

## Table of Figures

|     |                                                   |   |
|-----|---------------------------------------------------|---|
| 1.0 | MVC Design Pattern .....                          | 4 |
| 2.0 | Controller Patterns (Front and Page) .....        | 5 |
| 3.0 | Controller Patterns (Common Implementation) ..... | 6 |

## 1.0 Introduction

In his book "Patterns of Enterprise Application Architecture", Fowler discusses a number of key concepts that have been standard practice within enterprise application systems for some time. Fowler specifically talks about core architectural aspects of a systems design, which have in turn evolved into many of the modern design patterns documented today.

This report will aim to elaborate on some of the fundamental elements that Fowler draws attention to, specifically elements that have contributed to the Model, View, Controller (MVC) pattern.

## 2.0 Using MVC Within Enterprise Applications

MVC (Model, View, Controller), is a fast growing software design pattern created to deal with fundamental principles such as encapsulation of related logic. The major component parts of MVC are the models, views and controllers. These effectively interact as to provide a robust working system that meets user requirements [1]–[3].

The models are used to represent and encapsulate the business or data related logic, whilst views are used to represent the UI (User Interface) [4]. The controller can be thought of as the bridge between the model and view. The controller is in charge of relaying commands, gathering/formatting data and sending it back to the view for display to the user [4]. MVC effectively partitions, or layers the system which if used correctly, can reduce its complexity, size and maintainability.

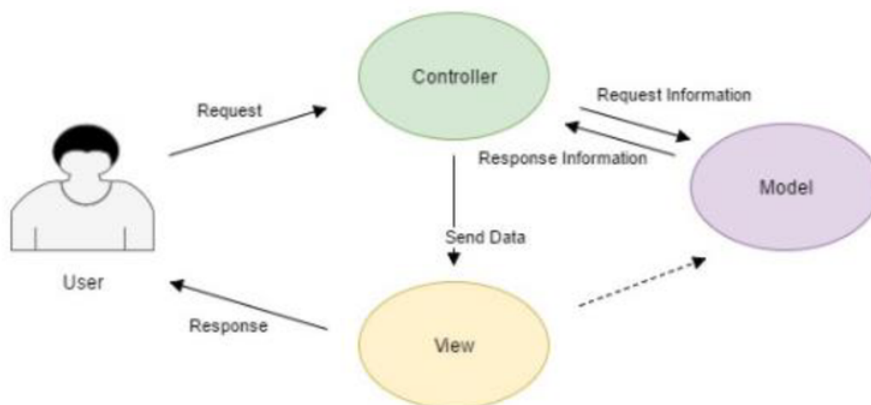


Figure 1 – MVC Design Pattern [1].

As systems become larger, they become increasingly difficult to maintain and update. MVC can potentially solve these issues due to its robust and flexible nature. MVC also provides a range of benefits that can be taken advantage of within many software projects, including enterprise applications. These include potentially increased development speed, better testing, loose coupling and better request handling. All this ultimately reduces the amount of resources it takes to maintain an application [2], [3].

### 3.0 MVC Controllers

As previously stated, the controller, as the name suggests, is an object with the responsibility of managing the flow of data between models and views. The controller encapsulates this kind of functionality, allowing for systems to be created in a more efficient manner [4]. As systems have grown larger and more complex, controllers have previously been used in a variety of ways. In modern applications, there are three main types of controller which generally crop up within many MVC implementations. These include application controllers, front controllers and page controllers.

Page controllers are those concerned with its own page, meaning they have a direct relationship with the view they are controlling. This essentially means that for every page of an application, there would be an associated view and page controller [5]. Although page controllers are relatively simple to approach, they can quickly become unmanageable as the application grows larger. This is where centralising key functionality could prove to be the better option.

The front controller is mostly implemented to solve this problem. Within MVC, the front controller is usually a single object in charge receiving and relaying requests to the relevant parts of the system [6]. In most cases, a request would come into the system and be received by the front controller. From there, the request is decomposed as to find out the correct pieces of the system the request is concerned with. These could be page controllers (each in charge of handling the request itself) or some other request/resolve implementation such as the command design pattern.

An application controller is similar to a front controller in a sense that there is usually only a single instance. The difference is the fact that the application controller is generally used to store application wide functionalities and properties [7]. For example, any configurations, view templates and page controllers that would be accessed throughout the entire application would generally be placed here. The application controller is also a common place to house flow logic and activity handling.

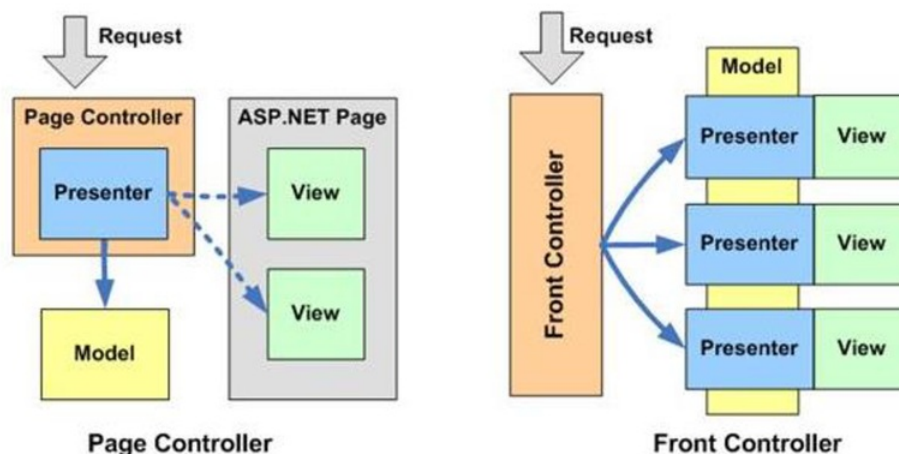


Figure 2 – Controller Patterns (Front and Page) [8].

Having a single front controller and using it to relay requests to individual page controllers, is not strictly part of MVC. It is mostly integrated as a sort of bootstrapping strategy, capable of taking URL's and transposing them into commands and actions.

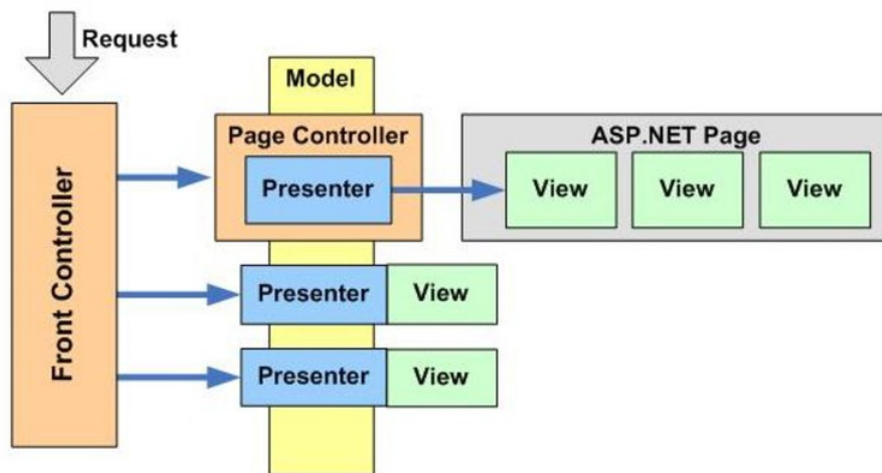


Figure 3 – Controller Patterns (Common Implementation) [8].

As shown in the preceding figures, all of the controllers are designed to receive, manipulate and respond to request objects. According to Fowler [9], handling a request could potentially involve a vast amount of contrasting or similar tasks such as user specific views, internationalization and security. Presenting these types of functionalities through a number of page controllers could prove to be challenging to maintain and can lead to an increased amount of duplicate code throughout the system.

Fowler highlights the exact problem the MVC pattern has set out to solve. By layering and encapsulating functionality in an efficient manner, these types of tasks can be stored in a global fashion, eliminating any need for them to be distributed throughout the code base.

Another key advantage to this layering is the improved testability. As objects become more cohesive with less coupling, the ability to test them in isolation becomes more apparent. This in turn opens the doors to mock testing, where core objects such as requests can be mocked and tested in a clean and time efficient manner [10].

Another advantage is the increased capability of collaborative working that is gained when using any layered architectural pattern. As core components are encapsulated, they can be worked on in tandem by a large number of developers. This in turn increases project turnovers as they can essentially be developed quickly by a skilled team [2], [11].

A key drawback to be noted is the increased learning curve in implementing an MVC approach. The increased complexity as compared to traditional static web pages is substantial, though; MVC does provide the ability to produce applications that are more secure, feature rich and efficient.

## 4.0 Presenting Data

It has already been identified that it is the views role to present the application to users. They are essentially the interface, in control of tasks such as UI configuration, taking input and relaying outputs back to the user [4].

Within MVC, there is generally at least one view associated with a page and controller. The view can be constructed much like a traditional web page using static HTML. However, within dynamic enterprise applications, this can become highly inflexible and would require a substantial amount of work change [10], [12]. A better approach would be to embed property markers within the HTML, capable of binding to the dynamic information required to build the page.

Template views are common implementations of this exact approach. A template view is a design pattern that essentially allows dynamic information to be fed into a HTML template using markers that are bound to data, retrieved from the applications backend [9], [10], [12], [13].



There are a number of advantages when using template views within MVC. Template views are usually cleaner and easier to manage because they tend to contain less logic. This inherently allows developers and web designers to work on them quickly and efficiently. Template views can also be created in a multitude of variations, whilst linking to a single controller. This means a page can be presented to a number of various users in differing configurations. This improves the usability and flexibility of an application.

## 5.0 Common MVC Frameworks

Because of the recent popularity in the use of MVC, a number of frameworks developed by various organisations have been developed or configured to take advantage of its many benefits.

A framework is essentially a set of reusable classes, methods and properties, which can be used as a standard skeleton to any application [14]. Many frameworks are clean, well designed and provide many common low level functionalities that are required within many applications. Having these available and working from the start allows developers to focus more on aspects related to the individual project.

Modern frameworks that support MVC can be accessed for a majority of languages and technologies.

### **ASP.NET MVC**

A popular framework developed by Microsoft, ASP.NET MVC is a server side development environment used within many software projects. The framework is regularly updated and has a growing community of developers. ASP.NET MVC is highly testable and supports many common patterns and methodologies such as inversion of control and dependency injection [15].

ASP.NET MVC can take advantage of a number of view engines, meaning developers have more options when it comes to constructing their projects. Two of the most common view engines used within the framework are web forms and razor [16]. These view engines provide much of the binding functionality required to construct reusable template views.

ASP.NET MVC also comes with a vast amount of pluggable components which provide much of the low level functionalities consistent between many applications. These are all highly extensible meaning they can easily be upgraded, changed or reconfigured in the future.

### ***Symfony***

Symfony is another server side framework developed using PHP. Symfony provides much of the same features as ASP.NET. Components are bundled meaning they can be configured as required without affecting other parts of the framework. As with ASP.NET, these components are again highly extensible, giving developers a substantial amount of tools to build any application that may be required [17].

Symfony has been growing for some time. The framework is regularly maintained and has a relatively large community of developers. This means that support for the framework is plentiful, again improving the development process.

### ***Angular***

Angular is a client side framework which primarily uses HTML and Typescript as its fundamental languages. In recent years, Angular has started to integrate MVC into its framework, especially within the later Angular2 editions [18].

As with the other frameworks, Angular is again a highly extensible and pluggable framework which provides a substantial amount of core functionality through components, modules and templates. The framework also provides much of the binding and routing operations required by most, if not all enterprise applications [18].

Angular works on multiple platforms and can also integrate other third party frameworks. For example, UI frameworks such as Ionic [19] can be easily integrated with angular to provide a full feature rich system.

In identifying a number of MVC inspired frameworks, a number of key similarities can be drawn out from them. They all aspire to achieve complete flexibility and extensibility with other programs. All identified frameworks also make a point of being incredibly clean and easy to work with. This seems to cater to simplifying the development process as much as possible.

Advantages developers can benefit from include the increased speed in which a system can be developed. This is because developers do not have to spend time creating repetitive functionalities which are common throughout many systems. Frameworks also tend to be highly tested and secure, taking many of these worries and issues away from the developers.

## **6.0 Conclusions and Recommendations**

Increasing the efficiency and maintainability of complex applications have been a major focus for some time. This has inevitably introduced a number of techniques and methodologies, specifically designed to improve code reusability, testability and structure. Within this report, the Model, View, Controller (MVC) pattern was critically researched for its suitability within enterprise applications.

The core components of MVC has been discussed and analysed to determine their correct usages. The report also effectively highlighted common issues and complexities when working within an MVC architecture. Combating these issues is mainly down to the developer, where careful planning has to be undertaken to ensure there are creating maintainable applications.

The report also identified a number of pre-fabricated frameworks which have been developed to take advantage of MVC. These are readily available and provide a multitude of benefits to developers and their organizations alike.

Due to its suitability for larger enterprise applications, the evidence gathered suggests that MVC will continue to grow throughout a number of development communities. This enforces the fact that MVC is now a fundamental design implementation that any software engineer or web developer should definitely make the effort to learn.

## 7.0 Reference List

- [1] K. Choudhary, "Difference Between MVC And Web Forms," 2017. [Online]. Available: <https://www.c-sharpcorner.com/article/difference-between-mvc-and-web-forms/>. [Accessed: 02-Mar-2018].
- [2] N. Heidke, J. Morrison, and M. Morrison, "Assessing the Effectiveness of the Model View Controller Architecture for Creating Web Applications."
- [3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*. 1994.
- [4] M. Kalelkar, P. Churi, and D. Kalelkar, "Implementation of Model-View-Controller Architecture Pattern for Business Intelligence Architecture," *Int. J. Comput. Appl.*, vol. 102, no. 12, pp. 975–8887, 2014.
- [5] M. Fowler, "Page Controller." [Online]. Available: <https://martinfowler.com/eaCatalog/pageController.html>. [Accessed: 02-Mar-2018].
- [6] M. Fowler, "Front Controller." [Online]. Available: <https://www.martinfowler.com/eaCatalog/frontController.html>. [Accessed: 02-Mar-2018].
- [7] M. Fowler, "Application Controller." [Online]. Available: <https://martinfowler.com/eaCatalog/applicationController.html>. [Accessed: 02-Mar-2018].
- [8] A. Homer, "Controller Patterns for ASP.NET," *DeveloperFusion*, 2011. [Online]. Available: <https://www.developerfusion.com/article/9450/controller-patterns-for-aspnet/>. [Accessed: 02-Mar-2018].
- [9] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- [10] M. Zandstra, *PHP Objects, Patterns and Practice*, 5th ed. Liverpool: Apress, 2016.
- [11] J. Deacon, "Model-View-Controller (MVC) Architecture," 2013.
- [12] J. E. Sweat, *Architect's Guide to PHP Design Patterns*. Toronto: Marco Tabini, 2005.
- [13] M. Fowler, "Template View." [Online]. Available: <https://martinfowler.com/eaCatalog/templateView.html>. [Accessed: 03-Mar-2018].
- [14] N. M. Edwin, "Software Frameworks, Architectural and Design Patterns," *J. Softw. Eng. Appl.*, 2014.
- [15] Microsoft, "ASP.NET MVC Overview." [Online]. Available: [https://msdn.microsoft.com/en-us/library/dd381412\(v=vs.108\).aspx](https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx). [Accessed: 03-Mar-2018].

- [16] Microsoft, "Working with HTML Forms in ASP.NET Web Pages (Razor) Sites | Microsoft Docs." [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/web-pages/overview/ui-layouts-and-themes/4-working-with-forms>. [Accessed: 03-Mar-2018].
- [17] Sensio Labs, "Symfony Components." [Online]. Available: <https://symfony.com/components>. [Accessed: 03-Mar-2018].
- [18] "Angular - Architecture overview." [Online]. Available: <https://angular.io/guide/architecture>. [Accessed: 03-Mar-2018].
- [19] "Ionic Documentation." [Online]. Available: <https://ionicframework.com/docs/>. [Accessed: 03-Mar-2018].

## 8.0 Implementation Code

For the implementation section of this project, the decision was taken to decompose my previous MVC implementation and rebuild it to take advantage of the request/response pattern learnt about during this half of the module. The reason for this is to build upon current understandings of MVC and to try and develop a custom MVC framework which can be updated and used in the future.

The decision was also taken to not use a pre build framework such as symfony. This was to ensure that the core functionalities and issues within MVC, were identified, addressed and an understanding was formed around them.

To keep the implementation as simple as possible, only the registration and login procedures have been focused on. As of now, a user can currently register and login using their credentials. The code completed here will form the basis to my own custom authentication service which can be reused in the future.

Some improvements that could be made include the integration of a page routing system. Currently the command pattern system is used which could potentially grow to be hard to maintain as the list of commands grows. The application could also do with the correct implementation of page controllers. These should be created and assigned dynamically and will greatly improve the readability and efficiency of the code. The in class MVC example was used as a guide during the development.

### ***File Structure***

#### MVCWebsite

##### Commands

- Command.php
- CommandResolver.php
- DefaultCommand.php
- DisplayRegisterCommand.php
- DisplaySignInCommand.php
- LoginCommand.php
- RegisterCommand.php

##### Controllers

- ApplicationController.php
- FrontController.php

##### Models

- BaseObject.php - Used my own earlier Implementation
- DataValidation.php - Used my own earlier Implementation
- DbConnection.php - Used my own earlier Implementation (Heavily Changed)
- Registry.php
- Request.php
- Response.php
- SPDO.php - Used my own earlier Implementation
- User.php - Used my own earlier Implementation

##### Views

- DefaultView.php
- HomeView.php
- RegisterView.php - Used my own earlier Implementation
- SignInView.php - Used my own earlier Implementation
- View.php

##### Index.php

- Styles.css - Used my own earlier Implementation (Heavily Changed)

### ***Command.php***

```
<?php
namespace Commands;

use \Models\Request;
use \Models\Response;
require_once 'Models/Request.php';
require_once 'Models/Response.php';

/*
 * Command Pattern
 */
abstract class Command
{
    public final function __construct() {}

    public abstract function execute(Request $request);
}
?>
```

### ***CommandResolver.php***

```
<?php
namespace Commands;

use \Models\Registry;
use \ReflectionClass;
use \Models\Request;
use \Models\Response;
use \Commands\DefaultCommand;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Response.php';
require_once 'Models/Registry.php';
require_once 'Commands/DefaultCommand.php';

class CommandResolver
{
    /*
     * Globals
     */
    static private $baseCommandCopy;
    static private $defaultCommand;
```

```

/*
 *      Constructor
 */
public function __construct()
{
    if(isset(self::$baseCommandCopy) == false)
    {
        self::$baseCommandCopy = new
ReflectionClass("\Commands\Command");
        self::$defaultCommand = new DefaultCommand();
    }
}

/*
 *      getCommand - Function to get the relevant command from the request
 */
public function getCommand(Request $request)
{
    $cmd = $request->getProperty(Registry::COMMAND_IDENTIFIER);
    $sep = DIRECTORY_SEPARATOR;

    if(isset($cmd) == false)
    {
        return clone self::$defaultCommand;
    }

    // Replace any dots or separator characters with empty string
    $cmd = str_replace(array('.', $sep), "", $cmd);

    $filepath = "Commands/{ $cmd }Command.php";

    // Class name including namespace
    $classname = "\Commands\{ $cmd }Command";

    if(file_exists($filepath))
    {
        @require_once $filepath;    // @ = Suppress error report

        // Having loaded the class file, check that the
        // class wasn't actually defined in this file
        if(class_exists($classname))
        {
            // Now use reflection to reverse-engineer classes, etc to
            // information about them. In our case, we will also use
            // to create an object from the class we just loaded.

```

```

read                                     // Start by created ReflectionClass object that allows us to
                                         // information about the class
                                         $reflectionClass = new ReflectionClass($classname);

                                         // One more security check, make sure this class inherits
from the                                // Command class
                                         if($reflectionClass-
>isSubclassOf(self::$baseCommandCopy))
{
    // Instantiate (create) and object using this class
    return $reflectionClass->newInstance();
}
else
{
    $response = Registry::getInstance()-
>getCurrentResponse();
    $response->setOutputProperty('Error', "Command
{$cmd}Command is not a command");
    return clone self::$defaultCommand; // Return
copy of default command
}
}
}

$response = Registry::getInstance()->getCurrentResponse();
$response->setOutputProperty('Error', "Command {$cmd}Command not
found");
return clone self::$defaultCommand; // Return copy of default command
}
}

?>

```

### ***DefaultCommand.php***

```
<?php
namespace Commands;

use \Commands\Command;
use \Models\Registry;
use \Models\Request;
use \Models\Response;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Registry.php';
require_once 'Models/Response.php';

class DefaultCommand extends Command
{
    /**
     * execute - Override Function to execute request
     */
    public function execute(Request $request)
    {
        $response = Registry::getInstance()->getCurrentResponse();

        $response->setResponseView("DefaultView");

        $response->setOutputProperty(Registry::COMMAND_IDENTIFIER,
        $request->getRequestProperty(Registry::COMMAND_IDENTIFIER));

        return $response;
    }
}
?>
```

### ***DisplayRegisterCommand.php***

```
<?php
namespace Commands;

use \Commands\Command;
use \Models\Registry;
use \Models\Request;
use \Models\Response;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Registry.php';
require_once 'Models/Response.php';

class DisplayRegisterCommand extends Command
{
    /*
     *      execute - Override Function to execute request
     */
    public function execute(Request $request)
    {
        $response = Registry::getInstance()->getCurrentResponse();

        $response->setResponseView("RegisterView");

        $response->setOutputProperty(Registry::COMMAND_IDENTIFIER,
        $request->getRequestProperty(Registry::COMMAND_IDENTIFIER));

        return $response;
    }
}
?>
```

### ***DisplaySignInCommand.php***

```
<?php
namespace Commands;

use \Commands\Command;
use \Models\Registry;
use \Models\Request;
use \Models\Response;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Response.php';
require_once 'Models/Registry.php';

class DisplaySignInCommand extends Command
{
    /**
     * execute - Override Function to execute request
     */
    public function execute(Request $request)
    {
        $response = Registry::getInstance()->getCurrentResponse();

        $response->setResponseView("SignInView");

        $response->setOutputProperty(Registry::COMMAND_IDENTIFIER,
        $request->getRequestProperty(Registry::COMMAND_IDENTIFIER));

        return $response;
    }
}
?>
```

### **LoginCommand.php**

```
<?php
namespace Commands;

use \Commands\Command;
use \Models\Registry;
use \Models\Request;
use \Models\Response;
use \Models\DbConnection;
use \Models\DataValidation;
use Models\User;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Registry.php';
require_once 'Models/Response.php';
require_once 'Models/DbConnection.php';
require_once 'Models/DataValidation.php';
require_once 'Models/User.php';

class LoginCommand extends Command
{
    /*
     *      execute - Override Function to execute request
     */
    public function execute(Request $request)
    {
        //Get Response Ready
        $response = Registry::getInstance()->getCurrentResponse();

        //Validate Data
        if (self::validateData($request))
        {
            //Login
            if (self::loginUser($request))
            {
                //Success
                $response->setResponseView("HomeView");

                $response-
>setOutputProperty(Registry::COMMAND_IDENTIFIER, $request-
>getRequestProperty(Registry::COMMAND_IDENTIFIER));

                //Add the user details to the response so we can use
them elsewhere

                $db = DbConnection::getInstance();
                $db->openConnection();
                $newUser = $db->getUser($request-
>getRequestProperty("usernameInput")); //Usernames are unique
                $db->closeConnection();

                $response-
>setOutputProperty(Registry::USER_IDENTIFIER, $newUser);
```

```

    }
    else
    {
        //Failed
        $response->setResponseView("SignInView");

        $response-
>setOutputProperty(Registry::COMMAND_IDENTIFIER, $request-
>getRequestProperty(Registry::COMMAND_IDENTIFIER));
    }
}
else
{
    //Build Response
    $response->setResponseView("SignInView");

    $response-
>setOutputProperty(Registry::COMMAND_IDENTIFIER, $request-
>getRequestProperty(Registry::COMMAND_IDENTIFIER));
}

return $response;
}

/*
 * validateData - Function to validate login data
 */
private function validateData(Request $request)
{
    if (DataValidation::isUsernameValid($request-
>getRequestProperty("usernameInput")))
    {
        if (DataValidation::isPasswordValid($request-
>getRequestProperty("userPassword")))
        {
            return true;
        }
        else
        {
            echo "<script type='text/javascript'>";
            echo "alert('Enter A Valid Password')";
            echo "</script>";
        }
    }
    else
    {
        echo "<script type='text/javascript'>";
        echo "alert('Enter A Valid Username')";
        echo "</script>";
    }
    return false;
}
}

```

```

        private function loginUser(Request $request)
        {
            $db = DbConnection::getInstance();
            $db->openConnection();
            $success = false;

            if ($db->login($request->getRequestProperty("usernameInput"),
            $request->getRequestProperty("userPassword")) == true)
            {
                $success = true;
            }
            else
            {
                echo "<script type='text/javascript'>";
                echo "alert('Incorrect Credentials')";
                echo "</script>";
            }

            $db->closeConnection();
            return $success;
        }
    }
?>

```

### ***RegisterCommand.php***

```

<?php
namespace Commands;

use \Commands\Command;
use \Models\Registry;
use \Models\Request;
use \Models\Response;
use \Models\DbConnection;
use \Models\DataValidation;
use \Models\User;

require_once 'Commands/Command.php';
require_once 'Models/Request.php';
require_once 'Models/Registry.php';
require_once 'Models/Response.php';
require_once 'Models/DbConnection.php';
require_once 'Models/DataValidation.php';
require_once 'Models/User.php';

class RegisterCommand extends Command
{
    /*
     *      validateData - Function to validate registration data
     */
    private function validateData(Request $request)
    {

```

```

        if (DataValidation::isEmailValid($request-
>getRequestProperty("emailInput")))
        {
            if (DataValidation::isUsernameValid($request-
>getRequestProperty("usernameInput")))
            {
                if (DataValidation::isPasswordValid($request-
>getRequestProperty("passwordInput")))
                {
                    if (strcmp($request-
>getRequestProperty("passwordInput"), $request-
>getRequestProperty("passwordConInput")) == 0)
                    {
                        if (strlen($request-
>getRequestProperty("fNameInput")) > 0)
                        {
                            if (strlen($request-
>getRequestProperty("lNameInput")) > 0)
                            {
                                return true;
                            }
                            else
                            {
                                echo "<script
                                echo "alert('Enter A Last
                                echo "</script>";

                                }
                            }
                        }
                        else
                        {
                            echo "<script
                            echo "alert('Enter A First Name)";
                            echo "</script>";

                        }
                    }
                }
            }
            else
            {
                echo "<script type='text/javascript'>";
                echo "alert('Passwords Do Not Match)";
                echo "</script>";

            }
        }
    }
    else
    {
        echo "<script type='text/javascript'>";
        echo "alert('Enter A Valid Password)";
        echo "</script>";

    }
}
else
{
    echo "<script type='text/javascript'>";

```

```

        echo "alert('Enter A Valid Username')";
        echo "</script>";
    }
}
else
{
    echo "<script type='text/javascript'>";
    echo "alert('Enter A Valid Email')";
    echo "</script>";
}

return false;
}

/*
 *   registerUser - Function to register a user
 */
private function registerUser(Request $request)
{
    //Register User (Check For First User and Whether email and username
    has been registered)
    $db = DbConnection::getInstance();
    $db->openConnection();

    if ($db->isEmailRegistered($request-
>getRequestProperty("emailInput")) == false)
    {
        if ($db->isUsernameRegistered($request-
>getRequestProperty("usernameInput")) == false)
        {
            $user = new User();
            $user->setFName($request-
>getRequestProperty("fNameInput"));
            $user->setLName($request-
>getRequestProperty("lNameInput"));
            $user->setEmail($request-
>getRequestProperty("emailInput"));
            $user->setUsername($request-
>getRequestProperty("usernameInput"));
            $user->setPass($request-
>getRequestProperty("passwordInput"));

            $userId = $db->addUser($user);

            if ($db->getUserCount() == 1)
            {
                $db->promoteToAdmin($userId);
            }
        }
        else
        {
            echo "<script type='text/javascript'>";
            echo "alert('Username Has Already Been Registered')";

```

```

        echo "</script>";
    }
}
else
{
    echo "<script type='text/javascript'>";
    echo "alert('Email Address Has Already Been Registered')";
    echo "</script>";
}
$db->closeConnection();
}

/*
 *      execute - Override Function to execute request
 */
public function execute(Request $request)
{
    //Get Response Ready
    $response = Registry::getInstance()->getCurrentResponse();

    //Validate Data
    if (self::validateData($request))
    {
        //Add User
        self::registerUser($request);

        //Build Response
        $response->setResponseView("SignInView");

        $response-
>setOutputProperty(Registry::COMMAND_IDENTIFIER, $request-
>getRequestProperty(Registry::COMMAND_IDENTIFIER));
    }
    else
    {
        //Build Response
        $response->setResponseView("RegisterView");

        $response-
>setOutputProperty(Registry::COMMAND_IDENTIFIER, $request-
>getRequestProperty(Registry::COMMAND_IDENTIFIER));
    }

    return $response;
}

}
?>

```

### ***ApplicationController.php***

```
<?php
namespace Ctrl;

use \Models\Request;
use \Models\Response;
use \Views\View;

require_once 'Models/Request.php';
require_once 'Models/Response.php';
require_once 'Views/View.php';

class ApplicationController
{
    /**
     *      Globals
     */
    private static $views;

    /**
     *      loadViews - Function to load the applications views into a map
     */
    public static function loadViews()
    {
        self::$views = array(
            'DefaultView' => new View("\Views\DefaultView.php"),
            'HomeView' => new View("\Views\HomeView.php"),
            'RegisterView' => new View("\Views\RegisterView.php"),
            'SignInView' => new View("\Views\SignInView.php")
        );
    }

    /**
     *      getView - Function geta specific view from the map
     */
    public static function getView(Response $response)
    {
        return self::$views[$response->getResponseView()];
    }
}
?>
```

### ***FrontController.php***

```
<?php
namespace Ctrl;

use \Models\Request;
use \Models\Response;
use \Models\Registry;
use \Commands\CommandResolver;
use \Ctrl\ApplicationController;
use \Views\View;

require_once 'Commands/CommandResolver.php';
require_once 'Models/Request.php';
require_once 'Models/Registry.php';
require_once 'Models/Response.php';
require_once 'Controllers/ApplicationController.php';
require_once 'Views/View.php';

class FrontController
{
    /**
     *      Globals
     */
    private static $instance;
    private $commandResolver;

    /**
     *      Constructor
     */
    private function __construct() {}

    /**
     *      getInstance - Static function to get class instance (singleton pattern)
     */
    public static function getInstance()
    {
        if(!isset(self::$instance))
        {
            self::$instance = new self();
            self::$instance->commandResolver = new CommandResolver();
            ApplicationController::loadViews();
        }
        return self::$instance;
    }

    /**
```

```

        *      handleRequest - Function to handle a request
        */
        public function handleRequest(Request $request)
        {
            //Default Request
            if ($request->getRequestProperty(Registry::COMMAND_IDENTIFIER)
== ""))
            {
                $request-
>setRequestProperty(Registry::COMMAND_IDENTIFIER, "DisplaySignIn");
            }

            //Get and execute the correct command
            $command = $this->commandResolver->getCommand($request);
            $response = $command->execute($request);

            //Get and Render the correct view
            $view = ApplicationController::getView($response);

            $view->render();
        }
    }
?>

```

### **Registry.php**

```
<?php
namespace Models;

use \Models\BaseObject;
require_once 'Models/BaseObject.php';
require_once 'Models/Request.php';
require_once 'Models/Response.php';

class Registry extends BaseObject
{
    /**
     *      Globals
     */
    private static $instance;
    private $request;
    private $response;
    const VIEW_KEY = "view";
    const COMMAND_IDENTIFIER = "cmd";
    const USER_IDENTIFIER = "usr";

    /**
     *      Constructor
     */
    private function __construct() {}

    /**
     *      getInstance - Static function to get class instance (singleton pattern)
     */
    public static function getInstance()
    {
        //If already exists, return it
        if(isset(self::$instance))
        {
            self::$instance = new self();
            self::$instance->request = new Request();
            self::$instance->response = new Response();
        }
        return self::$instance;
    }

    /**
     *      getCurrentRequest - Function to set the current request
     */
}
```

```

public function setCurrentRequest(Request $req)
{
    $this->request = $req;
}

/*
 *    getCurrentRequest - Function to get the current request
 */
public function getCurrentRequest()
{
    return $this->request;
}

/*
 *    getCurrentResponse - Function to set the current response
 */
public function setCurrentResponse(Response $res)
{
    $this->response = $res;
}

/*
 *    getCurrentResponse - Function to get the current response
 */
public function getCurrentResponse()
{
    return $this->response;
}

}
?>

```

### ***Request.php***

```
<?php
namespace Models;

class Request
{
    /**
     *      Globals
     */
    private $requestProperties;

    /**
     *      Constructor
     */
    public function __construct()
    {
        //Check for any properties coming in with the request
        if(isset($_SERVER['REQUEST_METHOD']) &&
($_SERVER['REQUEST_METHOD'] == "POST" || $_SERVER['REQUEST_METHOD']
== "GET"))
        {
            //Store Request Properties
            $this->requestProperties = $_REQUEST;
        }
        else
        {
            //Default Properties
            $this->requestProperties = array();
        }
    }

    /**
     *      setRequestProperty - Function to set and store a property
     */
    public function setRequestProperty($propertyKey, $value)
    {
        $this->requestProperties[$propertyKey] = $value;
    }

    /**
     *      getRequestProperty - Function to get a stored property
     */
    public function getRequestProperty($propertyKey)
    {
        if(isset($this->requestProperties[$propertyKey]))
```

```
        {
            return $this->requestProperties[$propertyKey];
        }
        return;
    }
?>
```

### ***Response.php***

```
<?php
namespace Models;

use \Models\Registry;
require_once 'Models/Registry.php';

class Response
{
    /**
     *      Globals
     */
    private $responseOutputProperties = array();

    /**
     *      Constructor
     */
    public function __construct(){}

    /**
     *      setOutputProperty - Function to set and store an output property
     */
    public function setOutputProperty($propertyKey, $value)
    {
        $this->responseOutputProperties[$propertyKey] = $value;
    }

    /**
     *      getOutputProperty - Function to set and store an output property
     */
    public function getOutputProperty($propertyKey)
    {
        if(isset($this->responseOutputProperties[$propertyKey]))
        {
            return $this->responseOutputProperties[$propertyKey];
        }
        return;
    }

    /**
     *      setResponseView - Function to set and store the correct view for this
     */
    response
```

```

        */
        public function setResponseView($view)
        {
            if(is_string($view))
            {
                //Use 'view' as the key
                $this->responseOutputProperties[Registry::VIEW_KEY] = $view;
            }
        }

        /*
        *      setResponseView - Function to set and store the correct view for this
response
        */
        public function getResponseView()
        {
            if(isset($this->responseOutputProperties[Registry::VIEW_KEY]))
            {
                return $this->responseOutputProperties[Registry::VIEW_KEY];
            }
            return false;
        }

    }
?>

```

### **DefaultView.php**

```
<?php
namespace Views;

use \Models\Registry;
use \Models\Response;

require_once 'Models/Registry.php';
require_once 'Models/Response.php';

$response = Registry::getInstance()->getCurrentResponse();
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>Default View</title>
</head>
<body>
<?php
    echo "Default View ... command &quot;{$response-
>getOutputProperty(Registry::COMMAND_IDENTIFIER)}&quot; is probably incorrect";
?>
</body>
</html>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
        <link rel='stylesheet' type='text/css'
href='http://localhost/secondWebAssignment/MVCWebsite/styles.css' />
        <title>Home</title>
    </head>
    <body>
        <?php
            echo "Error : Default View/Command &quot;{$response-
>getOutputProperty(Registry::COMMAND_IDENTIFIER)}&quot; is probably incorrect";
?>
    </body>
</html>
```

### **HomeView.php**

```
<?php
namespace Views;

use \Models\Registry;
use \Models\Response;

require_once 'Models/Registry.php';
require_once 'Models/Response.php';

$response = Registry::getInstance()->getCurrentResponse();
$response->getOutputProperty(Registry::USER_IDENTIFIER) ? $currentUsr =
$response->getOutputProperty(Registry::USER_IDENTIFIER) : $currentUsr = null;

echo '<script>';
echo 'console.log( . json_encode( $currentUsr->getEmail() ) .)';
echo '</script>';
?>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
        <link rel='stylesheet' type='text/css'
href='http://localhost/secondWebAssignment/MVCWebsite/styles.css' />
        <title>Home</title>
    </head>
    <body>
        <div id='signInForm'>
            <h1 class='basicLabel'>Successful Login</h1>
            <span class='formLabels'>First Name : <?php echo $currentUsr-
>getFName()?></span><br/>
            <span class='formLabels'>Last Name : <?php echo $currentUsr-
>getLName()?></span><br/>
            <span class='formLabels'>Email Address : <?php echo
$currentUsr->getEmail()?></span><br/>
            <span class='formLabels'>Username : <?php echo $currentUsr-
>getUsername()?></span><br/>
        </div>
    </body>
</html>
```

### **RegisterView.php**

```
<?php
namespace Views;

use \Models\Registry;
use \Models\Response;

require_once 'Models/Registry.php';
require_once 'Models/Response.php';

$response = Registry::getInstance()->getCurrentResponse();
?>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
        <link rel='stylesheet' type='text/css'
href='http://localhost/secondWebAssignment/MVCWebsite/styles.css' />
        <title>Register</title>
    </head>
    <body>
        <div id='signInForm'>
            <h1>Registration</h1>
            <form action='?cmd=Register' method='post'>
                <span class='formLabels'>First Name</span><br/>
                <input class='formInputs'type='text' name='fNameInput'
value=""/><br/><br/>
                <span class='formLabels'>Last Name</span><br/>
                <input class='formInputs'type='text' name='lNameInput'
value=""/><br/><br/>
                <span class='formLabels'>Email Address</span><br/>
                <input class='formInputs'type='text' name='emailInput'
value=""/><br/><br/>
                <span class='formLabels'>Username</span><br/>
                <input class='formInputs'type='text' name='usernameInput'
value=""/><br/><br/>
                <span class='formLabels'>Password</span><br/>
                <input class='formInputs' type='password' name='passwordInput'
value=""/><br/><br/>
                <span class='formLabels'>Confirm Password</span><br/>
                <input class='formInputs' type='password'
name='passwordConInput' value=""/><br/><br/>
                <div class='submitBtns'>
                    <input type = 'submit' value = 'Register' />
                </div>
            </form>
        </div>
    </body>
</html>
```

### ***SignInView.php***

```
<?php
namespace Views;

use \Models\Registry;
use \Models\Response;

require_once 'Models/Registry.php';
require_once 'Models/Response.php';

$response = Registry::getInstance()->getCurrentResponse();
?>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
        <link rel='stylesheet' type='text/css'
href='http://localhost/secondWebAssignment/MVCWebsite/styles.css' />
        <title>Sign In</title>
    </head>
    <body>
        <div id='signInForm'>
            <h1>Enter If You Dare</h1>
            <form action='?cmd=Login' method='post'>
                <span class='formLabels'>Username</span><br/>
                <input class='formInputs' type='text'
name='usernameInput' value='/'><br/><br/>
                <span class='formLabels'>Password</span><br/>
                <input class='formInputs' type='password'
name='userPassword' value='/'><br/><br/>
                <div class='submitBtns'>
                    <input type = 'submit' value = 'Sign In'>
                </div>
                <br/><br/>
                <a href='?cmd=DisplayRegister'
id='registerLink'>Register</a>
            </form>
        </div>
    </body>
</html>
```

### **View.php**

```
<?php
namespace Views;

class View
{
    /**
     *      Globals
     */
    private $filepath;

    /**
     *      Constructor
     */
    public function __construct($filepath)
    {
        $this->filepath = $filepath;
    }

    /**
     *      render - Function to render a view
     */
    public function render()
    {
        include($this->filepath);
        exit();
    }
}
?>
```

### ***Index.php***

```
<?php
use \Models\Registry;
use \Ctrl\FrontController;
use \Models\Request;

require_once 'Models/Registry.php';
require_once 'Models/Request.php';
require_once 'Controllers/FrontController.php';

//Get instance of front controller (using Singleton pattern)
$controller = FrontController::getInstance();

//Get the Request object and feed it to the front controller
$request = Registry::getInstance()->getCurrentRequest();
$controller->handleRequest($request);
?>
```

### ***Styles.php***

```
html, body {  
  margin: 0;  
  padding: 0;  
  font-size: 100%; }  
  
body {  
  font-family: "Roboto Condensed", Arial, sans-serif;  
  background: #000000; }
```

```
#signInForm  
{  
  text-align: center;  
  min-height: 50vh;  
}
```

```
#signInForm > form  
{  
  text-align: left;  
  margin-left: 39%;  
  margin-right: 39%;  
}
```

```
h1  
{  
  font-size: 400%;  
  margin-top: 10%;  
  margin-bottom: 60px;  
  color: #ffffff;  
}
```

```
.basicLabel  
{  
  text-align:center;  
}
```

```
.formLabels  
{  
  color: #ffffff;  
  margin-left: 10px;  
  margin-right: 20px;  
  font-size: 120%;  
}
```

```
.formInputs  
{  
  width: 100%;  
}
```

```
.submitBtns  
{
```

```
        text-align: center;
        margin-right: 20%;
    }

    .submitBtns > input
    {
        background-color: transparent;
        border: 0px solid transparent;
        color: #ffffff;
        font-size: 200%;
        font-weight: bolder;
        margin-top: 30px;
        margin-left: 30%;
    }

    .submitBtns > input:hover
    {
        color: #ffff00;
    }

    #registerLink
    {
        color: #ffffff;
        text-decoration: none;
        font-size: 120%;
        margin-left: 30%;
    }

    #registerLink:hover
    {
        text-decoration: underline;
    }
```

# khawkins\_p145441\_assignmentTwoReport

## GRADEMARK REPORT

### FINAL GRADE

78/100

### GENERAL COMMENTS

#### Instructor

Investigation of Fowler (30) 26

Investigation of MVC frameworks (20) 16

Quality of website and techniques used (25) 18

View handling (10) 6

Database handling (15) 12

Fowler concepts are covered in good detail, you could have provided some simple examples to illustrate your answer.

You have included a JS MVC framework in your discussion which is OK. Also it would have been a good idea to briefly identify a set of metrics by which to compare the frameworks.

MVC application built using provided framework demonstrating registering new users and user login. Code has been made cleaner, although application is a little small.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6



#### Comment 1

Very good point



## Comment 2

Good point

---

PAGE 33

---

PAGE 34

---

PAGE 35

---

PAGE 36

---

PAGE 37

---

PAGE 38

---

PAGE 39

---

PAGE 40

---

PAGE 41

---