

Assignment2LeahCahill

by Leah Cahill

FILE	2254_LEAH_CAHILL_ASSIGNMENT2LEAHCAHILL_93284_891421587.DOC X		
TIME SUBMITTED	22-MAR-2018 08:42AM (UTC+0000)	WORD COUNT	6905
SUBMISSION ID	85280596	CHARACTER COUNT	54669

Advanced Web Development

ASSIGNMENT 2

LEAH CAHILL (P140092)

Contents

Part A Understanding.....	3
Understanding the Front controller.....	3
Discussing Fowlers statement.....	3
Discussing the Page Controller.	4
Summary	4
Investigating the template pattern.....	5
Investigating PHP MVC Frameworks.....	6
Why should we use frameworks?	6
MVC PHP Frameworks	6
Understanding the structure of an MVC application.....	8
Exploring the concept of ORMs in PHP	12
Creating a simple REST API in PHP	15
Expanding upon the code created for simple REST API	17
Exploring the concept of "Convention over configuration"	18
Creating Views in the application	19
Displaying a database into the views	21
References	26
Appendix	29
Code for basic MVC application.....	29
Code for exploring the concept of creating an ORM	32
Code for creating a REST API.....	35
Full code for the MVC application	38
Commands	38
Views.	52

Figure 1- Template Pattern Structure [11].....	5
Figure 2- \$view->myTitle();	8
Figure 3- Building upon the code and displaying a database	9
Figure 4- Loading two templates	11
Figure 5- Screenshot showing structure of application	18

Part A Understanding

Understanding the Front controller.

A front controller handles all the requests that come into a web application. The front controller uses a dispatcher object which dispatches the incoming request, and everything is handled by a single handler. This approach of implementing a front controller integrates with the MVC pattern as it has a controller, dispatcher, helper, model and a view which are all separate components of the application [1].

The front controller logic is kept separate to the view so when requests are made, the views create an object of this request [2]. The front controller provides a single-entry point for managing the requests that the website receives but also has the disadvantage of potentially being complex to implement [3].

Discussing Fowlers statement

Martin Fowler states:

“In a complex Web site there are many similar things you need to do when handling a request. These things include security, internationalization, and providing views for certain users. If the input controller behaviour is scattered across multiple objects, much of this behaviour can end up duplicated. Also, it's difficult to change behaviour at runtime” [4]

Expanding upon this statement: If the behaviour of the code is duplicated then it increases the chance of errors such as the user inputting the wrong behaviour throughout the application which will make it harder to test and take longer to fix in comparison to all the logic being stored in the same place.

Having the code scattered throughout different files in an application could mean that maintaining it would be more complicated and time consuming, an example being if the user changes the name of an object that is related to a certain behaviour; it would mean going through all the other code to rename the other existing objects of the same name.

Duplicating code violates the DRY principle. DRY stands for Don't Repeat Yourself which states that duplication is waste [5]. Violating this principle could become a problem as it relates to other software design principles such as the Single Responsibility Principle.



The front controller is centralised with common logic, which helps solve the issue of the duplicated code and therefore, does not violate the DRY principle. The centralisation of the controller also makes it easier to configure the code when necessary which benefits the developer greatly when their application is flexible, and the centralisation also has the advantage of improving the reusability of the code [6].

As all requests go through the front controller rather than multiple objects, this helps the application enforce a level of security [7].

Discussing the Page Controller.

An alternative to the front controller is the page controller. The page controller handles specific requests on a website, it can be the page itself or a separate object. Each logical part of the website leads to a page controller which handles the specific request or action with an input controller [8].

The Front Controller handles all the requests coming into a website where as the Page Controller has a single object to each logical page. The page controller is used in situations when the navigation between the pages in the application are static, it directs the user to the correct view when they have clicked on an action [9].

In comparison to the front controller, the page controller is considered easier to implement. There are many frameworks available that have built in support to implement a page controller which reduces the complexity for the developer, however it does have the disadvantage of a lower level of security than the front controller [10].

Using the page controller approach integrates into the MVC pattern as it is the 'C'; Controller in the Model-View-Controller. The page controller sends requests to the view, so it is all kept in their separate components and shows how the logic is separate to the view.

Summary

To summarise front end controllers, this pattern:

- Make the code reusable.
- Enables the application to become flexible with high security.
- Centralising the control is easier to maintain than distributed code.
- Slightly complex to implement.

Investigating the template pattern

The template pattern is a behavioural pattern and helps dynamic web pages act as a template for text that is coming from a dynamic source such as a database. The focused data is dynamic rather than just standard and static HTML as the static HTML doesn't change between all the requests and would not change the layout or anything on the page each time it is loaded.

This pattern has been developed as dynamic information is likely to change every time with different requests and would result in the page looking different each time.

This pattern is called a template pattern as it is going to act as a literal template for dynamic information by using markers/placeholders on a web page, this ensures some consistency with how the information results upon the webpage and it uses inheritance [4].

The template can be described as a skeleton which enables subclasses to override parts of the algorithm but doesn't change the structure.

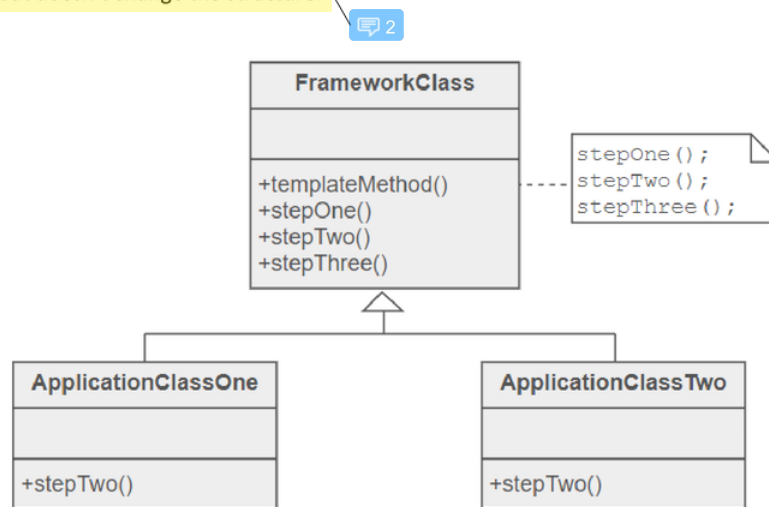


Figure 1- Template Pattern Structure [11]

Above is an example from Sourcingmaking.com [11] which demonstrates the structure of this pattern, it shows how the methods `stepTwo()` are the placeholders and defined in the derived classes.

The template pattern solves the issue of having duplicated code within your application and this is done by abstraction [12].

Investigating PHP MVC Frameworks

Why should we use frameworks?

Frameworks help developers when building complex and large applications. There are factors to consider when and why to apply a framework. First, the developer should consider the size of the project and how it may grow overtime, frameworks can help build manageable and structured applications [12].

Time is a factor to consider when choosing to work with a framework, as some developers are under pressure to meet a deadline. A framework may help with rapid development, or in larger teams, different developers could work on a component of the MVC application then integrate their work/files together when finished [13].

As many frameworks use the MVC structure for managing their code, this enables the benefit of the code becoming more extendable. Organising code with MVC enables the basic advantages of ensuring that the code is maintainable, and it becomes easier to debug when the developer is working through organised code.

MVC PHP Frameworks

There are many PHP MVC frameworks available and this section will discuss these and talk about the benefits and drawbacks of applying them. PHP MVC frameworks identified in the research are:

- CodeIgniter
- Zend Framework
- Phalcon
- PHPixie
- Fat-Free Framework
- CakePHP

Each framework has their own features, benefits and drawbacks to each one, the three chosen for discussion are CodeIgniter, Phalcon and CakePHP.

CodeIgniter is a lightweight open-source framework used to build dynamic websites with PHP, it was released in 2016. CodeIgniter includes features such as scaffolding, full page layout and meeting management [14].

The benefits to applying this type of framework are that it is very light, it supports rapid development, it is simple to configure and is flexible. An advantage is that this framework is considered easy to learn, customisable and has a high performance [15], however, a downside is that CodeIgniter currently does not have built in ORM [16].

Phalcon is another popular framework that creates web applications with PHP 5, the aim for this framework is to create highly configurable web applications and was released in 2012. Phalcon includes important features of PHP applications such as auto-loading and ORM (object-relational mapping) [17].

A benefit to applying this framework is that the learning curve is considered relatively easy and the overhead is little compared to others [18].

CakePHP is also a very popular open source framework that was released in December 2016 and is considered good for commercial applications. As it follows the MVC pattern, it has the advantages of defining and separating the business and logic layers [19].

A second advantage is that this framework can easily be extended with plug-ins. There are a lot of existing plug-in helpers available online, this helps the developer save time than writing a plug-in from scratch. CakePHP makes use of plug-ins and helpers but that is not all, its features also enable Ajax support, includes a useful ORM and has validation and authentication built in [20].

All these frameworks are suited towards a different level of expertise. CakePHP is considered most suited for experts to use, Phalcon is considered for intermediates and CodeIgniter is suited for beginners and those who are just getting started out with a basic knowledge of OOP and PHP. The disadvantage to the different levels of expertise is that every framework will take some time to learn and fully understand which will be an issue if the developer does not have much time to develop the application and have never used the framework before [21].

All three of these frameworks discussed are open source frameworks which benefits the developer when it comes to considering the cost of development.

An advantage for these three frameworks being so popular with PHP development, is that they have good documentation online and a lot of community support which will help the developer get started and learn about the framework, certain features and any help if required [22].

Each framework has their structures and different levels of user-friendliness. CodeIgniter and Phalcon are both considered developer friendly frameworks, however, the unit testing in CodeIgniter is considered not so user friendly compared to the other frameworks.

Considering the performance of these frameworks, Phalcon and CodeIgniter both enable high performance in their applications, whereas CakePHP is not so lightweight in comparison and results in a slower performance [23].

All three of these frameworks have the general advantages of supporting database connections such as MySQL, built in Core Libraries and allowing the code to become reusable which will save development time in future applications [24].

Understanding the structure of an MVC application.

This section begins by implementing a basic MVC structure to test and understand how each of the individual components work. This section displays a basic structure, then builds on that by displaying a database table from a function in the view and then goes on to loading two templates into the view [25]:

The Controller-

```
<?php
class Controller
{ private $model;
  public function __construct($model);
  $this->model = $model;

  } }
```

The View-

```
<?php
class View
{
  private $model;
  private $controller;
  public function __construct($controller,$model) {
    $this->controller = $controller;
    $this->model = $model;
  }
  public function myTitle (){
    return "<p>" . $this->model->string . "</p>" }
}
```

The Model-

```
<?php
class Model
{
  public $string;
  public function __construct(){
    $this->string = "Title of page"; } } ?>
```

Connecting them together, Index.php -

```
<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';

$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
?><html><head><h2><?php echo $view->myTitle(); ?></h2>
```



Figure 2- \$view->myTitle();

2. Displaying a database table – Adding a function into the View code shown above:

public function DisplayTable()

```
{
$dsn = 'mysql:host=localhost;dbname=assignmenttwo';
$dbUName = "root";
$dbPassword = "";
$dbh = new PDO($dsn, $dbUName, $dbPassword);

foreach($dbh->query('SELECT * FROM customer') as $record)

    {echo "<tr>" .
    "<td>" . $record["firstname"] . "</td>" .
    "<td>" . $record["lastname"] . "</td>" .
    "<td>" . $record["Email"] . "</td>" .
    "</tr>";
    }
}
```

Displaying the function in Index.php

```
<h2><?php echo $view->output(); ?></h2><h4>Customer Details</h4>
</head><body>
<table border=1>
<thead><tr><th>First name</th><th>Last name</th><th>Email</th></tr></thead>
<?php echo $view->DisplayTable();?>
</table>
```

Testing and running in the browser:

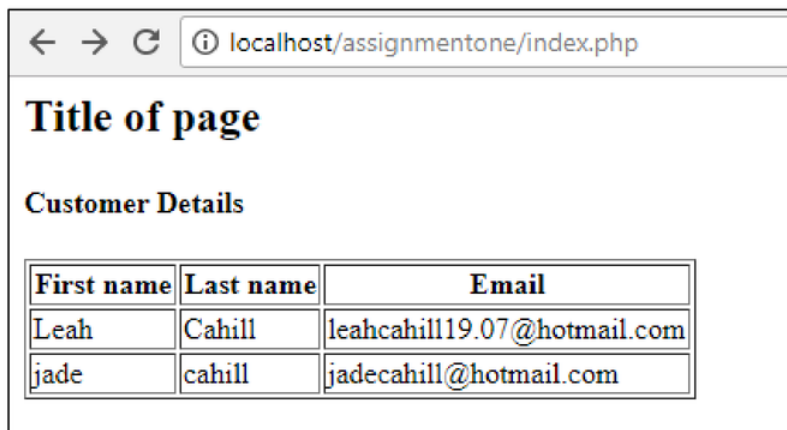


Figure 3- Building upon the code and displaying a database

3. Loading two Templates in the view - Adding the code into the existing Model [26]:

```
public $tempstring;

public function __construct() {
    $this->tempstring = "templatetext";
    $this->mytemplate = "template.php";
    $this->tempstring2 = "templatetext2";
    $this->mytemplate2 = "template2.php";
    $this->string = "Title of page";

    } }
```

Then into the View.

```
public function template1(){
    $intro = "<p>" . $this->model->tempstring . "</p>";
    require_once($this->model->mytemplate);
}
public function template2(){
    $page2 = "<p>" . $this->model->tempstring2 . "</p>";
    require_once($this->model->mytemplate2);

    }
```

The two template PHP files:

template.php-

```
<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';
$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
echo $view->template1();

?><!DOCTYPE html>
<html><head>
<meta charset="charset=utf-8">
<title></title>
</head><h1> Template Title Page </h1>
<body><h1><?php echo $intro; ?></h1></body></html>
```

template2.php-

```
<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';
$model = new Model();
$controller = new Controller($model);
```

```

$view = new View($controller, $model);
echo $view->template2();
?><!DOCTYPE html>
<html> <head><meta charset="utf-8"><title></title>
</head>
<h1> Template Title Two </h1><body><h1><?php echo $page2; ?></h1> </body></html>

```

Displaying the basic templates in Index.php:

```

<?php
echo $view->template1();
echo $view->template2();
?>

```

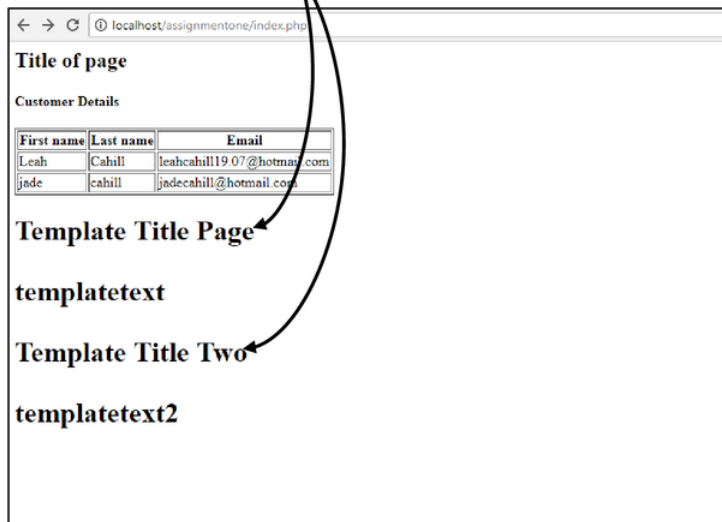


Figure 4- Loading two templates

Exploring the concept of ORMs in PHP

ORM is Object Relational Mapping and is used to convert/translate data between relational databases and the objects in object-oriented programming [27]. This section will create a simple data persistence layer that would be the start of implementing an ORM.

First, the code that interacts with mySQL database:

IdatabaseAdapter.php:

```
<?php

namespace Database;

interface DatabaseAdapterInterface
{
    function connectToDB();
    function disconnect();
    function query($query);
    function fetch();
    function select($table);
    function countRows();
    function freeData();
}
```

Then, this file below will implement the interface's methods.

SQL.php file:

```
<?php

namespace Database;

class SQL implements IdatabaseAdapter
{
    protected $_configuration = array();
    protected $_connectto;
    protected $_data;

    public function __construct(array $configuration)
    {
        if (count($configuration) != 4) {
            throw new InvalidArgumentException("Invalid");
        }
        $this->_configuration = $configuration;
    }

    public function connectToDB()
    {
        // Establishing the connection once
        if ($this->_connectto === null) {
```

```

        list($host, $user, $password, $database) = $this->_configuration;
        if (!$this->_connectto = @mysqli_connect("localhost", "root", "", "assignmenttwo")) {
            throw new RuntimeException('Error connecting to the server : ' . mysqli_connect_error());
        }
        unset($host, $user, $password, $database);
    }
    return $this->_connectto;
}

public function query($query)
{
    if (is_string($query) || empty($query)) {
        throw new InvalidArgumentException('invalid query');
    }
    $this->connectToDB();
    if (!$this->_data = mysqli_query($this->_connectto, $query)) {
        throw new RuntimeException('Error'. $query . mysqli_error($this->_connectto));
    }
    return $this->_data;
}

// Select Statement
public function select($table)
{
    $query = 'SELECT * FROM' $table;
    $this->query($query);
    return $this->countRows();
}

public function fetch()
{
    if ($this->_data !== null) {
        if (($row = mysqli_fetch_array($this->_result, MYSQLI_ASSOC)) === false) {
            $this->freeData();
        }
        return $row;
    }
    return false;
}

public function freeData()
{
    if ($this->_result === null) {
        return false;
    }
    mysqli_free_result($this->_data);
    return true; }

```

```

public function countRows()
{
    return $this->_data !== null
        ? mysqli_num_rows($this->_data) : 0;
}

public function disconnect()
{
    if ($this->_connectto === null) {
        return false;
    }
    mysqli_close($this->_connectto);
    $this->_connectto = null;
    return true;
}

// close database
public function __destruct()
{
    $this->disconnect();
}
}

```

The interface for the database can include all the CRUD operations but SELECT was included for this example. The class SQL.php implements IDatabaseAdapter.php, which is the database interface, so it can establish a connection with the selected MySQL database and this enables the user to write the CRUD operations or queries that they wish.

The code above is for the persistence layer. For the ORM to be fully up and running, it requires mappers to interact with the database by manipulating the data with CRUD operations.

The mapper interface would look something like this:

IMapper.php

```

<?php

public interface IMapper{

function findById($id); }?>

```

This code here was implemented by following a tutorial on building an ORM by Alexandro Gervasio [28], it was implemented to gain an initial understanding of ORMs and how the data persistence layer appears and works.

Creating a simple REST API in PHP

This section is going to create a simple REST API by using the HTTP GET request [29]. Once the code is set up, the data of the dresses (in this example) can then be retrieved by entering the following URLs:

<http://localhost/assignmentone/RESTAPI/api.php?name=Skater>
<http://localhost/assignmentone/RESTAPI/api.php?name=Midi>
<http://localhost/assignmentone/RESTAPI/api.php?name=Bodycon>
<http://localhost/assignmentone/RESTAPI/api.php?name=Maxi>

These four URLs output the data in a JSON format and informs the user whether the product exists and the availability of stock.

The code is as it follows:

API.php:

```
<?php

header("Content-Type:application/json");
require "dressdata.php";

if(!empty($_GET['name']))
{
    $name=$_GET['name'];
    $stock = get_stock($name);

    if(empty($stock))
    {
        response(200,"Product Not Found",NULL);
    }
    else
    {
        response(200,"That product is available",$stock);
    }
}
else
{
    response(400,"Invalid Request",NULL);
}

function response($status,$status_message,$data)
{
    header("HTTP/1.1 ".$status);

    $response['status']=$status;
    $response['status']=$status_message;
    $response['Stock']=$data;
```

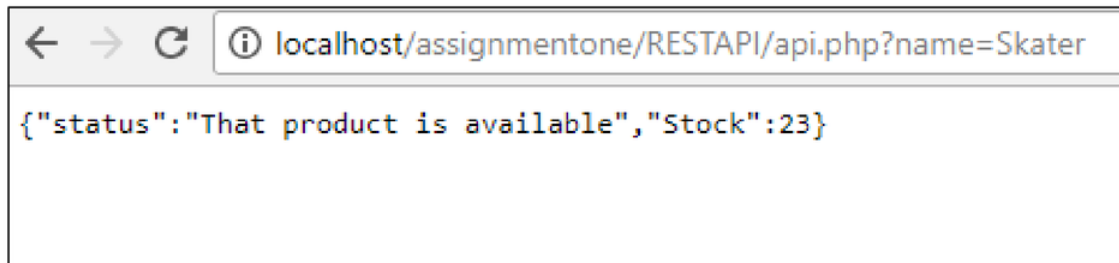
```
$json_response = json_encode($response);  
echo $json_response;  
}
```

Dressdata.php

```
<?php  
  
function get_stock($name)  
{  
    $dresses = [  
        "Skater"=>23,  
        "Bodycon"=>50,  
        "Midi"=>9,  
        "Maxi"=>10  
    ];  
  
    foreach($dresses as $dress=>$stock)  
    {  
        if($dress==$name)  
        {  
            return $stock;  
            break;  
        }  
    }  
}
```

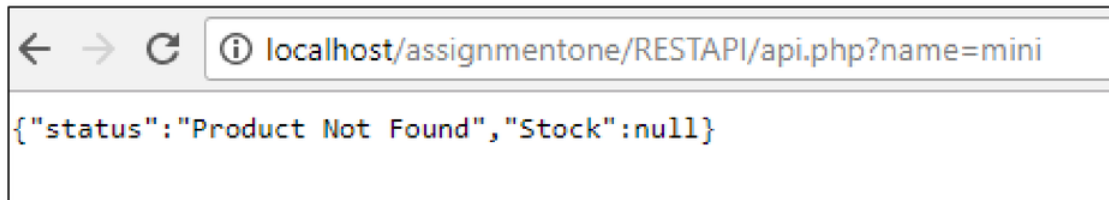
Testing with the URLs:

<http://localhost/assignmentone/RESTAPI/api.php?name=Skater>



Entering an invalid dress style at the end of the URL:

<http://localhost/assignmentone/RESTAPI/api.php?name=mini>



Expanding upon the code created for simple REST API

The code above fetches the data from a JSON object, in this step, a database has been set up in PHPmyAdmin called "Dresses" to try and retrieve the data from there instead. Below is the new PHP file called **dressDBdata.php** which uses a PDO to connect to the database:

```
<?php

function get_stock($name)    {

$dsn = 'mysql:host=localhost;dbname=assignmenttwo';
$dbUName = "root";
$dbPassword = "";
$dbh = new PDO($dsn, $dbUName, $dbPassword);
$stmt = $dbh->prepare('SELECT style, stock FROM dresses');

if($stmt->execute())

{
while($record = $stmt->fetch())

{
$dresses[] = $record;    }

foreach($dresses as $dress=>$stock) {

if($dress==$name) {
return $stock;
break; }    }

}
```

This file was connected to the file api_two.php (shown in appendix)

Testing fetching the data from the Database with the URL:

http://localhost/assignmentone/RESTAPI/api_two.php?name=Midi



It connects to the database and shows the product name and the numbers but has the error of the code retrieving all the data in the database. The database file should be reviewed to retrieve the data correctly.

Exploring the concept of “Convention over configuration”

This project was created following the concept of “coding by convention”, also known as “convention over configuration”. This term is essentially a design paradigm which is followed by developers who use technologies such as MVC ASP.NET or Ruby on rails, this design paradigm aims to limit the decisions made by the developer [30] [32].

Frameworks usually work with this concept, so the developer implement following conventions [31].

This project follows this concept as it is a MVC project [32] and follows a certain structure. For example, the Views of the application are stored in a folder called “Views” and all of the Views are named so that the user can identify that it is a view file before opening as they are named “__View.php” [33]. By naming the directory’s this, the convention has been followed.

All the controllers/commands in this project are named with the word “Command.php” at the end so the user again, knows that this is a controller file before opening it.

The naming conventions continue as the controllers and views are named so that they are recognised as relevant files, an example being “EmailCommand.php” which is linked to “EmailView.php” which is easy to tell by the name of the file. The names make sense, so the user can tell what the file contains, e.g. “StaffView” is a view that displays the staff database.

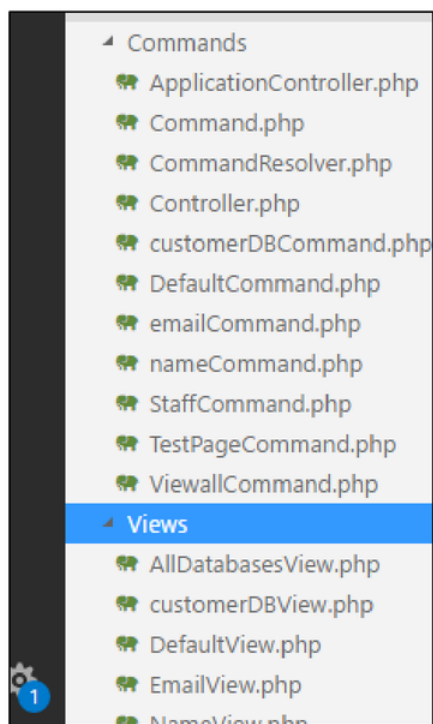


Figure 5- Screenshot showing structure of application

Creating Views in the application

This application demonstrates creating multiple views in an MVC application. The views are set in the application controller class (shown below). This class selects the Command classes and the according Views that are going to be displayed when the user selects the specific input. The buttons/links for other pages link to the command classes and then those command classes store the view name that the users are going to view.

The command classes direct the user to the View page by the code:

```
$response->SetViewName("EmailView");
```

ApplicationController.php

```
<?php

namespace Main\Control;

use \Main\Misc\Request;
use \Main\Misc\Response;
use \Main\View\View;
require_once 'Request.php';
require_once 'Response.php';
require_once 'Views/View.php';

class ApplicationController

{
    static private $_viewMap;
    static public function LoadViewMap()
    {
        self::$_viewMap = array(
            'DefaultView' => new View(".\Views\DefaultView.php"),
            'TestPage' => new View(".\Views\TestPageView.php"),
            'StaffPage' => new View(".\Views\StaffDBView.php"),
            'cusDBView' => new View(".\Views\customerDBView.php"),
            'DBEmailView' => new View(".\Views\EmailView.php"),
            'DBNameView' => new View(".\Views\NameView.php"),
            'Viewall' => new View(".\Views\AllDatabasesView.php")
        );
    }
    static public function GetView(Response $response)
    {
        return self::$_viewMap[$response->GetViewName()];
    }
}
```

This application has been implemented so that if there is an error with any of the Command classes or Views set up, they will direct back to the default View:

Default View:

```
<?php

namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
```

```

require_once 'Response.php';
$response = Registry::Instance()->Response;
?><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
<link rel="stylesheet" href="stylesheetpages1.css"/>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>Homepage</title>
</head><article>
<h1>Welcome to CoffeeHut.com</h1></article>
<body>
<a href="Index.php?cmd=TestPage">Page Two</a>
<a href="Index.php?cmd=staff">View staff Database</a>
<a href="Index.php?cmd=customerDB">View Customer database</a>
<a href="Index.php?cmd=Viewall">View All </a>
</body></html>

```

Not all views are linked to the default homepage, some Views are accessed through submit buttons on **DBView.php**:

```

<form action="Index.php?cmd=email" method="POST">
<input type="submit" value="View Customer Emails only" /></form>
<br><br>
<form action="Index.php?cmd=name" method="POST">
<input type="submit" value="View Customer names only" /></form> </body>

```

The views are accessed through unique URLs such as:

<http://localhost/assignmentone/Index.php?cmd=TestPage>
<http://localhost/assignmentone/Index.php?cmd=staff>

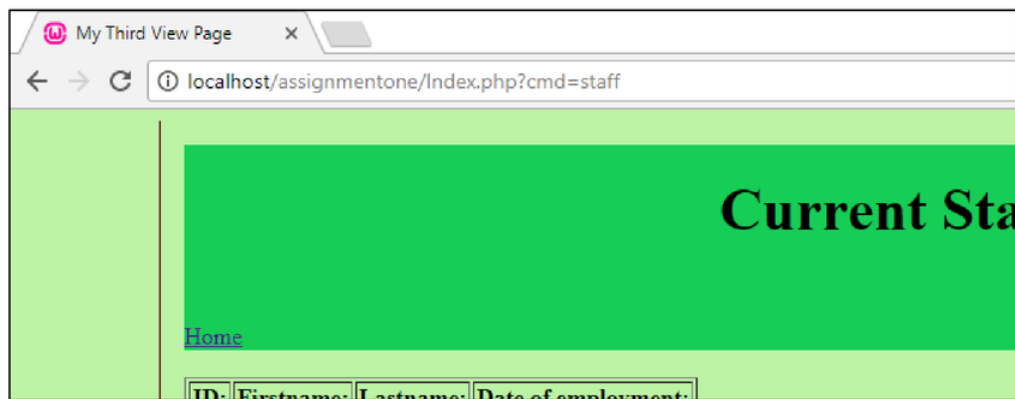


Figure 6

Figure 6 shows how the URL displays the selected View rather than connecting the pages through a HTML link.

Displaying a database into the views

The database connections and queries are written in the Command files of the classes:

```
class emailCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;
        $dsn = 'mysql:host=localhost;dbname=assignmenttwo';
        $dbUName = "root";
        $dbPassword = "";
        $dbh = new PDO($dsn, $dbUName, $dbPassword);
        $stmt = $dbh->prepare("SELECT email FROM customer");
        if($stmt->execute()) {
            while($record = $stmt->fetch())
            {
                $data[] = $record;
            }
        }

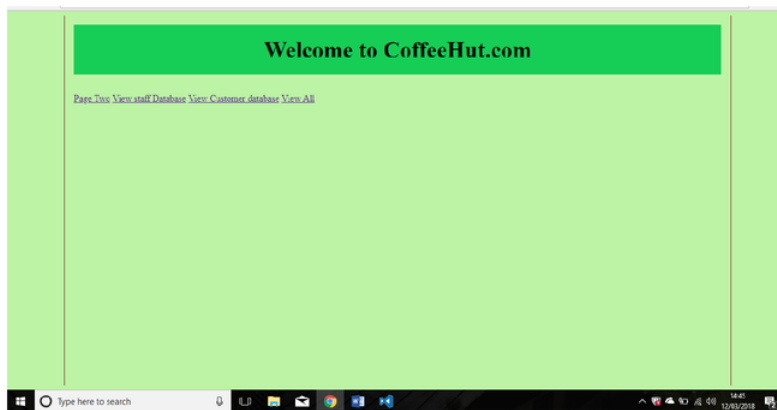
        $response = Registry::Instance()->Response;
        $response->SetViewName("DBEmailView");

        $response->IndexSetOut('data', $data);

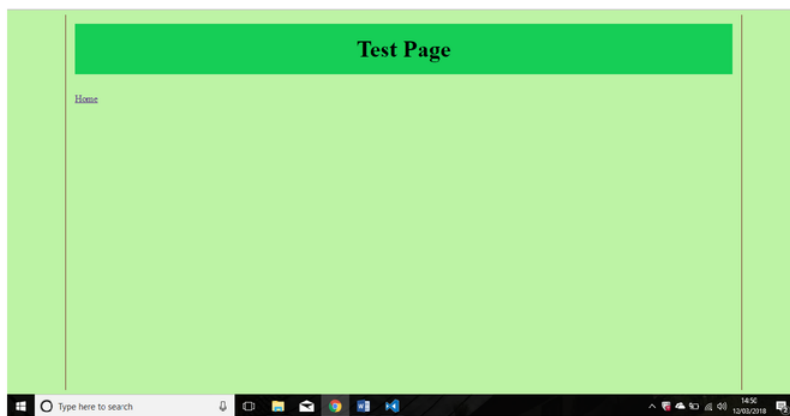
        return $response;    }
    }    ?>
```

The method IndexGetOut() is used in this case to contain the data that we are returning to the user, the array of data from the database has been stored into the value \$data when the query executes successfully. The code in the corresponding View retrieves that value by the following code:

```
$data = $response->IndexGetOut('data');
```



This is the default view page of the website, this does not display any database but sets up the links to the other views in the application.



Page two was essentially just a test page to ensure the Views worked.

Current Staff Database:			
Home			
ID:	Firstname:	Lastname:	Date of employment:
1	Narie	Smith	2018-03-11
2	Jack	Jones	2018-03-07
3	Iwan	Dyer	2018-03-01
4	Ashley	Morris	2017-12-26

This page displaying the current staff database is linked from the default page, this page just displays the staff database in PHPmyAdmin.

Current Customer Database:			
Home			
ID:	Firstname:	Lastname:	Email:
1	Leah	Cahill	leahcahill19 07@hotmail.com
2	Jade	Cahill	jadecahill@hotmail.com

View Customer Emails only

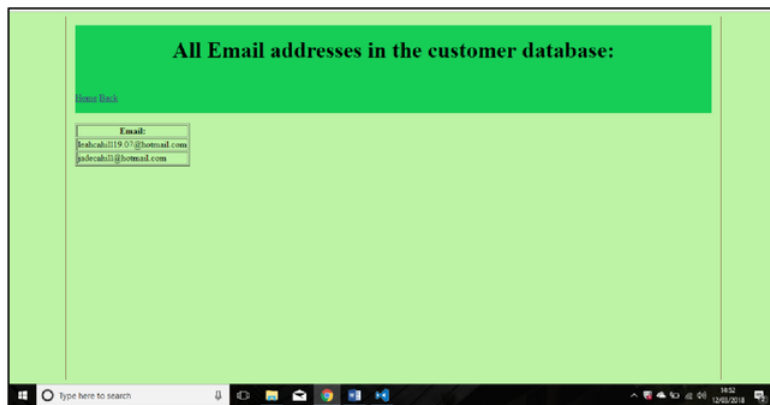
View Customer names only

This page is also linked from the default homepage. This time, it displays data from a different database and has two buttons which are named "View customer emails only" and "View customer names only" which directs the user two other Views.

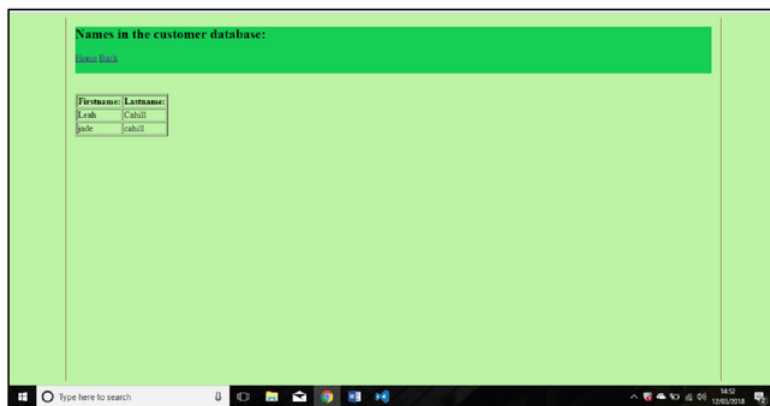
```

echo "</table>";
?><br><br><br><br>
<form action="Index.php?cmd=email" method="POST">
<input type="submit" value="View Customer Emails only" /></form>
<br><br>
<form action="Index.php?cmd=name" method="POST">
<input type="submit" value="View Customer names only" /></form>
</body>
</html>

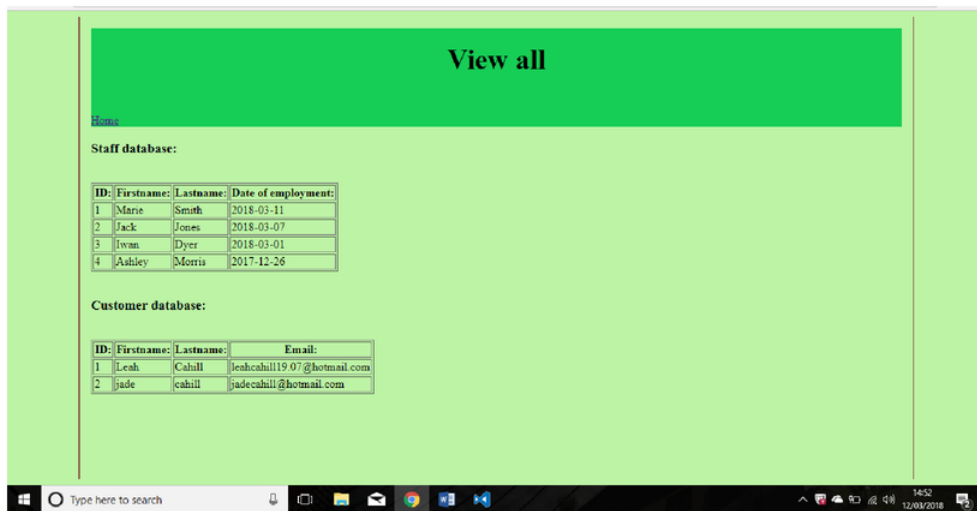
```



This page view stems from the button on the 'Customer database' page and cannot be accessed otherwise. The database displayed here is narrowed down to only displaying the customers email addresses.



This page is also linked with a button from the customer database page and cannot be accessed from the default page view. This view displays the names in the customer database. Both views include a link that allows the user to go back to the "Home" view.



This view displays both databases.

Ironically, this application was implemented following the “Convention over configuration” but the developer ended up configuring it and moving the commands to the main file, as the Views would not load when the Commands were in the Commands file.

References

- [1] "Front Controller Design Pattern," GeeksforGeeks, [Online]. Available: <https://www.geeksforgeeks.org/front-controller-design-pattern/>. [Accessed 8 March 2018].
- [2] "Design Pattern - Front Controller Pattern," Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/design_pattern/front_controller_pattern.htm. [Accessed 11 February 2018].
- [3] G. Sironi, "Practical PHP Patterns: Front Controller," DZone, 19 July 2010. [Online]. Available: <https://dzone.com/articles/practical-php-patterns/practical-php-patterns-front>. [Accessed 2 March 2018].
- [4] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2012.
- [5] "Don't Repeat Yourself," DevIQ, [Online]. Available: <http://deviq.com/don-t-repeat-yourself/>. [Accessed 05 March 2018].
- [6] "Core J2EE Patterns - Front Controller," Oracle, [Online]. Available: <http://www.oracle.com/technetwork/java/frontcontroller-135648.html>. [Accessed 8 March 2018].
- [7] "Front Controller," MSDN, [Online]. Available: <https://msdn.microsoft.com/en-us/library/ff648617.aspx>. [Accessed 05 March 2018].
- [8] M. Fowler, "Web Presentation Patterns," in *Patterns of Enterprise Application Architecture (P of EAA)*, Addison-Wesley, 2012, p. 333.
- [9] "Page Controller," MSDN, [Online]. Available: <https://msdn.microsoft.com/en-us/library/ff649595.aspx>. [Accessed 5 March 2018].
- [10] P. R. Pilaka, "Page Controller Vs Front Controller Pattern," Pratapreddypilaka, 03 September 2011. [Online]. Available: <http://pratapreddypilaka.blogspot.co.uk/2011/09/page-controller-vs-front-controller.html>. [Accessed 05 March 2018].
- [11] "Template Method Design Pattern," SourceMaking, [Online]. Available: https://sourcemaking.com/design_patterns/template_method. [Accessed 11 February 2018].
- [12] S. Jacobs, "When & Why to Use PHP Framework?," Template Monster, 2014. [Online]. Available: <https://www.templatemonster.com/blog/php-frameworks-when-and-why-to-use-them/>. [Accessed 8 March 2018].
- [13] Prava-MFS, "why to use Framework in PHP!!," CodeProject, 24 October 2014. [Online]. Available: <https://www.codeproject.com/Articles/826642/Why-to-use-Framework-in-PHP>. [Accessed 8 March 2018].
- [14] "The advantages and disadvantages of the Codeigniter framework," Downloadly, 29 December 2017. [Online]. Available: <https://downloadly.win/the-advantages-and-disadvantages-of-the-codeigniter-framework/>. [Accessed 05 March 2017].

- [15] T. Museth, "An introduction to CodeIgniter," Sitepoint, 6 April 2011. [Online]. Available: <https://www.sitepoint.com/an-introduction-to-codeigniter/>. [Accessed 2018 March 2018].
- [16] "CodeIgniter Advantages And Disadvantages," AngularMinds, 14 October 2015. [Online]. Available: <https://www.angularminds.com/blog/article/codeigniter-advantages-and-disadvantages.html>. [Accessed 08 March 2018].
- [17] A. Cogneau, "PhalconPHP: Yet Another PHP Framework?," Sitepoint, 29 July 2013. [Online]. Available: <https://www.sitepoint.com/phalconphp-yet-another-php-framework/>. [Accessed 05 March 2018].
- [18] M. Anand, "7 Best PHP Frameworks for Enterprise Application," TisIndia, 12 February 2014. [Online]. Available: <http://www.tisindia.com/blog/7-best-php-frameworks-2014/>. [Accessed 11 February 2018].
- [19] N. Technologies, "Benefits of cakephp web application development," linkedin, 24 June 2014. [Online]. Available: <https://www.linkedin.com/pulse/20140624060319-230230370-benefits-of-cakephp-web-application-development>. [Accessed 11 February 2018].
- [20] N. Chankov, "10 reasons to choose CakePHP as Framework," Nik.Chankov.net, 16 October 2007. [Online]. Available: <https://nik.chankov.net/2007/10/16/10-reasons-to-choose-cakephp-as-framework/>. [Accessed 11 February 2018].
- [21] J. Laine, "Should You Use a PHP Framework? Five Pros and Cons," EvatoTuts+, 11 July 2017. [Online]. Available: <https://code.tutsplus.com/tutorials/should-you-use-a-php-framework-five-pros-and-cons--cms-28905>. [Accessed 05 March 2018].
- [22] "11 Best PHP frameworks for modern web developers in 2018," Coders Eye, [Online]. Available: <https://coderseye.com/best-php-frameworks-for-web-developers/>. [Accessed 11 February 2018].
- [23] Agriya, "Pros and Cons of most Favoured PHP Frameworks," Agriya, 31 August 2015. [Online]. Available: <https://www.agriya.com/blog/pros-cons-favoured-php-frameworks/>. [Accessed 11 February 2018].
- [24] vikash.kumar, "Advantages of using PHP Frameworks," TeamGrowth, 19 April 2015. [Online]. Available: <http://blog.teamgrowth.net/index.php/programming/php/advantages-of-using-php-frameworks>. [Accessed 05 March 2018].
- [25] C. Hopkins, "The MVC Pattern and PHP, Part 1," SitePoint, 04 March 2013. [Online]. Available: <https://www.sitepoint.com/the-mvc-pattern-and-php-1/>. [Accessed 10 March 2018].
- [26] C. Hopkins, "The MVC Pattern and PHP, Part 2," SitePoint, 11 March 2013. [Online]. Available: <https://www.sitepoint.com/the-mvc-pattern-and-php-2/>. [Accessed 10 March 2018].
- [27] R. Knight, "Object Relational Mapping in PHP," Slideshare.net, 05 May 2009. [Online]. Available: https://www.slideshare.net/rob_knight/object-relational-mapping-in-php. [Accessed 10 March 2018].

- [28] A. Gervasio, "Building an ORM in PHP," DevShed, 18 November 2011. [Online]. Available: <http://www.devshed.com/c/a/mysql/building-an-orm-in-php/>. [Accessed 11 March 2018].
- [29] G. James, "Creating a simple REST API in PHP," SahreUrCodes, 21 April 2017. [Online]. Available: <https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php>. [Accessed 11 March 2018].
- [30] vladko, "Convention over Configuration – what's the big deal?," Zen of Coding, 29 January 2009. [Online]. Available: <https://www.zenofcoding.com/2009/01/29/convention-over-configuration-whats-the-big-deal/>. [Accessed 15 March 2018].
- [31] F. Sáez, "Convention Over Configuration," FacileThings, [Online]. Available: <https://facilethings.com/blog/en/convention-over-configuration>. [Accessed 15 March 2018].
- [32] "Naming conventions in ASP.NET MVC in ASP.NET MVC," Techfunda, [Online]. Available: <http://techfunda.com/howto/19/naming-conventions-in-asp-net-mvc>. [Accessed 15 March 2018].
- [33] V. V. Agarwal, "Software Design Paradigm - Convention Over Configuration (ASP.NET MVC)," C# Corner, 13 January 2017. [Online]. Available: <https://www.c-sharpcorner.com/article/software-design-paradigm-convention-over-configuration/>. [Accessed 15 March 2018].

Appendix

Code for basic MVC application

Controller.php

```
<?php
class Controller
{
    private $model;
    public function __construct($model) {
        $this->model = $model;
    }
}
```

Model.php

```
<?php

class Model

{
    public $string;
    public $tempstring;
    public function __construct(){
        $this->tempstring = "templatetext";
        $this->mytemplate = "template.php";
        $this->tempstring2 = "templatetext2";
        $this->mytemplate2 = "template2.php";
        $this->string = "Title of page";
    }
    }?>
```

View.php

```
<?php
class View
{
    private $model;
    private $controller;
    public function __construct($controller,$model) {
        $this->controller = $controller;
        $this->model = $model; }

    public function template1(){
        $intro = "<p>" . $this->model->tempstring . "</p>";
        require_once($this->model->mytemplate); }

    public function template2(){
        $page2 = "<p>" . $this->model->tempstring2 . "</p>";
        require_once($this->model->mytemplate2);
    }
}
```

```

public function DisplayTable()
{
    $dsn = 'mysql:host=localhost;dbname=assignmenttwo';
    $dbUName = "root";
    $dbPassword = "";
    $dbh = new PDO($dsn, $dbUName, $dbPassword);

    foreach($dbh->query('SELECT * FROM customer') as $record){
        echo "<tr>" .
            "<td>" . $record["firstname"] . "</td>" .
            "<td>" . $record["lastname"] . "</td>" .
            "<td>" . $record["Email"] . "</td>" .
            "</tr>";
    }

    }
    public function output(){
        return "<p>" . $this->model->string . "</p>";

    }
}

```

Template.php

```

<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';

$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
echo $view->template1();
?>
<!DOCTYPE html>
<html>
<head> <meta charset="charset=utf-8">
<title></title> </head>
<h1> Template Title Page </h1>
<body>
<h1><?php echo $intro; ?></h1>
</body>
</html>

```

Template2.php

```
<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';
$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
echo $view->template2();
?>
<!DOCTYPE html><html>
<head> <meta charset="charset=utf-8">
<title></title>
</head>
<h1> Template Title Two </h1> <body>
<h1><?php echo $page2; ?></h1>
</body>
</html>
```

Index.php

```
<?php
require_once 'controller.php';
require_once 'view.php';
require_once 'model.php';

$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);

?><html><head>

<h2><?php echo $view->output(); ?></h2>      <h4>Customer Details</h4>
</head>
<body><table border=1>
<thead><tr><th>First name</th><th>Last name</th><th>Email</th></tr></thead>
<?php echo $view->DisplayTable();?>

</table>

<?php
echo $view->template1();
echo $view->template2();

?>
</html>
```

[Code for exploring the concept of creating an ORM](#)

IdatabaseAdapter.php

```
<?php
namespace Database;
interface DatabaseAdapterInterface
{
    function connectToDB();
    function disconnect();
    function query($query);
    function fetch();
    function select($table);
    function countRows();
    function freeData();
}
```

SQL.php

```
<?php

namespace Database;
class SQL implements IdatabaseAdapter
{
    protected $_configuration = array();
    protected $_connectto;
    protected $_data;

    public function __construct(array $configuration)
    {
        if (count($configuration) !== 4) {
            throw new InvalidArgumentException("Invalid");
        }

        $this->_configuration = $configuration;
    }

    public function connectToDB()
    {
        // Establishing the connection once
        if ($this->_connectto === null) {
            list($host, $user, $password, $database) = $this->_configuration;
```

```

        if (!$this->_connectto = @mysqli_connect("localhost", "root", "", "assignmenttwo")) {
            throw new RuntimeException('Error connecting to the server : ' . mysqli_connect_error());
        }

        unset($host, $user, $password, $database);
    }
    return $this->_connectto;
}
public function query($query)
{
    if (!is_string($query) || empty($query)) {
        throw new InvalidArgumentException('invalid query');
    }
    $this->connectToDB();
    if (!$this->_data = mysqli_query($this->_connectto, $query)) {
        throw new RuntimeException('Error'. $query . mysqli_error($this->_connectto));
    }
    return $this->_data;
}
// Select Statement
public function select($table)
{
    $query = 'SELECT * FROM' $table;
    $this->query($query);
    return $this->countRows();
}
public function fetch()
{
    if ($this->_data !== null) {
        if (($row = mysqli_fetch_array($this->_result, MYSQLI_ASSOC)) === false) {
            $this->freeData(); }
        return $row; }
    return false; }
public function freeData()
{
    if ($this->_result === null) {
        return false;
    }
}

```

```

    }
    mysqli_free_result($this->_data);
    return true;
}
public function countRows()
{
    return $this->_data !== null
    ? mysqli_num_rows($this->_data) : 0;
}

public function disconnect()
{
    if ($this->_connectto === null) {
        return false;
    }
    mysqli_close($this->_connectto);
    $this->_connectto = null;
    return true; }
// close database
public function __destruct()
{
    $this->disconnect(); }
}

```

Imapper.php

```
<?php
```

```

public interface IMapper{
function findByld($id);
}
?>

```

Code for creating a REST API

Api.php

```
<?php

header("Content-Type:application/json");
require "dressdata.php";

if(!empty($_GET['name']))
{
    $name=$_GET['name'];
    $stock = get_stock($name);
    if(empty($stock))
    {
        response(200,"Product Not Found",NULL);
    }
else
    {
        response(200,"That product is available",$stock);
    }
}

Else {
    response(400,"Invalid Request",NULL);
}

function response($status,$status_message,$data)
{
    header("HTTP/1.1 ".$status);
    $response['status']=$status;
    $response['status']=$status_message;
    $response['Stock']=$data;
    $json_response = json_encode($response);
    echo $json_response;
}
```

Dressdata.php

```
<?php

function get_stock($name)
{
$dresses = [
"Skater"=>23,
"Bodycon"=>50,
"Midi"=>9,
"Maxi"=>10

];

foreach($dresses as $dress=>$stock)
{
if($dress==$name){
return $stock;
break; } } }
```

DressDBData.php

```
<?php

function get_stock($name){
$dsn = 'mysql:host=localhost;dbname=assignmenttwo';
$dbUName = "root";
$dbPassword = "";
$dbh = new PDO($dsn, $dbUName, $dbPassword);
$stmt = $dbh->prepare('SELECT style, stock FROM dresses');

if($stmt->execute())
{
while($record = $stmt->fetch())
{
$dresses[] = $record; } }

foreach($dresses as $dress=>$stock) {
if($dress==$name) {
return $stock;
break;}
}} }
```

api_two.php

```
<?php

header("Content-Type:application/json");

require "dressDBdata.php";

if(!empty($_GET['name']))
{
    $name=$_GET['name'];
    $stock = get_stock($name);

    if(empty($stock))
    {
        response(200,"Product Not Found",NULL);
    }
    Else {
        response(200,"That product is available",$stock);
    } }
else
{
    response(400,"Invalid Request",NULL);
}
function response($status,$status_message,$data)
{
    header("HTTP/1.1 ".$status);
    $response['status']=$status;
    $response['status']=$status_message;
    $response['Stock']=$data;
    $json_response = json_encode($response);
    echo $json_response;
}
```

Full code for the MVC application

Commands

ApplicationController.php

```
<?php

namespace Main\Control;
use \Main\Misc\Request;
use \Main\Misc\Response;
use \Main\View\View;
require_once 'Request.php';
require_once 'Response.php';
require_once 'Views/View.php';

class ApplicationController

{
    static private $_viewMap;
    static public function LoadViewMap()
    {
        self::$_viewMap = array(
            'DefaultView' => new View(".\Views\DefaultView.php"),
            'TestPage' => new View(".\Views\TestPageView.php"),
            'StaffPage' => new View(".\Views\StaffDBView.php"),
            'cusDBView' => new View(".\Views\customerDBView.php"),
            'DBEmailView' => new View(".\Views\EmailView.php"),
            'DBNameView' => new View(".\Views\NameView.php"),
            'Viewall' => new View(".\Views\AllDatabasesView.php")

            );
    }

    static public function GetView(Response $response)
    {
        return self::$_viewMap[$response->GetViewName()];
    }
}
}??>
```

Command.php

```
<?php

namespace Main\Command;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Request.php';
```

```

require_once 'Response.php';
abstract class Command
{
    public final function __construct()
    {
    }
    public abstract function Execute(Request $request);
}?:>

```

CommandResolver.php

```

<?php

namespace Main\Command;
use Main\Misc\Registry;
use \ReflectionClass;
use \Main\Misc\Request;
use \Main\Misc\Response;
use \Main\Command\DefaultCommand;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';
require_once 'DefaultCommand.php';

class CommandResolver
{
    static private $_baseCommandClass;
    static private $_defaultCommand;

    public function __construct()
    {
        if(isset(self::$_baseCommandClass) == false)
        {
            self::$_baseCommandClass = new ReflectionClass("\Main\Command\Command");
            self::$_defaultCommand = new DefaultCommand();
        }
    }

    public function GetCommand(Request $request)
    {
        $cmd = $request->IndexGetProperty('cmd');
        $sep = DIRECTORY_SEPARATOR;
        if(isset($cmd) == false)
        {

```

```

        return clone self::$_defaultCommand;
    }

    $cmd = str_replace(array('.', $sep), "", $cmd);
    $filepath = ".\\{$cmd}Command.php";
    $classname = "\\Main\\Command\\{$cmd}Command";
    if(file_exists($filepath))
    {
        @require_once $filepath;

        if(class_exists($classname))
        {
            $reflectionClass = new ReflectionClass($classname);

            if($reflectionClass->isSubclassOf(self::$_baseCommandClass))
            {
                return $reflectionClass->newInstance();
            }
            else
            {
                $response = Registry::Instance()->Response;
                $response->IndexSetOut('Error', "Command
                {$cmd}Command is not a command");
                return clone self::$_defaultCommand;
            }
        }
    }

    $response = Registry::Instance()->Response;
    $response->IndexSetOut('Error', "Command {$cmd}Command not found");
    return clone self::$_defaultCommand;    }    }
?>

```

TestPageCommand.php

```

<?php

namespace Main\Command;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;

```

```

require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';
class TestPageCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;
        $response->SetViewName("TestPage");
        $response->IndexSetOut('cmd', $request->IndexGetProperty('cmd'));
        return $response;
    }
}
?>

```

ViewAllCommand.php

```

<?php
namespace Main\Command;

use \PDO;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class ViewallCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;
        $dsn = 'mysql:host=localhost;dbname=assignmenttwo';
        $dbUName = "root";
        $dbPassword = "";
        $dbh = new PDO($dsn, $dbUName, $dbPassword);
        $stmt = $dbh->prepare("SELECT * FROM staff");
        if($stmt->execute())
        {
            while($record = $stmt->fetch())

```

```

        {
            $data[] = $record;
        }
    }

    $stmt2 = $dbh->prepare("SELECT * FROM customer");

    if($stmt2->execute())
    {
        while($record2 = $stmt2->fetch())
        {
            $data2[] = $record2;
        }
    }

    $response->SetViewName("Viewall");
    $response->IndexSetOut('data', $data);
    $response->IndexSetOut('data2', $data2);
    return $response;
}}?>

```

customerDBCommand.php

```

<?php

namespace Main\Command;
use \PDO;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class customerDBCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;
        $dsn = 'mysql:host=localhost;dbname=assignmenttwo';
        $dbUName = "root";
    }
}

```

```

$dbPassword = "";
$dbh = new PDO($dsn, $dbUsername, $dbPassword);
$stmt = $dbh->prepare("SELECT * FROM customer");
if($stmt->execute())
{
    while($record = $stmt->fetch())
    {
        $data[] = $record;
    }
}

$response->SetViewName("cusDBView");
$response->IndexSetOut('data', $data);
return $response;
}
}

?>

```

Default Command.php

```

<?php

namespace Main\Command;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class DefaultCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;
        $response->SetViewName("DefaultView");
        $response->IndexSetOut('cmd', $request->IndexGetProperty('cmd'));
        return $response;
    }
}

?>

```

Emailcommand.php

```
<?php

namespace Main\Command;
use \PDO;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class emailCommand extends Command
{
    public function Execute(Request $request)
    {
        $response = Registry::Instance()->Response;

        $dsn = 'mysql:host=localhost;dbname=assignmenttwo';
        $dbuname = "root";
        $dbpassword = "";

        $dbh = new PDO($dsn, $dbuname, $dbpassword);
        $stmt = $dbh->prepare("SELECT email FROM customer");

        if($stmt->execute())
        {
            while($record = $stmt->fetch())
            {
                $data[] = $record;
            }
        }
        $response = Registry::Instance()->Response;
        $response->SetViewName("DBEmailView");
        $response->IndexSetOut('data', $data);
        return $response;
    }
}
```

```
}
```

```
?>
```

Index.php

```
<?php
use \Main\Misc\Registry;
use \Main\Control\Controller;
use \Main\Misc\Request;
require_once 'Registry.php';
require_once 'Request.php';
require_once 'Controller.php';
```

```
$controller = Controller::Instance();
$request = Registry::Instance()->Request;
$controller->HandleRequest($request);
```

```
?>
```

nameCommand.php

```
<?php

namespace Main\Command;
use \PDO;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class nameCommand extends Command
{
    public function Execute(Request $request)
    {
```

```
$response = Registry::Instance()->Response;
```

```
$dsn = 'mysql:host=localhost;dbname=assignmenttwo';
$dbUName = "root";
$dbPassword = "";
$dbh = new PDO($dsn, $dbUName, $dbPassword);
$stmt = $dbh->prepare("SELECT firstname, lastname FROM customer");
if($stmt->execute())
```

```

        {
            while($record = $stmt->fetch())
            {
                $data[] = $record;
            }
        }

        $response = Registry::Instance()->Response;
        $response->SetViewName("DBNameView");
        $response->IndexSetOut('data', $data);
        return $response;
    }
}

?>

```

Registry.php

```

<?php

namespace Main\Misc;
use \Main\ObjectProperties;
require_once 'ObjectProperties.php';
require_once 'Request.php';
require_once 'Response.php';
class Registry extends ObjectProperties
{
    private static $_instance;
    private $_request;
    private $_response;
    private $_factories = array();

    private function __construct()
    {
        // Do nothing
    }

    static public function Instance() // Static so we use classname itself to create object
    i.e. Registry::Instance()
    {
        if(!isset(self::$_instance))
        {

```

```

self::$_instance = new self();
self::$_instance->_request = new Request();
self::$_instance->_response = new Response();
}
return self::$_instance;

    }

// Get method
// Return the Request object
public function GetRequest()
{
    Return $this->_request; }
public function SetRequest(Request $value)    {
    $this->_request = $value;
}
public function GetResponse()
{
    return $this->_response;
}

public function SetResponse(Response $value)
{
    $this->_response = $value;
}

public function IndexGetFactory($key)
{
    if(isset($this->_factories[$key]))
    {
        return $this->_factories[$key];
    }
    return;
}

public function IndexSetFactory($key, $value)
{
    $this->_factories[$key] = $value;
}
}
?>

```

Request.php

```
<?php

namespace Main\Misc;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';

class Request
{
    private $_properties;
    public function __construct()
    {
        $this->Init();
    }

    private function Init()
    {
        {
            if( isset($_SERVER['REQUEST_METHOD'])
            && ($_SERVER['REQUEST_METHOD'] == "POST"
            || $_SERVER['REQUEST_METHOD'] == "GET") )
            {
                $this->_properties = $_REQUEST;
            }
            else
            {
                $this->_properties = array();
            }
        }

        public function IndexGetProperty($key)
        {
            {
                if(isset($this->_properties[$key]))
                {
                    return $this->_properties[$key];
                }
            }
            return;

        public function IndexSetProperty($key, $value)
```

```

{
$this->_properties[$key] = $value;    }
public function Copy(Response $response)
{
foreach($this->_properties as $key => $value)
{
$response->IndexSetOut($key, $value);
}
}
}
?>

```

Response.php

```

<?php
namespace Main\Misc;
use \Main\Misc\Registry;
require_once 'Registry.php';

class Response
{
private $_output = array();
public function __construct()
{
}

public function IndexGetOut($key){
if(isset($this->_output[$key]))
{
return $this->_output[$key];
}
return;
}

public function IndexSetOut($key, $value)
{
$this->_output[$key] = $value;
}

public function GetViewName()
{
if(isset($this->_output['view']))

```

```

{
return $this->_output['view'];      }
return false;

}
public function SetViewName($viewName)
{
if(is_string($viewName))
{
$this->_output['view'] = $viewName;   }

}

}
?>

```

StaffCommand.php

```

<?php

namespace Main\Command;
use \PDO;
use Main\Command\Command;
use \Main\Misc\Registry;
use \Main\Misc\Request;
use \Main\Misc\Response;
require_once 'Command.php';
require_once 'Request.php';
require_once 'Response.php';

class StaffCommand extends Command
{
public function Execute(Request $request)
{
$response = Registry::Instance()->Response;
$dns = 'mysql:host=localhost;dbname=assignmenttwo';
$dbUName = "root";
$dbPassword = "";
$dbh = new PDO($dns, $dbUName, $dbPassword);

$stmt = $dbh->prepare("SELECT * FROM staff");
if($stmt->execute())
{
while($record = $stmt->fetch())

```

```

        {
            $data[] = $record;
        }
    }

    $response->SetViewName("StaffPage");
    $response->IndexSetOut('data', $data);
    return $response;
}

?>

```

ObjectProperties.php

```

<?php

namespace Main;
class ObjectProperties

{
    public function __get($property)
    {

        $method = "Get{$property}";
        if(method_exists($this, $method))
        {
            return $this->$method();
        }

        return;
    }

    public function __set($property, $value)
    {
        $method = "Set{$property}";
        if(method_exists($this, $method))
        {

            $this->$method($value);
        }

        return;
    }
}

?>

```

Stylesheetpages1.css

```
div.container {
width:100%;
border: 1px solid gray;
}

h1 {
color:black;
text-align: center;
padding: 20px;
font-size: 40px;
}

body {
margin-left: 100px;
margin-right: 100px;
border-left: 1px solid #701C1C;
border-right: 1px solid #701C1C;
padding: 1em;
overflow: hidden;
color:black;
background-color: #BDF3A5;
}

article {
background-color: #16CE56;
align-content: center;
}
```

Views.

View.php

```
<?php

namespace Main\View;
class View

{
private $_filepath;
public function __construct($filepath)
```

TestPageView.php

```
<?php
namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';
$response = Registry::Instance()->Response;

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>My Second View Page</title>
<link rel="stylesheet" href="stylesheetpages1.css"/>
<article>
<h1> Test Page </h1>

</article>

</head>
<body>
<a href="Index.php">Home</a>
</body></html>
```

staffDBView.php

```
<?php
namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';

$response = Registry::Instance()->Response;

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
<link rel="stylesheet" href="stylesheetpages1.css"/>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>My Third View Page</title>

<article>

<h1>Current Staff Database:</h1>
<a href="Index.php">Home</a><br></article>

<?php

    $data = $response->IndexGetOut('data');

    echo "<table border='1'>";

        echo "<thead><tr><th>ID:</th><th>Firstname:</th><th>Lastname:</th><th>Date of
employment:</th></tr></thead></br>";

        foreach($data as $record)

        {
            echo "<tr>";
            echo "<td>" . $record["staffID"] . "</td>";
            echo "<td>" . $record["firstname"] . "</td>";
            echo "<td>" . $record["lastname"] . "</td>";
            echo "<td>" . $record["dateofemployment"] . "</td>";
            echo "</tr>";
        }
        echo "</table>";
        ?><Br><br><br><br>
</body></html>
```

NameView.php

```
<?php

namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';
$response = Registry::Instance()->Response;
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html><head>
<link rel="stylesheet" href="stylesheetpages1.css"/>

<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">

<title></title>

</head>

<article>

<h2>Names in the customer database:</h2>

<a href="Index.php">Home</a>

<a href="Index.php?cmd=ShowDB">Back</a><br><br></article>

    <thead><tr></tr></thead></br>

<?php

    $data = $response->IndexGetOut('data');

    echo "<table border='1'>";
    echo "<thead><tr><th>Firstname:</th><th>Lastname:</th></tr></thead></br>";
    foreach($data as $record)
    {
        echo "<tr>";
        echo "<td>" . $record["firstname"] . "</th></td>";
        echo "<td>" . $record["lastname"] . "</td>";
        echo "</tr>";
    }
    echo "</table>";
```

```
?>
</body></html>
```

EmailView.php

```
<?php
namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';
$response = Registry::Instance()->Response;

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html><head>

    <link rel="stylesheet" href="stylesheetpages1.css"/>

<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">

<title></title>

</head><article>

<h1>All Email addresses in the customer database:</h1>

<a href="Index.php">Home</a>

<a href="Index.php?cmd=ShowDB">Back</a><br><br></article>

<?php

    $data = $response->IndexGetOut('data');

    echo "<table border='1'>";

    echo "<thead><tr><th>Email:</th></tr></thead></br>";
    foreach($data as $record)
    {
        echo "<tr>";
        echo "<td>" . $record["email"] . "</td>";
        echo "</tr>";
    }
    echo "</table>";
```

```
?>
</body></html>
```

DefaultView.php

```
<?php
namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';

$response = Registry::Instance()->Response;

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
<link rel="stylesheet" href="stylesheetpages1.css"/>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>Homepage</title>
</head>
<article>
<h1>Welcome to CoffeeHut.com</h1></article>
<body>
<a href="Index.php?cmd=TestPage">Page Two</a>
<a href="Index.php?cmd=staff">View staff Database</a>
<a href="Index.php?cmd=customerDB">View Customer database</a>
<a href="Index.php?cmd=Viewall">View All </a>
</body></html>
```

customerDBView.php

```
<?php
namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
```

```

require_once 'Response.php';
$response = Registry::Instance()->Response;

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title></title>
<link rel="stylesheet" href="stylesheetpages1.css"/>
</head> <body><article>
<h1>Current Customer Database:</h1>
<a href="Index.php">Home</a><br> </article>
<?php
$data = $response->IndexGetOut('data');

echo "<table border='1'>";

        echo
"<thead><tr><th>ID:</th><th>Firstname:</th><th>Lastname:</th><th>Email:</th></tr></thead></br>";

        foreach($data as $record)
        {
            echo "<tr>";

                echo "<td>" . $record["cusld"] . "</td>";

                echo "<td>" . $record["firstname"] . "</td>";
                echo "<td>" . $record["lastname"] . "</td>";
                echo "<td>" . $record["Email"] . "</td>";
            echo "</tr>";

        }

        echo "</table>";

        ?><Br><br><br><br>

<form action="Index.php?cmd=email" method="POST">
<input type="submit" value="View Customer Emails only" /></form>

<br><Br>

<form action="Index.php?cmd=name" method="POST">
<input type="submit" value="View Customer names only" /></form>

</body></html>

```

AllDatabaseView.php

```
<?php

namespace Main\View;
use \Main\Misc\Registry;
use \Main\Misc\Response;
require_once 'Registry.php';
require_once 'Response.php';

$response = Registry::Instance()->Response;

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html><head>
<link rel="stylesheet" href="stylesheetpages1.css"/>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>View All</title>
<article>

<h1>View all</h1>
<a href="Index.php">Home</a><br></article>

<h3>Staff database:</h3>
<?php

$data = $response->IndexGetOut('data');
echo "<table border='1'>";
echo "<thead><tr><th>ID:</th><th>Firstname:</th><th>Lastname:</th><th>Date of
employment:</th></tr></thead></br>";
foreach($data as $record)
{
echo "<tr>";
echo "<td>" . $record["staffID"] . "</td>";
echo "<td>" . $record["firstname"] . "</td>";
echo "<td>" . $record["lastname"] . "</td>";
echo "<td>" . $record["dateofemployment"] . "</td>";
echo "</tr>";
}

echo "</table>";

?><br>
```

```

        <h3>Customer database:</h3>

    <?php

        $data2 = $response->IndexGetOut('data2');

        echo "<table border='1'>";

            echo
            "<thead><tr><th>ID:</th><th>Firstname:</th><th>Lastname:</th><th>Email:</th></tr></thead></br>";

            foreach($data2 as $record2)
            {
                echo "<tr>";
                echo "<td>" . $record2["cusId"] . "</td>";
                echo "<td>" . $record2["firstname"] . "</td>";
                echo "<td>" . $record2["lastname"] . "</td>";
                echo "<td>" . $record2["Email"] . "</td>";

                echo "</tr>";
            }

            echo "</table>";

            ?><Br>

    </body>
</html>

```

Assignment2LeahCahill

GRADEMARK REPORT

FINAL GRADE

67 / 100

GENERAL COMMENTS

Instructor

Investigation of Fowler (30) 18

Investigation of MVC frameworks (20) 16

Quality of website and techniques used (25) 16

View handling (10) 6

Database handling (15) 11

Discussion of Fowler lacked a bit of detail but I like the DRY and SRP approach that you took to the discussion. The Template View explanation is confusing because you've mixed up two different concepts which have the same name.

When comparing frameworks it would have been useful to identify a set of criteria by which they could be compared.

Used the framework provided in class and demonstrated a number of controllers and views. Would like to have seen some CRUD operations. REST example provided and created your own DB class.

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

Good point

PAGE 5



Comment 2

You've mixed up two different concepts because they have the same name

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61
