CODEIGNITER – PART TWO - CMS

This lab will build upon concepts covered in the introductory lab on CodeIgniter in order to build a Content Management System (CMS). However, it is recommended that you build a new CodeIgniter project rather than adapt that from the previous lab.

SETTING UP THE FILE STRUCTURE CODEIGNITER

Unzip the files for the lab and download a copy of CodeIgniter into a folder called ***CI-CMS*** on your *f:/public_html*.

This lab will start with a number of views and controllers and a model already set up.

Add the files in *starter_views*, *starter_controllers* and *start_models* to the appropriate related folders in the *application* folder.

SETTING UP THE CONFIGURATION

Open the file *application/config/autoload.php* and ensure the following helpers are loaded:

```
 $autoload['helper'] = array('url', 'form');
```

Open the *application/config/database.php* file and change the database setting to match your MySQL account.

```
$db['default'] = array(
      'dsn' => '',
      'hostname' => 'localhost',
      'username' => '<YOURUSERNAME>',
      'password' => '<YOURPASSWORD>',
      'database' => '<YOURDATABASE>',
      'dbdriver' => 'mysqli',
…
```

Create a *.htaccess* file and place it into your *f:/public_html/CI-CMS* folder.

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond $1 !^(index\.php|images|robots\.txt|css)
RewriteRule ^(.*)$ index.php/$1 [L]
</IfModule>
```

The base URL value `$config['base_url']` set in file *application/config/config.php* can now be set to exclude the *index.php* part of a codeIgniter URI ie:

```
$config['base_url'] = 'https://homepages.shu.ac.uk/~<YOURSTUDENTNUMBER>/CI-
CMS;
```

A MORE SECURE FILE STRUCTURE

WARNING: THIS IS FOR INFORMATION
DO NOT TRY THIS ON HOMEPAGES.SHU.AC.UK AS IT WILL NOT WORK.

Given that connection details are stored in the configuration files a securer file structure is one where the *application* and *system* folders are above the root of the server.

On most Apache servers the main webserver root is *htdocs*. This is the folder which is visible to visitors to your website. As such a securer file structure would be:

```
├──── htdocs
|     ├──── index.php
|     ├──── css
|     ├──── images
|     ├──── js
├──── application
├──── system
```

In order for this structure to work the *index.php* needs to be edited so that the `$system_path` and `$application_folder` now match the revised file structure.

```
….
$system_path = '../system';
….
$application_folder = '../application';
…
```

If you are running CodeIgniter on your own server either hosted or the likes of XAMPP you could attempt to secure the folder structure as discussed. However, as stated above unfortunately this technique will not work on homepages.shu.ac.uk because of the nature of the drive mapping.

BOOTSTRAP

The sample files make use of the Bootstrap CSS framework. View your pages via *homepages.shu.ac.uk* to ensure that the home page appears appropriated styled. Your URL will be:

```
https://homepage.shu.ac.uk/~<YOUR-STUDENT-NUMBER/CI-CMS/All_films
```

Notice that in the view *views/header.php* is built to load all the necessary CSS and Javascript required by Bootstrap. There is also a view *views/footer.php* file to allow flexibility if we chose to add extra scripts.

CREATING A SEARCH BAR

In the *views/header.php* there is a form with an action of:

```
<?php echo base_url('S_bar');?>
```

This will call a Controller *controller/S_bar*. As we saw in the previous lab CodeIgniter uses URI with the following construct:

```
http://example.com/[controller-class]/[controller-method]/[arguments]
```

Ideally we'd like to use these SEO friendly URIs in our search. Therefore we'll use *S_bar* to redirect to another controller using the default *index()* method.

```
      public function index() {
            $searchTerm = $this->input->post('searchTerm');
            redirect('Search_bar/find_film/'.$searchTerm);
      }
```

The above will simply redirect to a second Controller called *controller/Search_bar.php*. Specifically it is redirected at the method `find_film()` and passes a parameter as the search term. As such there is a method of `find_film()` in *Search_bar.php*.

The `find_film()` method loads our model from *models/Movie.php* and calls the `search_films()` method.

Open *models/Movie.php* and create a `search_films()` method.

```
public function search_films($filmName){
            $this->db->like('filmName',$filmName);
            // order them
            $this->db->order_by('filmName');
            $query = $this->db->get('movies');
            return $query->result_array();
      }
```

Follow the trail of documents back through the controller to the view *view/search_bar.php*.

This now uses the data to create a loop of the matching films.

DETAILS PAGE

We want this list to link through to a details page. In the view *view/search_bar.php* amend the output of the loop so that the results are linked to a details page.

```
echo "<li><a href=\"";
echo base_url('Film_details/showFilm/'.$film['filmID']);
echo "\">{$film['filmName']}</a>";
echo "</li>";
```

We now need to build the logic around the controller `Film_details` and its method `showFilm()` to extract data specific to that film from the database.

To do this we'll need the same sequence of logic used in the original search:

1. Logic in the model to query the database and return the data to the controller
2. Logic in the controller to call the model passing the parameters as required. Logic in the controller to apply the data from the model to the view.
3. Presentation PHP in the view to display the data given to the view from the controller.

STEP 1. THE MODEL

Create a method in the model as follows:

```
        public function getDetails($filmID){
                $sql = "SELECT * FROM movies WHERE filmID = ?";
                $result_set = $this->db->query($sql, array($filmID));
                return $result_set->row();
        }
```

The above just makes use of the `query()` method in CodeIgniter. We could also write it as:

```
        public function getDetails($filmID){
                $where['filmID'] = $filmID;
                $result_set = $this->db->get_where('movies', $where);
                return $result_set->row();
        }
```

In either case as we are querying by the primary key we only expect one value return thus the use of `row()` not `result_array()`.

STEP 2. CREATE THE CONTROLLER

Create a controller *controllers/Film_details.php*.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Film_details extends CI_Controller {

      public function showFilm($id)
      {
            $this->load->model('Movies');
            $film = $this->Movies->getDetails($id);
            $data['film'] = $film;
            $headData = ['title'=>$film->filmName];

            $this->load->view('header', $headData);
            $this->load->view('film_details', $data);
            $this->load->view('footer');
      }
}
```

STEP 3. AMEND THE VIEW

To help you there is already a view *view/film_details.php*. We need Bootstrap to generate a new row to hold the film description and film image. This can be achieved with:

```
<div class="row">
      <div class="col-md-8">
            <?php

                  echo "<p>{$film->filmDescription}</p>";

            ?>

      </div>
      <div class="col-md-4">
            <?php
                  $imageURL = base_url('/images/'.$film->filmImage);

                  echo "<p class=\"\"><img src=\"{$imageURL}\"
alt=\"{$film->filmName}\" class=\"rounded mx-auto d-block\"></p>";

            ?>
      </div>
</div>
```

BUILDING A CMS

The examples so far have used SQL SELECT statements. In the CMS section of the site we'll build some SQL queries using INSERT, UPDATE AND DELETE.

In your web browser follow the link to the Login. Currently this does not check your credentials - just submit the form to enter the CMS. You should be at the URL

```
https://homepage.shu.ac.uk/~<YOUR-STUDENT-NUMBER/CI-CMS/Cms
```

The CMS presents a list of all the current films, so that they can be edited or deleted and a button allowing a new film to be inserted.

Note: There are tips at the end of this document for securing the CMS.

INSERTING A NEW FILM

To insert a new film we'll need:

1. Logic in the Model.
2. A Controller to communicate between the Model and the View.
3. A View to allow the user to add the details of the new film in a HTML form.

STEP 1: THE MODEL

Open the model and add a method:

```
        public function insertFilm($newData){
                $this->db->insert('movies', $newData);
        }
```

This makes use of CodeIgniter's query builder `insert()` method. The second parameter of the `insert()` method is an array of data to insert. This is an associate array with the name/value pairs representing the field names to be inserted ie:

```
["filmName"]=> "Batman"
["filmCertificate"]=> "PG"
["filmPrice"]=> (10.99)
```

STEP 2: THE CONTROLLER

The controller needs to take the input from a view and send it in the above format to the model.

Open the Controller *controller/cms/Insert_film.php*. When there is a `$_POST['filmName']` value the data is submitted to the Model.

STEP 3: THE VIEW

Create a view *view/insert_film.php*. The view needs to create a form that posts a value for *filmName*. This can be done with the CodeIgniter form helper methods as follows:

```
echo form_open('Insert_film');
echo form_input('filmName', '');
echo form_submit('mysubmit', 'Add');
echo form_close();
```

With the Model, Controller and View now in place you should be able to add a new film to the database. The only field currently been added is *filmName*.

ADDING SERVER SIDE FORM VALIDATION

The above example is lacking any server side validation. CodeIgniter has a library that can be used to offer form validation.

In the controller *controllers/Insert_film.php* after the line loading the form and url helpers add:

```
$this->load->library('form_validation');
```

We can now set validation rules. Add:

```
$this->form_validation->set_rules('filmName', 'filmName', 'required');
```

This indicates that *filmName* is required. Replace the conditional logic of:

```
if(isset($_POST['filmName'])){
```

with:

```
if ($this->form_validation->run() == FALSE){
```

and add the `load->view` methods into the `FALSE` side of the condition.

In the `TRUE` side place:

```
$filmName = $this->input->post('filmName');
$newData['filmName'] = $filmName;
$this->Movies->insertFilm($newData);
redirect('cms');
```

In the view *view/cms/insert_film.php* add:

```
echo validation_errors();
```

This will now echo an error if no *filmName* value is provided.

The error messages can be customized as explained in the document at:

https://www.codeigniter.com/userguide3/libraries/form_validation.html#setting-error-messages

For example:

```
$this->form_validation->set_rules('filmName', 'filmName', 'required',
array('required' => 'Please add a Film Name'));
```

ADDING MORE FIELDS

To add more fields we'll need to edit the view and controller.

In the view you could add a field for *filmPrice* with:

```
echo form_input('filmPrice');
```

In the controller then add:

```
$filmPrice = $this->input->post('filmPrice');
$newData['filmPrice'] = $filmPrice;
```

Tip: To see a more appropriately formatted version of this file open *views/cms/ insert_film-BOOTSTRAP.php*.

ADDING A DROP DOWN LIST OF FILM CERTIFICATES

Given we have a prescribed list of film certificates it will be better to offer the user a drop-down list of options rather than have a free text input.

We can build this by running a query to see what film certificates can be used and then using the data in the view. As before we'll need to:

1. Update the Model
2. Update the Controller.
3. Update the View.

STEP 1: THE MODEL

Add a method to the model *model/Movies.php* as follows:

```
      public function listCerts(){
            $result_set = $this->db->query('SELECT DISTINCT filmCertificate
FROM movies');
            return $result_set->result_array();
      }
```

This uses the SQL command DISTINCT that only returns one example of the different values in a particular field.

STEP 2: THE CONTROLLER

In the controller *controller/Insert_film.php* declare a private variable of `$data` to hold the data from the model.

Add the following request to load the certificate data. Place this after the `load>model()`.

```
// load certificate list
$certList = $this->Movies->listCerts();
$data['certs'] = $certList;
```

We now need to amend the calls to the view to pass the data ie:

```
$this->load->view('cms/insert_film', $this->data);
```

We also need to ensure we populate the `$newData` array with the value for *filmCertificate*. Use similar code to that used for the *filmName* and *filmPrice*.

STEP 3: THE VIEW

Add a new Bootstrap row to the view *view/cms/insert_film.php* as follows:

```
<div class="row">
      <div class="col-md-6">
        <div class="form-group">
        <label for="filmCertificate">Film Certificate</label>
<?php
      foreach($certs as $cert){
            $key = $cert['filmCertificate'];
            $options[$key] = $cert['filmCertificate'];
      }

      echo form_dropdown('filmCertificate', $options, [],
array('class'=>'form-control'));
?>
        </div>
      </div>
</div>
```

The above uses CodeIgniter's form builder to create a drop down list with the `foreach()` populating it with the necessary options.

The `form_dropdown()` method of the form builder takes a number of parameters including the options, the selected option and extras such as classes to be applied. See:

https://www.codeigniter.com/userguide3/helpers/form_helper.html#form_dropdown

You should now have a drop-down to help keep the integrity of your data.

UPDATING AN EXISTING RECORD

To edit an existing record we first will need to populate the view with the current record details. We already have a method `getDetails()` method on the model to help do this.

Notice, the links on the cms page link to:

```
Edit_film/showFilm/[filmID]
```

We'll use this to call a `showFilm()` method in the controller *controllers/Edit_film*. (This is essentially the same technique used for the 'details' example but instead of displaying film values this will place them as the default values in a HTML form).

We will also need a new method in the model to update the record and this will need to be sent the primary key of the record to be updated.

Again we need to tackle this in three steps:

1. Add the new method to update the record in the model.
2. Amend the controller to communicate between the model and the view.
3. Build a view to display a HTML form with the current data allowing changes to be added for submission to the controller.

STEP 1: AMENDING THE MODEL

Open the model *models/Movie.php* and add the following method.

```
public function updateFilm($filmID, $newData){
      $where['filmID'] = $filmID;
      $this->db->update('movies', $newData, $where);
}
```

Notice that this method takes two parameters, the primary key of the record to be updated and then the new data. This example uses the CodeIgniter query builder `update()` method.

STEP 2: PASSING DATA VIA THE CONTROLLER

Open the controller *controllers/Edit_film.php*. This file has a `showFilm()` method and conditional logic based around `form_validation->run()`.

At the moment, the logic in the file only edits the *filmName* field. Amend the file to allow the editing of *filmCertificate* and *filmPrice*.

Note: you may want to use the same dropdown technique used for certificates in the INSERT example.

STEP 3: UPDATING THE VIEW

Open the view *view/cms/edit.php*. This currently only has the HTML to allow the *filmName* to be edited. Edit the file to allow the *filmCertificate* and *filmPrice* values to be changed.

DELETING RECORDS

To delete a film is again a three step process.

1. Provide logic in the Model
2. Provide a controller to pass the primary key of the record to be deleted between the Model and View
3. Provide a view for the user to interact with.

STEP 1: THE MODEL

Open the model *models/Movies.php* and add the following method. This expects a parameter equal to the primary key of the film to be removed.

```
public function deleteFilm($id){
      $this->db->where('filmID', $id);
      $this->db->delete('movies');
}
```

STEP 2: THE CONTROLLER

The controller needs to send the primary key of the film to be deleted to the Model. Add the following to the controller when it receives a `$_POST` request.

```
$id = $this->input->post('filmID');
$this->Movies->deleteFilm($id);
redirect('index.php/cms');
```

STEP 3: THE VIEW

Open the view *view/confirm_delete.php*. This triggers the post request to the controller. We need to make sure that the primary key is displayed in the form. Add the following to the form in the view:

```
  <input type="hidden" name="filmID" value="<?php echo $film->filmID;?>">
```

IMPROVING SECURITY XSS

To improve the security of the application we should protect against Cross Site Scripting. This is where there is a malicious attempt to insert Javascript into your database.

CodeIgniter recommends that you filters outputs such that no `<script>` tags can be maliciously added to your pages.

This can be done in the model *models/Movies.php* by using the method `security->xss_clean()`.

For example the method `listCerts()` could be amended to:

```
public function listCerts(){
            $result_set = $this->db->query('SELECT DISTINCT filmCertificate
FROM movies');
            // cross site scripting clean up on output
            $data = $this->security->xss_clean($result_set-
>result_array());
            return $data;
}
```

Update all the methods of the Model that return data.

In addition to the above inputs can be filtered with the Input Class. We have seen `input->post()` and `input->get()`. These methods take two parameters. The first is the name of the parameter, the second optional value is a Boolean that can be used to apply XSS cleaning on input.

For example, the following amendment could be made to the controller *controllers/Edit_film.php*.

```
$filmName = $this->input->post('filmName', true);
```

When used on an input the XSS cleaning would replace:

```
<script>alert('hi');</script>
```

with the now safe:

```
[removed]alert('hi');[removed]
```

TIPS FOR THE LOGIN

CodeIgniter works with sessions in the same ways as generic PHP, in that they are controlled by the *PHPSESSID* cookie.

The configuration of Sessions is controlled by *config/config.php*. Look for the variables for Sessions and Cookies ie:

```
$config['cookie_secure']       = FALSE;
$config['cookie_httponly']     = FALSE;
```

To secure pages, it is useful to create a new controller called `MY_Controller`. This can then be used as the basis of all the controllers of pages that need to be secure ie:

```
//class Cms extends CI_Controller {
class Cms extends MY_Controller {
…
```

In *core* create a class called *MY_Controller* as follows:

```
<?php
class MY_Controller extends CI_Controller {

    function __construct()
    {
        parent::__construct();
            $CI = & get_instance();
            $CI->load->library('session');
            $CI->load->helper('url');
        if (!isset($_SESSION['login']))
        {
            redirect('login');
        }
    }
}
```

Then in the controller of any pages that you wish to protect, have the controllers extend *MY_Controller* rather than *CI_Controller*.