**CST-256 Activity 2 Guide**

Contents

[Part 1: Laravel Routes and Controllers 2](#_Toc528659305)

[Part 2: Laravel N-Layer Application 5](#_Toc528659306)

[Part 3: Laravel Blade Templates 7](#_Toc528659307)

This activity has multiple parts. All parts must be completed prior to documentation submission.

## Part 1: Laravel Routes and Controllers

**Special Note:** Part 1 should begin in Topic 1 but is not required to be finished until Topic 2.

**Overview**

In this activity, students will continue to explore Laravel Routes and Controllers in conjunction with learning how to process HTML Form Data in a Controller and pass this data onto a View.

**Execution**

Execute this assignment according to the following guidelines:

1. Create a new Controller:
   1. In Eclipse PHP, select the Terminal View (if this is not visible go to the Window->Show View->Other->Terminal menu options). Open a Terminal session by clicking on the Open Terminal icon. This will open the Command Line Tools Console where you can run Laravel Artisan commands. In the Terminal navigate thru the file system to reach the root directory of your project. One of the Artisan commands can be used to create a Laravel Controller class.
   2. Run the following Artisan command:
      1. artisan make:controller WhatsMyNameController
   3. Go to the /app/Http/Controllers and verify that a WhatsMyNameController.php was created. Notice this class extends from the Laravel Controller class.
   4. Add some implementation code to an index method index.

public function index(Request $request)

{

// Usage of path method

$path = $request->path();

echo 'Path Method: '.$path;

echo '<br>';

// Usage of is method

$method = $request->isMethod('get') ? "GET" : "POST";

echo 'GET or POST Method: '.$method;

echo '<br>';

// Usage of url method

$url = $request->url();

echo 'URL method: '.$url;

echo '<br>';

}

* 1. Open the web.php file in the /routes directory. Add a new Resourceful route:
     1. Route::get('/whoami','WhatsMyNameController@index');
     2. Test the Controller with the browser by navigating to /public/ whoami'.
     3. Take screenshots of your results.

1. Create a new View:
   1. Go to the /resources/views directory and create a new PHP page, whoami.php, with the following in the body of the page.

<form action = "whoami" method = "POST">

<input type = "hidden" name = "\_token" value = "<?php echo csrf\_token() ?>">

<h2> What's Your Name?</h2>

<table>

<tr>

<td>First Name: </td>

<td><input type = "text" name = "firstname" /></td>

</tr>

<tr>

<td>Last Name:</td>

<td><input type = "text" name = "lastname" /></td>

</tr>

<tr>

<td colspan = "2" align = "center">

<input type = "submit" value = "Ask Now" />

</td>

</table>

</form>

* 1. In the Routes change the /whoami URI mapping from a GET to POST.
  2. Open the web.php file in the /routes directory:
     1. Add a new route for the View:
        1. Route::get('/askme', function () { return view('whoami'); });
     2. Test the View and Controller with the browser by navigating to /public/askme.
     3. Take screenshots of your results.

1. Update the Controller to handle the Form Post Data:
   1. Add the following 2 lines of code to read the Form Variables in the Controller created in Step 1d:

$firstName = $request->input('firstname');

$lastName = $request->input('lastname');

echo "Your name is: " . $firstName . " " . $lastName;

echo '<br>';

* 1. Test the View and Controller with the browser by navigating to /public/askme and posting the form to the Controller.
  2. Take a screenshot of your result.

1. Create a new View:
   1. Go to the /resources/views directory and create a new PHP page, thatswhoami.php, with the following in the body of the page.

<h2> Hello!</h2>

<?php echo $firstName . " " . $lastName; ?>

<br>

<a href="askme">Go Again</a>

* 1. Add the following code to the bottom of the Controller created in step 3a:

$data = ['firstName' => $firstName, 'lastName' => $lastName];

return view('thatswhoiam')->with($data);

* 1. Test the Controller and Views with the browser by navigating to /public/askme.
  2. Take a screenshot of your result.

1. Research and Design:
   1. Read the "Layered Architecture Is Good" and "What Is N-Tier Architecture," located in the required readings.
   2. Review the Registration and Login pages you created in CST-236. Browse the Internet and review several registration pages for professional networking sites, similar to the one needed in this COURSE.
   3. Draw and provide a logical diagram illustrating an N-Layer application architecture for your application.

**Submission**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Create a Project Report using a GCU standard Project Header Page/Information containing your name, course, assignment name, and date. Make sure to include any screenshots demonstrating working code and write-ups as instructed.
2. Screenshots for the following URIs:
   1. /whoami (as GET route)
   2. /askme (as PHP page and Form Post to the Controller)
   3. /askme (as a Form Post to the Controller that prints Form Data)
   4. /askme (as a Form Post to the Controller that responses with a View)
3. A block diagram for each of the components (M, V, C) of the MVC design pattern. Map each component of the MVC design pattern to its equivalent component within the Laravel Framework and the code you ended up after completing Step 4b of Part 1: Laravel Routes and Controllers.
4. Logical Diagram of an N-Layer Application Architecture for a Login Module and Registration Module.

## Part 2: Laravel N-Layer Application

**Overview**

In this activity, students will use Laravel Routes, Controllers, Models, and Views to build a simple Login Page using an N-Layer Web Application Architecture.

**Execution**

Execute this activity according to the following guidelines:

1. Create a Database Schema and Table:
   1. Open your WebStart page from the MAMP Control Panel.
   2. From the Tools Menu select the phpMyAdmin menu item.
   3. Click the Database tab in phpMyAdmin.
   4. Create a Database using a database name of activity2.
   5. Create a Table named users with 3 columns and click the Go button.
   6. Create the following columns:
      1. ID of type INT and check the A.I. checkbox.
      2. USERNAME of type VARCHAR with length of 100.
      3. PASSWORD of type VARCHAR with length of 100.
      4. Click the Save button.
   7. Insert some test users in to the table.
2. Create a new Login Form and Login Controller:
   1. Go to the /resources/views directory and create a new PHP page, login.php, with an input text element named 'username', an input password text element named 'password', and a submit button. The Login Form should use a POST action to the Login Controller URI created in Step 2d below.
   2. In EclipsePHP select the Terminal View (if this is not visible go to the Window->Show View->Other->Terminal menu options). Open a Terminal session by clicking on the Open Terminal icon. This will open the Command Line Tools Console where you can run Laravel Artisan commands. In the Terminal navigate thru the file system to reach the root directory of your project. One of the Artisan commands can be used to create a Laravel Controller class. Run the following Artisan command:
      1. artisan make:controller LoginController
   3. Add some implementation code to an index method index that retrieves the Login Username and Password Form elements and prints them to the request.
   4. Open the web.php file in the /routes directory. Add the following new routes:
      1. Route::get('/login, function () { return view('login'); });
      2. Route::post('/dologin','LoginController@index');
   5. Test the Login Controller with the browser by navigating to /public/login.
      1. Take screenshots of your results.
3. Create a User Model class, Security Service class, and Security Data Access Service class:
   1. Create a User Model:
      1. Right click on the app folder and select New->Directory. Name your Directory Models.
      2. Right click on the Models folder and select New->PHP Class. Name your class UserModel and put in the namespace App\Models.
      3. Implement your UserModel class by adding private username and password class variables along public constructor that takes username and password arguments and also add getter methods for each property.
   2. Create a Security Data Access Service:
      1. Right click on the app folder and select New->Directory. Name your Directory Services.
      2. Right click on the Services folder and select New->Directory. Name your sub-folder Data.
      3. Right click on the Services\Data folder and select New->PHP Class. Name your class SecurityDAO and put in the namespace App\Services\Data.
      4. Implement your SecurityDAO class by adding a public method called findByUser that takes a UserModel as a single argument. Implement findByUser() by writing mysqli methods to query the user table created in step 1 and searching for the passed in username and password. Return true if the user is found else return false if the user is not found.
   3. Create a Security Business Service:
      1. Right click on the Services folder and select New->Directory. Name your sub-folder Business.
      2. Right click on the Services\Business folder and select New->PHP Class. Name your class SecurityService and put in the namespace App\Services\Business.
      3. Implement your SecurityService class by adding a public method called login that takes a UserModel as a single argument. Finish the implementation of your login() by calling SecurityDAO.findByUser(). The login() should return returning the value true if the user is found else return false if the user is not found.
   4. Update your Login Controller to call your security business logic:
      1. Create a UserModel from the username and password that was posted to the LoginController.
      2. Call the SecurityService.login() passing in the UserModel created in the previous step. Save the result of the login() method call in a local variable.
      3. If login() returns false navigate to a new View called loginFailed.php that simply renders a login error on the screen.
      4. If login() returns true navigate to a new View called loginPassed.php that simply renders a login success message on the screen. If desired pass the UserModel to the View and print the username on the screen.
   5. Test the Controller and Views with the browser by navigating to /public/login.
   6. Take a screenshot of the login results for a user that exists in the database and a user that does not exist in the database.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Add the screenshots for the following in your Project Report:
   1. Login Form
   2. Login Response for user found
   3. Login Response for user not found
2. Upload your code to Bitbucket and include the URL of your Bitbucket project in the Project Report.

## Part 3: Laravel Blade Templates

**Overview**

In this activity, students will use Laravel Blade Templates to enhance the Login Page created in Activity 2 Part 2: Laravel N-Layer Application.

**Execution**

Execute this assignment according to the following guidelines:

1. Create a Master Layout:
   1. Create a layouts directory in the resources/views directory by right clicking on the resources/views directory and selecting the New->Directory menu items.
   2. Create a new PHP File called 'appmaster.blade.php' in the resources/views/layouts directory. This template will have standard header and footer sections as well as a section for the main page content.
   3. Update the master template as follows:

<html lang="en">

<head>

<title>@yield('title')</title>

</head>

<body>

<div align="center">

@yield('content')

</div>

</body>

</html>

* 1. Copy the login.php file to login2.blade.php. Right click on the login.php file. Select the copy menu item. Right click on the views folder and select the paste menu item and name your file login2.blade. Create route for this view:

Route::get('/login2', function ()

{

return view('login2');

});

* 1. Update the login2.blade.php file to use the new appmaster template, set the title of the page, and set the content section to the Login Form as follows:

@extends('layouts.appmaster')

@section('title', 'Login Page')

@section('content')

<!-- Note Shown: Insert your Login Form from login.php Here -->

@endsection

* 1. Test the App Master template with the browser by navigating to /public/login2.
  2. Take a screenshot of the Login Form.

1. Enhance the App Master template with a Header and Footer:
   1. Add new Header and Footer sections to the App Master template.

@include('layouts.header')

<div align="center">

@yield('content')

</div>

@include('layouts.footer')

* 1. Create a Header file:
     1. Right click on the resources/views/layouts directory and select the File->New PHP File. Name your file header.blade. Remove all default PHP code that was generated. Add some HTML tags to display the Application Name 'Welcome to Activity 2' that is centered and styled using the h2 format.
     2. Test the App Master template with the browser by navigating to /public/login2.
     3. Take a screenshot of the Login Form.
  2. Create a Footer file:
     1. Right click on the resources/views/layouts directory and select the File->New PHP File. Name your file footer.blade. Remove all default PHP code that was generated. Add some HTML tags to display a copyright of 'Copyright @2018 My Own Company Name' that is centered and styled using the h5 format.
     2. Test the App Master template with the browser by navigating to /public/login2.
     3. Take a screenshot of the Login Form.

1. Add Conditional Logic in a View:
   1. Create a new PHP file under views called 'loginPassed2.blade'.
   2. Change the Login Controller to return the loginPassed2 View.
   3. Change the content of the loginPassed2 View" to the following:

@if($model->getUsername() == 'mark')

<h3>Mark you have logged in successfully.</h3>

@else

<h3>Someone besides mark logged in successfully.</h3>

@endif

* 1. Insert 2 users into the users table, one user should have a username of 'mark' and another user should not have a username of 'mark'
  2. Test the App Master template with the browser by navigating to /public/login2 and testing the Login Response for a username of 'mark' and a username that is not 'mark.'
  3. Take screenshots of both the Login Form responses.

**Submission**

Submit the following to the learning management system:

A Project Report to include:

1. Screenshots for the following URIs:
2. /whoami (as GET route)
3. /askme (as PHP page and Form Post to the Controller)
4. /askme (as a Form Post to the Controller that prints Form Data)
5. /askme (as a Form Post to the Controller that responses with a View)
6. A block diagram for each of the components (M, V, C) of the MVC design pattern. Map each component of the MVC design pattern to its equivalent component within the Laravel Framework and the code you ended up after completing Step 4b of Part 1.
7. Logical Diagram of an N-Layer Application Architecture for a Login Module and Registration Module.
8. Screenshots for the following:
   1. Login Form
   2. Login Response for user found
   3. Login Response for user not found
9. The URL of your Bitbucket project after uploading the code to Bitbucket.
10. Screenshots for the following:
    1. Login Form using Template (with no header and footer)
    2. Login Form using Template (with header and footer)
    3. Login Response for user named 'mark' and a user not named 'mark'