**CST-256 Activity 5 Guide**

Contents

[Part 1: Bootstrap 2](#_Toc528659494)

[Part 2: Laravel RESTful Services 4](#_Toc528659495)

[Part 3: Middleware 8](#_Toc528659496)

[Part 4: IoC Service Provider 11](#_Toc528659497)

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

## Part 1: Bootstrap

**Overview**

In this activity, students will add Bootstrap to a previous Laravel Activity to create a responsive application in Laravel. They will also explore a Bootstrap IDE that will be used to develop a Bootstrap Navigation Bar.

**Execution**

Execute this activity according to the following guidelines:

1. Install Bootstrap Studio
   1. Go to <https://bootstrapstudio.io/>.
   2. Scroll down to the bottom of the Home Page to the FAQ link. Click on the FAQ link. Scroll to the bottom of the FAQ where the question is asked if there "Is a student license." Click on the Apply link (to <https://bootstrapstudio.io/pages/student-license>).
   3. Fill out the Application for the Student License form and submit the request.
   4. Install the Bootstrap Studio IDE for your Operating System.

2. Create a new Bootstrap Studio Project:

1. Start the Bootstrap Studio IDE application.
2. Create a new Project by selecting the Create New Design button from the Bootstrap Studio main screen.
3. Name your Project Test, use the Blank Template, and click the Create button.

3. Create a NavBar using Bootstrap Studio:

1. Expand the Navigation options under the UI Components.
2. Drag the 'Navigation with Button' onto the Blank Page in your Project.
3. Make the following changes to the default Navbar:
   1. Change the 'Company Name' label to 'Welcome to Activity 5'.
   2. Change the 'First Item' label to 'GCU'. From the right pane select the Options button and replace the URL property to 'http://www.gcu.edu' (without single quotes) and change the Target property to Blank. Also click the Deactivate button from the Add to Nav toolbar.
   3. Optionally you can delete the 'Second Item' and 'Dropdown' options from the NavBar.
   4. Change the href for the Log In text button. From the right pane select the Options button and replace the URL property to 'login5' (without single quotes).
   5. Change the 'Sign Up' button label to 'Log Off' and also replace the URL in the properties from the right pane to '/' (without single quotes).
   6. Inspect the generated HTML by clicking the HTML tab below the source window.
   7. Save your project by clicking the Save menu option.
   8. Test the responsiveness of your page by clicking the XS, SM, and MD icons from the page toolbar.
   9. Create folder on your Desktop named 'bootstrap'.
   10. Export your page by clicking the Export menu option. Browse and select the 'bootstrap' folder from Desktop. Leave the default export options. Click the Save button. Click the Export button.

4. Update the root/URI file, resource assets, and route:

1. From the Bootstrap Studio project folder copy the generated assets folder to the Laravel Resources directory (resources).
2. From the Bootstrap Studio project folder copy the generated index.html page to a file named index.blade.php. Copy the index.blade.php file to the Laravel Views directory (resources/views).
3. Update the index.blade.php file so all of the Bootstrap and jQuery files are referenced from a URI of resources/assets.
4. Update the bootstrap.min.js file located in the assets/bootstrap/js directory and remove the line that references the bootstrap.bundle.min.js.map file.
5. Update the web.php routes for the / URI by replacing the welcome view with the index view (i.e., 'welcome' to 'index').
6. Deploy your code and test your changes by going to the root of your application as well as resizing your application.
7. Take a screenshot of the default application page showing the Navigation Bar in full screen as well as resized to show the "hamburger" menu.

5. Refactor the application to use Bootstrap and Navigation Bar across all pages:

1. Open the appmaster.blade.php file.
2. Copy the head section from index.blade.php into this file.
3. Open the header.blade.php file.
4. Copy the Navigation Bar (all content from opening and closing body tag) from index.blade.php into this file.
5. Remove the remaining content from index.blade.php.
6. Update index.blade.php to extend from layouts.appmaster, with a section title of 'ICA Page', and with an empty content section.
7. Deploy your code and test your changes by going to the root of your application as well as resizing your application.
8. Take a screenshot of the default application page showing the Navigation Bar in full screen as well as resized to show the "hamburger" menu.

**Extra Practice:**

1. Why is it important to consider designing an application today using responsive design frameworks?
2. What extra work is required for designers and developers when supporting a responsive design?

**Documentation**

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. Add the screenshots for the following in your Project Report:
   1. Part 1 Step 4g
   2. Part 1 Step 5h

## Part 2: Laravel RESTful Services

**Overview**

In this activity, students will explore creating a REST(ful) service that is based on using a Laravel REST(ful) Controller. They will use their Browser and Postman to test the REST(ful) Services.

**Execution**

Execute this activity according to the following guidelines:

1. Create a RESTful Controller:
   1. In EclipsePHP select the Terminal View (if this is not visible go to theWindow->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 UsersRestController --resource
   3. Go to the /app/Http/Controllers and verify that a UsersRestController.php was created.
   4. Delete all methods from the UsersRestController.php controller class except the index() and show() methods.
   5. Open the web.php file in the /routes directory. Add a new Resourceful route:
      1. Route::resource('/usersrest', 'UsersRestController');
      2. Test all the GET verb URIs per the following table.
      3. Take screenshots of all your results.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Action | Route Name |
| GET | /usersrest | index | UsersRestController.index |
| GET | /usersrest/{id} | show | UsersRestController.show |

1. Create a DAO and Business Service Functions:
   1. In the Security DAO add a method findAllUsers() that returns a List of UserModel for all the users in the database.
   2. In the Security Service add a method getAllUsers() that returns a List of UserModel by calling the Security DAO findAllUsers().
   3. In the Security DAO add a method findUserByID($id) that returns a UserModel for User ID (from the method argument) in the database.
   4. In the Security Service add a method getUser($id) that returns a UserModel by calling the Security DAO findUserByID($id).
2. Make your UserModel serializable.
   1. In the UserModel implement the \JsonSerializeable interface.
   2. Implement the publc jsonSerialize():

return get\_object\_vars($this);

1. Create a Data Transfer Object:
   1. In the Models namespace create a class called DTO.
   2. Make the DTO class serializable and implement the public jsonSerialize() method.
   3. Add the following properties with a non-default constructor that initializes all properties:
      1. errorCode
      2. errorMessage
      3. data
2. Update the REST Controller to call Security Services:
   1. Call SecurityService.getAllUsers() method from the UsersRestController.index() method. Create a DTO using an appropriate error code, error message, and data payload. Serialize the DTO using the json\_encode(). Return the JSON from the Controller.
   2. Call SecurityService.getUser() method from the UsersRestController.show() method. Create a DTO using an appropriate error code, error message, and data payload. Serialize the DTO using the json\_encode(). Return the JSON from the Controller.
3. Test the REST API using your desktop Browser and Postman:
   1. Take screenshots for each REST API call from your Browser:
      1. localhost/playlaravel/usersrest
      2. localhost/playlaravel/usersrest/{id that exists}
      3. localhost/playlaravel/usersrest/{id that does not exist}
   2. Download Postman from <https://www.getpostman.com>.
   3. Refer to the "How-to Guides" resource located within the Course Materials.
   4. Click on the New Collection icon. Enter a desired Collection name. Click the Create button.
   5. Click the new Collection name in the Collection list.
   6. Click the Add Requests link. Enter a desired Request name. Click the Save to button.
   7. Click the new Request name listed within the Collection.
   8. From the Request enter the following information:
      1. GET Command
      2. Request URL
      3. Click the Save button
   9. Repeat steps g-i for the following URLs, click the Send button to test each URL, and take a screenshot from Postman for each tested URL:
      1. localhost/playlaravel/usersrest
      2. localhost/playlaravel/usersrest/{id that exists}
      3. localhost/playlaravel/usersrest/{id that does not exist}
4. Test the REST API using PHP Client Code:
   1. Update your project to include the Guzzle Library. Refer to the "How-to Guides" resource located within the Course Materials.
   2. Create a Test Client Controller:
      1. Using artisan create a Controller called RestClientController.
      2. Add the following code to the RestClientController index() after adding a route to Controller's index() method:
         1. Add using namespace: use GuzzleHttp\Client;
         2. See image below (you may have a different host, port, and application name):



* 1. Take screenshots of the REST Client response using your Browser:
     1. localhost/playlaravel/usersrest
     2. localhost/playlaravel/usersrest/{id that exists}
     3. localhost/playlaravel/usersrest/{id that does not exist}

**Extra Practice:**

1. What advantages are there to building REST(ful) based services versus SOAP based services?
2. What disadvantages are there to building REST(ful) based services versus SOAP based services?
3. Research the standard verbs that are defined for REST(ful) based interfaces. In the context of the REST(ful) Controller created in exercise 1 describe (in 50-100 words) some advantages to creating a REST(ful) Controller?
4. Why is a tool like Postman (see link below) necessary to learn when developing REST(ful) based services?

**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 browser screenshots for the following URIs to your Project Report:
   1. /usersrest
   2. /usersrest/{id that exists}
   3. /usersrest/{id that not does exist}.
2. Add the Postman screenshots for the following URIs to your Project Report:
   1. /usersrest
   2. /usersrest/{id that exists}
   3. /usersrest/{id that not does exist}.
3. Add the Rest Client screenshot for the following URIs to your Project Report:
   1. /usersrest
   2. /usersrest/{id that exists}
   3. /usersrest/{id that not does exist}.

## Part 3: Middleware

**Overview**

In this activity, students will explore some of the misc. services that are available to build enterprise class applications in Laravel. They will explore how to build Middleware, a Data Cache, and a Security Filter (i.e., their own custom Security Framework ignoring what is already built into Laravel).

**Execution**

Execute this activity according to the following guidelines:

1. Create a Middleware Service:
   1. 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 Middleware Controller class.
   2. Run the following Artisan command:

artisan make:middleware MyTestMiddleware

* 1. Go to the /app/Http/Middleware and verify that a MyTestMiddleware.php class was created.
  2. Register the Middleware by opening the Kernel.php class located in the /app/Http directory. Add the following line to the $middlewareGroups array:

\App\Http\Middleware\MyTestMiddleware::class,

* 1. Use the Logging Utility created in Activity 4 and add a logging statement showing when the Middleware handle() was called. Run your application.
     1. Take a screenshot of the logging statement.

1. Create a Data Cache:
   1. Modify the Middleware Service to test Laravel's File Based Cache.
   2. Check for a non-null value for the username parameter posted from the Login Form.
   3. If the username value is not null then store the username in a File Store Cache in a key called 'mydata' for a duration of 1 minute. Use the logger to indicate that a value was put in the cache.
   4. If the username value is null then read the username from the File Store Cache. If the value is not null then use the logger to indicate that the value was obtained from the cache. If the value is null then use the logger to indicate that the value was not available in the cache.
   5. Run the application. Log into the application. Validate that the username was stored in the cache. Then access the Logout menu and do this a few times until the cache expires after 1 minute. Validate that the username was expired from the cache.
      1. Take a screenshot of the logging statements.
2. Create a Security Filter:
   1. 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 Middleware Controller class.
   2. Run the following Artisan command:

artisan make:middleware MySecurityMiddleware

* 1. Go to the /app/Http/Middleware and verify that a MySecurityMiddleware.php class was created.
  2. Register the Middleware by opening the Kernel.php class located in the /app/Http directory. Add the following line to the $middlewareGroups array:

\App\Http\Middleware\MySecurityMiddleware::class,

* 1. Add the following logic in the Middleware handle() method:
     1. Step 1: Get the route URI path for any request handled by the Middleware:

$path = $request->path();

MyLogger2::info("Entering My Security Middleware in handle() at path: " . $path);

* + 1. // Step 2: Run the business rules that check for all URIs that you do not need to secure (this is a smaller number of URIs as opposed to all the URIs that need to be secure) and also make sure you consider the REST API URI (NOTE, the code below may be using URIs not in your ICA code and will need to updated accordingly based on your code):

$secureCheck = true;

if ($request->is('/') || $request->is('login3') || $request->is('dologin3') ||

$request->is('usersrest') || $request->is('usersrest/\*') ||

$request->is('loggingservice'))

{ $secureCheck = false;}

MyLogger2::info($secureCheck ? "Security Middleware in handle().....Needs Security" : "Security Middleware in handle().....No Security Required");

* + 1. Step 3: If entering a secure URI with no security token then do a redirect to the root URI or Login page:

if($secureCheck)

{

MyLogger2::info("Leaving My Security Middleware in handle() doing a redirect back to login");

return redirect('/login3');

}

* + 1. Make sure to return the next filter in the chain in the method return (i.e., return $next($request);).
    2. Take a screenshot of your logging statements validating that your Security Filter is functioning properly.

**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 activities to your Project Report:
   1. Middleware Service
   2. Data Cache
   3. Security Service

## Part 4: IoC Service Provider

**Overview**

In this activity, students will use Laravel's Inversion of Control (IoC) container to inject a Logging Service into a Controller. They will explore how to build a Laravel Service Provider and leverage Dependency Injection.

**Execution**

Execute this assignment according to the following guidelines:

1. 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.
2. Run the following command to create a Logging Service Provider:

artisan make:provider LoggingServiceProvider

1. Create a Logging Service Interface and Implementation:
   1. Copy ILogger.php to ILoggerService.php
   2. Copy MyLogger1.php to MyLogger3.php
   3. Remove the getLogger() method
   4. Remove static from all methods
2. Register the Logging Service Interface:
   1. Open the LoggingServiceProvider in the Providers directory
   2. Bind a singleton to the My Logging Service:

$this->**app**->singleton(**'App\Services\Utility\ILoggerService'**, **function** ($app) {  
 **return new** MyLogger3();  
});

1. Add the Logging Service Provider:
   1. Open the application configuration file in config\app.php.
   2. Add the Provider to the list of providers array:

App\Providers\LoggingServiceProvider::***class***,

1. Create a Test Logging Service Controller:
   1. Create a TestLoggingController.
   2. Create a protected class variable called $logger.
   3. Create a \_\_constructor() that takes an ILoggerService method argument.
   4. Set the method argument to the $logger class variable.
   5. Create an index() method and in its implementation call the info() method on the $logger class variable.
2. Create a Route to the Test Controller:
   1. Create a route to the test() using a GET request at URI \loggingservice:

Route::*get*(**'/loggingservice'**,**'TestLoggingController@index'**);

1. Validate the Test Logging Service Controller at \loggingservice and verify the log message in the Laravel log file located in the storage/logs directory with in htdocs.
   1. Take a screenshot of the log message.

**Submission**

Submit the following to the learning management system:

A Project Report to include:

1. Screenshots for the following:
   1. Part 1 Step 4g
   2. Part 1 Step 5h
2. Browser screenshots for the following URIs:
3. /usersrest
4. /usersrest/{id that exists}
5. /usersrest/{id that not does exist}.
6. Postman screenshots for the following URIs:
7. /usersrest
8. /usersrest/{id that exists}
9. /usersrest/{id that not does exist}.
10. Rest Client screenshots for the following URIs:
11. /usersrest
12. /usersrest/{id that exists}
13. /usersrest/{id that not does exist}.
14. Screenshots for the following activities:
15. Middleware Service
16. Data Cache
17. Security Service
18. Screenshots for the following activities:
    1. Log message