The PHP Company

Participant Exercises Guide

Zend Framework 2: Fundamentals

 version 1.4

Page left blank for online viewing

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

In this lab, you will be building on top of the basic **onlinemarket.work** project using the Zend Skeleton App. Using the Zend Skeleton App allows you to realize immediate results with Zend Framework 2. As the course progresses, the architecture behind Zend Framework 2 will be explained, providing a greater understanding of exactly what is happening in the internal workings of the Zend Skeleton App.

During this lab, you will complete the following tasks:

- Install the Zend Skeleton App
- Install Zend Framework 2
- Create a virtual host definition for **onlinemarket.work**
- Test the results in your browser

Depending on the lab system being used in your course, additional notes might appear as part of the lab instructions.

- *CLOUD* indicates notes specific to the **my.phpcloud.com** environment
- *VM* indicates notes specific to a **Virtual Machine**
- *RT* indicates notes specific to a **ReadyTech Virtual Session**

The basic workspace folder is indicated in all labs as "**path/to/workspace**". The actual path varies depending on your lab system:

CLOUD: **path/to/workspace** is the **Zend Studio workspace** folder you created as part of the initial cloud setup instructions you received

VM: **path/to/workspace** is **/workspace**

RT: **path/to/workspace** is **/workspace**
If asked for the `root` password (using the `sudo` command), it is `password`

My Notes

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

1. INSTALL THE ZEND SKELETON APP

The Zend Skeleton App is a generic MVC-based ZF2 application that provides a solid foundational structure for developing a project. There are several ways in which this application can be obtained and installed:

- Using *zftool.phar*
- Using *Composer*
- Using *Composer* and *git*
- Downloading and unzipping

For the purposes of this lab you will be using `zftool.phar`

1. Download the latest version of `zftool.phar` from the following URL:
   `https://packages.zendframework.com/zftool.phar`

2. Move `zftool.phar` to the folder
   `path/to/workspace/onlinemarket.start/files`

3. Test that your version of `zftool.phar` works as follows:

   ```
   php path/to/workspace/onlinemarket.start/files/zftool.phar --help
   ```

   You should see a help screen listing the available commands.

4. Open a terminal window so that you have access to the command prompt

5. Change to the `path/to/workspace` folder

My Notes

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

6. To create the **onlinemarket.work** project issue this command:

   ```
   php path/to/workspace/onlinemarket.start/files/zftool.phar create project onlinemarket.work
   ```

   NOTE: if you have a working Internet connection, `zftool.phar` will obtain the latest version of the Zend Skeleton App from `github.com`. Otherwise, the version of the Zend Skeleton App that is stored in a compressed format inside `zftool.phar` is used

7. Import the new project into Zend Studio:

   - **File | New | PHP Project from Existing Directory**
   - Project Name: `onlinemarket.work`
   - Location: `path/to/workspace/onlinemarket.work`
   - Content: `Basic`
   - Version: `PHP 5.3` or `PHP 5.4` as indicated by your instructor

8. Copy the `public` folder from `onlinemarket.start` into the new project, overwriting any files created by `zftool.phar`

My Notes

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

2. INSTALL ZEND FRAMEWORK 2

ZF2 can be installed in a number of ways:

- Using `zftool.phar`
- Using `composer`
- Download and unzip

For the purposes of this lab, you will be using `zftool.phar`

1. Open a terminal window so that you have access to the command prompt
2. Remove the folder: `path/to/workspace/onlinemarket.work/vendor/ZF2`
3. Issue the following command:

   ```
   php /path/to/workspace/onlinemarket.start/files/zftool.phar install zf path/to/workspace/onlinemarket.work/vendor/ZF2
   ```

4. Verify that you now have a working installation of ZF2 in the folder

   `path/to/workspace/onlinemarket.work/vendor/ZF2`

My Notes

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

3. CREATE VHOST (VIRTUAL HOST) DEFINITION

NOTE: the vhost file may already exist in the folder: `/etc/apache2/sites-available`

1. Open up a terminal window in order to gain access to your lab system's command prompt
2. Edit the `/etc/hosts` file and add an entry for a new host `onlinemarket.work` assigned to IP address `127.0.0.1` using the command: `sudo gedit /etc/hosts`
3. Change to the `/etc/apache2/sites-available` directory on your lab system
4. Create the virtual host definition file `onlinemarket.work.conf` using the command: `sudo gedit onlinemarket.work.conf`
5. Create a link for this file in `/etc/apache2/sites-enabled` using the command:

   ```
   sudo ln -s /etc/apache2/sites-available/onlinemarket.work.conf
   /etc/apache2/sites-enabled/onlinemarket.work.conf
   ```

6. Restart the server using the command: `sudo /usr/local/zend/bin/apachectl restart`
7. Observe and correct any errors

   NOTE: to perform these commands you will need to be the `root` user. The `sudo` command allows you to perform a command a the `super user`. When prompted, the password is `password`

My Notes

Module 2

MVC-BASED ZEND SKELETON APP LAB … M2Ex1

4. TEST RESULTS IN BROWSER

1. Open the browser
2. Enter the URL for the new project: `http://onlinemarket.work`
3. You should see a "splash" screen, "**Welcome to Zend Framework 2**"
4. Observe and correct any errors

My Notes

Module 3

EVENT MANAGER LAB … M3Ex1

In this lab, you will be working with the Event Manager to attach an event handler to the `dispatch` event (explained in more detail in the course module on MVC). The MVC `dispatch` event is triggered when an MVC controller is instantiated, and one of its methods (called "actions") are invoked. The best place to assign a `listener` (that is, a handler method you create) for this event is in the `Module.php` file. For the purposes of the lab you will be working with

`path/to/workspace/onlinemarket.work/module/Application/Module.php`

Note the `onBootstrap()` method inside `Module.php`. This method accepts a `Zend\Mvc\MvcEvent object` as an argument. This method is convenient in that it is invoked automatically by a handler which listens for an MVC `bootstrap` event. The theory behind this is explained in more detail in a later course module.

---

1. In Zend Studio, open `Module.php`
2. In the `onBootstrap()` method, at the end, attach a listener with the following characteristics:
    - Listens For: the `dispatch` event
    - Context: current object
    - Handler: `onDispatch`
    - Priority: 100
3. Define method `onDispatch()`, which accepts an `MvcEvent` as an argument.
4. In the `onDispatch()` method, use the `MvcEvent` to acquire an instance of the view model. The view model is typically what is returned by a controller action, and is used as part of the final view rendering process. The view model is explained in more detail in a later course module
5. In the `onDispatch()` method, use the `setVariable()` method of the view model (**`Zend\View\Model\ViewModel`**) to assign a value of `CATEGORY LIST` to a variable `categories`. This variable will be changed in a later lab
6. Open `Application/view/layout/layout.phtml`. This is a master template, which is used for the entire application.

My Notes

EVENT MANAGER LAB ... M3Ex1

7. Locate the line `echo $this->content;`
8. Create two `<div>` tags, one which represents the left column, the other the right column. There are a number of `spanX` styles in `public/css/bootstrap.min.css`, which can be used. `span2` is 140px, for example, and `span8` is 620px
9. Place the line `echo $this->content;` inside the right `<div>` tag
10. Add a new line `echo $this->categories;` inside the left `<div>` tag
11. Save your work
12. Open your browser and test
13. Observe and correct any errors

My Notes

Module 4

SERVICE MANAGER LAB ... M4Ex1

In this lab, you will be defining a simple type of service referred to as "service" or "registration". In future labs you will gain experience defining "invokable" and "factory" services.

A. Define a registration (or "service") type of service:

1. In Zend Studio open `Application/config/module.config.php`
2. Look for the array key `'service_manager'`. Note that it defines a factory which implements a translation class.

   NOTE: further down in the file, notice the key `controllers`. Although the syntax for defining controllers is similar to the syntax used for defining services, items identified in the `controllers` key are obtained from the `ControllerManager` rather than the `ServiceManager`. You will work extensively with the `controllers` key in future labs

3. Add a new key under `'service_manager'` called `'services'`
4. Under the new `'services'` key define a sub-key `'categories'`
5. Under the `'categories'` sub-key, define an array of categories as follows:

   - Barter
   - Beauty
   - Clothing
   - Computer
   - Entertainment
   - Free
   - Garden
   - General
   - health
   - household
   - phones
   - property
   - sporting
   - tools
   - transportation
   - wanted

   NOTE: be sure you have placed commas and parentheses correctly!

My Notes

Module 4

SERVICE MANAGER LAB … M4Ex1

B. Retrieve the service:

1. Open `Application/Module.php`
2. Add a line after the line in which you obtain an instance of the view model
3. From the MvcEvent instance `$e`, use `getApplication()` to obtain an instance of the `application` object
4. From the application object, use `getServiceManager()` to obtain an instance of the service manager
5. From the service manager, use the `get()` method to retrieve the `categories` service
6. Modify the command created in the Event Manager lab so that you assign the array of categories, obtained from the service manager, in place of `'CATEGORY LIST'`

C. Display the results in the layout:

7. Open `Application/view/layout/layout.phtml`
8. Locate the line from the Event Manager lab where you echoed the contents of the variable representing categories
9. Implement a `foreach()` loop to display the categories wrapped in `<ul>` and `<li>` tags

   NOTE: there is a `view helper` called `htmlList()` that could also be used

   NOTE: in later labs this will be modified so that each category will become a link to a separate page
10. Save your work
11. Open your browser and test … observe and correct any errors

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

In this lab, you will be working with modules. In the first lab, you will create a new module, "**Market",** based on the Zend Skeleton Module. In the second lab, you will be integrating an existing module – **Search** - into the **onlinemarket.work** application.

A. Build a new module "Market"

1. Download the **ZendSkeletonModule** from github

   https://github.com/zendframework/ZendSkeletonModule/archive/master.zip

2. Using your filesystem tools, open the **ZIP** file, and extract the **ZendSkeletonModule** master folder into the

   path/to/workspace/onlinemarket.work/module directory structure

3. From Zend Studio, select the **onlinemarket.work** project and click on **File | Refresh**… wait for the workspace to be rebuilt

4. You will now need to rename several folders in the skeleton module as follows:

FROM	TO
ZendSkeletonModule-master	**Market**
src/ZendSkeletonModule	**src/Market**
tests/ZendSkeletonModule	**tests/Market**
view/zend-skeleton-module/skeleton	**view/market/index**

NOTE:

If using Zend Studio to do this, you do not have to checkmark **update references**…

If *not*, make sure you select **File | Refresh** inside Zend Studio when done!

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

5. Using Zend Studio, you can rename all textual occurrences of **ZendSkeletonModule** to **Market**:

   - Open the **PHP Explorer** pane and select the newly renamed folders
   - Select **Search | Search**
   - In the **File Search** tab, enter the following info:

PROMPT	ANSWER
Containing Text	**ZendSkeletonModule**
Case Sensitive	*make sure this box is checked*
Filename Patterns	*****
Scope	**Selected Resources**

:

   - Click **Replace**
   - In the **Replace** dialog, enter the following info:

REPLACE	WITH
ZendSkeletonModule	**Market**

   - Click **OK**... When you see "**- 0 - matches ...**" to the right, you're done

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

6. Change the controller name and references from **SkeletonController** to **IndexController**:

   - Change the name of the file

     `Market/src/Market/Controller/SkeletonController.php` to

     `Market/src/Market/Controller/IndexController.php`

   - Open the file `Market/src/Market/Controller/IndexController.php`

   - Change the class from `SkeletonController` to `IndexController`

7. Change module configuration file to reflect the name changes:

FROM	TO
`'Market\Controller\Index' => 'Market\Controller\IndexController'`	`'market-index-controller' => 'Market\Controller\IndexController'`

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

- Open the file `Market/config/module.config.php`
- Under the `"controllers" => "invokables"` key, change the reference:

  NOTE: there's nothing wrong with the current reference. Using a key such as `market-index-controller` rather than `Market\Controller\Index` will avoid confusion between a key and an actual class
- Make changes to the array key `"router" => "routes"` as follows:

  NOTE: routing & controllers are covered in more detail later in the course

FROM	TO
`module-name-here`	`market`
`module-specific-root`	`/market`
`'__NAMESPACE__' => 'Market\Controller',`	`Remove or comment out this line`
`'controller' => 'Index',`	`'controller' => 'market-index-controller',`

8. Activate the module:

- From Zend Studio, open the file `path/to/workspace/onlinemarket.work/config/application.config.php`
- Under the array key `modules`, add a new entry `Market`

  NOTE: be sure to place commas correctly!

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

9. Regenerate the `autoload_classmap.php` file:

   - `autoload_classmap.php` is used in conjunction with the ZF2 classmap autoloader… It's a highly efficient alternative to the standard autoloader
   - When you define a new module, and create new class files to be used with the module, it is important to update this file
   - ZF2 provides a tool, `classmap_generator.php`, that can be used to scan a module directory structure and update the `autoload_classmap.php` file; this tool is in the **bin** folder of your ZF2 distribution

     Note: the **bin** folder was either unzipped or linked to the following location in the first lab: **path/to/workspace/onlinemarket.work/vendor/ZF2/bin**
   - Open a terminal window on your system to access the command prompt
   - Change to the **path/to/workspace/onlinemarket.work/module/Market** folder
   - Run `classmap_generator.php` by issuing this command:

     ```
     php path/to/workspace/onlinemarket.work/vendor/ZF2/bin/
     classmap_generator.php
     ```

10. Save your work

11. Open your browser and test … observe and correct any errors

    http://onlinemarket.work/

    http://onlinemarket.work/market

    http://onlinemarket.work/market/imarket-index-controller/index

    http://onlinemarket.work/market/imarket-index-controller/foo

    NOTE: you can also use zftool.phar by issuing the command

    `php zftool.phar autoload_classmap.php`

My Notes

Module 5

MVC AND MODULES LAB ...M5Ex1

B. Add an existing module "Search"

1. Using your filesystem tools, open the ZIP file
   `path/to/workspace/onlinemarket.start/files/search_module.zip`
2. Under **path/to/onlinemarket.work/module** create a new folder **Search**
3. Unzip the search module files into the new folder **Search**
4. Using Zend Studio, select the **onlinemarket.work** project and click on **File | Refresh** … wait for the workspace to be rebuilt
5. Open the file
   `path/to/workspace/onlinemarket.work/config/application.config.php`
6. Under the array key `modules` add a new entry, `'Search'`

   NOTE: be sure to place commas correctly!
7. Save your work
8. Open your browser and test … observe and correct any errors

   http://onlinemarket.work/

   http://onlinemarket.work/search/test

   NOTE: none of the other search module actions will work until the database lab has been completed!

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

In this lab, you will be creating two MVC controllers. First, you will create a new controller, **ViewController**, which will eventually be used to display items from the database. The controller will be initially configured as an "invokable" class. Next, you will create a new controller, **PostController**, which will use a factory to produce the controller. The post controller will be reconfigured in a later lab to create a new posting to the Online Market.

You will also configure "actions" as part of designing the new controllers. Actions are class methods that can be dispatched by specific user browser requests. The ZF2 MVC system will perform a process referred to as "routing", during which the user request is broken down and assigned to specific modules, controllers and actions. (Routing is discussed in more detail in the next course module. You will incorporate three controller plugins into the new controllers for that module's exercise.)

---

1. Build a new "invokable" controller

1. Create the controller class
    - Using Zend Studio, under the `onlinemarket.work` project, select `module/Market/src/Market/Controller`
    - Click **File | New | PHP File**; name the file `ViewController.php`
    - In the new file, specify a class name of `ViewController`; you should extend `Zend\Mvc\Controller\AbstractActionController`
    - Make sure that you "`use`" the appropriate classes and indicate the appropriate namespace
2. Create an index `action`
    - Using Zend Studio, open `module/Market/src/Market/Controller/ViewController.php`
    - Create a public method `indexAction()`
    - For now, merely have this method return an instance of `Zend\View\Model\ViewModel`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

- In the constructor for the `ViewModel`, supply an array as an argument
- For a key, use `listings` with a value of `LIST OF ITEMS`

  In a later lab, you will modify this to pull online market listings from a database table
- Make sure that you `use` the appropriate class

3. Create the view template
   - Create a new folder `Market/view/market/view`
   - Copy the file `Market/view/market/index/index.phtml` to `Market/view/market/view/index.phtml`
   - Modify the newly copied file to echo the value of `$this->listings` (which is passed from the view model)
4. Add the new controller to the `"controllers" => "invokables"` key
   - Open the file `Market/config/module.config.php`
   - Under the `"controllers"`, key, create a sub-key `"invokables"` if it does not already exist
   - Under the `"invokables"` sub-key, add a new reference to the new controller
   - Call the new key `"market-view-controller"` and have it reference the new `ViewController` class
5. Open a terminal window, and regenerate the `autoload_classmap.php` file using `zftool.phar`:

   ```
   php path/to/workspace/onlinemarket.start/files/zftool.phar classmap generate path/to/workspace/onlinemarket/module/Market -a
   ```

   NOTE: you can specify a filename as well, but the default `autoload_classmap.php` is fine

   NOTE: use the `-a` option to append to an existing file, use `-w` to overwrite

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB …M6Ex1

6. Using Zend Studio, select **File | Refresh** and then open `module/Market/autoload_classmap.php` and make sure the new controller has been identified; if not, add an appropriate entry

7. Save your work

8. Open your browser and test … observe and correct any errors

   `http://onlinemarket.work/`

   NOTE: it's a good idea to test the main page to make sure nothing is broken!

   `http://onlinemarket.work/market`

   `http://onlinemarket.work/market/market-view-controller`

   `http://onlinemarket.work/market/market-view-controller/index`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

2. Use a factory to generate the controller

The reason why you would use a factory (rather than invokable) is to be able to perform additional processing for the new object. In this case, you will be using "setter injection" to inject the list of categories defined as a service into the new controller

1. Create the controller class
    - Using Zend Studio, under the `onlinemarket.work` project, locate `module/Market/src/Market/Controller`
    - Click on **File | New | PHP File**, with `PostController.php` as the filename
    - In the new file, specify a class name of `PostController`; you should extend `Zend\Mvc\Controller\AbstractActionController`
    - Make sure that you `use` the appropriate classes and indicate the appropriate namespace
2. Define a `setter` method, which will be called by the factory:
    - Using Zend Studio, continue to edit `module/Market/src/Market/Controller/PostController.php`
    - Create a public property `categories`
    - Create a public method `setCategories()`, which accepts an array of categories as a parameter
    - Have this method assign the categories parameter to the categories property
3. Create an index `action`
    - Using Zend Studio, open `module/Market/src/Market/Controller/PostController.php`
    - Create a public method `indexAction()`
    - For now, have this method return an instance of `Zend\View\Model\ViewModel`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

- In the constructor for the view model, pass in an array with a key "categories", which is assigned a value of the public property $this->categories
- Make sure that you "use" the appropriate class

4. Create a factory which will be used to generate the new controller

- Using Zend Studio, under the onlinemarket.work project, select module/Market/src/Market
- Click on **File | New | Folder**, naming the folder Factory
- Again, under the onlinemarket.work project, select module/Market/src/Market/Factory
- Click on **File | New | Folder**, naming the file PostControllerFactory.php
- From Zend Studio, open module/Market/src/Market/Factory/PostControllerFactory.php
- Make sure the factory class PostControllerFactory implements Zend\ServiceManager\FactoryInterface
- Create a method createService(), which accepts an object of type Zend\ServiceManager\ServiceLocatorInterface as a parameter
- Retrieve an instance of the service manager

  NOTE: the ServiceManager is not directly provided as a parameter. Instead, the factory will be sent an instance of the ControllerManager. This is by design and is for security reasons (otherwise any user could invoke any service by launching a specific url!)

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

- In order to retrieve the ServiceManager from the ControllerManager, proceed as follows inside the `createService()` method

  For the sake of this step, assume that you have defined the `createService(ServiceLocatorInterface $controllerManager)`:

  - Use the `getServiceLocator()` method of `$controllerManager` to retrieve service locator for the controller factory (in this case a plugin manager)
  - From the service locator instance, use the call `get('ServiceManager')` to retrieve an instance of the service manager
  - From the service manager, retrieve the list of categories defined as the `categories` service in an earlier lab
  - Create an instance of the new post controller
- Create an instance of the post controller
- Call the post controller's `setCategories()` method and assign the values retrieved from the `"categories"` service
- Return the controller instance
- Make sure that you "`use`" the appropriate class

5. Create the view template

- Create a new folder `Market/view/market/post`
- Copy the file `Market/view/market/view/index.phtml` to `Market/view/market/post/index.phtml`
- Modify the newly copied file as appropriate
- For now, have the view script perform `Zend\Debug\Debug::dump()` on `$this->categories`; later this will be changed

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

6. Add the new controller to the `"controllers" => "factories"` key
    - Open the file `Market/config/module.config.php`
    - Under the `controllers` key, create a sub-key `"factories"` if it does not already exist
    - Under the `factories` sub-key add a new reference to the new controller factory
    - Call the new key `market-post-controller` and have it reference the new `PostControllerFactory` class
7. Open a terminal window, and regenerate the `autoload_classmap.php` file using `zftool.phar`
8. In Zend Studio, select **File | Refresh** and open `module/Market/autoload_classmap.php`; make sure the new controller has been identified; if not, add an appropriate entry
9. Save your work
10. Open your browser and test … observe and correct any errors

    `http://onlinemarket.work/`

    `http://onlinemarket.work/market`

    `http://onlinemarket.work/market/market-post-controller`

    `http://onlinemarket.work/market/market-post-controller/index`

    NOTE: none of the other search module actions will work until the database lab has been completed!

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB …M6Ex1

3. Utilize Controller Plugins

Using the params() plugin: The `params()` plugin is used to obtain inbound request information from `$_GET` or `$_POST`

1. Using Zend Studio, open `module/Market/src/Market/Controller/ViewController.php`
2. Create a new action `itemAction()`
3. Inside this method, capture a parameter `category` using the `params()` plugin method `fromQuery()`

   NOTE:

   `params()->fromQuery()` is used to obtain information found in `$_GET`

   `params()->fromPost()` is used to obtain information found in `$_POST`
4. Add this parameter to the array being passed to the view model
5. Create a new view template `Market/view/market/view/item.phtml`
6. Echo the value of the category parameter captured from the request
7. Save your work
8. Open your browser and test … observe and correct any errors

   `http://onlinemarket.work/market/market-view-controller/item?category=ABC`

   `http://onlinemarket.work/market/market-view-controller/item?category=XYZ`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB …M6Ex1

Using the redirect() plugin: The `redirect()` plugin is used to redirect your users from the current web page to another.

Important Note: you should prefix any use of the `redirect()` plugin with `return`, as it effectively "short circuits" the normal handoff to the view rendering process

1. Using Zend Studio, open `module/Market/src/Market/Controller/ViewController.php`

2. Inside `itemAction()`, add an `if ()` clause that checks to see if the category parameter is empty; if so, use the `redirect()` plugin to redirect to the `onlinemarket.work` main page

3. From the `redirect()` plugin, you can use either the following methods to specify the target of redirection:

METHOD	PURPOSE
`toUrl()`	Lets you specify a URL as a target
`toRoute()`	Lets you specify a route named in the `"router" => "routes"` configuration. This is explained in more detail later in the course

4. Save your work

5. Open your browser and test … observe and correct any errors

   `http://onlinemarket.work/market/market-view-controller/item?category=TEST`

   `http://onlinemarket.work/market/market-view-controller/item`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

Using the flashMessenger() plugin: The `flashMessenger()` plugin is used to store messages now, to be retrieved in subsequent page requests

1. Using Zend Studio, open `module/Market/src/Market/Controller/ViewController.php`
2. Inside `itemAction()`, locate the `if()` clause added above
3. If the category parameter is empty, use the `flashMessenger()` plugin to add a message "`Item Not Found`"
4. Open `module/Market/src/Market/Controller/IndexController.php`
5. Inside `indexAction()` check to see if the `flashMessenger()` plugin has any messages
6. If there are messages, send the messages to the view template via the view model
7. Open the view template `Market/view/market/index/index.phtml`
8. If there are any messages, echo them

   NOTE: flashMessenger messages are in the form of an array.
9. Save your work
10. Open your browser and test … observe and correct any errors

    `http://onlinemarket.work/market/market-view-controller/item?category=TEST`

    `http://onlinemarket.work/market/market-view-controller/item`

My Notes

Module 6

CONTROLLERS AND PLUGINS LAB ...M6Ex1

4. Creating an Alias

As you have noticed, having to type in lengthy controller name keys in the URL is not very "friendly". One technique that can be introduced at this point is to create an "alias". Another, even better, technique would be to define a proper route. Routing is covered in a later lab

1. In Zend Studio, open the file `Market/config/module.config.php`
2. Locate the `"controllers"` key
3. Add a new key

   ```
   'controllers' => 'aliases' => 'alt' => 'market-view-controller',
   ```

   NOTE: be sure to use the correct number of parentheses and commas!
4. Save your work
5. In the browser test these URLs:

   `http://onlinemarket.work/market/alt`

   `http://onlinemarket.work/market/alt/index`

   `http://onlinemarket.work/market/alt/item?category=TEST`

My Notes

Module 7

ROUTING LAB ...M7Ex1

In this lab, you will be creating routing configurations for the controllers and actions thus far built. As you have noticed, it is not very "user friendly" to ask a website visitor to type something like this:

http://onlinemarket.work/market/market-view-controller/item?category=XYZ

Accordingly, you will be building "SEO friendly" routes using ZF2's MVC routing capabilities.

In the lecture for this course module, you learned there are several types of routes which can be configured. For the purposes of this lab we will be focusing on "literal" and "segment" routes.

A. Create a home route

A `home` route would map to "`/`". This route is also referred to as the "home page"

Before you begin this part of the lab, verify that when you enter the base url - http://onlinemarket.work/ - you see the "splash" screen that was originally part of the Zend Skeleton App, "Welcome to Zend Framework 2"

1. Using Zend Studio, open the file `Market/config/module.config.php`

2. Immediately under the `"router" => "routes"` key, create a sub-key `home`

   - `home` will contain an array, that itself contains two keys: `type` and `options`
     - Set `type` to `literal`
   - `options` will contain an array with two further keys: `route` and `defaults`
     - Set `route` to `/`
   - `defaults` will contain array with two keys: `controller` and `action`
     - Set `controller` to the key that identifies the controller key (should be `market-index-controller`)
     - Set `action` to `index`

My Notes

Module 7

ROUTING LAB ...M7Ex1

To summarize the parameters for the `home` route:

1. Open the **PHP Explorer** pane and select the newly renamed folders
2. Select **Search | Search**
3. In the **File Search** tab, enter the following info:

KEY	SETTING
`home => type`	`literal`
`home => options => route`	`/`
`home => options => defaults => controller`	`market-index-controller`
`home => options => defaults => action`	`index`

3. Save your work
4. Open your browser and test

   http://onlinemarket.work/

   NOTE: now your application defaults to the `Market module`, `Index controller`, `index action`

5. Observe and correct any errors

   NOTE: make sure you have the correct number and placement of parentheses "`()`" and commas "`,`"!

My Notes

Module 7

ROUTING LAB ...M7Ex1

B. Create a route for the view controller

It is still recommended that you include the word "`market`" in the request URL in order to distinguish requests for the `Market` module. However ,it is obvious that asking a visitor to type "`/market/market-view-controller`" is asking too much. Accordingly, you will now craft a shorter request route for the view controller.

1. Using Zend Studio, open the file `Market/config/module.config.php`
2. Immediately under new key `home` and before the `market` key, add a new key `market-view` with these options:

KEY	SETTING
`type`	`literal`
`options => route`	`/market/view`
`options => defaults => controller`	`market-view-controller`
`options => defaults => action`	`index`

3. Save your work
4. Open your browser and test

   http://onlinemarket.work/market/view

My Notes

Module 7

ROUTING LAB ...M7Ex1

5. You should receive an error at this point! Why? The current default `market` route is interfering with specific `market` routes. Accordingly, you will need to disable the current `market` route:
   - Locate the key that starts with "`market`"
   - Place a PHP open multi-line comment "`/*`" in front of this key
   - Locate the last closing parenthesis after "`child_routes`"
   - Close the multi-line comment by placing "`*/`" after this parenthesis
6. Insert a new, re-stated `market` key above the `market-view` just created, with these options:

KEY	SETTING
`type`	`literal`
`options => route`	`/market`
`options => defaults => controller`	`market-index-controller`
`options => defaults => action`	`index`

7. Save your work
8. Re-test the new `market/view` route ... observe and correct any errors

   `http://onlinemarket.work/market`

   `http://onlinemarket.work/market/view`

My Notes

Module 7

ROUTING LAB ...M7Ex1

C. Create a route for the post controller

Follow the steps listed above and create a new route, `/market/post`, for the post controller, with the following parameters:

KEY	SETTING
`type`	`literal`
`options => route`	`/market`
`options => defaults => controller`	`market-post-controller`
`options => defaults => action`	`index`

D. Create a route to capture parameters

In the previous lab, you configured the view controller to accept a parameter representing the category. You can configure a flexible route that can be mapped to not only actions and controllers, but also parameters. This is accomplished with the "`segment`" route.

When configuring a `segment` route, you can define optional routing entities by using square brackets, a colon, and a label (ex: `[:label]`). Furthermore, literal routes can be combined with segment routes using the `child_routes` sub-key

1. Using Zend Studio, open the file `Market/config/module.config.php`
2. Inside the `market-view` key, after the closing parenthesis of `options`, add a new key `'may_terminate' => true,'`

   The purpose for this option is to allow the route to be specified as-is, with no options

My Notes

Module 7

ROUTING LAB ...M7Ex1

3. After the `may_terminate` key, add another key `child_routes` with defaults for `type` and `route` options
4. To summarize the new options:

KEY	SETTING
`may_terminate`	`true`
`child_routes => default => type`	`segment`
`child_routes => default => options => route`	`/[:action][/:category]`

5. Save your work
6. Open your browser and test

```
http://onlinemarket.work/market/view
http://onlinemarket.work/market/view/
http://onlinemarket.work/market/view/index
http://onlinemarket.work/market/view/item
http://onlinemarket.work/market/view/index/TEST
http://onlinemarket.work/market/view/item/TEST
```

You will notice that the last item, "http://onlinemarket.work/market/view/item/TEST", does not work. The reason for this has to do with how the controller retrieves the parameter. As you will recall, the view controller is using the `params()` plugin and the `fromQuery()` method. The `fromQuery()` method looks for its information from `$_GET`, whereas the parameter is now being filtered through the routing process.

My Notes

Module 7

ROUTING LAB ...M7Ex1

7. Using Zend Studio, open the file
   `Market/src/Market/Controller/ViewController.php`
8. In both `indexAction()` and `itemAction()`, switch from "`fromQuery('category')`
   to `fromRoute('category')`
9. [OPTIONAL] Add a child route to the `market-post` route to account for a trailing "`/`"
10. Save your work
11. From the browser re-run the test … observe and correct any errors
    `http://onlinemarket.work/market/view`
    `http://onlinemarket.work/market/view/`
    `http://onlinemarket.work/market/view/index`
    `http://onlinemarket.work/market/view/item`
    `http://onlinemarket.work/market/view/index/TEST`
    `http://onlinemarket.work/market/view/item/TEST`

    You should now see that value of "`TEST`" is being routed to "`category`"

My Notes

Module 8

VIEW LAYER LAB … M8Ex1

In this lab, you will be working with the view aspects of the MVC design pattern. You will be taking a closer look at view models, using built-in view helpers as well as creating a custom one.

A. Returning a view model:

The Zend Skeleton App implements the MVC design pattern. The default behavior for an MVC-based ZF2 app is to invoke the PhpRenderer whenever an MVC controller returns either an array or a view model. The `onlinemarket.work` application was configured in an earlier lab, through route manipulation, to invoke the `Market` module, `Index` controller, and `index` action.

1. In Zend Studio, open the file
   `Market/src/Market/Controller/IndexController.php`
2. Notice that it currently returns an array as follows:
   `return array('messages' => $messages);`
3. Inside `indexAction()` modify the logic so that the intial value for `$messages` is
   `array('Welcome to the Online Market')`
4. Open your browser and test: `http://onlinemarket.work/`
5. Note the current behavior
6. Copy the return array `array('messages' => $messages)` inside the parentheses when you instantiate the view model (pass this array to the class constructor)
7. Modify the `return` statement to return the view model instead of an array
8. Save your work
9. Open your browser and test … observe and correct any errors
   `http://onlinemarket.work/`
   The behavior of the application should be the same as before. Why? Because when you return an array, the default behavior is to automatically create a ViewModel instance, and to pass the array into the class constructor

My Notes

Module 8

VIEW LAYER MANAGER LAB … M8Ex1

B. Working with a view model:

There are advantage in working directly with the view model in the controller. First, it makes ZF2 "work" less hard in that the framework does not have to create a view model - it uses the one you return. Secondly, and more importantly, you can create "child" view models and attach them to a parent, creating an elaborate layering effect. For example, you could create "ad blocks" as child view models, and attach them to the view model being returned. Another practical use for a view model is the ability to assign a view template (also referred to as a view script). In this lab, you will be working with the post controller to establish an alternate view template for the "`index`" action in case a form posting is invalid. **NOTE**: as you have not yet implemented a form, the alternate template will be directly assigned to the view model. In a later lab, it will be assigned in an "`if`" construct to test the post form for validity.

1. Open your browser and test this URL: `http://onlinemarket.work/market/post`
2. Note the current behavior
3. In Zend Studio, copy the file `index.pthml` under `Market/view/market/post/` to `invalid.pthml`
4. Modify `invalid.pthml` by adding a header at the top indicating that one or more form inputs are invalid
5. In Zend Studio, open the file `Market/src/Market/Controller/PostController.php`
6. Assign the new view template to the view model returned by the `index` action

   NOTE: specify the path to the new template as follows: `market/post/invalid.phtml`
7. Save your work
8. From the browser, re-test the URL … observe and correct any errors

   `http://onlinemarket.work/market/post`

My Notes

Module 8

VIEW LAYER MANAGER LAB ... M8Ex1

C. Using built-in view helpers:

View helpers encapsulate view-related logic. Normally, in a view template or script, you want to minimize the amount of PHP code. If you find yourself implementing elaborate loops or other control structures, it might be time to use a built-in view helper, or create one of your own.

1. In Zend Studio, open the file: `Appliction/view/layout/layout.pthml`
2. Make a list of built-in view helpers used in `layout.phtml` (supplied initially in the Zend Skeleton App)
3. Be prepared to share your list with the class
4. Open your browser and enter this URL:

   `http://onlinemarket.work/market/view/item/TEST`
5. You should see the word "TEST" appear near the center of the screen
6. Again using your browser, try the following URL:

   ```
   http://onlinemarket.work/market/view/item/<span
   style="color:red;font-weight:bold;font-size:18px;">TEST
   ```

   Notice that everything is from the word "**TEST**" is now bold, red, and in a large font!
7. In Zend Studio, open the file: `Market/view/market/view/item.pthml`
8. Wrap the echo of `$this->categoryParam` using `escapeHtml()`
9. Save your work
10. From the browser, try the test URL again:

    ```
    http://onlinemarket.work/market/view/item/<span
    style="color:red;font-weight:bold;font-size:18px;">TEST
    ```

    Notice that the HTML coming from the URL is simply displayed but not rendered

My Notes

Module 8

VIEW LAYER MANAGER LAB ... M8Ex1

D. Create a custom view helper:

The objective of this part of the lab is to create a custom view helper that iterates through the array of categories (provided in an earlier lab), and presents a link that, when clicked, passes the category as a parameter to the `Market` module, `View` controller, and `index` action.

NOTE: in a later lab you will be configuring the `index` action to display a list of online market items for this category.

Create the View Helper

1. In Zend Studio, select the file: `onlinemarket.work/module/Application/src/Application` folder
2. Create a new folder named "`Helper`"
3. Select this new folder and then from the Studio menu select: **File | New | PHP File**, naming the file `LeftLinks.php`
4. The new class will have the following characteristics:

namespace	`Application\Helper`
classname	`LeftLinks`
extends	`Zend\View\Helper\AbstractHelper`
methods	`render($values, $urlPrefix)` `__invoke($values, $urlPrefix)`

My Notes

Module 8

VIEW LAYER MANAGER LAB ... M8Ex1

5. Define the `render()` method as follows:
   - Displays the array `$values` as a series of `<ul><li>` tags
   - Each `<li>` tag will include an `<a href="">` tag which links to `$urlPrefix/CCC`, where "`CCC`" corresponds to a different category
   - The return value should be a string containing HTML
6. The `__invoke()` method should return the output of `render()`, passing it the same parameters `$values` and `$urlPrefix`

Register the Custom View Helper

1. Open the file `Application/config/module.config.php`
2. If there is no primary key `view_helpers`, create it
3. Under the `view_helpers` key, add a sub-key `invokables`
4. Under `invokables`, add a reference as follows:

```
'leftLinks' => 'Application\Helper\LeftLinks',
```

5. In Zend Studio, open the file: `Application/view/layout/layout.pthml`
6. Locate the code in the `container` class `div` tag that displays the contents of `$this->categories`
7. Change the echo (or `htmlList()` call) to use the new view helper

   NOTE: when you supply the url prefix of `/market/view/index`, consider using the `basePath()` view helper as well

My Notes

Module 8

VIEW LAYER MANAGER LAB … M8Ex1

Show the Category in the View

1. In Zend Studio, open the file:
   `Market/src/Market/Controller/ViewController.php`
2. In `indexAction()`, modify the parameter sent to the view model to reflect the category parameter (retrieved from routing)
3. Open the file: `Market/view/market/view/index.pthml`
4. Modify the file to include a header which echos the upper case value of the category parameter

   NOTE: don't forget to properly escape the value!
5. Save your work
6. From the browser, test the category links for the URL … observe and correct any errors

   `http://onlinemarket.work/`

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

In the first part of this lab, you will be working with forms and filters. We will concentrate on the process of posting an item to the Online Market. In the database lab (the next course module lab), you will "complete the circuit" and insert validated form data into the online market database.

In the second part, you will design a form filter that allows you to filter and validate items to be posted.

In the third part, you will inject the form and filter into the post controller by way of the post controller factory.

Here is the structure of the database table listings, which will eventually house the post data:

```
CREATE TABLE `listings` (

  'listings_id' int(10) unsigned NOT NULL AUTO_INCREMENT,
  'category' char(16) NOT NULL,
  'title' varchar(128) NOT NULL,
  'date_created' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  'date_expires' timestamp NULL DEFAULT NULL,
  'description' varchar(4096) DEFAULT NULL,
  'photo_filename' varchar(1024) DEFAULT NULL,
  'contact_name' varchar(255) DEFAULT NULL,
  'contact_email' varchar(255) DEFAULT NULL,
  'contact_phone' varchar(32) DEFAULT NULL,
  'city' varchar(128) DEFAULT NULL,
  'country' char(2) NOT NULL,
  'price' decimal(12,2) NOT NULL,
  'delete_code' char(16) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT
NULL,
  PRIMARY KEY ('listings_id'),
  KEY 'title' ('title'),
  KEY 'category' ('category'),
  KEY 'delete_code' ('delete_code')
) ENGINE=InnoDB AUTO_INCREMENT=46 DEFAULT CHARSET=utf8;
```

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

A. Forms Lab:

1. In Zend Studio, select the folder:
   `onlinemarket.work/module/Market/src/Market`

2. Create a new folder named "`Form`"

3. Select this new folder and then from the Studio menu select: **File | New | PHP File**, naming the file `PostForm.php`

4. The new class will have the following characteristics:

namespace	`Market\Form`
classname	`PostForm`
extends	`Zend\Form\Form`
method	`prepareElements()`

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

5. In the `prepareElements()` method, you will need to define elements that correspond with the database table `listings` (shown above). Here are some suggestions for the various elements that need to be created:

Element Name	Zend\Form\Element*	Notes
`listings_id`	n/a	Do not capture this in the form: use the database auto-increment mechanism
`category`	`Select`	Use the categories `service`
`title`	`Text`	
`price`	`Number`	
`date_created`	n/a	Do not create: let the database do the work
`date_expires`	`Radio`	Present the user with a series of options which represents days. Create a static array property for this purpose. Do the date calculations later in the controller.
`description`	`Textarea`	
`photo_filename`	`Url`	
`contact_name`	`Text`	
`contact_email`	`Email`	
`contact_phone`	`Text`	

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

Element Name	Zend\Form\Element*	Notes
cityCode	Select	Use setOptions() to set an array key, options, assigned to a temporary static array of a few cities and countries. For example: array('London,UK', 'Paris,FR', 'New York,US', 'Sydney,AU', 'Toronto,CA') In a later lab you can retrieve the list of city + country codes from the database table world_city_area_codes
country	n/a	Don't capture this: it can be derived from the city data
delete_code	Number	
captcha	Captcha	Set up a "dumb" CAPTCHA to help block SPAM
submit	Submit	Submit button

6. NOTE: don't forget to use the setLabel() method on form elements so that they display a label when rendered. You might also want to consider using the setAttribute() method on form elements to set various HTML attributes such as size, title, maxlength, etc.

   NOTE: don't forget to add() each element to the form!

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

Set up the form using a factory service

1. In Zend Studio, select the `Market/src/Market/Factory` folder
2. Copy `PostControllerFactory.php` to `PostFormFactory.php`
3. Change the reference for the parameter from `$controllerManager` to `$serviceManager`
4. NOTE: this is not mandatory, but helps you to recall that the entity being passed in this case will be an instance of ServiceManager, *not* ControllerManager!
5. Remove the line that calls `getServiceLocator()` and the one that calls `get('ServiceManager')`
6. Change the reference from `$allServices` to the incoming parameter `$serviceManager`
7. Create a form instance
8. Extract the `'categories'` service
9. Reformat the array from numeric to associative where the key = value
10. Call `prepareElements()` and pass associative array as a parameter
11. Have the factory return the form instance
12. In Zend Studio, select the `Market/config/module.config.php`
13. Add a new key `'service_manager' => 'factories' => 'market-post-form' => 'Market\Factory\PostFormFactory'`
14. NOTE: be sure to "`use`" the appropriate namespaces and classes

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

Configure the post controller to use the form

1. In Zend Studio, open the file `Market/src/Market/Controller/PostController.php`
2. Add a new property `$postForm`
3. Define a method `setPostForm()` that sets this property
4. In the `index` action, assign the form a method of `POST` and an action that corresponds to this module, controller and action
5. NOTE: you could use the `url()` controller plugin, and use the `market-post` route assigned in the routing lab
6. For test purposes, also send post data to the view using the `fromPost()` method of the `params()` plugin
7. Assign this data set to the form using the `setData()` method
8. Pass the new `$postForm` property to the view model
9. In Zend Studio, open the file `Market/src/Market/Factory/PostControllerFactory.php`
10. Call `setPostForm()` from the controller object, using the new `market-post-form` factory service

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

Render the form in the view

```
<h3>Render the form in the view</h3>
```

1. In Zend Studio, open the file `Market/view/market/post/index.phtml`
2. Call the `form()` view helper's `prepare()` method
3. Open the form by calling the `openTag()` method of the `form()` view helper
4. Render each form element using the `formRow()` view helper
5. Close the form by calling the `closeTag()` method of the `form()` view helper
6. Echo any post data collected using `Zend\Debug\Debug::dump()` or `var_dump()`
7. In `PostController.php` don't forget to set the template back to `index.phtml`!
8. Modify the layout adding a header link to the new post form

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

B. Filters Lab:

1. In Zend Studio, select the folder `onlinemarket.work/module/Market/src/Market/Form`

2. Select **File | New | PHP File**, naming the file `PostFormFilter.php`

3. The new class will have the following characteristics:

namespace	`Market\Form`
classname	`PostFormFilter`
extends	`Zend\InputFilter\InputFilter`
method	`prepareFilters()`

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

4. In the `prepareFilters()` method, you will need to define input filter elements that correspond with the database table `listings` (shown above). Here are some suggestions for the various elements that need to be created:

Element Name	Zend\Validator*	Notes
`category`	`InArray, Filter: StringToLower`	The source array is supplied as an argument to `prepareFilters()`
`title`	`Alnum`, `StringLength`	
`price`	`Regex`	
`date_expires`	`Digits`	
`description`	`StringLength`	
`photo_filename`	`Regex, Not Required`	
`contact_name`	`Regex`	
`contact_email`	`EmailAddress`	
`contact_phone`	`Regex`	
`city`	`InArray`	The source array is provided from `PostForm::$cityCodes`. Remember to validate against `array_keys()`!
`Delete_code`	`Alnum`	

5. Filters, for the most part, would likely be `StripTags` and `StringTrim`

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

Set up the filter using a factory service

1. In Zend Studio, select the folder: `Market/src/Market/Factory`
2. Copy `PostFormFactory.php` to `PostFormFilterFactory.php`
3. Create a filter instance
4. Change `prepareElements()` to `prepareFilters()` and pass the categories in as a parameter
5. Have the factory return the filter instance
6. Make sure you "`use`" the appropriate classes
7. From Zend Studio, select the `Market/config/module.config.php`
8. Add a new key `'service_manager' => 'factories' => 'market-post-form-filter' = 'Market\Factory\PostFormFilterFactory'`

Configure the post controller to use the filter and validate the data

1. In Zend Studio, open the file: `Market/src/Market/Factory/PostControllerFactory.php`
2. Call `setPostFormFilter()` from the controller object, using the new `market-post-form-filter` factory service
3. In Zend Studio, open the file: `Market/src/Market/Controller/PostController.php`
4. Add a new property `$postFormFilter`
5. Define a method `setPostFormFilter()` that populates this property

My Notes

Module 9

FORMS / FILTERS/ VALIDATORS LAB ...M9Ex1

6. Modify the `index` action as follows:
    - Assign `$postFormFilter` to the form using the `setInputFilter()` method
    - Assign post data to a variable $data using the `fromPost()` method of the `params()` plugin
    - Assign `$data` to the form using its `setData()` method
    - Test to see if the submit button has been pressed
    - Check to see if the form data is valid
    - If not valid, create a new view model `$invalidViewModel`
    - Set the template for `$invalidViewModel` to `market/post/invalid.pthml`
    - Set the original view model as a child of `$invalidViewModel`
    - Assign the input filter to the form, and pass the form to the view
    - NOTE: the validation messages should automatically appear, presuming you've configured the view template properly.
    - Otherwise, send an appropriate success message using the `flashMessenger()` plugin, and redirect home
7. Test that form data validates properly

My Notes

Module 10

DATABASE LAB … M10Ex1

In this lab you will first set up a class which represents a table in the database. You will then set up the table class as a service, and inject it into all three controllers created thus far. Then you will implement a lookup to display the most recent posting on the home page. Next you will configure the application to present a list of items by category. Finallly, you will "complete the circuit" and insert validated form data into the online market database.

IMPORTANT NOTE: this is a very lengthy lab, and will reinforce concepts presented in all the previous labs. There are also a number of labs marked [OPTIONAL]. Optional labs will give you additional experience in the various concepts presented, but will not impact the course if not completed.

A. Set up a table model:

The table you will be working with initially is the `listings` table. You will create an MVC Model class which represents the table. Here is the structure of this table:

```
CREATE TABLE `listings` (
  `listings_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `category` char(16) NOT NULL,
   `title` varchar(128) NOT NULL,
   `date_created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   `date_expires` timestamp NULL DEFAULT NULL,
   `description` varchar(4096) DEFAULT NULL,
   `photo_filename` varchar(1024) DEFAULT NULL,
   `contact_name` varchar(255) DEFAULT NULL,
   `contact_email` varchar(255) DEFAULT NULL,
   `contact_phone` varchar(32) DEFAULT NULL,
   `city` varchar(128) DEFAULT NULL,
   `country` char(2) NOT NULL,
   `price` decimal(12,2) NOT NULL,
   `delete_code` char(16) CHARACTER SET utf8 COLLATE utf8_bin
DEFAULT NULL,
  PRIMARY KEY (`listings_id`),
  KEY `title` (`title`),
  KEY `category` (`category`),
  KEY `delete_code` (`delete_code`)
) ENGINE=InnoDB AUTO_INCREMENT=46 DEFAULT CHARSET=utf8;
```

My Notes

Module 10

DATABASE LAB … M10Ex1

1. In Zend Studio, select the folder:
   `onlinemarket.work/module/Market/src/Market`
2. Create a new folder, "`Model`"
3. Select this new folder and then **File | New | PHP File**, naming the file `ListingsTable.php`
4. The new class will have the following characteristics:

namespace	`Market\Market`
classname	`ListingsTable`
extends	`Zend\Db\TableGateway\TableGateway`
property	`Public static $tableName = 'listings'`

My Notes

Module 10

DATABASE LAB ... M10Ex1

B. Inject the table into the three controllers

The first thing which needs to be done is to establish a system-wide "adapter" service. You can then create a table factory, and finally modify the controller factories to perform injection.

Create a system-wide adapter service

1. In Zend Studio, open the file:

   `path/to/workspace/onlinemarket.work/config/autoload/local.php.dist`

2. Add a key `'db'` with the following options:

Key	Setting
driver	`pdo`
dsn	`mysql:dbname=onlinemarket;host=localhost`
username	`zend`
password	`password`
driver_options	`array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)`

3. This key is the default database key, and is used by the `AdapterFactory` class

My Notes

Module 10

DATABASE LAB ... M10Ex1

4. Create a new key to represent the adapter factory:
   `'service_manager' => 'factories' => 'general-adapter' => 'Zend\Db\Adapter\AdapterServiceFactory'`

5. Save the file as:
   `path/to/workspace/onlinemarket.work/config/autoload/db.local.php`

   NOTE: have a look at the `application.config.php` file key
   `'module_listener_options' => 'config_glob_paths'`

Create a listings table factory

1. In Zend Studio, select the folder:
   `onlinemarket.work/module/Market/src/Market/Factory`

2. Select **File | New | PHP File**, naming the file `ListingsTableFactory.php`

3. Make sure the class implements `Zend\ServiceManager\FactoryInterface`

4. Create a method `createService()` that accepts an object of type `Zend\ServiceManager\ServiceLocatorInterface` as a parameter

5. Create an instance of the table model, with the following two parameters:

Parameter	Source
Table Name	`ListingsTable::$tableName`
Adapter	From the service manager get the `general-adapter` service defined above

6. Return the table instance

7. Make sure that you "`use`" the appropriate classes

   Hint: copy the `PostFormFactory.php` file and make modifications to save time

My Notes

Module 10

DATABASE LAB ... M10Ex1

Create a listings table service that uses the factory

8. In Zend Studio, open the file: `Market/config/module.config.php`
9. Add a new key `'service_manager' => 'factories' => 'listings-table' => 'Market\Factory\ListingsTableFactory'`

Configure the 3 controllers for table injection

1. In Zend Studio, open the file: `Market/src/Market/Controller/IndexController.php`
2. Define a property `$listingsTable`
3. Define a method `setListingsTable()` that accepts a `ListingsTable` object as an argument, and sets the `$listingsTable` property
4. Repeat these steps for `ViewController.php` and `PostController.php`

Modify the factory for the "post" controller to inject the table

1. In Zend Studio, open the file: `Market/src/Market/Factory/PostControllerFacctory.php`
2. Add a line, before the controller is returned, which calls `setListingsTable()`
3. As an argument to `setListingsTable()`, use the table returned from the `listings-table` service

My Notes

Module 10

DATABASE LAB ... M10Ex1

Create factories for the"index" and "view" controllers based on the "post" controller factory

1. In Zend Studio, open the file:
   `Market/src/Market/Factory/PostControllerFacctory.php`
2. Search for `Post` and replace it with `Index`
3. Set the controller's `listingsTable` property from the `listings-table` service
4. Remove references to setting the form and filter
5. Save the file as `Market/src/Market/Factory/IndexControllerFactory.php`
6. Save the file (again) as
   `Market/src/Market/Factory/ViewControllerFactory.php`
7. Search for `Index` and replace it with `View`
8. Save the file again

My Notes

Module 10

DATABASE LAB ... M10Ex1

Move the definitions for "market-index-controller" and "market-view-controller"

1. In Zend Studio, open `Market/config/module.config.php`

2. Move the key from

   ```
   'controllers' => 'invokables' => 'market-index-controller' =>
   'Market\Controller\IndexController'
   ```

   to

   ```
   'controllers' => 'factories' => 'market-index-controller' =>
   'Market\Factory\IndexControllerFactory'
   ```

3. Move the key from

   ```
   'controllers' => 'invokables' => 'market-view-controller' =>
   'Market\Controller\IndexController'
   ```

   to

   ```
   'controllers' => 'factories' => 'market-view-controller' =>
   'Market\Factory\ViewControllerFactory'
   ```

4. From the browser test the following URLs ... observe and correct any errors

   ```
   http://onlinemarket.work/
   http://onlinemarket.work/market/view
   http://onlinemarket.work/market/post
   ```

   NOTE: read any errors carefully. If you see an indication that the URL "maps to an invalid controller class" there is a good chance one of your factory classes has failed to produce its target object correctly.

My Notes

Module 10

DATABASE LAB ... M10Ex1

C. Display a list of items in a category

In this part of the lab you will be working with the `view` controller and the `index` action. As you will recall, the `index` action has already been configured to accept a parameter `category`. In this lab you will use the `category` parameter to extract a list of items from the `listings` table.

1. In Zend Studio, open the file: `Market/src/Market/Model/ListingsTable.php`
2. Create a method `getListingsByCategory()`, which returns an appropriate result set from the `listings` table

   Using `Zend\Db\Sql` construct the equivalent of this SQL statement:

   ```
   SELECT * FROM `listings` WHERE `category` = ?
   ```

   - Create a `Zend\Db\Select` object
   - Use the `from()` method to indicate the table name
   - HINT: use the static `$tableName` property
   - Create a `Zend\Db\Where` object
   - Use the `Where::equalTo()` method to compare the `'category'` column with the `$category` parameter
   - Assign the `Where()` object to the `Select()` object
   - Return the result using the table's "`selectWith()`" method, with the `Select()` object as an argument

   HINT: if you are not sure the correct SQL statement is being generated, consider echoing the following, which will give you the SQL string for analysis:

   ```
   $select->getSqlString($this->getAdapter()->getPlatform());
   ```

My Notes

Module 10

DATABASE LAB … M10Ex1

3. Modify the `view` controller's `index` action to display a list of items in one category
   - From Zend Studio, open the file: `Market/src/Market/Controller/ViewController.php`
   - In `indexAction()`, call the new `getListingsByCategory()` method from the table
   - Assign the results to the view model
4. Modify the view to display a list of items by category, with links to view each item
   - From Zend Studio, open the file: `Market/view/market/view/index.phtml`
   - Build a `foreach()` loop, going through the result set produced by the database query
   - HINT: initially, consider using var_dump() to see exactly what is being passed to the view
   - HINT: consider using the `cycle()` view helper to alternate colors between rows
   - For each item, display the title, price, date created, date expires, city and country
   - Create a link for each item as `http://onlinemarket.work/market/item/:listings_id`
5. From the browser, test the links created on the left side, for example:

   `http://onlinemarket.work/market/view/index/beauty`
6. Observe and correct any errors

   NOTE: the link will work until you complete the next step!

My Notes

Module 10

DATABASE LAB ... M10Ex1

D. Display one item

In this part of the lab you will be working with the `view` controller and the `item` action.

1. Create a method `getListingById()` in the model class that returns a single table row:
   - From Zend Studio, open the file `Market/src/Market/Model/ListingsTable.php`
   - Create a method `getListingById()` that accepts an integer `$id` as a parameter
   - Add logic which implements this SQL statement:

     ```
     SELECT * FROM `listings` WHERE `listings_id` = ? LIMIT 1
     ```

   - Create a `Zend\Db\Select` object
   - Use the `from()` method to indicate the table name
   - Create a `Zend\Db\Where` object
   - Use the `Where::equalTo()` method to compare the `'listings_id'` column with the `$id` parameter
   - Assign the `Where()` object to the `Select()` object
   - Use the `limit()` method to indicate only 1 row
   - Return the result using the table's `selectWith()` method, with the `Select()` object as an argument

     NOTE: consider returning the result using the `current()` method of the object returned by `selectWith()`

My Notes

Module 10

DATABASE LAB ... M10Ex1

2. Modify the `item` action of the `view` controller to accet a `listings ID` and display one item:
    - From Zend Studio, open the file `Market/src/Market/Controller/ViewController.php`
    - As you will recall, the `item` action has already been configured to accept a parameter `category`. You will need to modify this to accept a parameter `id` that you will be configuring using a new route in a later step
    - In `itemAction()`, call the new `getListingById()` method from the table, passing the `id` parameter
    - Assign the results to the view model
3. Create a new route to view an item:
    - From Zend Studio, open the file: `Market/config/module.config.php`
    - Create a new key `market-item` based on the existing route key `market-view` making these changes:

From	To
`'route' => '/market/view'`	`'route' => '/market/item'`
`'action' => 'index',`	`'action' => 'item',`
Under `'child_routes': 'route' => '/[:action][/][:category]',`	`'route' => '/[:id]',`

My Notes

Module 10

DATABASE LAB … M10Ex1

4. Modify the view template to display one item:
    - From Zend Studio, open the file: `Market/view/market/view/item.phtml`
    - Consult the `listings` table database structure above, and display all possibe information about the item (except for the delete code!)
    - If the value of `photo_filename` does not contain `http://` you may want to use the `basePath()` view helper to supply the initial part of the URL. You can then display the actual photo using an HTML `<img>` tag
5. Copy `onlinemarket.start/public/images` to `onlinemarket.work/public/images`

E. Display most recent posting

In this part of the lab you will be working with the `index` controller and the `inde` action. You can use the logic created for displaying one item to accomplish a similar result for the most recent listing.

1. Create a method `getMostRecentListing()` in the model class that returns a single table row:
    - From Zend Studio, open the file `Market/src/Market/Model/ListingsTable.php`
    - Create a method `getMostRecentListing()` that accepts an integer `$id` as a parameter
    - Add logic which implements this SQL statement:

      ```
      SELECT * FROM `listings` WHERE `listings_id` IN (SELECT
      MAX(`listings_id`) FROM `listings`)
      ```

        - Create a `Zend\Db\Select` object
        - Use the `from()` method to indicate the table name
        - Create a `Zend\Db\Sql\Expression` object with `MAX(`listings_id`)` as a constructor argument

My Notes

Module 10

DATABASE LAB ... M10Ex1

- Create a second `Zend\Db\Select` object, to be used as a sub-select
- In the sub-select, add a `from()` clause identify the (same) table, and a `columns()` clause using the Expression created above
- Create a `Zend\Db\Where` object
- Use the `in()` method to compare the `'listings_id'` column with the sub-select object
- Assign the `Where()` object to the `Select()` object
- Return the result using the table's `selectWith()` method, with the `Select()` object as an argument.
- NOTE: consider returning the result using the `current()` method of the object returned by `selectWith()`

2. Modify the `index` action of the `index` controller to accept a listings ID and display one item:
   - From Zend Studio, open the file `Market/src/Market/Controller/IndexController.php`
   - In `indexAction()`, call the new `getMostRecentListing()` method from the table
   - Assign the results to the view model
3. Modify the view template to display one item:
   - From Zend Studio, open the file `Market/view/market/index/index.phtml`
   - Copy the logic from `Market/view/market/view/item.phtml` to display the most recent posting

My Notes

Module 10

DATABASE LAB ... M10Ex1

F. Insert form data into the database

In the forms lab you configured the "`post`" controller and related view template to display a form for posting data.

1. From Zend Studio, open the file `Market/src/Market/Model/ListingsTable.php`
2. Create a new method `addPosting()` that accepts an array of data as a parameter
3. Split out city and country:
   - Get the city and country from `Market\Form\PostForm::$cityCodes`
   - Use `cityCode` coming from the form as a key
   - Use `explode()`, with "`,`" as the split character, to separate city from country
4. Use date arithmetic to calculate `date_expires`:
   - Create a new `\DateTime` object
   - Create an `interval` that looks like this:

     `+ NNN` day (where "`NNN`" is the number of days captured from the form)
   - Use the `modify()` method of the `DateTime` object, specifying the interval as an argument
   - Store the date in the `listings` table using the `DateTime::format()` method, with a string of `'Y-m-d H:i:s'`
5. Use the table's `insert()` method to add the posting
6. From Zend Studio, open the file `Market/src/Market/Controller/PostController.php`
7. In `indexAction()`, locate where you checked to see if the form was valid
8. If the form is valid, call the `listings` table's new `addPosting()` method, using the sanatized data from the form filter
9. From the browser, test by posting a couple of entries

My Notes

Module 10

DATABASE LAB … M10Ex1

G. [OPTIONAL] Incorporate the "world-city-area-codes" table into the form

This is an optional lab. If you decide to finish this lab, you will gain additional experience in controllers, forms, filters, and services. If you choose not to complete this lab, it will have no impact on the Online Market class project, nor will it have any impact on subsequent labs.

1. Create a table model which represents the `world_city_area_codes` table
2. Create a method `getCodesForForm()` that returns an array where the key = the `world_city_area_codes_id` field, and the value is a combination of city & country
3. Create a lookup method `getCodeById()` that accepts a `world_city_area_codes_id` and returns a single listing
4. Create a factory to generate an instance of the `world_city_area_codes` table
5. Set up a service which points to the `world_city_area_codes` table factory
6. In `Market\Form\PostFormFactory` , add a line that calls the `getCodesForForm()` method and injects this into the form
7. Modify the `cityCode` form element to accept the array created above instead of a hard-coded array of city names
8. Modify `Market\Controller\PostController` as follows:
   - Add a `$cityCodeTable` property
   - Add a method to set this property
   - In `indexAction()`, before form data is inserted into the DB, do a lookup on the cityCode field, calling `getCodeById()` from the `world_city_area_codes` table
9. Modify `Market\Form\PostFormFactory` to inject the `world_city_area_codes` table

My Notes

Module 10

DATABASE LAB … M10Ex1

H. [OPTIONAL] Connect the search module

In an earlier lab, you added the `Search` module to the application. As you may recall, it was not possible to fully test this module as the database "`db`" key and adapter factory had not yet been configured. Now that these services are in place, you can connect the `Search` module to rest of the application by supplying a link to the layout. Looking at `Search/config/module.config.php` you can see a route with the url "`/search`", the link that needs to be added.

I. [OPTIONAL] Configure the application to delete a posting given the delete code

This is an optional lab. If you decide to complete this lab, you will gain additional experience in controllers, forms, filters, and services. If you choose not to pursue this lab, it will have no impact on the Online Market class project, nor will it have any impact on subsequent labs.

1. Add a `deleteByDeleteCode()` method to the `ListingsTable` class that accepts a listings ID and a delete code as an argument; the method should then delete a database row if both parameters match

2. Create a `delete` controller that has the following features:
    - Delete form public property
    - Delete filter public property
    - Listings table public property
    - A `setDeleteForm()` method
    - A `setDeleteFilter()` method
    - A `setListingsTable()` method

My Notes

Module 10

DATABASE LAB … M10Ex1

3. The `delete` controller will have an `index` action that does the following:
    - Captures data from `$_POST` using the `params()` plugin
    - Assigns the input filter to the form
    - Assigns the data to the form
    - Checks to see if the form is valid
    - If valid, calls the `deleteByDeleteCode()` method of the listings table
    - Redirects back home with a success message
    - If not valid, redisplay the form with its error messages
4. Create a delete form
    - Include elements for delete code and item ID
    - Make sure you include all the appropriate HTML attributes and labels
    - Add a CAPTCHA to prevent SPAM
    - Don't for the submit button!
5. Create a delete filter
    - Have an input element which matches each element you need to filter or validate from the form
6. Create factories for the delete form and for the filter
7. Define factory services for the form and filter
8. Create a factory for the delete controller that injects the table, form and filter
9. Define a route for the delete controller
10. Create a view template that renders the form
11. Test the new controller … observe and correct any errors

My Notes

Module 11

SESSIONS LAB ... M11Ex1

This module lab is composed of three separate parts (topics) that may be completed independently, in any order. In this Session lab, you will use `Zend\Session\*` family components to prevent an invalid posting from occurring more than three times. In the Log lab, you will be creating a log of those items being viewed. In the Mail lab, you will send an email notification when a user posts an item that reminds them of their delete code.

---

1. In Zend Studio, select the file:
   `Market/src/Market/Controller/PostController.php`

2. Create a public property `$maxAttempts` and set it to a value (ex: 3)

3. Create a new method `invalidPostCount()` as follows:
   - Set a flag $invalid = FALSE
   - Create a `Zend\Session\Container` object. Supply a container name `'post'` in the constructor.
   - Use `offsetExists()` to check to see if a key `'count'` has been set
     - If not, assign an initial value of `1` to the count
     - If the key has been set, check to see if the value exceeds `$maxAttempts`
       - If so, set the `$invalid` to `TRUE` and reset the count to `1`
       - If not, increment the count
       - Make sure the new count is saved back to the session container using `offsetSet()`

4. Inside `indexAction()` incorporate the new method:
   - Locate where the form is validated
   - Add a call to the new method
   - If the invalid count has been exceeded (if the return value is TRUE):
     - Use the `flashMessenger()` plugin to send an appropriate message
     - Redirect back to the home page

5. Test the new functionality by selecting "post" and hitting the submit button a few times

6. Observe and correct any errors

My Notes

Module 11

LOG LAB ... M11Ex2

In this lab, you will be working with the `Market\Module` class, defining an event and listener which records items viewed. You will also need to configure a new service instance in a local override file which defines the location of the log file.

1. Create a directory structure to contain log files:
   - From Zend Studio, select the folder `onlinemarket.work`
   - Select the folder `onlinemarket.work/data`
   - Select **File | New |Folder** and name the new folder `logs`
   - Make sure the app has the right permissions to read & write files in the folder

2. Configure system-wide parameters in a local autoload override file:
   - From Zend Studio, select the folder: `onlinemarket.work/config/autoload`
   - Select **File | New | PHP File** and name the new file `params.local.php`
   - Return an array with a key `"service_manager" => "services" => "params"`
   - Assign an array to the `"params"` key with the following information:

Key	Value
`log_filename`	`__DIR__ . '/../../data/data/logs/items_viewed.log'`

   IMPORTANT: make sure that your application has permissions to write to this file!

3. From Zend Studio, select the file `Market/Module.php`

4. At the end of the `onBootstrap()` method, attach a listener with these characteristics:
   - *Listens For:* the `dispatch` event
   - *Context:* current object
   - *Handler:* `onDispatch`
   - *Priority:* 100

   NOTE: be sure to "`use`" the appropriate classes

My Notes

Module 11

LOG LAB ... M11Ex2

5. Define a method `onDispatch()` that accepts an MvcEvent as an argument, as follows:
   - From the incoming MvcEvent use the "`getRouteMatch()`" method to retrieve routing information
   - From the route match retrieve the controller and action names
   - Retrieve the system-wide parameters from the service manager using `getServiceManager()`, which in turn can be obtained from the application using `getApplication()`
   - If the controller is `'market-view-controller'` and the action is `'item'`, log the item viewed:
     - Retrieve the item ID from the route match using `getParam()`
     - Construct the message string to include the item ID
     - Create a `Log\Writer\Stream` object using the parameters obtained from the service
     - Create a `Log\Logger` object
     - Assign the writer to the logger
     - Log the message
6. Test the new functionality by viewing several items and then by reviewing the log file

   NOTE: you will need to select **File | Refresh** before being able to view the file inside Zend Studio
7. Observe and correct any errors

My Notes

Module 11

MAIL LAB … M11Ex3

1. If it does not already exist, create a directory structure to contain the email test files:
   - In Zend Studio, select the folder `onlinemarket.work`
   - Select the folder `onlinemarket.work/data`
   - Select **File | New | Folder** with a name `logs`
   - Make sure the application has the right permissions to read and write files in the new folder
2. Configure email parameters in the local autoload override file:
   - In Zend Studio, select or create the file `onlinemarket.work/config/autoload/params.local.php`
   - Add new key `"service_manager" => "services" => "params" => "email-params"`
   - Add the following information to the `"email-params"` key:

Key	Value
`to`	destination email address (ex: `'admin@company.com'`)
`from`	source email address (ex: `'market@company.com'`)
`email_dir`	`__DIR__ . '/../../data/logs'`

3. In Zend Studio, select the file `Market/src/Market/Controller/PostController.php`
4. Define a property `$mailTransport` that represents the mechanism for sending email. The email transport will defined as a service, and will later be injected into the controller

   NOTE: normally this would be `sendmail`, but for the purposes of the lab, you will be defining a file-based email transport

My Notes

Module 11

MAIL LAB … M11Ex3

5. Define a method `setMailTransport()` that sets the `$mailTransport` property
6. Define a property `$emailParams` that represents the email parameters
7. Define a method `setEmailParams()` that sets the `$emailParams` property; this will be called (later) from the `PostControllerFactory`
8. Create a new method `sendNofication()` as follows:
    - Have the new method accept the delete code as an argument
    - Create a new `Zend\Mail\Message` object
    - Configure the message components as follows:

Component	Method	Notes
`Recipient`	`addTo()`	Use `'to'` from the `params` service
`From`	`addFrom()`	Use `'from'` from the `params` service
`Subject`	`setSubject()`	An appropriate subject
`Subject`	`setSubject()`	An appropriate message that includes the delete code
`Encoding`	`setEncoding()`	utf-8 is recommended

    - Send the email message using the email transport mentioned above
9. Locate in `indexAction()` where the form is valid. If the form is valid, and a success message is posted, call the `sendNotification()` method

My Notes

Module 11

MAIL LAB … M11Ex3

10. Create an email transport factory class as follows:
    - From Zend Studio, select the `Market/src/Market/Factory` folder
    - Copy `PostFormFactory.php` to `MailTransportFactory.php`
    - Extract the `email-params` service
    - Create a `Zend\Mail\Transport\FileOptions` object, and as a constructor argument:
        - Specify an array with a key `'path'`
        - Use the information in the `'email_dir'` key of the `'email-params'` service as the path
    - Create a `Zend\Mail\Transport\File` object, with the `FileOptions` object as a constructor argument
    - Have the factory return the `Zend\Mail\Transport\File` object
11. Inject the mail transport and parameters into the `post` controller
    - From Zend Studio, open the file `Market/config/module.config.php`
    - Add a new key `'service_manager' => 'factories' => 'market-mail-transport' => 'Market\Factory\MailTransportFactory'`
    - In Zend Studio, open the file `Market/src/Market/Factory/PostControllerFactory.php`
    - Retrieve the email transport from the `'market-mail-transport'` service
    - Call `setMailTransport()` from the controller object, using the new `'market-mail-transport'` factory service
    - Retrieve the email params from the `'email-params'` service
    - Call `setEmailParams()` from the controller object, using the email params retrieved above

My Notes

Module 11

MAIL LAB … M11Ex3

12. Test the new functionality by posting an item and then by reviewing the `data/logs` folder, looking for new messages

    NOTE: you will need to select **File | Refresh** before being able to view the newly created file inside Zend Studio

13. Observe and correct any errors

    NOTE: don't forget to "`use`" the appropriate classes!

My Notes