DOWNLOAD PDF ZEND FRAMEWORK IN ACTION

Chapter 1 : Collections - zend-form

Zend Framework in Action is a comprehensive tutorial that shows how to use Zend Framework to create web-based applications and web services. This book takes you on an "over-the-shoulder" tour of the components of Zend Framework as you build a high quality, real-world web application.

How do you do it without overriding the layout for every action throughout the entire application? Thanks to GeeH, over in zftalk , for providing corrections to the earlier draft of this post. Zend Framework 2 implements the 2-Step View pattern , which allows for one set of view templates to handle logic such as headers, footers, sidebars and navigation blocks, logic not specific to any one action be kept separate from the view templates which render the content of a specific action. For the most part, that works fine. Zend Framework 2 makes it easy to do this. What if we want to override it for every action in the controller? In your controller add in the following use statement: The higher the priority the more important that event is. By passing the current controller object to the closure we can call the layout function on it, specifying the same template alias as before. Actions in any other controller, in any other module, will use the default layout template as normal. For more information on the available Module options, check out the manual. What it does is to get access to the EventManager and attach a listener to the dispatch event as before, using a closure, with one small difference. If not, the original layout template is rendered. And there you have it. Three simple ways to override layouts in your Zend Framework 2 modules. Do you change layouts in your modules this way, or do you take a different approach? Share your approach in the comments. Drop your email in the box below, and get awesome tutorials â€" just like this one â€" straight to your inbox, PLUS exclusive content only available by email. Join The Community Drop your email address in the box below to join a community of over 1, like-minded developers getting great tutorials, podcasts, screencasts, tips, pointers and much, much more. Want to learn all about Zend Expressive without going overboard? Then this book is for you! Learn the essentials you need to know to start building applications with Zend Expressive.

Chapter 2 : Zend Framework 2 in Action â€“ Rob Allen's DevNotes

Zend Framework in Action is a comprehensive tutorial that shows how to use the Zend Framework to create web-based applications and web services. This book takes you on an over-the-shoulder tour of the components of the Zend Framework as you build a high quality, real-world web application.

Preventing validation from wiping out previous collection items Form Collections Often, fieldsets or elements in your forms will correspond to other domain objects. In some cases, they may correspond to collections of domain objects. In this latter case, in terms of user interfaces, you may want to add items dynamically in the user interface â€" a great example is adding tasks to a task list. This document is intended to demonstrate these features. N relationship one Product has many Category instances. Creating Fieldsets The first step is to create three fieldsets. Each fieldset will contain all the fields and relationships for a specific entity. Here is the Brand fieldset: When we validate incoming data, the form will automatically iterate through all the fieldsets it contains populate all sub-objects, in order to return a complete entity. This information will be used to validate the input field. Finally, getInputFilterSpecification gives the specification for the remaining input "name" , indicating that this input is required. Note that required in the array "attributes" when elements are added is only meant to add the "required" attribute to the form markup and therefore has semantic meaning only. Here is the Category fieldset: Finally, the Product fieldset: First, notice how the brand element is added: This is how you handle a 1: When the form is validated, the BrandFieldset will first be populated, and will return a Brand entity as we have specified a ClassMethods hydrator, and bound the fieldset to a Brand entity using the setObject method. This Brand entity will then be used to populate the Product entity by calling the setBrand method. The next element shows you how to handle 1: As you can see, the name of the element "categories" perfectly matches the name of the property in the Product entity. This element has a few interesting options: This, of course, depends on what you want to do. The Form Element So far, so good. We now have our fieldsets in place. But those are fieldsets, not forms. And only Form instances can be validated. So here is the form: This option is here to say to the form: This approach allows each entity to have its own Fieldset, and enables re-use. You describe the elements, the filters, and validators for each entity only once, and the concrete Form instance will only compose those fieldsets. You no longer have to add the "username" input to every form that deals with users! Create a form instance. Bind it to an object. Integration with zend-view And finally, the view: You must call it prior to rendering anything in the view this function is only meant to be called in views, not in controllers. If you need more control about the way you render your forms, you can iterate through the elements in the collection, and render them manually one by one. Here is the result: Collections are wrapped inside a fieldset, and every item in the collection is itself wrapped in the fieldset. In fact, the Collection element uses label for each item in the collection, while the label of the Collection element itself is used as the legend of the fieldset. You must have a label on every element in order to use this feature. As soon as the form is valid, this is what we get: The bound object is completely filled with the object instances we specified, not with arrays! Often, forms are not completely static. The collection generates two fieldsets the two categories plus a span with a data-template attribute that contains the full HTML code to copy to create a new element in the collection. First, count the number of elements we already have. Change the placeholder to a valid index. Add the element to the DOM. The following is a potential implementation: One small remark about template. Now if we validate the form, it will automatically take into account this new element by validating it, filtering it and retrieving it: If the initial count is 2, you must have at least two elements. Dynamically added elements have to be added at the end of the collection. They can be added anywhere these elements will still be validated and inserted into the entity , but if the validation fails, this newly added element will be automatically placed at the end of the collection. Validation groups for fieldsets and collection Validation groups allow you to validate a subset of fields. In fact, this is normal. Of course, you could create a BrandFieldsetWithoutUrl fieldset, but this would require a lot of duplicate code. A validation group is specified in a Form object hence, in our case, in the CreateProduct form by giving an array of all the elements we want to validate. Our CreateProduct class now looks like this: You can also recursively select the elements if desired. There is one limitation currently: Preventing validation from wiping out previous collection items In some cases, you may be representing collections within a model, but not validating them; as an example, if you use a validation group that excludes the collections from validation so that they remain untouched after binding. One such change is that if a collection is found in a form, but has no associated data, an empty array is assigned to it, even when not in the validation group. This effectively wipes out the collection data when you bind values. To prevent this behavior, starting in 2. The Form class has been updated to pass the validation group, if present, on to fieldset and collection instances when performing bindValues operations. For more details, refer to the following issues:

Chapter 3 : Change Layout in Controllers and Actions in Zend Framework 2 â€” Master Zend Framework

However, Zend Framework is really a heavy framework and not flexible. (Ruby on Rails is a much better framework.) After reading this book, I decided not to go ahead to use Zend Framework.

There are two bits to this part: Create a file called AlbumForm. For each item we set various attributes and options, including the label to be displayed. We also need to set up validation for this form. In Zend Framework 2 this is done using an input filter, which can either be standalone or defined within any class that implements the InputFilterAwareInterface interface, such as a model entity. In our case, we are going to add the input filter to the Album class, which resides in the Album. We only need to implement getInputFilter so we simply throw an exception in setInputFilter. Within getInputFilter , we instantiate an InputFilter and then add the inputs that we require. We add one input for each property that we wish to filter or validate. For the id field we add an Int filter as we only need integers. We now need to get the form to display and then process it on submission. After adding the AlbumForm to the use list, we implement addAction. We then set the posted data to the form and check to see if it is valid using the isValid member function of the form. In this case, just the form object. Note that Zend Framework 2 also allows you to simply return an array containing the variables to be assigned to the view and it will create a ViewModel behind the scenes for you. This saves a little typing. We now need to render the form in the add. Zend Framework provides some view helpers to make this a little easier. The form view helper has an openTag and closeTag method which we use to open and close the form. Then for each element with a label, we can use formRow , but for the two elements that are standalone, we use formHidden and formSubmit. Alternatively, the process of rendering the form can be simplified by using the bundled formCollection view helper. For example, in the view script above replace all the form-rendering echo statements with: You still need to call the openTag and closeTag methods of the form. You replace the other echo statements with the call to formCollection, above. This helps reduce the complexity of your view script in situations where the default HTML rendering of the form is acceptable. This time we use editAction in the AlbumController: This code should look comfortably familiar. Firstly, we look for the id that is in the matched route and use it to load the album to be edited: If the id is zero, then we redirect to the add action, otherwise, we continue by getting the album entity from the database. We have to check to make sure that the Album with the specified id can actually be found. If it cannot, then the data access method throws an exception. We catch that exception and re-route the user to the index page. This is used in two ways: When displaying the form, the initial values for each element are extracted from the model. After successful validation in isValid , the data from the form is put back into the model. These operations are done using a hydrator object. We have already written exchangeArray in our Album entity, so just need to write getArrayCopy: The view template, edit. You should now be able to edit albums. This would be wrong. We use the table object to delete the row using the deleteAlbum method and then redirect back the list of albums. If the request is not a POST, then we retrieve the correct database record and assign to the view, along with the id. The view script is a simple form: At the moment, the home page, http:

Chapter 4 : Zend Framework in Action - Rob Allen, Nick Lo, Steven Brown - Google Books

Zend Framework in Action has 46 ratings and 4 reviews. Mike said: It is a very good introduction to the Zend Framework. Through the book, you build an ex.

Chapter 5 : calendrierdelascience.com: Manning | Zend Framework in Action

For Zend Framework applications, the controller system is based on the design pattern known as Front Controller which uses a handler (Zend_Controller_Front) and action commands (Zend_Controller_Action) which work together in tandem.

Chapter 6 : zend framework: set action in form class - Stack Overflow

DOWNLOAD PDF ZEND FRAMEWORK IN ACTION

Action Helpers in Zend Framework are often considered a fairly arcane subject, something for experts only. However, they are meant to be an easy way to extend the capabilities of Action.

Chapter 7 : Zend Framework In Action (calendrierdelascience.com) - Manning | Zend Framework in Action

We collected the majority of metadata history records for calendrierdelascience.com Zend Framework In Action has a poor description which rather negatively influences the efficiency of search engines index and hence worsens positions of the domain.

Chapter 8 : Zend Framework in Action

Zend Framework (ZF) is an open source, object-oriented web application framework implemented in PHP 7 and licensed under the New BSD License. The framework is basically a collection of professional PHP based packages.

Chapter 9 : Zend Framework - Wikipedia

20 CHAPTER 2 Hello Zend Framework! The model The model part of the MVC pattern is all the business-logic code that works behind the scenes in the application. This is the code that decides how to apply the shipping costs.