Zend Framework Interview Question Answers - Part II

Zend-MVC

- Model - The "stuff" you are using in the application -- data, web services, feeds,etc.
- View -The display returned to the user.
- Controller - Manages the request environment, and determines what happens.

Bootstrap

What is Bootstrapping? Many PHP applications funnel server requests into a single (or few) PHP source file that sets up the environment and configuration for the application, manages sessions and caching, and invokes the dispatcher for their MVC framework. They can do more, but their main job is to take care of the consistent needs of every page of a web application.
In our Blueprint for PHP Applications, we will have a core bootstrapper that receives all dynamic requests for an application and applies a template for application behavior that we can later extend. It will allow us to later customize the functionality for each unique application.

Zend registry A registry is a container for storing objects and values in the application space. By storing the value in a registry, the same object is always available throughout your application. This mechanism is an alternative to using global storage.
The typical method to use registries with Zend Framework is through static methods in the Zend_Registry class. Alternatively, the registry can be used as an array object, so you can access elements stored within it with a convenient array-like interface.

Zend form, decorator Zend_Form simplifies form creation and handling in your web application. It performs the following tasks:

- Element input filtering and validation
- Element ordering
- Element and Form rendering, including escaping
- Element and form grouping
- Element and form-level configuration

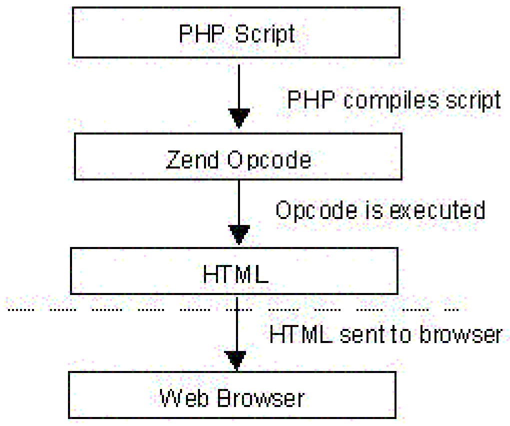
Zend_Form makes use of several Zend Framework components to accomplish its goals, including Zend_Config, Zend_Validate, Zend_Filter, Zend_Loader_PluginLoader, and optionally Zend_View.

Zend helpers In your view scripts, often it is necessary to perform certain complex functions over and over: e.g., formatting a date, generating form elements, or displaying action links. You can use helper classes to perform these behaviors for you.

E-g

# Action View Helper
# BaseUrl Helper
# Currency Helper
# Cycle Helper
# Partial Helper
# Placeholder Helper

Zend engine **Zend Engine is used internally by PHP as a complier and runtime engine. PHP Scripts are loaded into memory and compiled into Zend opcodes**
These opcodes are executed and the HTML generated is sent to the client. The same is depicted below



What is Zend engine in PHP? Zend engine is like a virtual machine and is an open source, and is known for its role in automating the web using PHP. Zend is named after its developers Zeev and Aandi. Its reliability, performance and extensibility has a significant role in increasing the PHP's popularity. The Zend Engine II is the heart of PHP 5. It is an open source project and freely available under BSD style license.

Zend layout Zend_Layout implements a classic Two Step View pattern, allowing developers to wrap application content within another view, usually representing the site template. Such templates are often termed *layouts* by other projects, and Zend Framework has adopted this term for consistency. The main goals of Zend_Layout are as follows:

- Automate selection and rendering of layouts when used with the Zend Framework MVC components.
- Provide separate scope for layout related variables and content.
- Allow configuration, including layout name, layout script resolution (inflection), and layout script path.
- Allow disabling layouts, changing layout scripts, and other states; allow these actions from within action controllers and view scripts.
- Follow same script resolution rules (inflection) as the ViewRenderer, but allow them to also use different rules.
- Allow usage without Zend Framework MVC components.

Front controllers Zend_Controller_Front implements a » Front Controller pattern used in » Model-View-Controller (MVC) applications. Its purpose is to initialize the request environment, route the incoming request, and then dispatch any discovered actions; it aggregates any responses and returns them when the process is complete.
Zend_Controller_Front also implements the » Singleton pattern, meaning only a single instance of it may be available at any given time. This allows it to also act as a registry on which the other objects in the dispatch process may draw.
Zend_Controller_Front registers a plugin broker with itself, allowing various events it triggers to be

observed by plugins. In most cases, this gives the developer the opportunity to tailor the dispatch process to the site without the need to extend the front controller to add functionality.
At a bare minimum, the front controller needs one or more paths to directories containing action controllers in order to do its work. A variety of methods may also be invoked to further tailor the front controller environment and that of its helper classes

Zend components Goals of Zend Framework components
The following lists the components of Zend Framework, each with a brief description and list of goals for each component.

Zend_Acl provides lightweight and flexible access control list (ACL) functionality and privileges management.

- includes basic implementations for both Roles and Resources
- Roles and Resources may be instances of user-defined classes
- simplifies the specification of access control rules with inheritance support
- supports conditional access control rules via an assertion interface

Zend_Auth provides an API for authentication and includes concrete authentication adapters for common use case scenarios, as well as "Identity 2.0" adapters such as OpenID and Microsoft InfoCard.

- provides adapter interface for customized authentication mechanisms
- automatic identity storage is abstracted for easy customization
- simple and extensible API

Zend_Cache provides a flexible approach toward caching data, including support for tagging, manipulating, iterating, and removing subsets.

- provides multiple storage back-ends (File, Sqlite, Memcached, etc.)
- provides multiple front-ends (helpers for caching function or method calls, in addition to caching full pages)
- simple and flexible for generic uses

Zend_Config
Zend_Config simplifies the use of configuration data for web applications.

- provides a property-based interface for reading configuration data
- supports a variety of hierarchical data storage formats
- supports inheritance of configuration data between two sections

Zend_Console_Getopt
Command-line PHP applications benefit from this convenient object-oriented interface for declaring, parsing, and reporting command-line arguments and options.

- supports GNU getopt syntax
- supports more extensive option declaration syntax
- supports automatic reporting of option usage help

Zend_Controller and Zend_View

These components provide the infrastructure for a Model-View-Controller (MVC) website.

- provides simple and extensible MVC pattern
- provides PHP-based template engine by default
- provides support for application modules
- provides configuration-less architecture

Zend_Date
Zend_Date offers a detailed but simple API for manipulating dates and times.

- supports I18N and L10N throughout its API
- supports ISO and GNU/PHP standard tokens
- provides handling for dates bigger than 64bit
- provides sunset and sunrise calculation based on cities

Zend_Db
This is a lightweight database access layer, providing an interface to PDO and other database extensions in PHP. It includes adapters for each database driver, a query profiler, and an API to construct most SELECT statements.

- provides abstract interface to multiple PHP database extensions
- based on PDO interface, but extends beyond that
- provides query profiler
- provides query builder
- provides robust SQL support including parameters and quoting

Zend_Db_Table
The Zend_Db_Table component is a lightweight solution for object-oriented programming with databases.

- implements the Table Data Gateway and Row Data Gateway patterns
- discovers database metadata without the need for complex configuration files to maintain
- provides a solution for querying related tables

Zend_Feed
This component provides a very simple way to work with live syndicated feeds.

- consumes RSS and Atom feeds
- provides utilities for discovering feed links
- imports feeds from multiple sources
- provides feed building and posting operations

Zend_Filter and Zend_Validate
These components encourage the development of secure websites by providing the basic tools necessary for input filtering and validation.

- provide an extensible architecture for filters and validators
- support Unicode text data

- support user-configurable messages for validation failures

Zend_Filter_Input
This is a configurable solution for declaring and enforcing filtering and validation rules. This component serves as a "cage" for input data, so they are available to your application only after being validated.

- does not require configuration files
- supports declarative syntax for applying rules to collections of input data
- supports chaining of filters and validators
- supports automatic escaping of validated data values

Zend_Form
This component provides an object-oriented interface for building forms, complete with input filtering and rendering capabilities.

- provides classes for elements, forms, display groups, and sub forms
- supports per-element input filters
- supports per-element validations, including context-sensitive validations
- supports per-element, group, and form rendering via flexible decorators
- extensive plugin system for customizing all aspects of forms and elements

Zend_Gdata (Zend Google Data Client)
The Google Data APIs provide read/write access to such services hosted at google.com as Spreadsheets, Calendar, Blogger, and CodeSearch.

- supports both authentication mechanisms of Google Data servers
- supports queries and posting changes against Google Data services
- supports service-specific element types in an object-oriented interface
- matches functionality and design of other Google Data API clients

Zend_Http_Client
This component provides a client for the HTTP protocol, without requiring any PHP extensions. It drives our web services components.

- supports URL validation
- supports cookies
- supports proxy servers

Zend_Json
Easily convert PHP structures into JSON and vice-versa for use in AJAX-enabled applications.

- uses PHP's ext/json when available
- supports decoding Javascript objects to native PHP structures
- supports encoding native PHP objects to JSON notation
- supports XML to JSON conversions

Zend_Layout Easily provide sitewide layouts for your MVC applications.

- supports use with or without MVC layer

- decorates Zend_View, inheriting capabilities of that component
- provides a variety of helpers and plugins for accessing the layout object from within other MVC components

Zend_Loader Load files, classes, and resources dynamically in your PHP application.

- supports SPL autoloader
- supports include_path
- provides exception-based failure mechanism
- provides mechanism for loading plugins based on class prefix and path

Zend_Locale Zend_Locale is the Framework's answer to the question, "How can the same application be used around the whole world?" This component is the foundation of Zend_Date, Zend_Translate, and others.

1. provides access to CLDR, an international data repository for I18N issues, for all framework classes
2. provides localizing of numbers
3. provides normalizing of dates, times and numbers

Zend_Log Log data to the console, flat files, or a database. Its no-frills, simple, procedural API reduces the hassle of logging to one line of code and is perfect for cron jobs and error logs.

- provides a simple object-oriented interface inspired by log4j
- supports extensible output channels
- supports extensible output formats

Zend_Mail and Zend_Mime Almost every Internet application needs to send email. Zend_Mail, assisted by Zend_Mime, creates email messages and sends them.

- supports attachments
- supports MIME types
- supports a variety of mail storage protocols
- supports multiple mail transport agents
- supports a variety of authentication mechanisms

Zend_Measure Using Zend_Measure, you can convert measurements into different units of the same type. They can be added, subtracted, and compared against each other.

- supports localized handling of measurements and numbers
- supports converting of measurements and numbers

Zend_Memory Zend_Memory offers an API for managing data in a limited memory mode. A PHP developer can create a Zend_Memory object to store and access large amounts of data, which would exceed the memory usage limits imposed by some PHP environments.

- provide transparent mechanism to work with swappable memory blocks
- support all existing Zend_Cache back-ends as storage providers as well as the 'None' back-end which gives an ability to work in non-limited memory mode through the same API and with minimal overhead

Zend_Pdf
Portable Document Format (PDF) from Adobe is the de facto standard for cross-platform rich documents. Now, PHP applications can create or read PDF documents on the fly, without the need to call utilities from the shell, depend on PHP extensions, or pay licensing fees. Zend_Pdf can even modify existing PDF documents.

- supports Adobe PDF file format
- parses PDF structure and provides access to elements
- creates or modifies PDF documents
- utilizes memory efficiently

Zend_Registry The registry is a container for storing objects and values in the application space. By storing an object or value in the registry, the same object or value is always available throughout your application for the lifetime of the request. This mechanism is often an acceptable alternative to using global variables.

- provides globally accessible storage for objects and values
- provides iterator, array, and indexed access

Zend_Rest_Client and Zend_Rest_Server
REST Web Services use service-specific XML formats. These ad-hoc standards mean that the manner for accessing a REST web service is different for each service. REST web services typically use URL parameters (GET data) or path information for requesting data and POST data for sending data.

- provides capabilities to access REST web services
- provides capabilities to expose APIs as REST services

Zend_Search_Lucene
The Apache Lucene engine is a powerful, feature-rich Java search engine that is flexible about document storage and supports many complex query types. Zend_Search_Lucene is a port of this engine written entirely in PHP 5.

- allows PHP-powered websites to leverage powerful search capabilities without the need for web services or Java
- provides binary compatibility with Apache Lucene
- matches Apache Lucene in performance

Zend_Service: Akismet, Amazon, Audioscrobbler, Delicious, Flickr, Nirvanix, Simpy, StrikeIron and Yahoo!
Web services are important to the PHP developer creating the next generation of mashups and composite applications. Zend Framework provides wrappers for service APIs from major providers to make it as simple as possible to use those web services from your PHP application.

- fetch web service data from popular providers with just a few lines of code
- simplified object-oriented API encapsulates the underlying protocols and formats
- features an ever-growing set of components to accommodate new and relevant services

Zend_Session Zend_Session helps manage and preserve session data across multiple page requests by the same client.

- provides an object-oriented interface to access session data
- provides optional security features to help protect against session hijacking
- supports namespaced access to the PHP session for interoperability

Zend_Translate component provides Zend Framework with message translation functionality.

- provides a simple and consistent object-oriented interface to translated message storage
- supports industry-standard message storage formats such as gettext, TMX, Qt, XLIFF and others
- provides thread-safe gettext implementation

Zend_Uri

Zend_Uri is a component that aids in manipulating and validating Uniform Resource Identifiers (URIs). Zend_Uri exists primarily to service other components such as Zend_Http_Client but is also useful as a standalone utility.

- create URIs
- manipulate URIs
- validate URIs

Zend_XmlRpc

Zend_XmlRpc makes it easy to communicate with and create XML-RPC services from PHP.

- mimics PHP's SOAP extension
- flexible request and response implementation allows for use with non-HTTP services
- server implementation allows attaching existing classes to quickly expose APIs as XML-RPC services

Zend plugins, default functins The controller architecture includes a plugin system that allows user code to be called when certain events occur in the controller process lifetime. The front controller uses a plugin broker as a registry for user plugins, and the plugin broker ensures that event methods are called on each plugin registered with the front controller.
The event methods are defined in the abstract class Zend_Controller_Plugin_Abstract, from which user plugin classes inherit:

- routeStartup() is called before Zend_Controller_Front calls on the router to evaluate the request against the registered routes.
- routeShutdown() is called after the router finishes routing the request.
- dispatchLoopStartup() is called before Zend_Controller_Front enters its dispatch loop.
- preDispatch() is called before an action is dispatched by the dispatcher. This callback allows for proxy or filter behavior. By altering the request and resetting its dispatched flag (via Zend_Controller_Request_Abstract::setDispatched(false)), the current action may be skipped and/or replaced.
- postDispatch() is called after an action is dispatched by the dispatcher. This callback allows for proxy or filter behavior. By altering the request and resetting its dispatched flag (via Zend_Controller_Request_Abstract::setDispatched(false)), a new action may be specified for dispatching.
- dispatchLoopShutdown() is called after Zend_Controller_Front exits its dispatch loop

What is routing and how it's work? Zend_Controller_Router_Rewrite is the standard framework router. Routing is the process of taking a URI endpoint (that part of the URI which comes after the base URL) and decomposing it into parameters to determine which module, controller, and action of that controller should receive the request. This values of the module, controller, action and other parameters are packaged into a Zend_Controller_Request_Http object which is then processed by Zend_Controller_Dispatcher_Standard. Routing occurs only once: when the request is initially received and before the first controller is dispatched.
Zend_Controller_Router_Rewrite is designed to allow for mod_rewrite-like functionality using pure PHP structures. It is very loosely based on Ruby on Rails routing and does not require any prior knowledge of webserver URL rewriting. It is designed to work with a single Apache mod_rewrite rule

How create form element using Zend form? A form is made of elements that typically correspond to HTML form input. Zend_Form_Element encapsulates single form elements, with the following areas of responsibility:

- validation (is submitted data valid?)
- capturing of validation error codes and messages
- filtering (how is the element escaped or normalized prior to validation and/or for output?)
- rendering (how is the element displayed?)
- metadata and attributes (what information further qualifies the element?)

E-G

```
class Storefront_Form_LoginForm extends Zend_Form{
public function init(){
$username = $this->createElement("text","username");
$username->setRequired(true)
->setOptions(array('class'=>'textbox'))
->addFilters(array(
new Zend_Filter_StripTags(),
new Zend_Filter_StringTrim()
))
-
>addValidator("NotEmpty",true,array('messages'=>array('isEmpty'=>'Emai
l address cannot be empty')))
-
>addValidator("emailAddress",true,array('messages'=>array('emailAddres
sInvalidFormat'=>'Email address is not valid')));


$password = $this->createElement("password","password");
$password->setRequired(true)
->setOptions(array('class'=>'textbox'))
->addFilters(array(
new Zend_Filter_StripTags(),
new Zend_Filter_StringTrim()
))
-
>addValidator("NotEmpty",true,array('messages'=>array('isEmpty'=>'Pass
word cannot be empty')));
$signin = $this->createElement("submit","signin");
$signin->setLabel("Sign in")
->setIgnore(true);
$this->addElements(array(
$username,
$password,
$signin
)
);
// Decoding the all elements into empty
$this->setElementDecorators(array('ViewHelper'));
}
}

//We can access the form at controller using following method
$login_form = new Storefront_Form_LoginForm();
assign this variable object to view

accessing form elemnt at view
$this->form->username
$this->form->password
```

What are the validator having in Zend and it's syntax? If you subscribe to the security mantra of "filter input, escape output," you'll should use validator to filter input submitted with your form. In Zend_Form, each element includes its own validator chain, consisting of Zend_Validate_* validators. Validators may be added to the chain in two ways:

- passing in a concrete validator instance
- providing a short validator name

e-g
$element->addValidator(**new** Zend_Validate_Alnum());

- *// Short validator name:*
- $element->addValidator('Alnum');
- $element->addValidator('alnum');

Validator class:
1)**Zend_Validate_Alnum** allows you to validate if a given value contains only alphabetical characters and digits. There is no length limitation for the input you want to validate. $validator = **new** Zend_Validate_Alnum();

```
1. if ($validator->isValid('Abcd12')) {
2.     // value contains only allowed chars
3. } else {
4.     // false
```

5. }

2)**Zend_Validate_Alpha** allows you to validate if a given value contains only alphabetical characters. There is no length limitation for the input you want to validate. This validator is related to the Zend_Validate_Alnum validator with the exception that it does not accept digits.
3)**Zend_Validate_Barcode** allows you to check if a given value can be represented as barcode. Zend_Validate_Barcode supports multiple barcode standards and can be extended with proprietary barcode implementations very easily
4)**Zend_Validate_Between** allows you to validate if a given value is between two other values. **Note**: **Zend_Validate_Between supports only number validation**
It should be noted that Zend_Validate_Between supports only the validation of numbers. Strings or dates can not be validated with this validator.
5)**Zend_Validate_Callback** allows you to provide a callback with which to validate a given value.
Supported options for Zend_Validate_Callback

The following options are supported for Zend_Validate_Callback:

* callback: Sets the callback which will be called for the validation.
* options: Sets the additional options which will be given to the callback.
6) **Zend_Validate_CreditCard** allows you to validate if a given value could be a credit card number.
A creditcard contains several items of metadata, including a hologram, account number, logo, expiration date, security code and the card holder name. The algorithms for verifying the combination of metadata are only known to the issuing company, and should be verified with them for purposes of payment. However, it's often useful to know whether or not a given number actually falls within the ranges of possible numbers *prior* to performing such verification, and, as such, Zend_Validate_CreditCard simply verifies that the credit card number provided is well-formed.
7)The **Ccnum** validator has been deprecated in favor of the CreditCard validator. For security reasons you should use CreditCard instead of Ccnum.
8) **Zend_Validate_Date** allows you to validate if a given value contains a date. This validator validates also localized input.
Supported options for Zend_Validate_Date

The following options are supported for Zend_Validate_Date:

* format: Sets the format which is used to write the date.
* locale: Sets the locale which will be used to validate date values.

Zend ACL Zend_Acl provides a lightweight and flexible access control list (ACL) implementation for privileges management. In general, an application may utilize such ACL's to control access to certain protected objects by other requesting objects.
For the purposes of this documentation:

- a *resource* is an object to which access is controlled.
- a *role* is an object that may request access to a Resource.

Put simply, *roles request access to resources*. For example, if a parking attendant requests access to a car, then the parking attendant is the requesting role, and the car is the resource, since access to the car may not be granted to everyone.
Through the specification and use of an ACL, an application may control how roles are granted access to resources. There are two key components in this ACL process:

- A Front Controller Plugin: This component resolves if the current user has access to the page which is being opened.
- An Action Helper: This component allows you to check whether the current user has access inside a controller.

Zend config Zend_Config is designed to simplify the access to, and the use of, configuration data within applications. It provides a nested object property based user interface for accessing this configuration data within application code. The configuration data may come from a variety of media supporting hierarchical data storage. Currently Zend_Config provides adapters for configuration data that are stored in text files with Zend_Config_Ini and Zend_Config_Xml.

How to create custom plugin

What are Plugins?

• Triggered by front controller events
• Events bookend each major process of the front controller
• Allow automating actions that apply globally

Creating Plugins:

• Extend Zend_Controller_Plugin_Abstract
• Extend one or more of the event methods
♣ Create multi-purpose plugins by extending multiple methods
♣ Create targetted plugins by extending a single method

Zend auth Zend_Auth provides an API for authentication and includes concrete authentication adapters for common use case scenarios.

Zend_Auth is concerned only with *authentication* and not with *authorization*. Authentication is loosely defined as determining whether an entity actually is what it purports to be (i.e., identification), based on some set of credentials. Authorization, the process of deciding whether to allow an entity access to, or to perform operations upon, other entities is outside the scope of Zend_Auth. For more information about authorization and access control with Zend Framework, please see Zend_Acl.

Note: The Zend_Auth class implements the Singleton pattern - only one instance of the class is available - through its static getInstance() method. This means that using the *new* operator and the *clone* keyword will not work with the Zend_Auth class; use Zend_Auth::getInstance() instead.

Cache Zend_Cache provides a generic way to cache any data.

Caching in Zend Framework is operated by frontends while cache records are stored through backend adapters (*File*, *Sqlite*, *Memcache*...) through a flexible system of IDs and tags. Using those, it is easy to delete specific types of records afterwards (for example: "delete all cache records marked with a given tag").

The core of the module (Zend_Cache_Core) is generic, flexible and configurable. Yet, for your specific needs there are cache frontends that extend Zend_Cache_Core for convenience: *Output*, *File*, *Function* and *Class*.