**1)** **Is ZF a component library or a framework?**

ZF is both. Zend Framework provides all the components required for most web applications in a single distribution. But Zend Framework components are also loosely coupled, making it easy to use just a few components in a web application- even alongside other frameworks! Using this use-at-will architecture, we are implementing features commonly found in more monolithic frameworks. In fact, we are currently working on a tooling component for the 1.8 release that will make it simpler to build applications using ZF components, yet will not sacrifice the use-at-will nature of existing ZF components. It’s a testament to the use-at-will architecture of Zend Framework that the tooling component itself can be used standalone.

**2)What is autoloader?**

Autoloader is function that load all the object on start up.

**3) What is use of Zend front controller?**

Routing and dispatching is managed in the front controller. It collects all the request from the server and handles it.

**4) What is the use of Bootstrap?**

Apart from index if we want to do any extra configuration regarding database and other things that is done within bootstrap.

**5) Zend auth**

It is used to authenticate user, for example like admin, general etc.

**6) Zend Acl**

Based on the zend authentication it allows the user to access certain actions.

**7) How do u set Module name, Controller name, and Action name in Zend framework?**

a)$request->setModuleName(‘front’);

b)$request->setControllerName(‘address’);

c) $request->setActionName(‘addresslist’);

**8) Configuration in Zend Framework, application.ini file?**

Configuration can be done in application.ini file in Zend framework. This file in the path application/configs/application.ini.

**9) Checking whether form posted or not in Zend framework?**

$request=$this->getRequest();

$\_GET=$request->getParams();

$\_POST = $request->getPost();

**10) Does Zend Framework support PHP 4?**

No. Zend Framework was built to use all of the sophisticated object oriented features of PHP 5 and take advantage of significant performance and security enhancements.

**11) Fetch last inserted id, fetch all record and fetch a single record.**

$this->\_db->lastInsertId();

$this->\_db->fetchAll($sql);

$this->\_db->fetchRow($sql);

**12) Difference between Zend\_Registry and Zend\_Session?**

The basic difference between these objects is the ‘scope’ in which they are valid:

a)Zend\_Registry:request scope

b) Zend\_Session : session scope

Zend\_Registry is used to store objects/values for the current request. In short, anything that you commit to Registry in index.php can be accessed from other controllers/actions (because EVERY request is first routed to the index.php bootstrapper via the .htaccess file). Config parameters and db parameters are generally prepped for global use using the Zend\_Registry object.

Zend\_Session actually uses PHP sessions. Data stored using Zend\_Session can be accessed in different/all pages. So, if you want to create a variable named ‘UserRole’ in the /auth/login script and want it to be accessible in /auth/redirect, you would use Zend\_Session.

**13) When do we need to disable layout?**

At the time of calling AJAX to fetch we need to disable layout.

$this->\_helper->layout()->disableLayout();

$this->\_helper->viewRenderer->setNoRender(true);

**14) Filters in Zend Framework with Examples?**

The Zend\_Filter component provides a set of commonly needed data filters. It also provides a simple filter chaining mechanism by which multiple filters may be applied to a single datum in a user-defined order.

Example:

//Add an email element

$this->addElement(‘text’, ‘email’, array(

‘label’ => ‘Your email address:’,

‘required’ => true,

‘filters’ => array(‘StringTrim’),

‘validators’ => array(

‘EmailAddress’,

)

));

Other Filters:

Alnum – Zend\_Filter\_Alnum is a filter which returns only alphabetic characters and digits. All other characters are supressed.

Alpha – Zend\_Filter\_Alpha is a filter which returns the string $value, removing all but alphabetic characters. This filter includes an option to also allow white space characters.

**15) Name some Important component in zend framework?**

Uses of Zend\_Controller

Gives the request & reponse methods by using its sub-classes.

$request = new Zend\_Controller\_Request\_Http()

$response = new Zend\_Controller\_Response\_Http()

Uses of Zend\_Date

Date related processing can be done using this component.

Uses of Zend\_File\_Transfer

it provides extensive support for file uploads and downloads.

Uses of Zend\_Db

It is used to doing database related purpose in our appication.

Uses of Zend\_Paginator

Doing the pagination in our application.

Uses of Zend\_Auth

It is used to authenticate a user.

$auth = Zend\_Auth::getInstance();

$results = $auth->authenticate($adapter);

if ($results->isValid()){

/\* user successfully authenticate into login process \*/

}

Zend\_Session\_Namespace

This is a simple proxy class to use API into the Zend\_Session managed $\_SESSION Superglobal.

**16) Where is the model in ZF’s MVC implementation?**

The model component can vary dramatically in responsibilities and data store from one MVC application to the next.

**17) How to call two different views from same action?**

Example1:

Public function indexAction() {

If(condition)

$this->render(‘yourview.phtml’);

Else

Index.phtml;

Example2:

Public function indexAction() {

}

Now in your index.phtml you can have this statement to call other view

$this->action(‘action name’,’controller name’,’module name’,array(‘parameter name’=>’parameter value’));

**18) Can we call a model in view?**

Yes, you can call a model in view. Simple create the object and call the method.

**19) how do u define the library path in zend ?**

create directory ‘library’, and put ‘Zend’ directory in it. Your directory structure will look like this:

wwwroot

application

lib

Zend

public

index.php

now you should add library to your include path. Edit index.php file:

$includePath = array();

$includePath[] = ‘.’;

$includePath[] = ‘./../application’;

$includePath[] = ‘./../library’;

$includePath[] = get\_include\_path();

$includePath = implode(PATH\_SEPARATOR,$includePath);

set\_include\_path($includePath);

**19) Can we rename the application folder ?**

yes you can rename the application folder

**20) Can we move the index.php file outside the public folder?**

yes you can move index.php file outside the public folder.

**21) How to include js from controller and view in zend**

From within a view file: $this->headScript()->appendFile(‘filename.js’);

From within a controller: $this->view->headScript()->appendFile(‘filename.js’);

And then somewhere in your layout you need to echo out your headScript object:

<?=$this->headScript();?>

**22) How to include css from controller and view in zend**

From within a view file: $this->headLink()->appendStylesheet(‘filename.css’);

From within a controller: $this->view->headLink()->appendStylesheet(‘filename.css’);

And then somewhere in your layout you need to echo out your headLink object:

<?=$this->headLink();?>

**23) How do you protect your site from sql injection in zend when using select query?**

You have to quote the strings,

$this->getAdapter ()->quote ( <variable name> );

$select->where ( ” <field name> = “, <variable name> );

OR (If you are using the question mark after equal to sign)

$select->where ( ” <field name> = ? “, <variable name> );

**24) How can you get a module name in bootstrap file.**

$router = new Zend\_Controller\_Router\_Rewrite();

$request = new Zend\_Controller\_Request\_Http();

$router->route($request);

$moduleName = $request->getModuleName();

**1.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.

**2.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.

**3.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.

**4.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

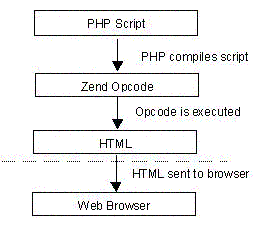
Zend\_Form makes use of several Zend Framework components to accomplish its goals, including Zend\_Config, Zend\_Validate, Zend\_Filter, Zend\_Loader\_PluginLoader, and optionally Zend\_View.

**5.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

**6.zend engine**What is zend engine?

**AnswerZend Engine is used internally by PHP as a complier and runtime engine. PHP Scripts are loaded into memory and compiled into Zend opcodes**

**Explain the architecture of Zend engine with the diagram.**

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.

**7.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](http://framework.zend.com/manual/en/zend.controller.actionhelpers.html#zend.controller.actionhelpers.viewrenderer), but allow them to also use different rules.
* Allow usage without Zend Framework MVC components.

**8.Front controllers**Zend\_Controller\_Front implements a [» Front Controller pattern](http://www.martinfowler.com/eaaCatalog/frontController.html) used in [» Model-View-Controller (MVC)](http://en.wikipedia.org/wiki/Model-view-controller) 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](http://en.wikipedia.org/wiki/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](http://framework.zend.com/manual/en/zend.controller.plugins.html) 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](http://framework.zend.com/manual/en/zend.controller.action.html) 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

**9.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](http://framework.zend.com/manual/en/zend.acl.html)**

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](http://framework.zend.com/manual/en/zend.auth.html)**

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](http://framework.zend.com/manual/en/zend.cache.html)**

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](http://framework.zend.com/manual/en/zend.config.html)**

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](http://framework.zend.com/manual/en/zend.console.getopt.html)**

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](http://framework.zend.com/manual/en/zend.controller.html) and**[**Zend\_View**](http://framework.zend.com/manual/en/zend.view.html)

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](http://framework.zend.com/manual/en/zend.date.html)**

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](http://framework.zend.com/manual/en/zend.db.html)**

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](http://framework.zend.com/manual/manual/en/zend.db.table.html)**

The Zend\_Db\_Table component is a lightweight solution for object-oriented programming with databases. 

* implements the [Table Data Gateway](http://www.martinfowler.com/eaaCatalog/tableDataGateway.html) and [Row Data Gateway patterns](http://www.martinfowler.com/eaaCatalog/rowDataGateway.html)
* discovers database metadata without the need for complex configuration files to maintain
* provides a solution for querying related tables

**[Zend\_Feed](http://framework.zend.com/manual/en/zend.feed.html)**

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](http://framework.zend.com/manual/en/zend.filter.html) and**[**Zend\_Validate**](http://framework.zend.com/manual/en/zend.validate.html)

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](http://framework.zend.com/manual/en/zend.filter.input.html)**

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](http://framework.zend.com/manual/en/zend.form.html)**

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)](http://framework.zend.com/manual/en/zend.gdata.html)**

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](http://framework.zend.com/manual/en/zend.http.html)**

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](http://framework.zend.com/manual/en/zend.json.html)**

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](http://framework.zend.com/manual/en/zend.layout.html)**

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](http://framework.zend.com/manual/en/zend.loader.html)**

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](http://framework.zend.com/manual/en/zend.locale.html)**

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. 

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

**[Zend\_Log](http://framework.zend.com/manual/en/zend.log.html)**

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](http://framework.zend.com/manual/en/zend.mail.html) and**[**Zend\_Mime**](http://framework.zend.com/manual/en/zend.mime.html)

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](http://framework.zend.com/manual/en/zend.measure.html)**

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](http://framework.zend.com/manual/en/zend.memory.html)**

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](http://framework.zend.com/manual/en/zend.pdf.html)**

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](http://framework.zend.com/manual/en/zend.registry.html)**

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](http://framework.zend.com/manual/en/zend.rest.html)**

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](http://framework.zend.com/manual/en/zend.search.lucene.html)**

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!](http://framework.zend.com/manual/en/zend.service.html)**

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](http://framework.zend.com/manual/en/zend.session.html)**

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](http://framework.zend.com/manual/en/zend.translate.html)**

The 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](http://framework.zend.com/manual/en/zend.uri.html)**

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](http://framework.zend.com/manual/en/zend.xmlrpc.html)**

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

**10.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](http://framework.zend.com/manual/en/zend.controller.router.html) to evaluate the request against the registered routes.
* routeShutdown() is called after [the router](http://framework.zend.com/manual/en/zend.controller.router.html) 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](http://framework.zend.com/manual/en/zend.controller.dispatcher.html). 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](http://framework.zend.com/manual/en/zend.controller.dispatcher.html). 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

**11.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

**12.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'=>'Email address cannot be empty')))

->addValidator("emailAddress",true,array('messages'=>array('emailAddressInvalidFormat'=>'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'=>'Password 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

**13.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.

**14. 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:

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

**15.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](http://framework.zend.com/manual/en/zend.config.adapters.ini.html) and [Zend\_Config\_Xml](http://framework.zend.com/manual/en/zend.config.adapters.xml.html).

**16. 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♣

**17.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](http://framework.zend.com/manual/en/zend.acl.html).

**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.

**18.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*.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Is Zend Framework a component library or a framework?**  
  
ZF is both. Zend Framework provides all the components required for most web applications in a single distribution. But Zend Framework components are also loosely coupled, making it easy to use just a few components in a web application- even alongside other frameworks! Using this use-at-will architecture, we are implementing features commonly found in more monolithic frameworks. In fact, we are currently working on a tooling component for the 1.8 release that will make it simpler to build applications using ZF components, yet will not sacrifice the use-at-will nature of existing ZF components.

**What is autoloader?**  
Autoloader is function that load all the object on start up.  
  
**What is use of Zend front controller?**  
Routing and dispatching is managed in the front controller. It collects all the request from the server and handles it.  
  
**What is the use of Bootstrap?**  
Apart from index if we want to do any extra configuration regarding database and other things that is done within bootstrap.  
  
**How you can set Module name, Controller name, and Action name in Zend framework?**

* $request->setModuleName(‘front’);
* $request->setControllerName(‘address’);
* $request->setActionName(‘addresslist’);

**Configuration in Zend Framework, application.ini file?**  
Configuration can be done in application.ini file in Zend framework. This file in the path application/configs/application.ini.  
  
**Checking whether form posted or not in Zend framework?**  
$request = $this->getRequest();  
$getData = $request->getParams();  
$postData = $request->getPost();

$isPost = $request->isPost();  
  
  
**Fetch last inserted id, fetch all record,find and fetch a single record.**  
$this->\_db->lastInsertId();  
$this->\_db->fetchAll($sql);

$this->\_db->find($id);  
$this->\_db->fetchRow($sql);  
  
**Difference between Zend\_Registry and Zend\_Session?**  
  
Zend\_Registry is used to store objects/values for the current request. In short, anything that you commit to Registry in index.php can be accessed from other controllers/actions (because EVERY request is first routed to the index.php bootstrapper via the .htaccess file). Config parameters and db parameters are generally prepped for global use using the Zend\_Registry object.  
  
Zend\_Session actually uses PHP sessions. Data stored using Zend\_Session can be accessed in different/all pages. So, if you want to create a variable named ‘UserRole’ in the /auth/login script and want it to be accessible in /auth/redirect, you would use Zend\_Session.  
  
**When do we need to disable layout?**  
At the time of calling AJAX to fetch we need to disable layout.  
$this->\_helper->layout()->disableLayout();  
$this->\_helper->viewRenderer->setNoRender(true);  
  
  
**How to call two different views from same action?**  
Example1:  
Public function indexAction() {  
If(condition)  
$this->render(‘yourview.phtml’);  
Else  
Index.phtml;  
  
Example2:  
Public function indexAction() {  
}  
Now in your index.phtml you can have this statement to call other view  
$this->action(‘action name’,’controller name’,’module name’,array(‘parameter name’=>’parameter value’));  
  
**Can we call a model in view?**  
Yes, you can call a model in view. Simple create the object and call the method.

$modelObj = new Application\_Model\_User();  
  
**Can we rename the application folder ?**  
yes, we can  
  
**Can we move the index.php file outside the public folder?**  
yes, we can  
  
**How to include js from controller and view in zend?**  
  
From within a view file: $this->headScript()->appendFile(‘filename.js’);  
From within a controller: $this->view->headScript()->appendFile(‘filename.js’);  
And then somewhere in your layout you need to echo out your headScript object:  
<?=$this->headScript();?>  
  
  
**How to include css from controller and view in zend?**  
From within a view file: $this->headLink()->appendStylesheet(‘filename.css’);  
From within a controller: $this->view->headLink()->appendStylesheet(‘filename.css’);  
And then somewhere in your layout you need to echo out your headLink object:  
<?=$this->headLink();?>  
  
**How do you protect your site from sql injection in zend when using select query?**  
  
We have to quote the strings,  
$this->getAdapter ()->quote ( <variable name> );  
$select->where ( ” <field name> = “, <variable name> );  
OR (If you are using the question mark after equal to sign)  
$select->where ( ” <field name> = ? “, <variable name> );

Create Form & Form Elements

//create a form object  
$form = new Zend\_Form  
$form->setAction('/resource/process')  
     ->setMethod('post');  
  
$form->setAttrib('id', 'login');  
?>

//Form support Following elements

button,

checkbox (one/many),

hidden,

image,

password,

radio,

reset,

select (single select/many select),

submit,

text,

textarea

Zend Framework Central Authentication

* Zend Framework  follow MVC Design with Dispatcher.
* Auth / ACL Implement
* Validate & Filter the URL with using dispatcher

*class FilterPlugin extends Zend\_Controller\_Plugin\_Abstract*  
*{*  
*public function preDispatch(Zend\_Controller\_Request\_Abstract $request)*  
*{*  
*$params     = $request->getParams();*  
*$controller = $request->getControllerName();*  
*$action     = $request->getActionName();*  
*$filter = $GLOBALS['filters'][$controller][$action];*  
*$validator = $GLOBALS['validators'][$controller][$action];*  
*$inputdata = new Zend\_Filter\_Input($filter, $validator, $params);*  
*if (!$inputdata* *->isValid()) {*  
*$request->setModuleName('default')*  
*->setControllerName('error')*  
*->setActionName('illegalparam')*  
*->setDispatched(false);*  
*return;*  
*}*  
*}*

What is Zend\_Form

It uses to create a form in zend framework. In this each input/select/radio/checkbox known as element.

Following are the zend form features.

         Filtering and validation

         Ordering

         Element and Form rendering, including escaping

         Element aSnd form grouping

         Element and form-level configuration

Zend\_ACL

Zend\_Acl provides lightweight and flexible access control list (ACL) functionality and privileges management. In general, an application may utilize such functionality to control access to certain protected objects by other requesting objects.   
  
**Resource**  
 creating a Resource is very simple. Zend\_Acl provides Zend\_Acl\_Resource\_Interfaceto facilitate developers' creating Resources. A class need only implement this interface, which consists of a single method, *getResourceId()*;  
Role  
Zend\_Acl providesZend\_Acl\_Role\_Interface to facilitate developers' creating Roles. A class need only implement this interface, which consists of a single method, getRoleId(), in order for Zend\_Acl to consider the object to be a Role.  In Zend\_Acl, a Role may inherit from one or more Roles. This is to support inheritance of rules among Role.

**Example of ACL**

<?php

require\_once 'Zend/Acl.php';

$acl = **new** Zend\_Acl();

require\_once 'Zend/Acl/Role.php';

$acl->addRole(**new** Zend\_Acl\_Role('guest'))

->addRole(**new** Zend\_Acl\_Role('member'))

->addRole(**new** Zend\_Acl\_Role('admin'));

$parents = array('guest', 'member', 'admin');

$acl->addRole(**new** Zend\_Acl\_Role('someUser'), $parents);

require\_once 'Zend/Acl/Resource.php';

$acl->add(**new** Zend\_Acl\_Resource('someResource'));

$acl->deny('guest', 'someResource');

$acl->allow('member', 'someResource');

echo $acl->isAllowed('someUser', 'someResource') ? 'allowed' : 'denied';//allowed

?>

*// Guest may only view content of all controller, Here 'view' can be an array having all the function which guest may access. null can be a string or an array, Here Null means all controller*

$acl->allow($roleGuest, **null**, 'view');

*// Here no controller or action is given, it means Administrator is allowed all privileges*

$acl->allow('administrator');

// Remove the denial of revising latest news to staff

$acl->removeDeny('staff', 'latest', 'revise');

**Assertions**

Sometimes a rule for allowing or denying a Role access to a Resource should not be absolute but dependent upon various criteria. For example, suppose that certain access should be allowed, but only between the hours of 8:00am and 5:00pm. Another example would be denying access because a request comes from an IP address that has been flagged as a source of abuse.

<?php

require\_once 'Zend/Acl/Assert/Interface.php';

class CleanIpaddressAssertion implements Zend\_Acl\_Assert\_Interface

{

public **function** assert(Zend\_Acl $acl, Zend\_Acl\_Role\_Interface $role = null,

Zend\_Acl\_Resource\_Interface $resource = null, $privilege = null)

{

**return** $this->\_isCleanIP($\_SERVER['REMOTE\_ADDR']);

}

protected **function** \_isCleanIP($ip)

{

*// ...*

}

}

$acl->allow(**null**, **null**, **null**, **new** CleanIpaddressAssertion());

zend layout

A website have different pages but have a page where all the footer/header/left/right section is included know as template/layout. Each page is called from that layout. A site may have one or more layout depend on website.

* Automate selection and rendering of layouts.
* Provide different scope for layout related variables and content.eg render(), partial().
* Allow configuration, including layout name, layout script resolution (inflection), and layout.
* You can disable 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.

There are different options to set the options for layout. //static method  Zend\_Layout::startMvc($optionArray)  Zend\_Layout::startMvc($optionIniObject) //constructor $layout = Zend\_Layout($optionArray);  $layout = Zend\_Layout($optionIniObject); //methods $layout = Zend\_Layout();  $layout->setOptions($optionArray);  $layout->setOptions($optionIniObject);  //using accessors $ayout->setLayout('layout) ->setLayoutPath('/views/layouts/mylayout.phtml'); You can use Zend\_Layout as an standalone component but in that case all full features will not available. following are available. Scoping of layout. isolation of layout view scripts from another view scripts

zend-action-helper

A helper can be initialized in several different ways, based on your needs as well as the functionality of that helper. The helper broker is stored as the $\_helper member of Zend\_Controller\_Action; use the broker to retrieve or call on helpers. Some methods for doing so include:  Following are the 3 different but equal method to set the message

*$flashMessenger = $this->\_helper->getHelper('FlashMessenger'); $flashMessenger->addMessage('We did something in the last request');*

 OR

*$flashMessenger = $this->\_helper->FlashMessenger; $flashMessenger->addMessage('We did something in the last request');*

 OR

*$this->\_helper->FlashMessenger('We did something in the last request');*

[**bootstrap**](http://zend-framework-certification.blogspot.in/2012/06/bootstrap.html)

Bootstrap: In bootstrap you can setting up the database, configuring your view and view helpers, configuring your layouts, registering plugins, registering action helpers, cron job & service srcipt. /\*\* Include Bootstrap file in public/index.php \*/ require\_once 'Zend/Application.php'; *// Create application, bootstrap, and run $application = new Zend\_Application(     APPLICATION\_ENV,     APPLICATION\_PATH . '/configs/application.ini' ); $application->bootstrap()             ->run();* /\*\* Include Bootstrap file in public/index.php \*/   /\*in application/bootstrap.php \*/***class****Bootstrap****extends****Zend\_Application\_Bootstrap\_Bootstrap{}****class****Zend\_Application\_Bootstrap\_Bootstrap****extends****Zend\_Application\_Bootstrap\_BootstrapAbstract{}* *abstract class Zend\_Application\_Bootstrap\_BootstrapAbstract     implements Zend\_Application\_Bootstrap\_Bootstrapper,                Zend\_Application\_Bootstrap\_ResourceBootstrapper* How to call a function from bootstrap

***$bootstrap->bootstrap([array](http://www.php.net/array)('foo'));***

In bootstrap class there should be function with name of **\_initFoo()**

[**Design Patern**](http://zend-framework-certification.blogspot.in/2012/06/design-patern.html)

**Design Patern**: It a formal way of documenting a solution to a problem.

Zend Famework mainly use following Patterns.

**Singleton Pattern**: It is a design pattern that restricts the instantiation of a class to one object.

 Example:

*class Database {*

*private static $\_singleton;*

*private $\_connection;*

*private function \_\_construct($host,$user,$pass){*

*$this->\_connection = mysql\_connect($host,$user,$pass);*

*}*

*public static function getInstance(){*

*if (is\_null (self::$\_singleton)) {*

*self::$\_singleton = new Database();*

*}*

*return self::$\_singleton;*

*}*

*}*

*$db = Database::getInstance();*

**Factory Pattern:** In this Pattern we can create an instance of any of one class in-directly from an static function of another class.

example

*$db = Zend\_Db::factory( ...options... );*

[**components**](http://zend-framework-certification.blogspot.in/2012/06/components.html)

**Zend\_Acl**

Provides lightweight and flexible access control list (ACL) functionality and privileges management.**Role**: It is an object that may request to access the Resource. **Resource**: Object which access is controller. e.g. Driver request to access the car. Here driver is Role & Car is resource.

**Zend\_Auth**

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.

**Zend\_Cache**

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

**Zend\_Config**

Zend\_Config simplifies the use of configuration data for web applications.

**Zend\_Console\_Getopt**

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

**Zend\_Controller and Zend\_View**

Controller is for logical part and view is for Html Part.

**Zend\_Date**

Zend\_Date offers a detailed but simple API for manipulating dates and times.

**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.

**Zend\_Db\_Table**

The Zend\_Db\_Table component is a lightweight solution for object-oriented programming with databases.

**Zend\_Feed**

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

**Zend\_Filter and Zend\_Validate**

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

**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.

**Zend\_Form**

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

**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.

**Zend\_Http\_Client**

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

**Zend\_Json**

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

**Zend\_Layout**

Easily provide sitewide layouts for your MVC applications.

**Zend\_Loader**

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

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**Zend\_Session**

Zend\_Session helps manage and preserve session data across multiple page requests by the same client.

**Zend\_Translate**

The Zend\_Translate component provides Zend Framework with message translation functionality.

**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.

**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

[**Model View Controller Architecture**](http://zend-framework-certification.blogspot.in/2012/06/mvc.html)

|  |
| --- |
| [MVC Diagram](http://1.bp.blogspot.com/-aPQzsHMNchg/T-u_w0ea-dI/AAAAAAAAAAc/MyBLw4Js584/s1600/zf_zce_logo.gif) |
| MVC Diagram |

 In **MVC** pattern, we differentiate our logic(controller), Mysql(model) and html(view). **View**: It have all the Html parts of the page. Here we use the variables which we have set in controllers. **Model**: MYSQL queries parts of the page, we make an object of the model in controller & call the functions of the model. after fetching the data, we set it in controller & use in view. **Controller**: It have all the logic part of the site.

[**Zend Framework Certification Topics**](http://zend-framework-certification.blogspot.in/2012/06/zend-framework-certification15-topics.html)

**MVC**

* Components
* Pattern Basics
* Bootstrap
* Zend\_Controller
* Zend\_Layout

**Infrastructure**

* Zend\_Config
* Zend\_Session
* Zend\_Registry
* Zend\_Loader

**Internationalization**

* Zend\_Date
* Zend\_Translate
* Zend\_Locale
* Zend\_Currency
* Zend\_View\_Helper\_ Translate
* Internationalization Performance

**Authentication and Access**

* Zend\_Acl
* Zend\_Auth
* Zend\_Auth Adapters

**Performance**

* Zend\_Cache
* Script inclusion
* Optimization
* Zend\_Memory

**Forms**

* Zend\_Form
* Validation/Filtering
* Decorators
* Elements
* Forms
* Display Groups & Sub Forms
* Configuration
* Internationalization

**Security**

* Cross Site Request Forgery
* Secure Authentication
* Escaping for output
* Cross Site Scripting
* Security Best Practices

**Diagnosis and Maintainability**

* Zend\_Debug
* Zend\_Log

**Filtering and Validation**

* Filtering Chains
* Custom Filters
* Standard Validation Classes
* Validator Chains
* Custom Validators

**search**

* Zend\_Search\_Lucene
* Indexing
* Querying
* Performance

**Database**

* Zend\_Db
* Zend\_Db\_Statement
* Zend\_Db\_Table
* Zend\_Db\_Profiler
* Zend\_Db\_Select
* Table Data Gateway Pattern
* Row Data Gateway Pattern

**Mail**

* Constructing
* Storage
* Message retrieval
* Storage providers
* Message sending

**Coding Standards**

* Coding conventions
* Zend\_Loader

**Web Services**

* XML-RPC Client
* XML-RPC Server
* REST Client
* REST Server
* Zend\_Service Web Services

**What is the difference between Zend\_Auth and Zend\_Acl?**

- Zend\_Auth provides the authentication protocol for the users using many methods whereas; Zend\_Acl is used for authorization purpose.  - Zend\_Auth uses the methods like LDAP, OpenID and HTTP to provide the authentication, whereas Zend\_Acl uses Access Control List for authorization. - Zend\_Auth provide the authentication in the form of verifying and providing the credentials for the user’s system, whereas Zend\_Acl uses list of roles that is being implemented only those who are authorized to it.  - Zend\_Auth provides an environment to the system through which user who is authenticated for use, whereas Zend\_Acl performs some operations on the specific resources that need to be written on the system.  - Zend\_Auth supports the authentication platforms features, whereas Zend\_Acl supports the advanced definitions with features of multiple inheritance and other similar features.

**Write a program that performs the calculations using the SQL function?**

- The SQL functions are used to perform the calculations of the statements using the Zend\_Db\_Select command.  - The expression needs to have parentheses and the command that is given as Zend\_Db\_Select need to provide the statement appropriately.  - The Zend\_Db\_Expr is used to explicitly create the expressions that are given below: $select = $db->select() ->from(array('p' => 'products'), array('product\_id', 'cost\_plus\_tax' => new Zend\_Db\_Expr('p.cost \* 1.08'), ));

**What is the way in which the HTML can be used to form elements?**

- The form elements can be customized using the decorators and using the Zend\_Form for the rendering of the output. - The description decorator can be used to have an instance running that allow the HTML to create the form elements.  - The turning of escape characters is required for the output of the decorator and it can be represented as shown below:  $element->setDecorators(array(array('ViewHelper'), array('Description', array('escape', false)), array('Errors'), array('HtmlTag', array('tag' => 'dd')), array('Label', array('tag' => 'dt')), )); - The element can be set to use the description by adding the extra HTML to this and it can be done using this: $element->setDescription('<strong>Hello World</strong>');

**What is the procedure to see that the optional file is detected?**

- There is a use of receive() method that returns true for the file element which is not required.  - The file can be neglected when using the method mentioned above, as soon as the file gets opened it detected. - The method returns the false in case of any error or the file hasn’t been copied or uploaded directly.  - There are other methods used to provide the detection of the file using the elements in normal HTML file.  - The methods can detect the file and provide the tools to make the file being detected.

**What are the ways in which the file can be detected as being uploaded or not?**

- There are certain functions used for the detection of the file and check whether the file is being uploaded properly or not. - The use of isUploaded() function with the pre-defined parameters are used to check and it returns the boolean result.  - The use of getFileName() method is used to provide the latest releases of the file and returns the NULL. - The use of getFileInfo() provides a way to make an empty file key that can be used to check whether the file is being uploaded or not.  - The flag is set for the isUploaded() method to false that allow the detection to be made easy.

**What is the use of Zend framework?**

- Zend framework is a robust framework that provides the components used by the web applications.  - It provides inbuilt libraries and functions to make the programming and integration easier.  - Zend framework consists of loosely coupled components that can be used to easily integrate in an application. - It is used for the web application that uses lots of functions and libraries without writing much of the coding.  - It provides tools that can be used to have the standalone component based architecture.

**What is the function of model in MVC architecture?**

- MVC is made up of model, view and controller components to, represents the data in the architectural form. - The model component can vary from one requirement to another for MVC application.  - The function model represents the application that has the abstract processing of the components.  - It uses the zend framework that defines MVC architecture at one place and allow the applications to use it.  - It is having the model interface, class and other components that used to identify the value and provide the limitations on the use.

**Write a program to retrieve the view object within plugin?**

- Zend uses the framework that allows easy retrieving of the object within the arbitrary code.  - It uses the action helper and other class methods to view the object within the plugin.  - The method ViewRenderer() is used by default in the zend framework to view and render the components. $viewRenderer = Zend\_Controller\_Action\_HelperBroker::getStaticHelper('viewRenderer'; $view = $viewRenderer->view;

**What is the function of action helper in Zend?**

- Zend allows the creation of application using the framework that provides MVC architecture.  - The action helper is used to instantiates and store the object i.e Zend\_View and other objects that are related to view.  - It also provides ways to inject the instantiation process in different objects by the help of action helper.  - The view object is stored in the view property provided by the action helper and a view instance is created using this.  - ViewRenderer instance is being created using the action helper function and it allows creating sync between all the objects

**Write a program to show the function of action helper?**

- Action helper is used to create the application and instantiate the object of it using the Zend\_View. - The object is used and stored using the ViewRenderer’s property that creates an instance and provides more options.  - It includes the dispatch process before any action is being dispatched from the view and the example code is given as: if (!isset($viewRenderer->view)) { $viewRenderer->initView(); } $view = $viewRenderer->view;

**What is the function Lucene?**

- Lucene is an open source binary format used to have features for high performance and provide other tools to manage the resources.  - It provides fully featured text search engine that can be utilized by the applications and be used to execute it.  - It provides the storing of the data and the search indexes by providing the standardized query format for querying for other indexes. - Lucene is a search engine provider that allows the functionality to be implemented for the indexes.  - Lucene uses the Apache project and having the zend library with the name Zend\_Search\_Lucene.

**Why are in-built libraries used for the web services?**

- Zend framework is used to solve the development problems of the web applications and it provides resources for use.  - Web services are used for the development environment and to make it accessible to all the platforms.  - Web services are being provided by the zend framework and used in the application development process.  - The users uses the web services to access their work and other services and components.  - It uses the Zend\_Registry and Zend\_Log methods to have the registry setup and log being maintained while setting up the libraries for the web services.

**What are the steps kept in mind while designing the Zend\_Form?**

- The markups are placed in the proper places with each element and with the use of form.  - The markup should be generated for the displaying of the element and in the form as well.  - Inject the values in the element markup with the metadata to design it more effectively.  - The reporting errors need to be kept in check to see the repetitive task that is being done using the code.  - The form solution is being provided to address the actions and other features that need to be addressed.

**Write a program to show the instantiation of the window class?**

- The decorator is used to extend the functionality of the class that wraps up the structure. - The decorator extends the class and its functionality by wrapping up in a defined pattern.  - The use of it allows the user to add or modify the existing behaviour of the API that is same.  - Decorators wrap other that identifies the original object and follows with the same API.  - The WindowDecorator is being created to display the window, by showing scrollbars, window title, etc;  $window = new WindowScrollbarDecorator( new WindowTitleDecorator( new WindowDecorator( new Window() ) ) );

**What are the methods used in Decorator?**

- The methods used to render the view of the final window are shown as: $window->render(); - The render view is being executed by the use of WindowScrollbarDecorator::render() method.  - It uses the WindowTitleDecorator::render() method, to be used as the decorator for the title and the rendering of it.  - WindowDecorator::render() method, is used as a Decorator of the window class that allow the main execution of the application.  - The rendering of the window is being done by using the function or method as: Window::render() that allows the simple use of state object.

**What is being presented by the Decorators?**

- Decorators are used to provide the Zend\_Form in the case of developer’s use and provide them the functionality to write the code.  - It determines the display and provides the final output of the input that is being provided by the developer.  - The solution is being managed by the modified decorator and it is used to decorate the object.  - The content can be generated by using the decorate methods and function or using the metadata.  - The metadata is being used with the elements or objects through which the content will be generated.

**What are the basic operations performed by the Decorator?**

- Decorator provides the ability to append, prepend, and replace the content that is passed to the method.  - The content that is being initially placed provides the empty string and being processed with the decorator. **The operations that can be performed are shown below:** - Append: this allows the appending of the functions and methods that can be used to build an application. - Prepend: this allows the content to be placed at the beginning of the file. - Replace: this operation will allow the replacement of the content from any of the content that is present.

**What are the default methods provided by decorators?**

- Decorators are used in the application to provide the functionality and make elements to work.  - The decorators being provided as default are as follows: - ViewHelper: is a method that allows the viewing of the helper files used. It allows the replacement of the content that is being provided.  - Errors: provide the error codes and messages that came during the execution of the application. It provides appending of the content.  - HTMLTag: that allows the tags to be written by using the parameters and attributes.  It provides the wrapping up of the content and decreasing the length of the code.  - Labels: are used to provide the informative keywords as it provides prepending of the content that is being provided.

**Write a program to show the execution of the application?**

- The execution of the program can be done by using the methods of Decorators.  - Decorators provide the methods like errors, htmltag, labels, etc. to help the developers to use it and execute their applications. - The program is written like this: $label->render($htmlTag->render($errors->render($viewHelper->render('')))) <input name="hello" id="hello" type="text" value="" /> <div class="error"><ul> <li>...</li> </ul></div> <dd> <input name="hello" id="hello" type="text" value="" /> <div class="error"><ul> <li>...</li> </ul></div> </dd>

**What are the elements used in customizing the output of using the standard decorators?**

The elements used by the decorator to be used in the application are as follows: - ViewHelper: It allows the replacement of the content that is being provided.  - Errors: It provides appending of the content.  - HtmlTag (

): It provides the wrapping up of the content and decreasing the length of the code.  - Label: labels are used with the wrapping of the code using the tag

**.  The form objects are also involved in it and are as follows: - FormElements: are used to iterate the elements, display groups, and sub-forms that are used to render the elements used above.**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

<http://www.f2finterview.com/web/ZendFramework/?page=1>

<http://www.withoutbook.com/Technology.php?tech=46&subject=Zend%20Framework%20Interview%20Questions%20and%20Answers>