Localization of E-Governance Project

Overall Architecture

Low-Level e-platform model

March 30, 2009

Vishanta Rayamajhi
International IT Expert
UNDP Bhutan

Submitted to:

Department of Information Technology
Ministry of Information and Communication
Royal Government of Bhutan

UNDP Funded Project

Table of Contents

Introduction

This report is a comprehensive architecture of the e-platform model. This gives an insight on the core architecture and flow of the e-platform model.

The report describes the e-platform model in correlation with Zend Framework core engines, Zend_Controller, Zend_Auth, Zend_Acl and so on. The architecture also describes the core elements of the database schema, namely, roles, resources and privileges of the e-platform model as well as multi-layered verification process flow. UML and ER diagrams are also used to describe the core elements of the e-platform model.

Security considerations and implementation is also described in detail.

1. The e-platform model

1.1 What is e-platform model?

The standardized *low-level e-platform* model serves as a base for deploying and developing custom e-service prototypes. It integrates several generic service modules that make the e-service prototypes development faster and without the need to develop from scratch. The development becomes easier and existing generic service modules or components of e-platform model is tailored to meet specific business needs at the web application level. The e-platform, in essence, is a building block for e-services developments to meet business processes of an organization.

The e-platform contains a set of standard components which implement concrete functionality for utilization within e-service applications. Implementation of such application amounts to selecting the proper components and requires minor configuration to achieve the business needs of the e-service prototypes.

The e-platform model is designed to speed up the development of scalable and maintainable online web applications. The model is based on enterprise application architecture, with the intention of tailoring solutions using standard components. The solution is adaptable to specific business needs, both in terms of functionality and integration with existing systems.

The concept of separation will be availed extensively ensuring e-platform flexibility, maintainability and extensibility. Processes and functions will separated meaning that changes in processes can easily be made. Business logic and design will be separated enabling the maintainability of the e-platform to bring changes in design or business process changes to be done independently by concerned stakeholders.

1.2 Functionality of e-platform model

The framework includes a number of standard components. These components offer concrete business level functionality which is usable at application level.

A set of chosen components are configured and combined within the framework to provide the required functionality of the application. In this way the development effort is considerably diminished.

As the components themselves do not change they are effectively re-used within a number of different applications. This is known as *component-based services* and component / *code re-usability*. This is also called *SOA* (*Service-Oriented Architecture*) and provides methods for web applications (e-service prototypes) development and integration where applications group functionality around business processes and package these as interoperable services. Common service modules are User Management and Approval system. The aim is to separate functions into distinct units or services, and developers can combine and reuse them in the production of business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services.

In this way the components will be improved over several years and as such will reach a high level of reliability, availability, maintainability and security.

2. e-platform development in Zend Framework

2.1 What is Zend Framework?

Zend Framework is a simple, straightforward, open-source software framework for PHP 5 designed to eliminate the tedious details of coding and let you focus on the big picture. Its strength is in its highly-modular MVC design, making your code more reusable and easier to maintain.

The Zend Framework is a PHP library for building PHP web application. It provides a set of components to enable developers to build PHP applications more easily which will be easier to maintain and extend over the lifetime of the application.

Zend Framework was designed and built to improve developer productivity. Unlike other frameworks that require large configuration files to work, most aspects of a Zend Framework application can be defined at runtime using simple PHP commands. This saves developers time because instead of complex configuration files controlling every aspect of the application, developer only configure the parts that deviate from the norm. Zend Framework was written entirely in PHP 5. It will not run on any server that does not have a minimum of PHP 5.1.4 installed.

MVC stands for Model View Controller. It is a design pattern. A design pattern is a code structure that allows for common coding frameworks to be replicated quickly. One can think of a design pattern as a skeleton or framework on which applications will be built. PHP Zend Framework is a framework based on MVC architecture.

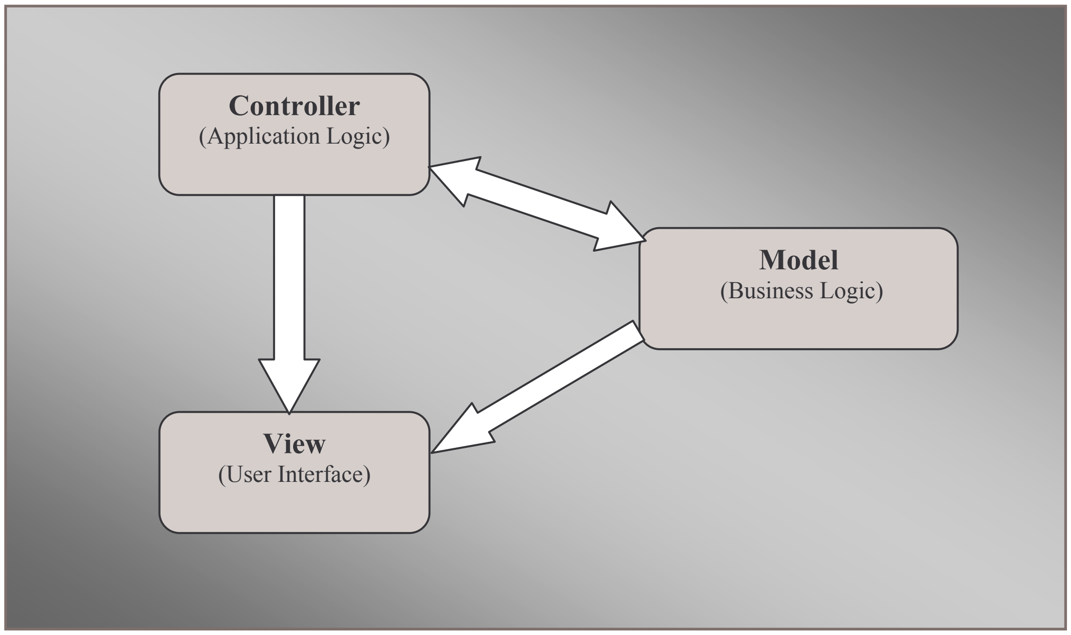

Fig : The MVC pattern of Zend Framework

3. Why Zend Framework

The Zend Framework is essentially a hybrid framework and as such can be used in a much larger range of projects than strict "application frameworks". While many components in Zend Framework can be used stand-alone like a component library; it is, at its core an implementation of the "Model-View-Controller" (MVC) pattern.

Zend Framework introduces a standardized set of components that allow for easy development of web applications. These applications can be easily developed, maintained and enhanced.

The key features of the Zend Framework are:

1. Everything in the box
2. Modern design
3. Easy to learn
4. Full documentation
5. Simpler Development
6. Rapid development

3.1 Everything in the box

The Zend Framework is a comprehensive full stack framework that contains everything you need to develop your application. This includes a robust MVC component to ensure that your website is structured according to best practices. Accompanying the MVC component, there are components for authentication, searching, localization, PDF creation, email and connecting to web services, along with a few other more esoteric items.

3.2 Modern design

The Zend Framework is written in object-oriented PHP5 using the modern design techniques, known as design patterns. Software design patterns are recognized high level solutions to design problems and, as such, are not a specific implementation of the solution. The actual implementation depends on the nature of the rest of the design. The Zend Framework makes use of many design patterns and its implementation has been carefully designed to allow the maximum flexibility for application developers without making them do too much work!

The framework recognizes the PHP way and doesn't force you into using all the components, so you are free to pick and choose between them. This is especially important as it allows you to introduce specific components into an existing site. The key is that each component within the Framework has very few dependencies on other components.

3.3 Easy to learn

Zend Framework is modular and has a design goal of simplicity which makes it easy to learn, one step at a time. Each component doesn't depend on lots of other components and so is easy to study. The design of each component is such that you do not need to understand how it works in its entirety before you can use it and benefit from it. Once you have some experience of using the component, building up to use the more advanced features is straight-forward as it can be done in steps. This is the key to reducing the barrier to entry for most users.

3.4 Full documentation

No matter how good the code is, lack of documentation can kill a project through lack of adoption. The Zend Framework is aimed at developers who do not want to have to dig through all the source code to get their job done and so the Zend Framework puts documentation on an equal footing with the code. This means that the core team will not allow new code into the framework unless it has accompanying documentation.

There are two types of documentation supplied with the framework: API and end-user. The API documentation is created using PHPDocumenter and is automatically generated using special “docblock” comments in the source code. These comments are typically found just above every class, function and member variable declaration. The key advantage of using docblocks is that IDEs such as PHPIDE in Eclipse or Zend’s Studio are able to supply auto-completion tool tips whilst coding and so improve developer productivity.

3.5 Simpler development

As we have noted, one of PHP’s strengths is that developing simple dynamic web pages is very easy. This has enabled millions of people to have fantastic websites who may not have had them otherwise. The ability of PHP programmers range from people who are beginners to programming through to enterprise developers needing to meet their deadlines. The Zend Framework is designed to make development simpler for every type of developer. So how does it make development simpler? The key feature that the framework brings to the table is tested, reliable code that does the “grunt” work of an application. This means that the code you write is the code you need for your application. The code that does the “boring” bits is taken care of for you and is not cluttering up your code.

3.6 Rapid Development

The Zend Framework makes it easy to get going on your web application or add new functionality to a current website. As the framework provides many of the underlying components of an application, you are free to concentrate on the core parts of your application, rather than on the underlying foundation. Hence, it is easy to get started quickly on a given piece of functionality and immediately see the results.

Another way the framework speeds up development is that the default use of most components is the common case. In other words, you don’t have to worry having to set lots of configuration settings for each component just so that you can get started using it.

4. Top-Level view of e-platform Architecture

4.1 Architecture

The e-platform model is best described as common, industry-wide, open-standards-based, interoperable platform facilitating the reliable and pervasive availability of, access interfaces with, and processing for, the various distributed information processing environment. It defines various technologies required to deliver individual agencies' and the State's business application systems and services to the citizens. It allows individual agencies to deploy and support effective and efficient end-user access interfaces to business application systems, as well as providing the processing capability to execute business application systems, while increasing the use of e-government solutions and maintaining traditional methods of service delivery to citizens.

Platform is a complete environment that provides rich functionality by interacting with the existing PHP in a simple and generic way. Platform is a non-intrusive extension to an existing environment with minimal overhead that helps obtain enhanced performance and reliability.

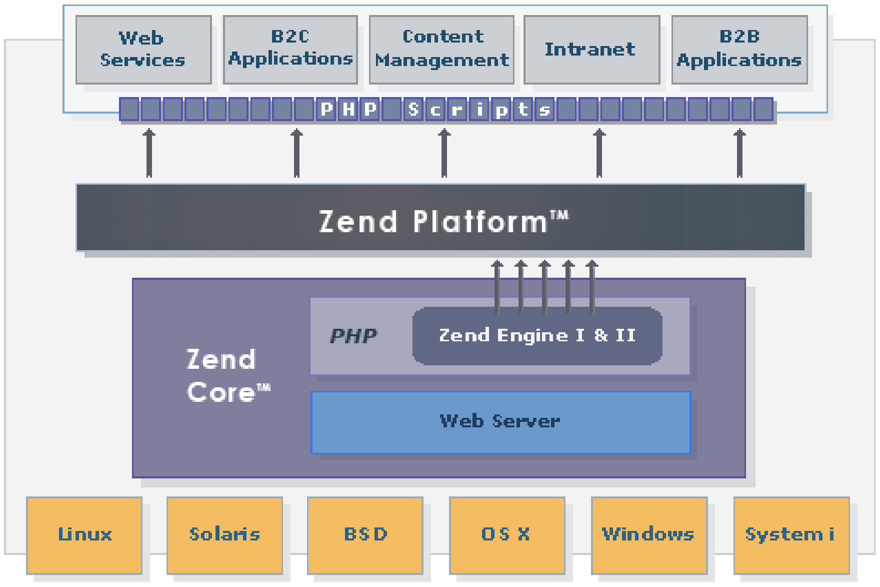

Fig - Platform and the PHP-enabled Enterprise

Platform extends the Zend Engine with the organization's execution environment, providing the platform on which to base the e-Platform model. Basically the inner most layer is the Hardware and on top of which the Kernel or the operating system being stacked. The PHP layer which stands as the backbone for Zend Framework is established on top of the Kernel. Finally the e-platform model which has been total based on Zend Framework.

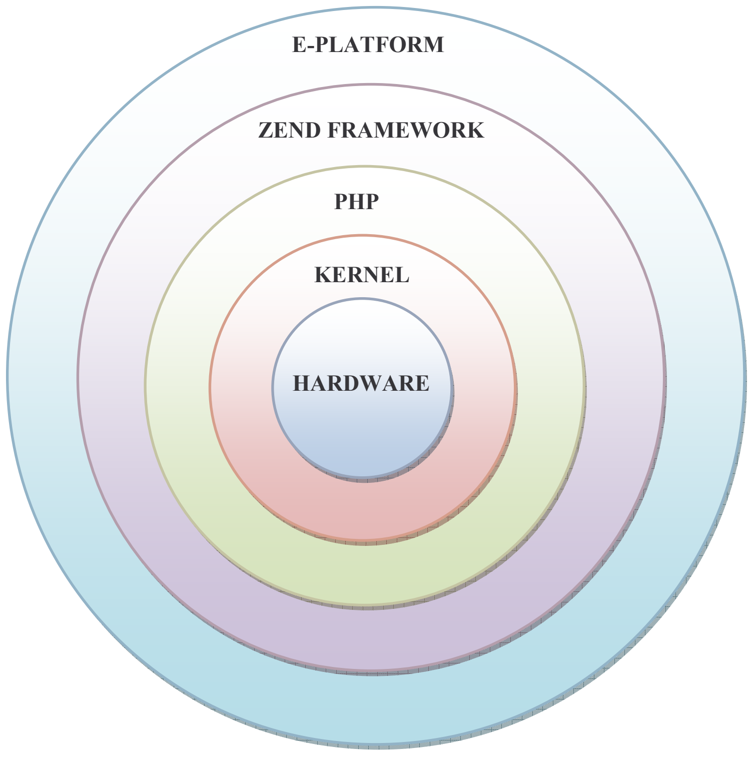

Fig - Top Level view of e-platform architecture

4.2 The Core Components

The core components of Zend provide a full-features Model-View-Controller (MVC) system for building applications that separate out the view templates from the business logic and controller files. There are three families of classes that make up the MVC system: Zend_Controller (Controller), Zend_View (View) and Zend_Db (Model). Figure below shows the basics of the Zend Framework's MVC system.

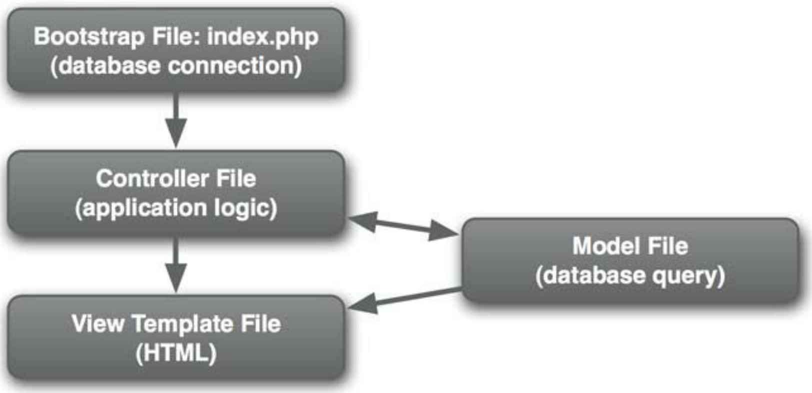

Fig - MVC

The Zend_Controller family of classes provides a front controller design which dispatches requests to controller actions (also known as commands) so that all processing is centralized. The controller supports plug-ins at all levels of the process and has built in flex-points to enable you to change specific parts of the behaviour without having to do too much work.

The view template system is called Zend_View which provides a PHP based template system. This means that, unlike Smarty, all the view templates are written in PHP. Zend_View provides a helper plugin system to allow for creation of reusable display code. It is designed to allow for overriding for specific requirements, or even replacing entirely with another template system such as Smarty.

Zend_Db_Table implements a table row gateway pattern to form the basis of the model within the MVC system. The model provides the business logic for the application which is usually database-based in a web application. Supporting Zend_Db_Table is Zend_Db which provides object oriented database independent access to a variety of different databases, such as MySQL, PostgresSQL, SQL Server and Oracle.

4.3 The Zend Framework’s Controller

The Zend Framework’s front controller code is spread over a number of classes that work together to provide a very flexible solution to the problem of routing a web request to the correct place to do the work. Zend_Controller_Front is the foundation and it processes all requests received by the application and delegates that actual work to action controllers.

4.4 Zend_Controller_Front

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.
Zend_Controller_Front is the foundation and it processes all requests received by the application and delegates that actual work to action controllers.

The Zend_Controller workflow is implemented by several components. While it is not necessary to completely understand the underpinnings of all of these components to use the system, having a working knowledge of the process is helpful.

Zend_Controller_Front orchestrates the entire workflow of the Zend_Controller system. It is an interpretation of the FrontController pattern. Zend_Controller_Front processes all requests received by the server and is ultimately responsible for delegating requests to ActionControllers (Zend_Controller_Action).

5. Details of e-platform model Architecture

5.1 The Model-View-Controller Design Pattern

The Zend Framework controller system is an implementation of the Model-View-Controller software design pattern. A software design pattern is a standard general solution to a common problem. This means that whilst the exact implementation will differ, the concepts used to solve a problem using a given pattern will be the same. The MVC pattern describes a way to separate out the key parts of an application into three main sections.

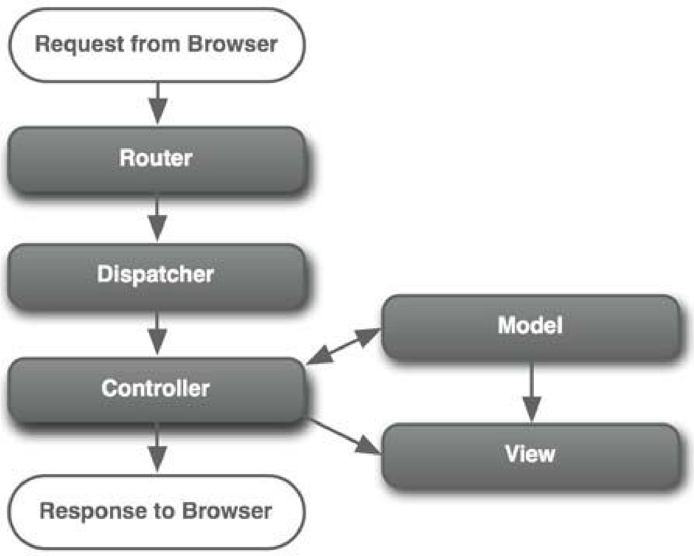

Figure - MVC pattern diagram showing the three main sections of a web application along with the dispatcher that find the correct controller to be executed in response to a request.

5.1.1 The Model

The model part of the MVC pattern is all the code that works behind the scenes related to how this particular application works. This is known as business logic. This is the code that decides how to apply the shipping cost to an e-commerce order or the code that knows that a user has a first name and a surname. It follows therefore that retrieving and storing data to a database is within the model layer. In terms of the code, the Zend Framework provides the Zend_Db_Table class which provides table level access to the database and allows for easily manipulating the data used by the application.

5.1.2 The View

The view is the display logic of the application. For a web application, this is usually the HTML code that makes up the web pages, but can include, say, XML that is used for an RSS feed. Also, if the website allows for export in CSV format, the generation of the CSV would be part of the view. The view files themselves are known as templates as they usually have some code that allows for the displaying of data created by the model. It is also usual to move the more complex template related code into functions known as View Helpers, View Helpers improve the re-usability of the view code. By default the Zend Framework's view class (Zend_View) uses PHP within the template files, but another template engine such as Smarty or PHPTAL may be substituted.

5.1.3 The Controller

The controller is the rest of the code that makes up the application. For web applications, the controller code is the code that works out what to actually run in response to the web request. For Zend Framework applications, the controller system is based on the design pattern known as Front Controller which uses a handler (Zend_Controller_Front) and action commands (Zend_Controller_Action) which work together in tandem. The front controller handler accepts all server requests and runs the correct action function within the action command. This process is known as routing and dispatching. The action class is responsible for a group of related action functions which perform the "real" work required from the request. Within the Controller of the Zend Framework, it is possible to have a single request result in the dispatch of multiple actions.

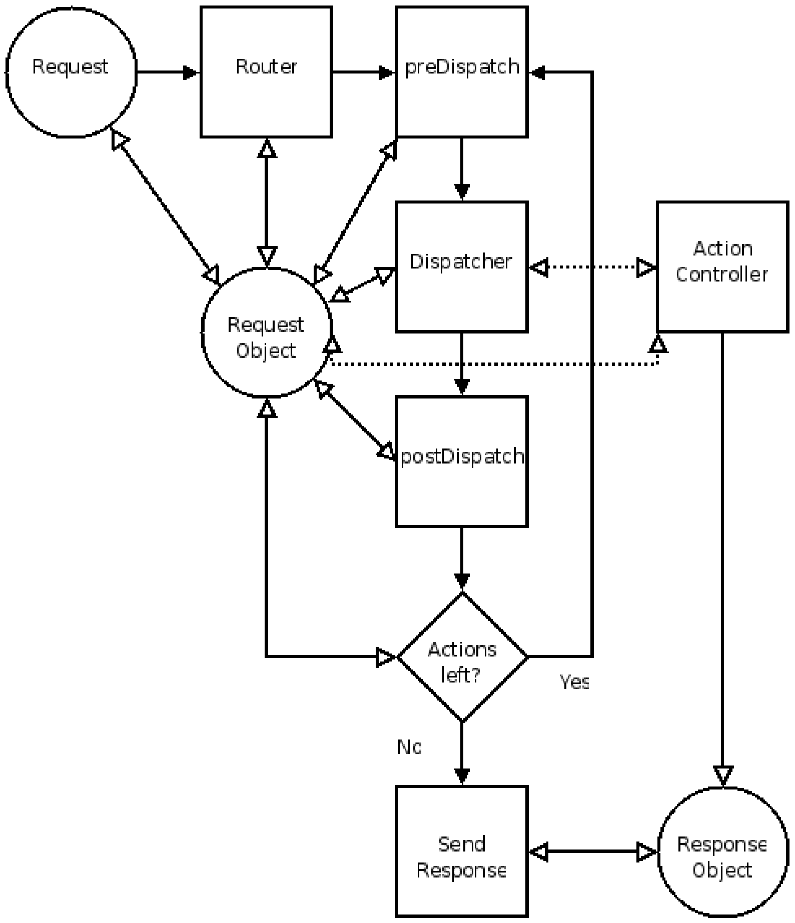

Fig – Request and Response handling in the e-platform model

5.1.3.1 Request

The request is encapsulated within an instance of Zend_Controller_Request_Http which provides access to the entire HTTP request environment. A request environment is all the variables received by the application from outside the application along with relevant controller parameters such as the controller and action router variables.
The HTTP request environment contains all the super globals ($_GET, $_POST, $_COOKIE, $_SERVER and $_ENV) along with the base path to the application. The router also places the module, controller and action names into the request object once it has worked them out. Zend_Controller_Request_Http provides the function getParam() to allow the application to collect the request variables and so the rest of the application is protected from a change in environment. For example, a command line request environment wouldn't contain the HTTP specific items, but would include the command line arguments passed to the script. Thus the code: $items = $request->getParam('items'); will work unchanged when run as a web request or as a command line script.

In general, the request object should be treated as read only to the application as, implicitly, the values set by the user shouldn't be changed.

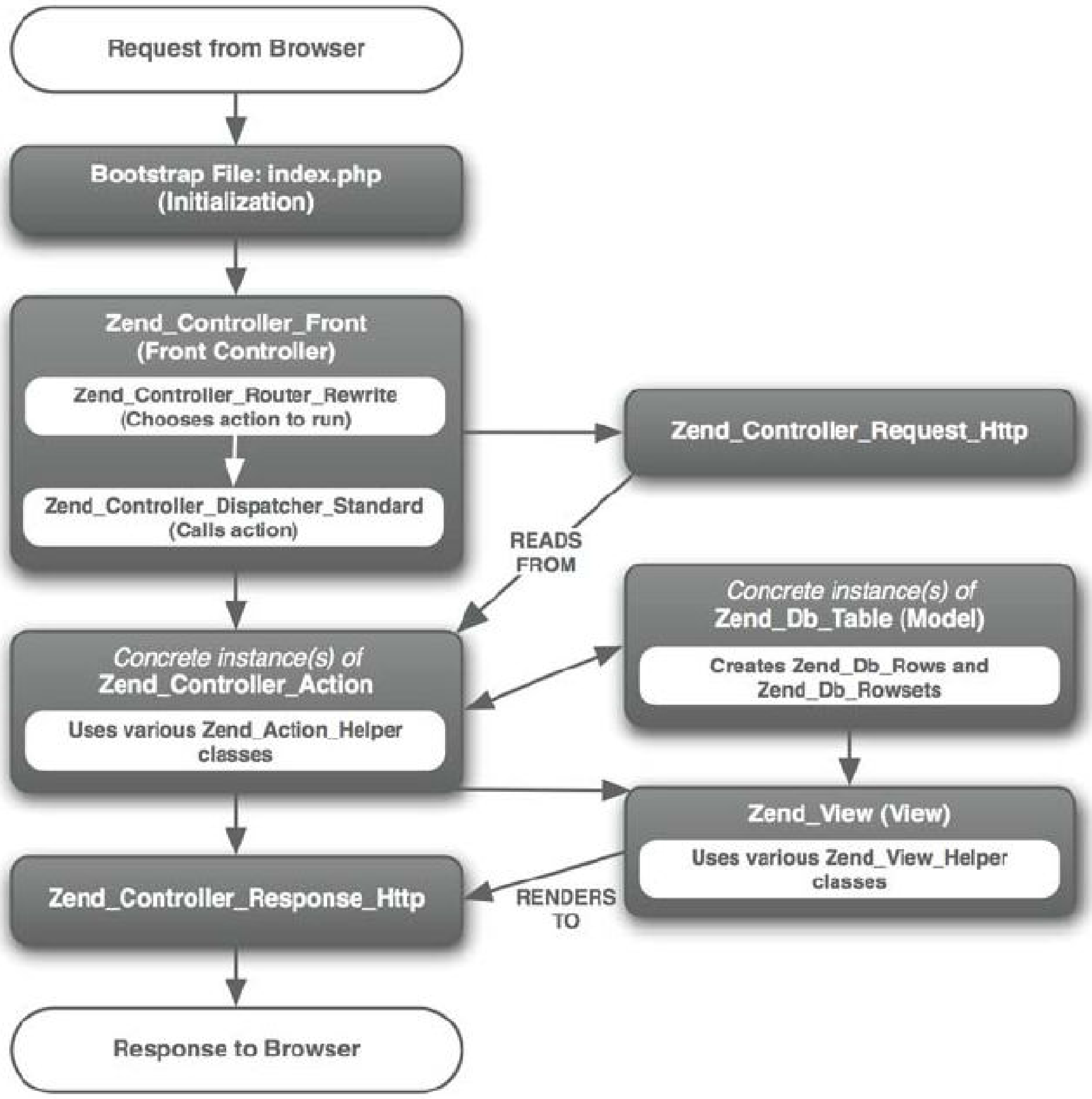
Request from Browser
Bootstrap File: index.php (Initialization)
Zend_Controller_Front (Front Controller)
Zend_Controller_Router_Rewrite (Chooses action to run)
Zend_Controller_Dispatcher_Standard (Calls action)
Zend_Controller_Request_Http
READS FROM
Concrete instance(s) of Zend_Db_Table (Model)
Creates Zend_Db_Rows and Zend_Db_Rowsets
Concrete instance(s) of Zend_Controller_Action
Uses various Zend_Action_Helper classes
Zend_View (View)
Uses various Zend_View_Helper classes
Zend_Controller_Response_Http
RENDERS TO
Response to Browser

Figure : MVC: the Zend Framework way

5.1.3.2 Bootstrapping

Bootstrapping is the term used to describe starting the application up. With the Front Controller pattern, this file is the only file needed in the web root directory and so is usually called index.php. As this file is used for all page requests, it is used for setting up the application's environment, setting up the Zend Framework's controller system and then running the application itself.

Initially, the environment is set up correctly to ensure that all errors or notices are displayed. PHP 5.1 introduced new time and date functionality that needs to know where in the world we are. There are multiple ways to set this, but the easiest user-land method is date_default_timezone_set().

The Zend Framework is written with the assumption that the library directory is available on the php_include path. There are multiple ways of doing this and the fastest for a global library is to alter the include_path setting directly in php.ini. A more portable method, especially if you use multiple versions of the framework on one server, is to set the include path within the bootstrap. The Zend Framework applications does not depend on any particular file, however it is useful to have a couple of helper classes loaded early. Zend_Loader::loadClass() is used "include" the correct file for the supplied class name. The function converts the underscores in the class's name to directory separators and then, after error checking, includes the file. Hence the code line Zend_Loader::loadClass('Zend_Controller_Front'); and include_once 'Zend/Controller/Front.php'; have the same end result. Zend_Debug::dump() is used to output debugging information about a variable by providing a formatted var_dump() output.

The final section of the bootstrap sets up the front controller and then runs it. The front controller class, Zend_Controller_Front implements the Singleton design pattern. This means that the class definition itself ensures that there can only be one instance of the object allowed. A Singleton design is appropriate for a front controller as it ensures that there is only ever one class that is processing the request. One of the consequences of the Singleton design is that you cannot use the new operator to instantiate it and must, instead, use the getInstance() static member function.

The front controller has a feature that captures all exceptions thrown by default and stores them into the Response object that it creates. This Response object holds all information about the response to the requested URL and for HTML applications this is the HTTP headers, the page content and any exceptions that were thrown. The front controller automatically sends the headers and displays the page content when it finishes processing the request.

5.1.3.3 Routing

Routing is the process of determining which controller's action needs to be run in order to satisfy the request. This is performed by a class that implements Zend_Controller_Router_Interface and the framework supplies Zend_Controller_Router_Rewrite which will handle most routing requirements. Routing works by taking the part of the URI after the base URL (known as the URI endpoint) and decomposing it into separate parameters. For a standard URL such as http://example.com/index.php?controller=news&action=list the decomposition is done by simply reading the $_GET array and looking for the 'controller' and 'action' elements. As a modern framework, it is expected that most applications built using the Zend Framework will use pretty URLs of the form http://example.com/news/list. In this case, the router will use the $_SERVER['REQUEST_URI'] variable to determine the which controller and action has been requested.

5.1.3.4 Dispatching

Dispatching is the process of actually calling the correct function in the correct class. As with everything in the Zend Framework, the standard dispatcher provides enough functionality for nearly every situation, but if you need something special, it is easy to write your own and fit it into the front controller. The key things that the dispatcher controls are formatting of the controller class name, formatting of the action function name and calling the action function itself.

Zend_Controller_Dispatcher_Standard is where the rules concerning case are enforced, such that the name format of the controller is always TitleCase and only contains alphabetic characters. The dispatcher's dispatch() method is responsible for loading the controller class file, instantiating the class and then calling the action function within that class. Hence, if you decided that you wanted to reorganize the structure so that each action lived in its own class within a directory named after the controller, you would supply your own dispatcher.

Zend_Controller_Dispatcher_Interface is used to define dispatchers. Dispatching is the process of pulling the controller and action from the request object and mapping them to a controller file/class and action method in the controller class. If the controller or action does not exist, it handles determining default controllers and actions to dispatch.

The actual dispatching process consists of instantiating the controller class and calling the action method in that class. Unlike routing, which occurs only once, dispatching occurs in a loop. If the request object's dispatched status is reset at any point, the loop will be repeated, calling whatever action is currently set in the request object. The first time the loop finishes with the request object's dispatched status set (boolean true), it will finish processing.

The default dispatcher is Zend_Controller_Dispatcher_Standard. It defines controllers as MixedCasedClasses ending in the word Controller, and action methods as camelCasedMethods ending in the word Action: FooController::barAction(). In this case, the controller would be referred to as foo and the action as bar.

5.1.3.5 The Action

Zend_Controller_Action is an abstract class that all action controllers are derived from. Zend_Controller_Action is the base action controller component. Each controller is a single class that extends the Zend_Controller_Action class and should contain one or more action methods. The dispatcher enforces that your action controllers derive from this class to ensure that it can expect certain methods to be available. The action contains an instance of the request for reading parameters from and an instance of the response for writing to. The rest of the class concentrates on ensuring that writing actions and managing changes from one action to another one are easy to do. There are accessor functions to get and set parameters, and redirection functions to redirect to another action or another URL entirely.

Assuming that the standard dispatcher is used, the action functions are all named after the action's name with the word "Action" appended. You can therefore expect a controller action class to contain functions such as indexAction(), viewAction(), editAction(), deleteAction() etc. Each of these is discrete functions that are run in response to a specific URL. There are, however, a number of tasks that you will want to do regardless of which action is run. Zend_Controller_Action provides two levels of functionality to accommodate this requirement: init() and pre/postdispatch(). The init() function is called whenever the controller class is constructed. This makes it very similar to the standard constructor, except that it does not take any parameters and does not require the parent function to be called.

preDispatch() and postDispatch() are a complementary pair of functions that are run before and after each action function is called. For an application where only one action is run in response to a request, there is no difference between init() and preDispatch() as each are only call once. If, however, the first action function uses the _forward() function to pass control to another action function, then preDispatch() will be run again, but init() will not be. To illustrate this point, we could use init() to ensure that only administrators are allowed access to any action function in the controller and preDispatch() to set the correct view template file that will be used by the action.

The workflow of Zend_Controller is relatively simple. A request is received by Zend_Controller_Front, which in turn calls Zend_Controller_Router_Rewrite to determine which controller (and action in that controller) to dispatch. Zend_Controller_Router_Rewrite decomposes the URI in order to set the controller and action names in the request. Zend_Controller_Front then enters a dispatch loop. It calls Zend_Controller_Dispatcher_Standard, passing it the request, to dispatch to the controller and action specified in the request (or use defaults). After the controller has finished, control returns to Zend_Controller_Front. If the controller has indicated that another controller should be dispatched by resetting the dispatched status of the request, the loop continues and another dispatch is performed. Otherwise, the process ends.

5.1.3.6 The Response

The final link in the front controller chain is the response. For a web application Zend_Controller_Reponse_Http is provided, but if you are writing a command line application, then Zend_Controller_Response_Client would be more appropriate. The response object is very simple and is essentially a bucket to hold all the output until the end of the controller processing. This can be very useful when using front controller plugins as they could alter the output of the action before it is sent back to the client.

Zend_Controller_Response_Http contains three types of information: header, body and exception. In the context of the response, the headers are HTTP headers, not HTML headers. Each header is an array containing a name along with its value and it is possible to have two headers with the same name but different values within the response's container. The response also holds the HTTP response code (as defined in RFC 2616) which is sent to the client at the end of processing. By default, this is set to 200 which mean OK. Other common response codes are 404 (Not Found) and 302 (Found) which is used when redirecting to a new URL.

The body container within the response is used to contain everything else that needs to be sent back to the client. For a web application this means everything you see when you view source on a web page. If you are sending a file to a client, then the body would contain the contents of the file.

5.1.4 Front Controller Plugins

The front controller's architecture contains a plug-in system to allow user code to be executed automatically at certain points in the routing and dispatching process. All plug-ins are derived from Zend_Controller_Plugin_Abstract and there are six event methods that can be overridden:

1. **routeStartup()** is called just before the router is executed.
2. **dispatchLoopStartup()** is called just before the dispatcher starts executing.
3. **preDispatch()** is called before each action is executed.
4. **postDispatch()** is called after each action is executed.
5. **dispatchLoopShutdown()** is called after all actions have been dispatched.
6. **routeShutdown()** is called after the router has finished.

One problem with the current router is that if you specify a controller that does not exist, then an exception is thrown. A front controller plugin is a good way to inject a solution into the routing process and redirect to a more useful page. The Zend Framework supplies the ErrorHandler plug-in for this purpose.

6. Architecture from application perspective

6.1 Role

A role is a responsibility that a user has within the system. A typical role would be “operator” or “member”. This is encapsulated with Zend_Acl_Role which is a very simple class that just holds the name of the role. Once a role is created, its name cannot be changed. Note also, that within the context of the ACL(Access Control List), each role name must be unique. A given role may have a parent which means that the new role can do everything the parent role can do. A role may inherit from one or more roles. This is to support inheritance of rules among roles.

6.2 Resource

A resource is something that you want to protect. A typical resource in a Zend Framework application would be a controller or action. For example you may want to protect access to the forums module controller so that only those users belonging to the member role have access to the controllers within it. A resource is attached to Zend_Acl in a similar manner to a role using the addResource() member function.

Creating a resource in Zend_Acl is very simple. Zend_Acl provides the resource, Zend_Acl_Resource_Interface, to facilitate creating resources in an application. A class need only implement this interface, which consists of a single method, getResourceId(), so Zend_Acl to recognize the object as a resource.

Zend_Acl provides a tree structure to which multiple resources can be added. Since resources are stored in such a tree structure, they can be organized from thse general (toward the tree root) to the specific (toward the tree leaves). Queries on a specific resource will automatically search the resource's hierarchy for rules assigned to ancestor resources, allowing for simple inheritance of rules.

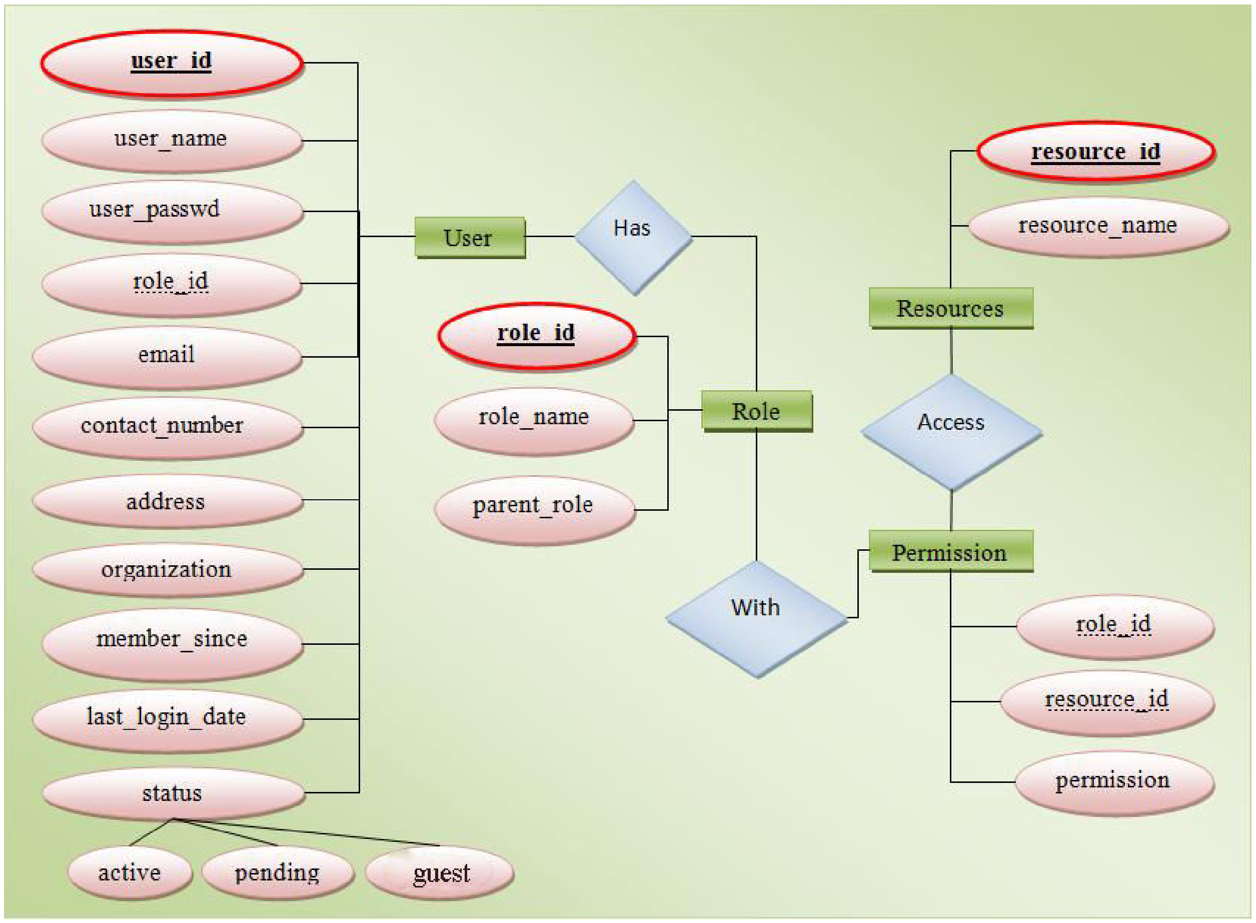

Fig – ER diagram of ACL (user, role, resource and privilege)

6.3 Permissions

The final part of the setting up a Zend_Acl object for use is telling the object which permissions a given role has for accessing a given resources. To do this we need to look at the concept of privileges. A privilege is the type of access required. Typically, privileges are based around the operations that will be performed, so have names like "view", "create", "update", etc.

Zend_Acl has two functions for setting permissions: allow() and deny(). We start off in state where all roles are denied access to all resources. The allow() function then provides access to a resource for a role and deny() and remove a subset of the allowed access for a particular case. Inheritance also comes into play here as permissions set for a parent role will cascade to child roles.

6.4 Authentication and Authorisation

Not every application needs to identify their users, but it is a surprisingly common requirement. Authorization is the process of providing access to a given resource, such as a web page, to an authenticated user. That is, authentication is the process of identifying and entity, usually via a token such as a username/password pair, but could equally be via a fingerprint. Authorization is the process of deciding if the authenticated entity is allowed to have access to, or perform operations on, a given resource, such as a record from a database.

As there are two separate processes required, the Zend Framework provides two separate components: Zend_Acl and Zend_Auth. Zend_Auth is used to identify the user and is typically used in conjunction with Zend_Session to hold that information across multiple page requests (known as token persistence). Zend_Acl is then uses the authentication token to provide access to private information using the Role Based Access Control List system.

The process flow while visiting and logging in the e-platform model is as follows:

1. When a user visits the e-platform web site, the user is assigned with a role called 'guest', which is a lowest level role in the platform
2. When user has to access some privileged resource (web page or menu item) not listed in his defined set of privilege, the user has to login
3. After user login to the application, Role-Based ACL verifies whether the user has privilege to access the resource (menu item)
4. User is granted access to resource if he has privilege over the resource requested, else, he is redirected to a page stating the user 'You are not authorized to view the page'
5. Access to non-existent resource or action throws an exception which is logged into the application error file which can be referenced (viewed) from the e-platform model application itself

The flowchart below depicts the same process flow as described above.

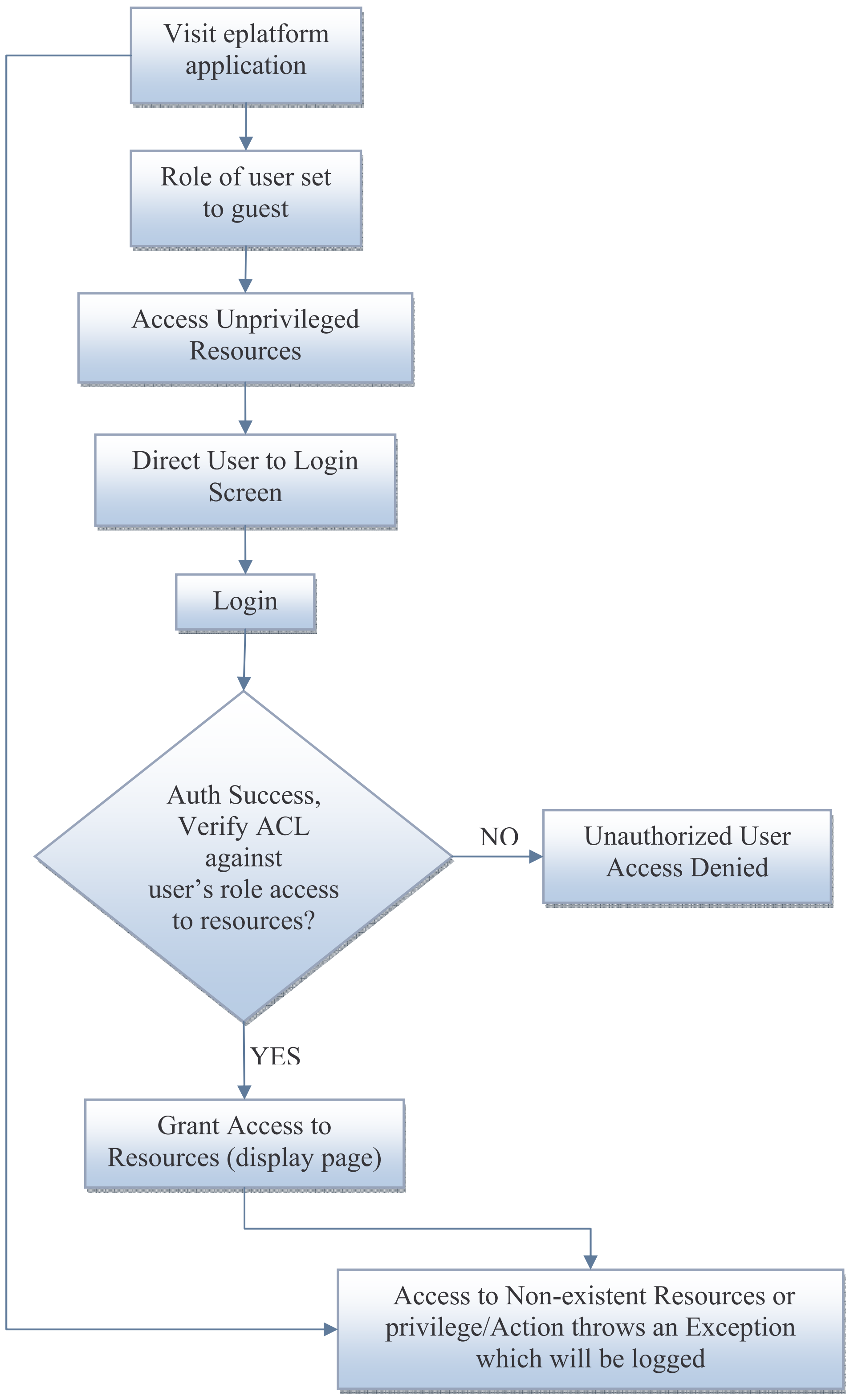
Visit eplatform application
Role of user set to guest
Access Unprivileged Resources
Direct User to Login Screen
Login
Auth Success, Verify ACL against user’s role access to resources?
NO
Unauthorized User Access Denied
YES
Grant Access to Resources (display page)
Access to Non-existent Resources or privilege/Action throws an Exception which will be logged

Fig – Process of authentication and authorization in e-platform model

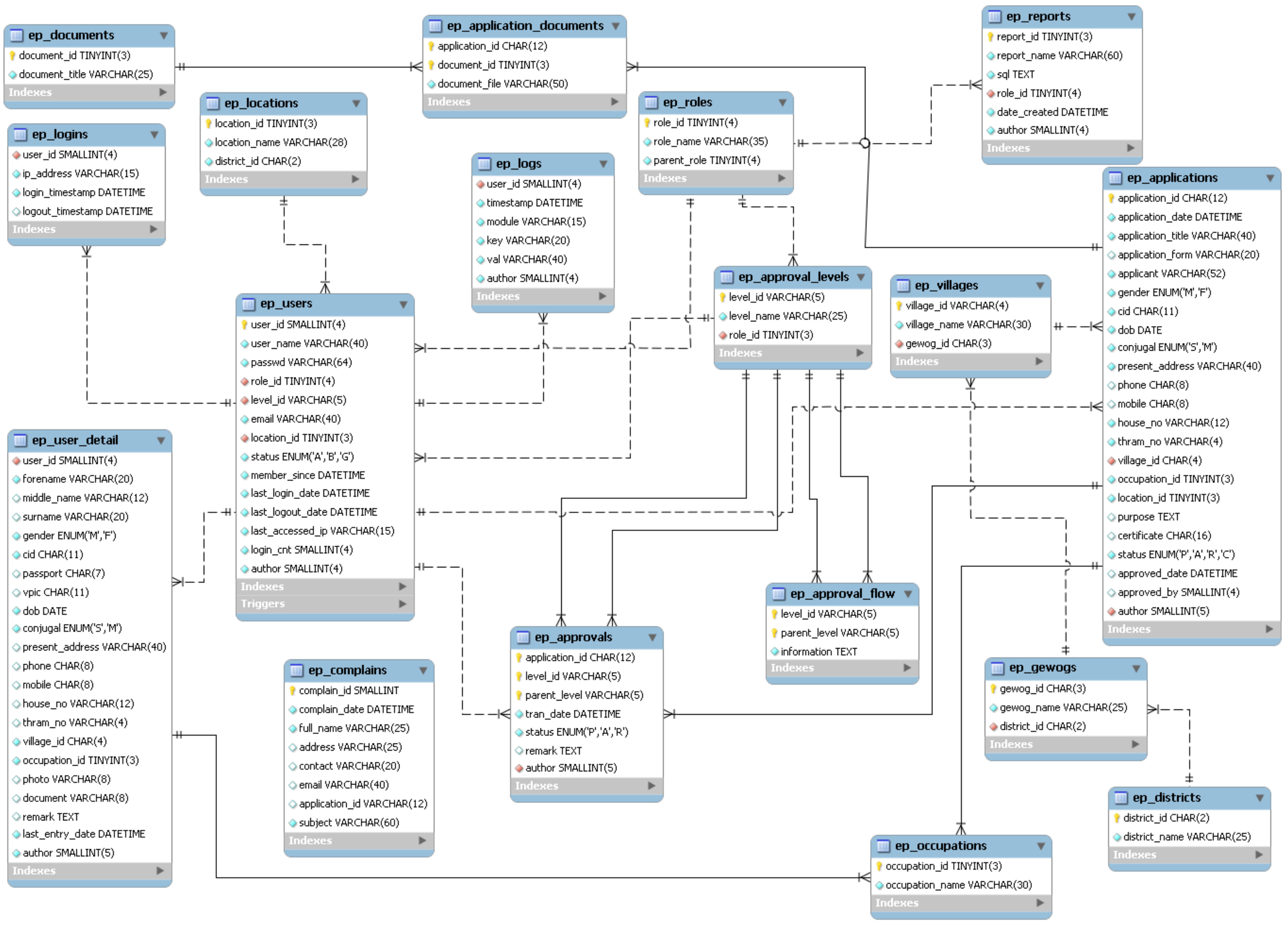

Fig – Database Schema of the e-platform model

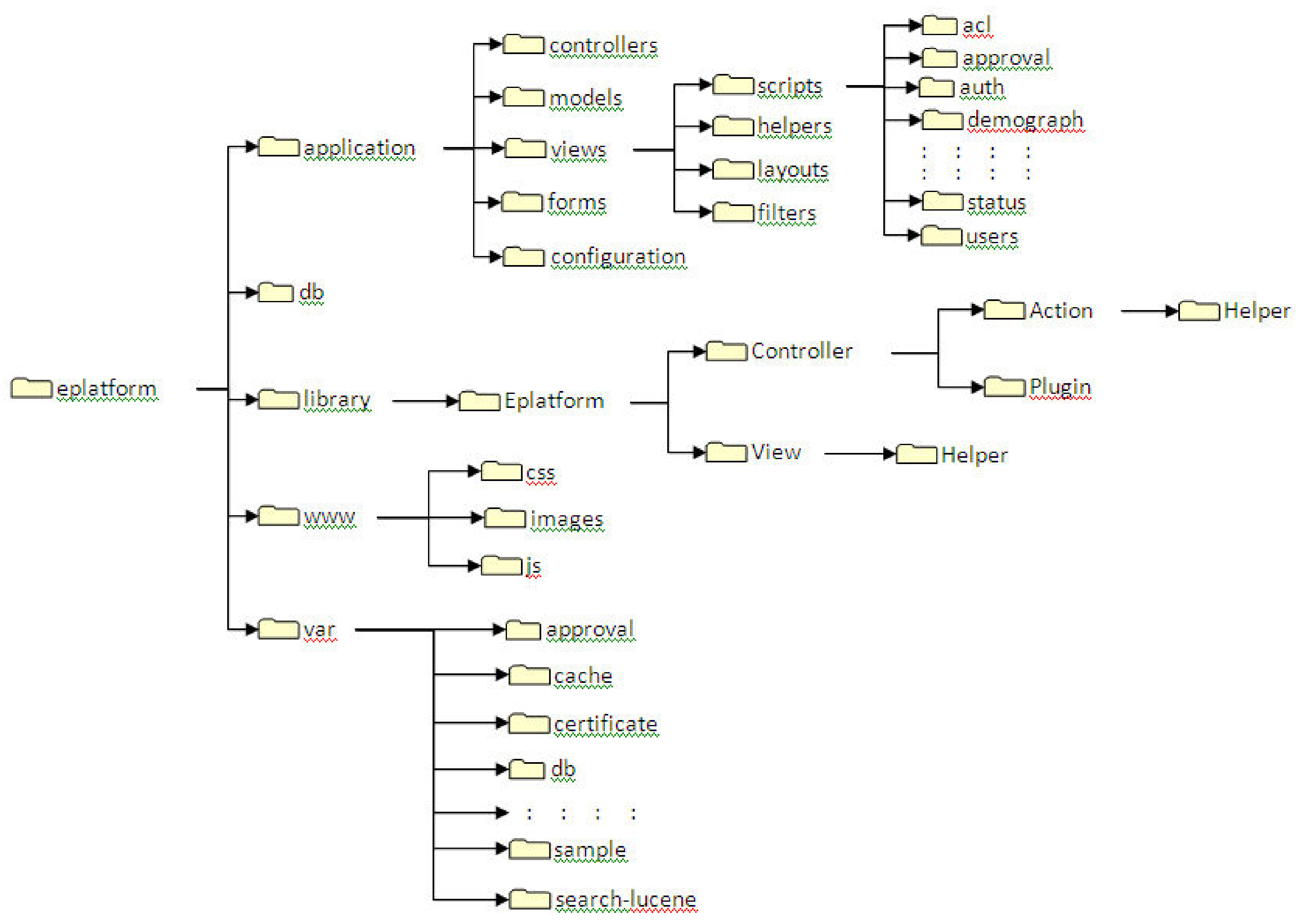

Fig - Directory structure of the e-platform model

6.5 Approval System Process

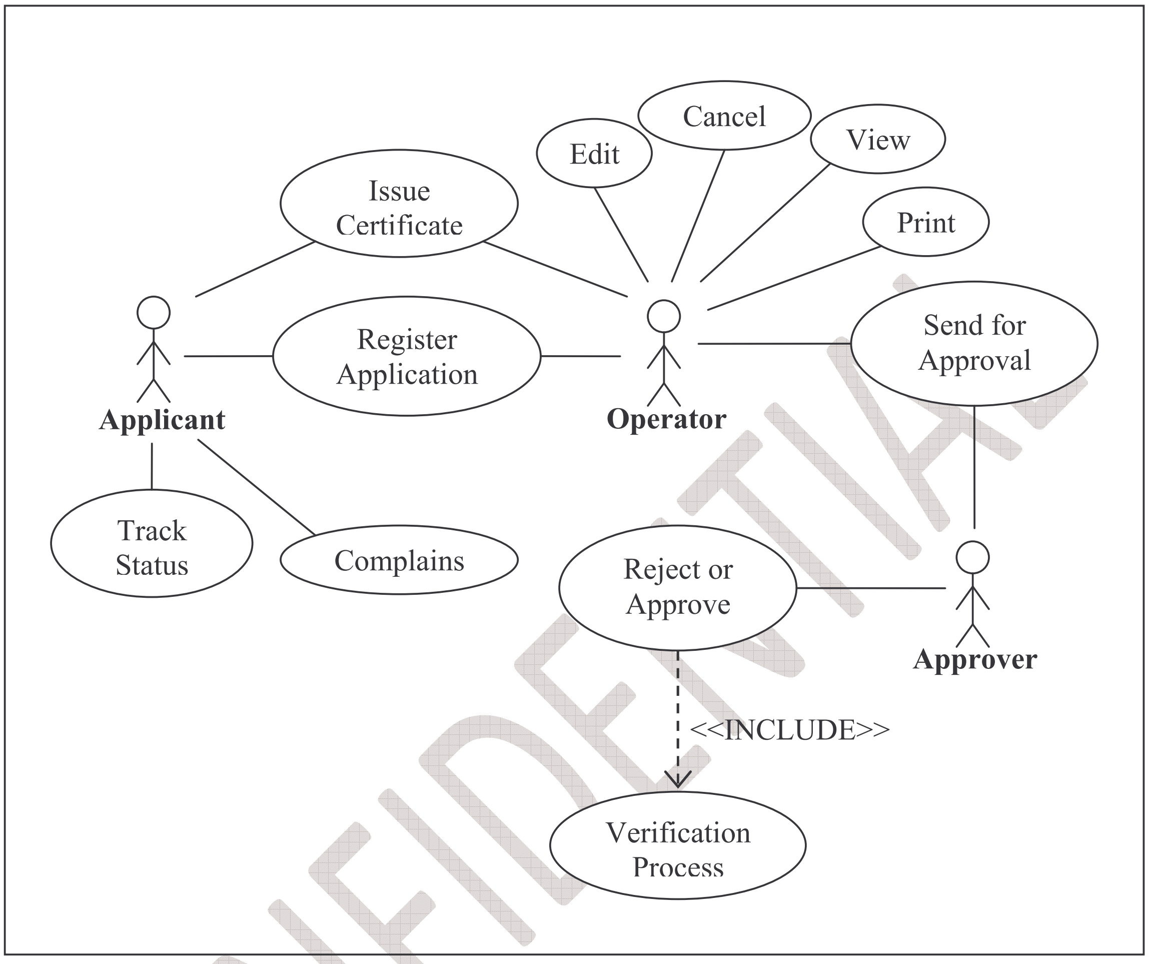

Fig - UML Use Case Diagram for Approval System

UML Use Case Diagram is a narrative document that shows sequence of events of a user using a system to complete a process or function. Use case is drawn as an oval that represent the systems function. It always starts with a verb.

From the diagram above, we see the process of the approval system as follows:

An applicant registers his application. He/she fills up an application form and submits the application along with the necessary required document. The operator receives the applications and feeds the information into the system. The operator can edit, cancel, view and print the application. Once the information is completely inserted into the system, the Approver on the higher level will carry out the verification process and approve or reject depending on the previous record and the documents that the applicant submitted. An applicant can routinely track the status of the application with the Application No which is provided by the operator during the time of registration. A complain lodging mechanism is available in the system whereby common people can make inquiries and/or lodge complains regarding their submitted application and/or general inquiry on the services.

7. Security Implementations

7.1 Security considerations and implementations

When writing the view code, the most important security issue to be aware of is **Cross Site Scripting** (also known as **XSS**). Cross site scripting vulnerabilities occur when unexpected HTML, CSS or Javascript is displayed by the website. Generally, this happens when a website displays data created by a user without checking that it is safe for display. As an example, this could happen when the text from a comment form contains HTML and is displayed on a guestbook page "as is".

The easiest way to preventing XSS vulnerabilities is to encode the characters that have sepecial meaning in HTML. That is, we need to change all instances of < to &lt;, & to &amp; and > to &gt; so that the browser treats them as literals rather than HTML. Within the Zend Framework, we use the helper function **escape()** to do this. Every time that you display a PHP variable within a template file, you should use escape() unless you need it to contain HTML in which case, you should write a sanitizing function to allow only HTML codes that you trust.

7.2 Security issues with databases

The most common type of database security problems are known as **SQL injection security** breaches. These occur when your user is able to trick your code into running a database query that you didn't intend to happen. Consider this code:

```
$result = $db->query("SELECT * FROM users
WHERE name='" . $_POST['name'] . "'");
```

This typical code might be used to authorize a user after they have submitted a login form. The coder has ensured that the correct superglobal, $_POST, is used, but hasn't checked what it contains. Suppose that $_POST['name'] contains the string "' OR 1 OR name = '" (single quote, followed by “OR 1 OR name=” followed by another single quote). This would result in the perfectly legal SQL statement of:

```
SELECT * from users where name='' OR 1 OR name= ''
```

As you can see, the OR 1 in the SQL statement will result in all the users being returned from the database table. With SQL injection vulnerabilities like this, it can be possible to retrieve username and password information or to maliciously delete database rows causing your application to stop working. As should be obvious, the way to avoid SQL injection attacks is to ensure that the data that you are putting into the SQL statement has been escaped using the correct functionality for your database. For MySQL, you would use the function mysql_real_escape_string() and for PostgreSQL, you would use pg_escape_string().

From Zend Framework perspective, the business logic and interaction with database is carried out using Zend_Db component, and hence, the member function **quote()** is used to take care of this issue. The quote() function will call the correct underlying database specific function and if there isn’t one, then it will escape the string using the correct rules for the database involved. Usage is very easy:

```
$value = $db->quote("It's a kind of magic");
```

An alternative solution is to use parameterized queries, where variables are denoted by placeholders and are substituted by the database engine with the correct variable. The Zend_Db provides the quoteInto() function for this. For example:

```
$sql = $db->quoteInto('SELECT * FROM table WHERE id = ?', 1);
$result = $db->query($sql);
```

7.3 Cross-Browser Compatibility

The e-platform model is developed to work in all the available browsers. The model including graphics, design, layout, menu, CSS, AJAX, JavaScript and Forms working have been tested successfully in major browser including Internet Explorer, Firefox, Google Chrome and Opera.

7.4 Recommendations

It is very much recommended to host the e-services in Secured Socket Layer (**SSL**) for desired data encryption to avoid potential spoofing of data traversed in a network, both LAN and WAN.