Your African solutions in databases, mobile and web applications

BenSino Information Technology Corporation

http://it.bensino-eg.com

ASDBMS Case Study

Design Artifacts

Ralph D'Almeida

2013

Table of Contents

Overview

Comergence, the solidarity cooperative is a network of organizations and connivance of people who respond to the recognized need for an organization to coordinate and integrate the activities of existing organizations within African Immigrants Communities (CIIA in French).

I. Business needs

Searching for members' contact information from different organizations requires tedious cross-reference of all the organizations data.

Members' skills and abilities to perform various activities within different organizations cannot be acknowledged.

II. Solutions

ASDBMS allows centralizing all members' contact information from all organizations into one database management system.

Members can also register their skills and accomplishments with ASDBMS so that other organizations can utilize their abilities wisely.

III. Benefits

Having one database for all members' information makes searching easier and reduce the time to reach any one of them.

Accessing all members' skills enhances human resource management daily activities especially fulfilling internal position.

Case Study

This section describes the software engineering solutions used to satisfy the above business needs. In other words, the design artifacts of ASDBMS are highlighted below.

I. Review

African Skills Database Management System (ASDBMS) is initially a database that gathers information about all members from different African organizations in order to facilitate and coordinate communication between those organizations. It then has evolved to include the members' skills and constitute a data center for African skills. The actual objective is to coordinate all efforts deployed by Africans around the world toward the development of the beautiful continent – Africa.

II. Design artifacts

All the artifacts used to design and develop ASDBMS are presented in this section from the requirements specifications to the implementation of the system.

A. Requirements

The high-level technical requirements of the system which constitutes the solution for centralizing all the members' information are detailed here.

1. Stakeholders Summary

Name	Description	Responsibilities
Managers	*Organization's team and project managers*	*The manager coordinates communication with other organizations.*
Human Resource	*Organization's recruiters*	*The human resource internally fulfills available positions and assesses members' skills.*
Operators	*Staff members and receptionist*	*The operator contacts members.*

2. Needs and Features

Needs	Priority Level	Features	Planned Release
The system should allow the managers to view other administrators' information from another organization.	*High*	*Search By Name*	*Version 1.0.0*
	High	*Search By Country*	*Version 1.0.0*
The system should allow the human resource personnel to assess any member's skills.	*High*	*Register Profile*	*Version 1.0.0*
	High	*Register and Add Skills*	*Version 1.0.0*
	High	*Search By Profession*	*Version 1.0.0*
	High	*View Skills*	*Version 1.0.0*
The system should allow the operators to contact members	*High*	*Register Personal Information*	*Version 1.0.0*
	High	*View All Members*	*Version 1.0.0*
The system should allow members to manage their own accounts	*Normal*	*Update account information*	*Version 1.0.0*
	Normal	*Delete account*	*Version 1.0.0*
	Normal	*Retrieve Password*	*Version 1.0.0*

3. Use Case Model

a. Actors

Name	Description	Goals
Administrator	*The administrator enforces the system's policies and standards.*	*The administrator wants to delete the account of any member that has infringed the system's terms and conditions.*
Guests	*The guests have access to public information only*	*The guests want to quickly view and search for member's information.*
Members	*The members can manage their own accounts and have access to other members' private information*	*The members want to create, update and delete their own account.* *The members want to view and search for other members' information.*

b. Context Diagram

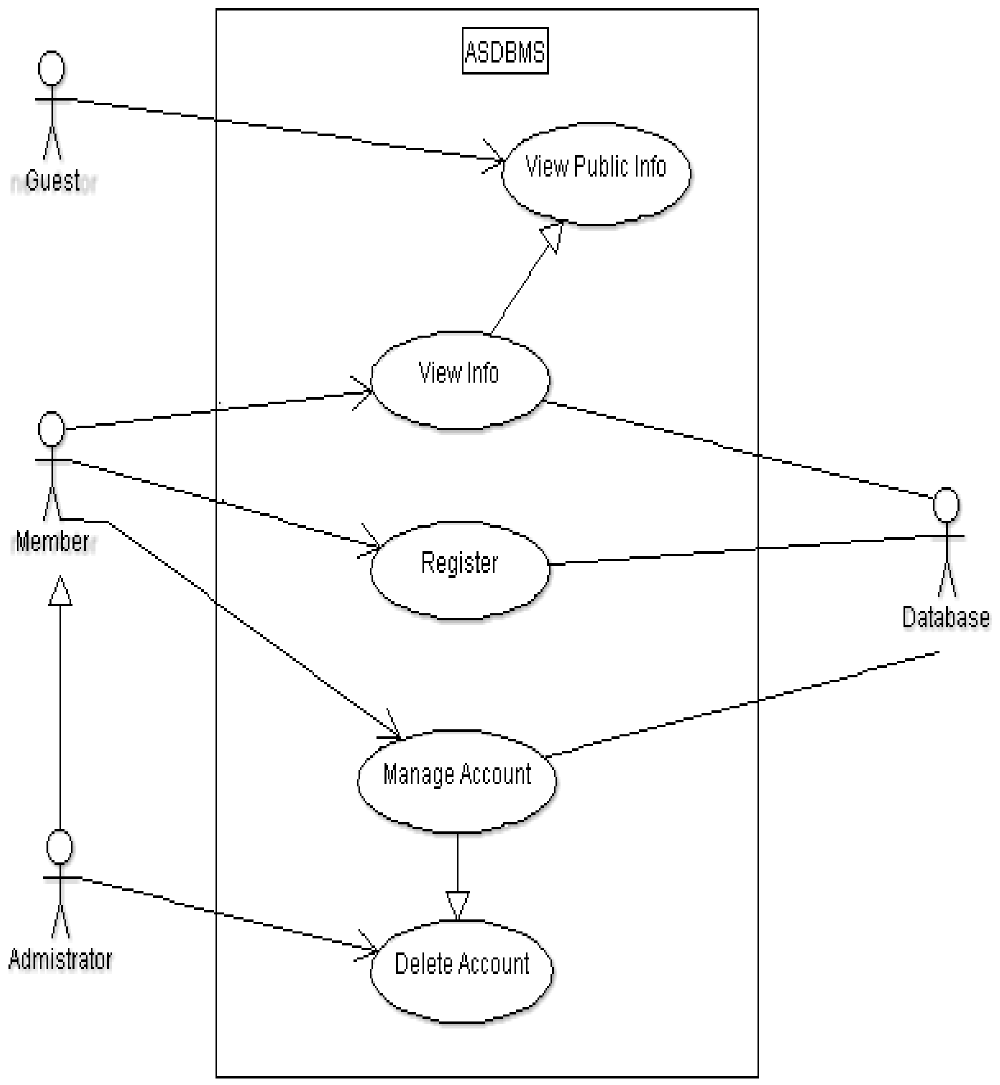


Figure 1: Use Case Diagram

c. Domain Model

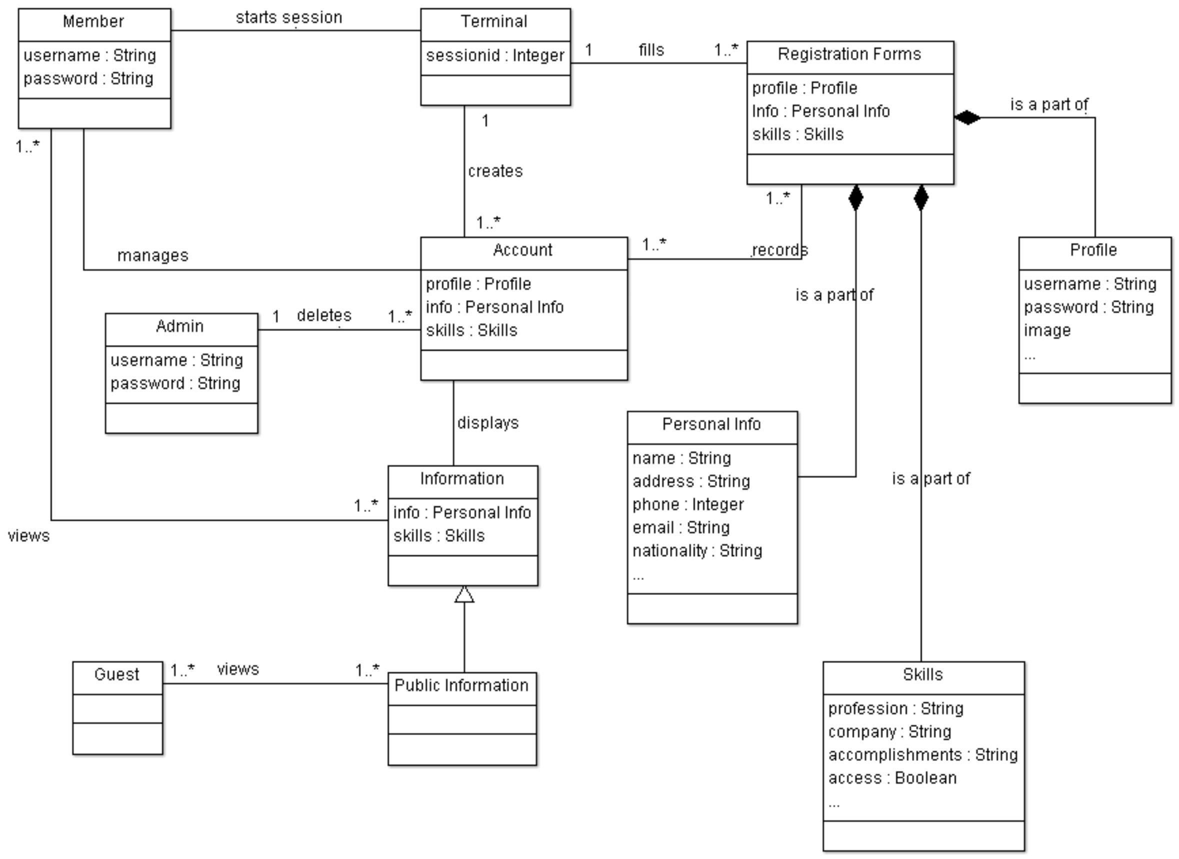


Figure 2: Domain Model Diagram

4. Supplementary Specifications

The functional requirements not captured in the use case model, as well as the non-functional requirements which the system must satisfy are described in this section. That is, the quality of the system and the regulation that governs it are presented below.

a. Functionality

- There are User and Admin access level to the system. User can register, view member's information, and manage account. Admin are given the same access as User with the added functionality of deleting any member's account. It follows that Human Resource and Operators are User while Managers are Admin. Member and Guest are also User with the different that Guest has restricted access and need to register to become a member in order to be granted full access.

- A user is able to log out from the system at any time. Additionally, the user must be logged out when the application is closed.
- The system must be able to automatically update the member’s information at a regular time basis.

b. Reliability

- The system will be available 24/7 unless it experiences downtime due to technical difficulties or for maintenance. Unforeseen downtimes should not exceed 1 day. Scheduled maintenance should not exceed 6 hours and should happen during overnight only.
- The Mean Time Between Failures (MTBF) is the sum of the availability periods divided by the number of observed failures. Assuming there are only 2 observed failures per month, the expected time will be 14 days.
- In case of a fault occurring in the system, the Mean Time To Repair (MTTR) or the longest down time the system will experience until repair will not exceed 24 hours.
- The system must always deliver reliable and accurate data.
- In term of maximum bugs or defect rate, the system defective rate will not be more than 2% of the KLOC. Major bugs will be fixed as soon as possible, significant ones in 3 days and minor ones in at most 24 hours. The maximum bugs allowed are 10 bugs/KLOC and a critical bug is considered as loss of data.

c. Performance

- The response time of the system when viewing the member’s information should not exceed 2 seconds.
- The system will be able to handle a minimum of 1000 users logged in at a given time.
- The system shall provide access to the database with no more than 3 seconds latency.
- The system shall perform actions and have a user command response time of less than 2 seconds. It will display a notification detailing any error that might occur and/or the action to be taken by the user. The length of time actions are completed within the system depends on the complexity of the user’s inputs.

d. Design Constraints

- The system will use PHP libraries and extensions with PHP and SQL programming languages. HTML5, CSS, and JavaScript technologies are also used.
- The system requires the following software tools: PHP plug-in for Eclipse IDE, Apache HTTP Server, and MySQL Server.
- The system will use the Model View Controller architecture model and the client's browser must allow the execution of JavaScript file.

B. Architecture

The system needs to be designed to satisfy most of the stakeholders' concerns and to provide different architectural views with respect to the separation of concerns principle. The view model that fully satisfies this specification is the 4 + 1 architectural view model and its components are described below:

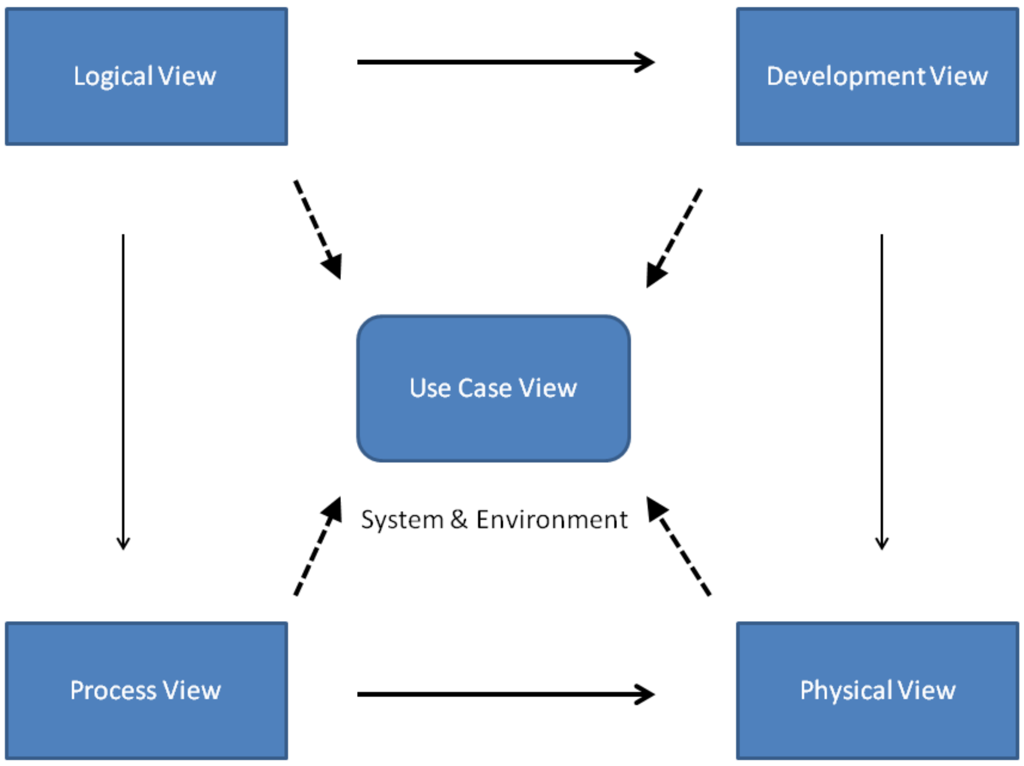


Figure 3: 4 + 1 View Model

1. Logical View

This view addresses the functionalities that ASDBMS will provide to the end-user and is presented here in form of UML class diagram.

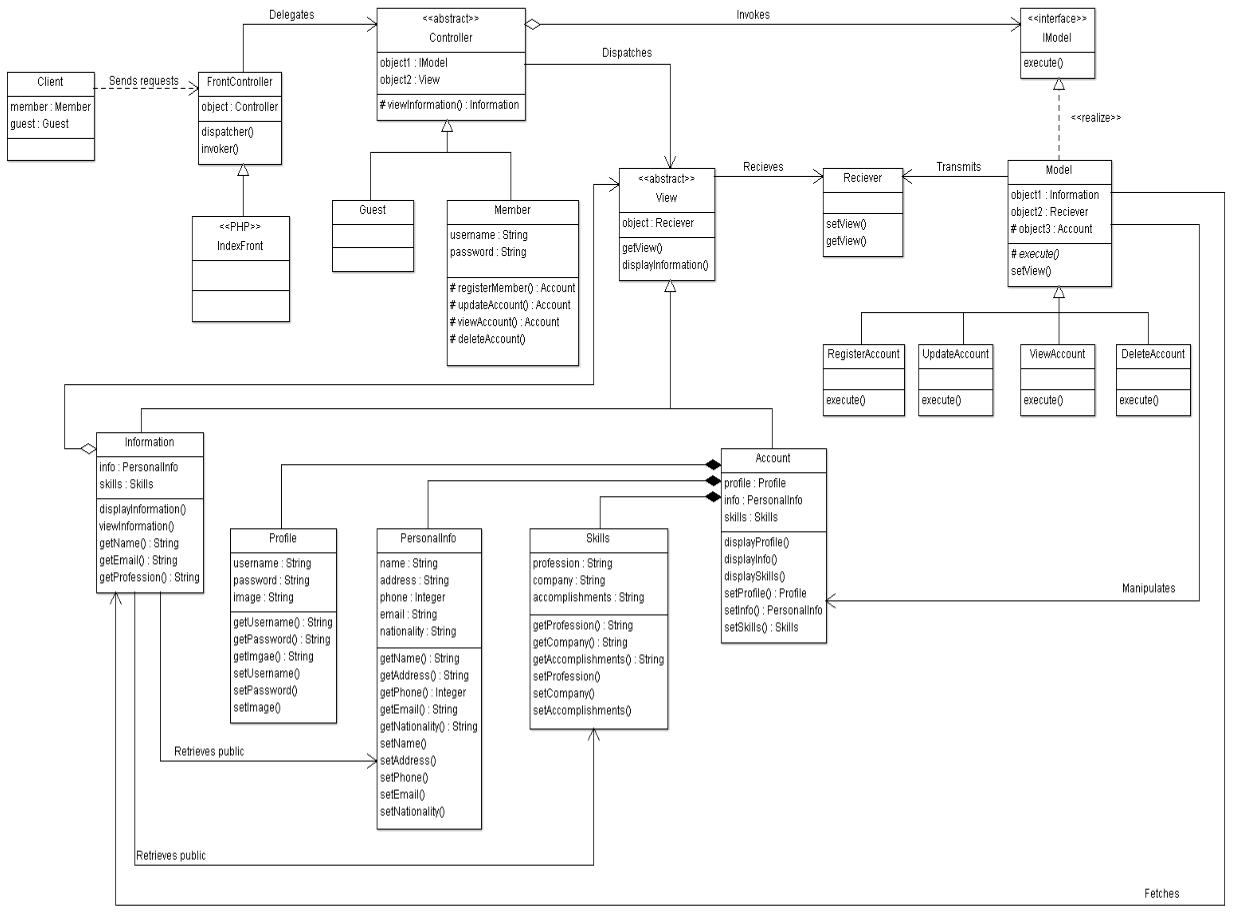


Figure 4: Class Diagram

The adopted architecture for ASDBMS is based on MVC pattern on which some additional design patterns are applied. That is, the Model incorporates Active Record design pattern, the View integrates Factory and Composite, and the Controller implements Front Controller with Command design patterns.

2. Development View

Also known as implementation view, it describes the software management from the developer's perspective and is presented here in form of package diagram.

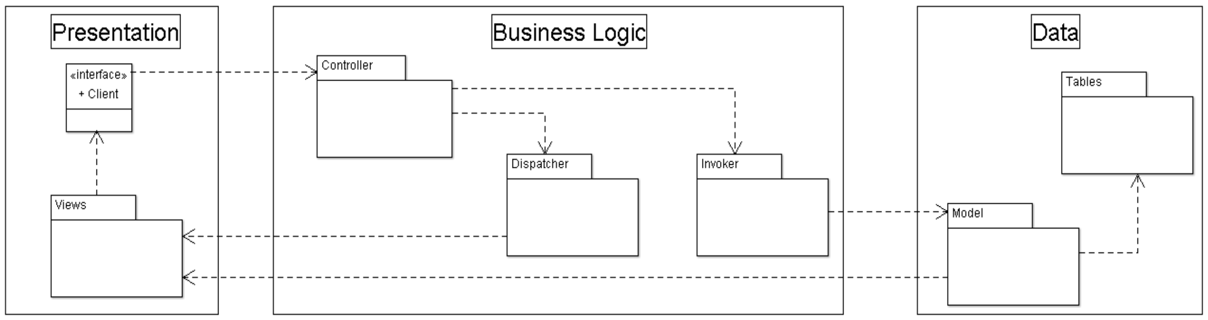


Figure 5: Package Diagram

This section can also encompass the data view of the system by translation the design model into data model. This is done by applying Active Record design pattern to the Model part of the system's MVC architecture; as a result, the following ER diagram represents the system's database.

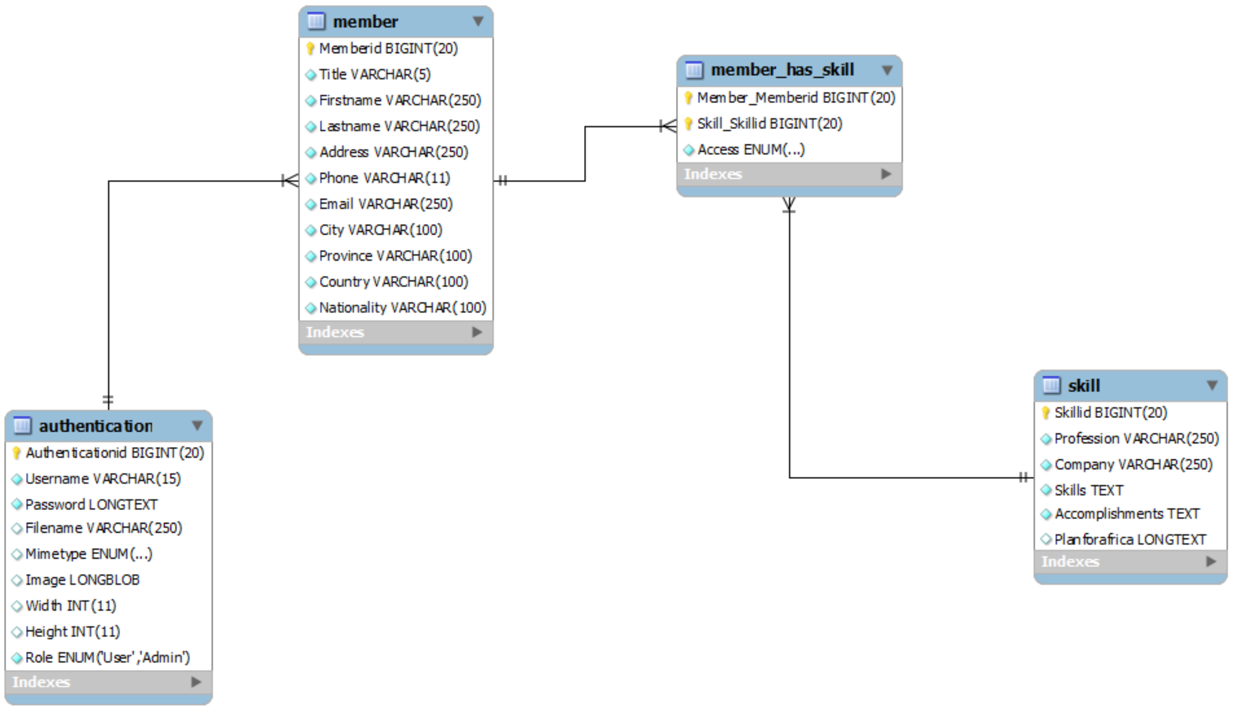


Figure 6: ER Diagram

3. Process View

Here, the runtime behavior of the system such as its processes and their communication are highlighted. It basically ensures the system concurrency, distribution, performance, and scalability. It is described in form of activity diagram in this section.

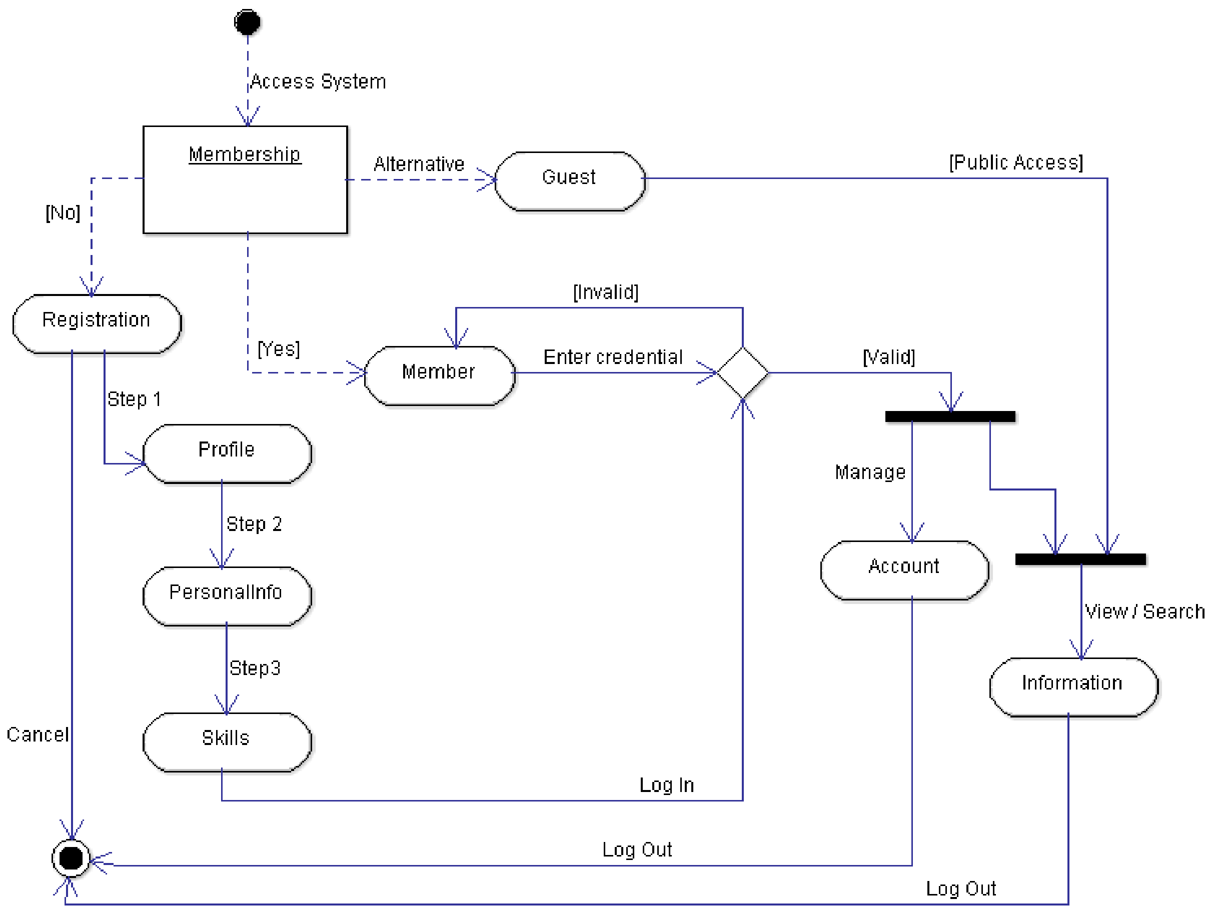


Figure 7: Activity Diagram

4. Physical View

The physical or deployment view represents ASDBMS from a system engineer's perspective. That is, it describes the physical layer components' topology and their interconnections. It is presented here in form of deployment diagram.

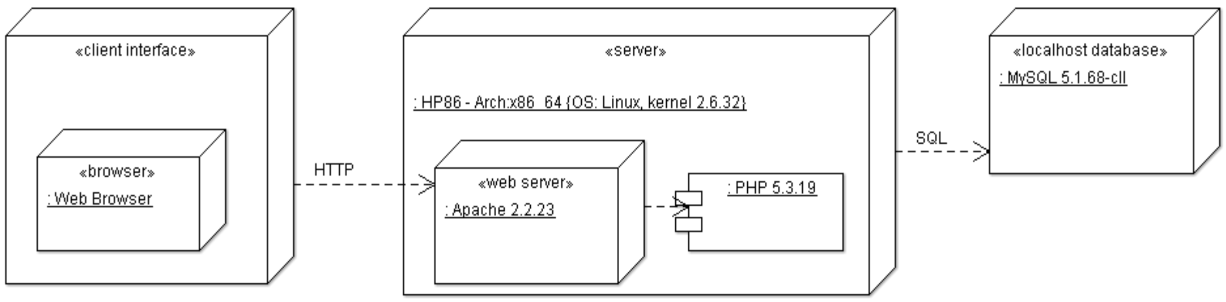


Figure 8: Deployment Diagram

The above diagram includes two physical nodes and two execution environment nodes. The physical nodes are first the server and then the localhost database. The first physical node hosts one of the other execution nodes; that is, Apache web server with PHP libraries and extensions. The second physical node hosts the system's database – MySQL server. Finally, the last execution node is composed of the client interface including the web browser.

5. Use Case View

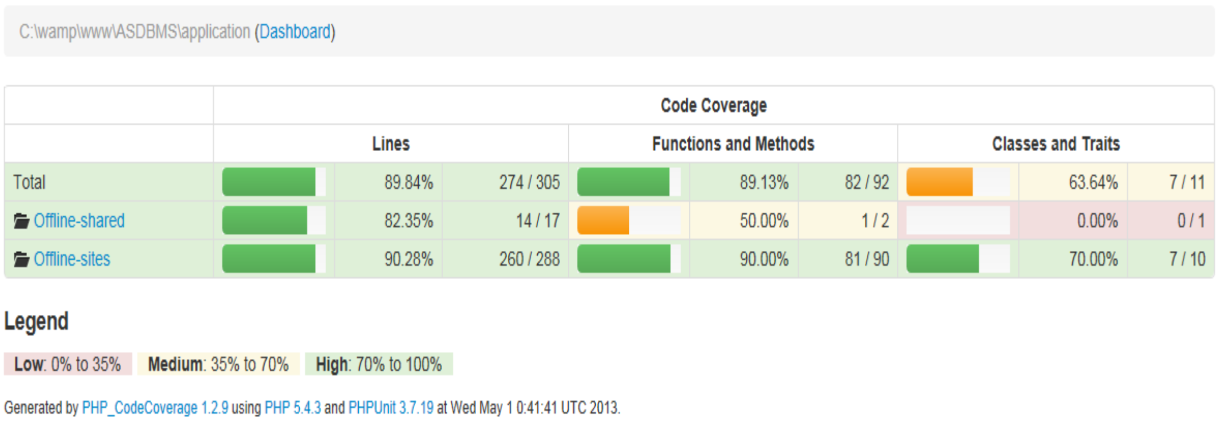
It describes architectural elements' interactions that illustrate and validate ASDBMS architecture design. This view is already represented within the requirements section by the use case diagram.

C. Testing

This section describes the test plan adopted to ensure that ASDBMS performs as expected in its execution environment. The testing plan is then to conduct the following levels of testing:

1. Unit Testing

This level of testing verifies the functionality of a specific section of code, usually at the function level or class level. It is a white-box testing style and includes static code analysis such as code coverage analysis. The goal is to increases the efficiency of the quality assurance process with at least 80% functions' coverage and 60% classes' coverage.



C:\wamp\www\ASDBMS\application (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		89.84%	274 / 305		89.13%	82 / 92		63.64%	7 / 11
Offline-shared		82.35%	14 / 17		50.00%	1 / 2		0.00%	0 / 1
Offline-sites		90.28%	260 / 288		90.00%	81 / 90		70.00%	7 / 10

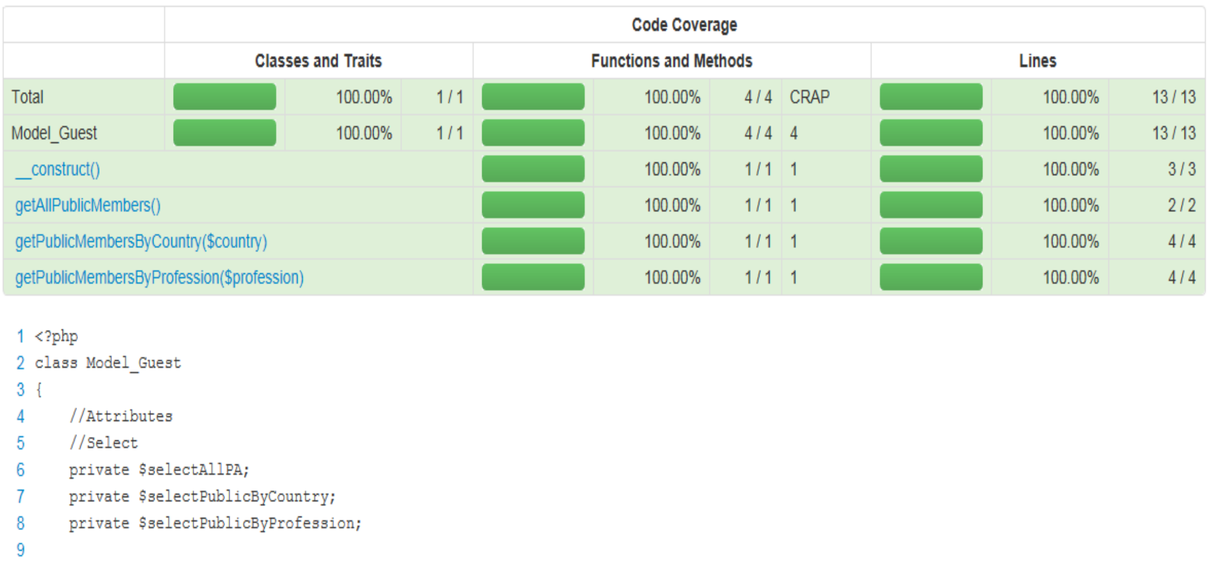
Legend

Low: 0% to 35% **Medium**: 35% to 70% **High**: 70% to 100%

Generated by PHP_CodeCoverage 1.2.9 using PHP 5.4.3 and PHPUnit 3.7.19 at Wed May 1 0:41:41 UTC 2013.

Figure 9: ASDBMS Code Coverage

The above code coverage is the result of the test suite that groups each tested class from the system packages. Below is a code coverage sample of one of the tested classes.



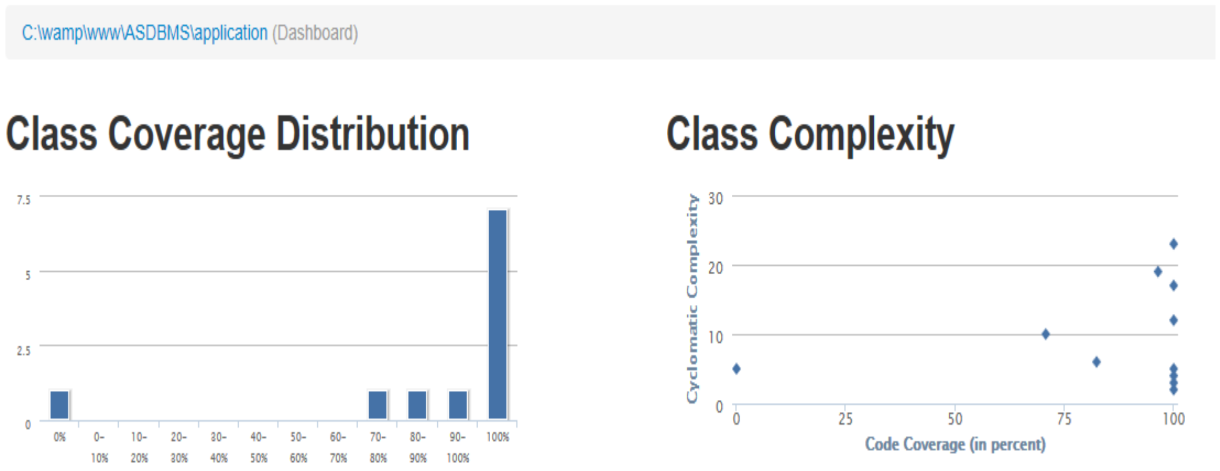
	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total	🟩	100.00%	1 / 1	🟩	100.00%	4 / 4	CRAP	🟩	100.00%	13 / 13
Model_Guest	🟩	100.00%	1 / 1	🟩	100.00%	4 / 4	4	🟩	100.00%	13 / 13
__construct()				🟩	100.00%	1 / 1	1	🟩	100.00%	3 / 3
getAllPublicMembers()				🟩	100.00%	1 / 1	1	🟩	100.00%	2 / 2
getPublicMembersByCountry($country)				🟩	100.00%	1 / 1	1	🟩	100.00%	4 / 4
getPublicMembersByProfession($profession)				🟩	100.00%	1 / 1	1	🟩	100.00%	4 / 4

```
<?php
class Model_Guest
{
    //Attributes
    //Select
    private $selectAllPA;
    private $selectPublicByCountry;
    private $selectPublicByProfession;

```

Figure 10: Guest Class Code Coverage

2. Integration Testing

It verifies the interfaces between components against the software design in order to ensure consistency.



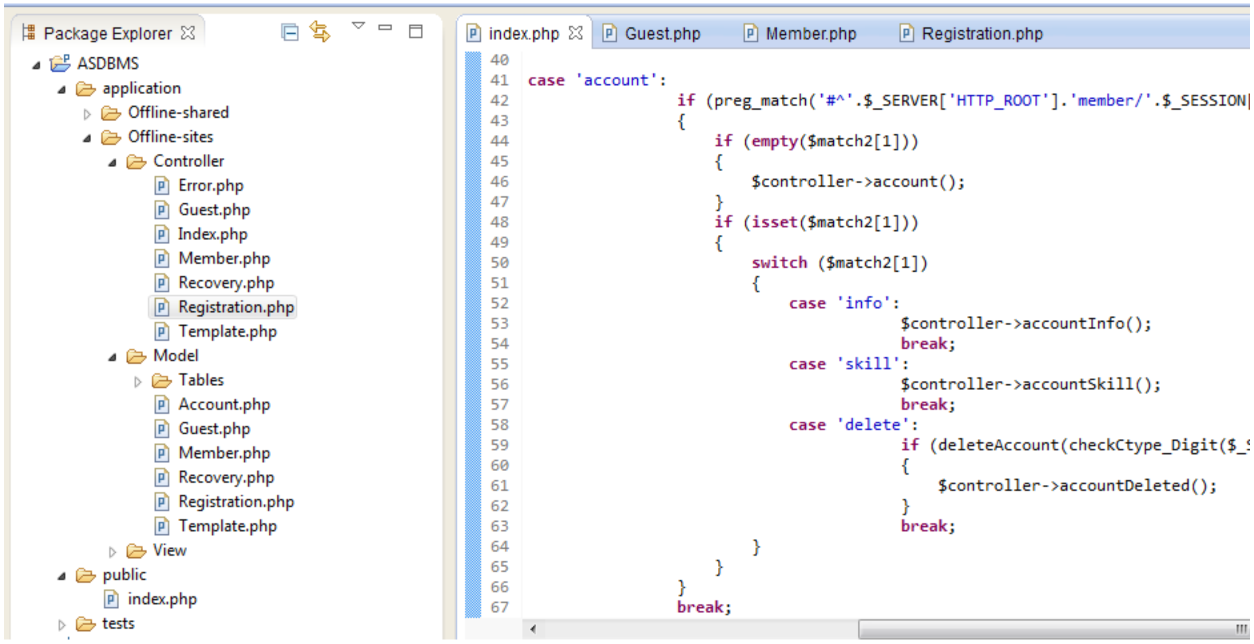
C:\wamp\www\ASDBMS\application (Dashboard)

3. System Testing

In this level of testing, the integrated system is completely tested in order to verify that the system meets its requirements.

D. Implementation

The organization of the system’s packages and the implementation details are described in this section.

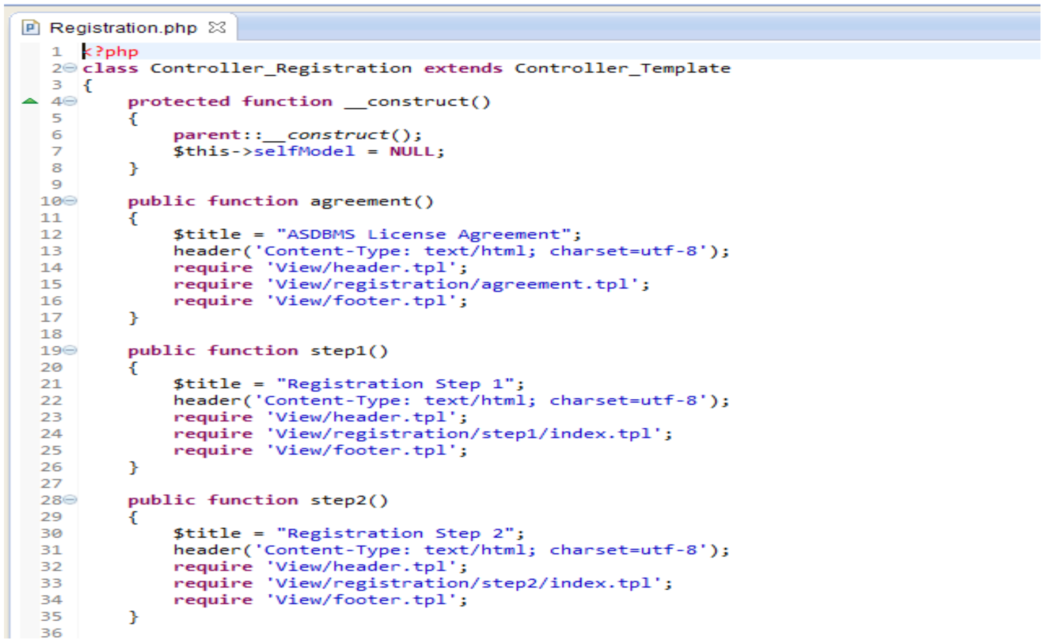


```
ASDBMS
    application
        Offline-shared
        Offline-sites
            Controller
                Error.php
                Guest.php
                Index.php
                Member.php
                Recovery.php
                Registration.php
                Template.php
            Model
                Tables
                Account.php
                Guest.php
                Member.php
                Recovery.php
                Registration.php
                Template.php
            View
    public
        index.php
    tests
```

```php
case 'account':
                if (preg_match('#^'.$_SERVER['HTTP_ROOT'].'member/'.$_SESSION
                {
                    if (empty($match2[1]))
                    {
                        $controller->account();
                    }
                    if (isset($match2[1]))
                    {
                        switch ($match2[1])
                        {
                            case 'info':
                                        $controller->accountInfo();
                                        break;
                            case 'skill':
                                        $controller->accountSkill();
                                        break;
                            case 'delete':
                                        if (deleteAccount(checkCtype_Digit($_
                                        {
                                            $controller->accountDeleted();
                                        }
                                        break;
                        }
                    }
                }
                break;
```

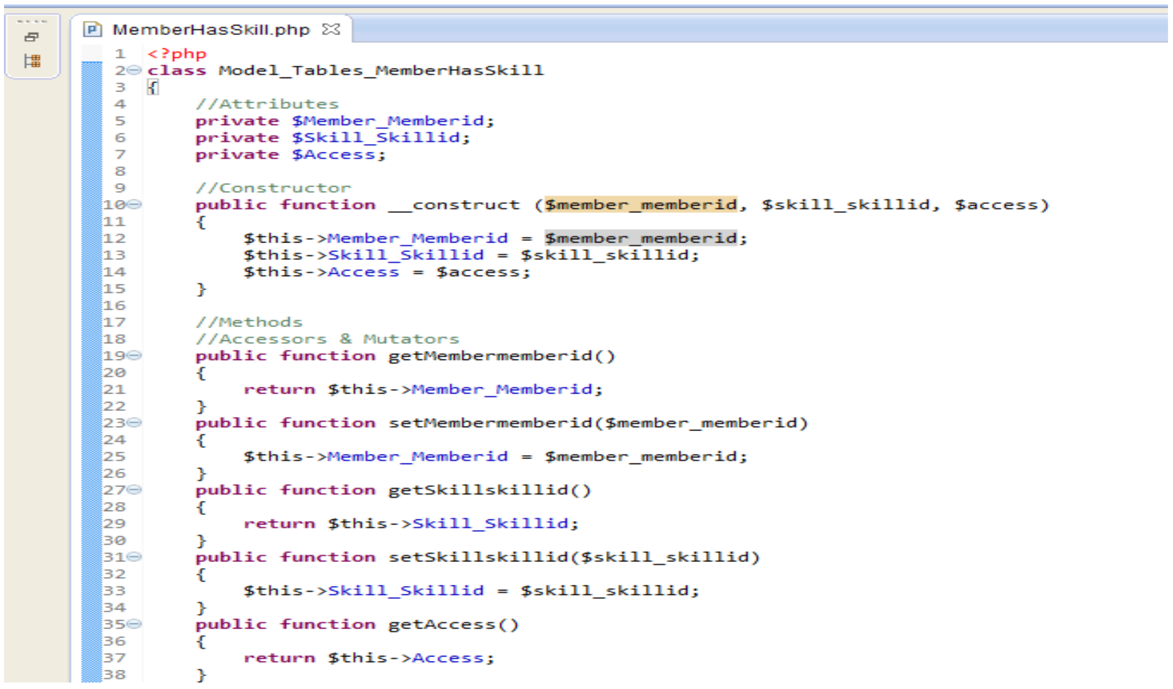
Figure 11: Packages Organization

Following the class diagram developed earlier, each class has been implemented with respect to the association between each one. Some classes’ code samples are presented below.



```php
<?php
class Controller_Registration extends Controller_Template
{
    protected function __construct()
    {
        parent::__construct();
        $this->selfModel = NULL;
    }

    public function agreement()
    {
        $title = "ASDBMS License Agreement";
        header('Content-Type: text/html; charset=utf-8');
        require 'View/header.tpl';
        require 'View/registration/agreement.tpl';
        require 'View/footer.tpl';
    }

    public function step1()
    {
        $title = "Registration Step 1";
        header('Content-Type: text/html; charset=utf-8');
        require 'View/header.tpl';
        require 'View/registration/step1/index.tpl';
        require 'View/footer.tpl';
    }

    public function step2()
    {
        $title = "Registration Step 2";
        header('Content-Type: text/html; charset=utf-8');
        require 'View/header.tpl';
        require 'View/registration/step2/index.tpl';
        require 'View/footer.tpl';
    }

```

Figure 12: Registration Class Sample Code



```php
<?php
class Model_Tables_MemberHasSkill
{
    //Attributes
    private $Member_Memberid;
    private $Skill_Skillid;
    private $Access;

    //Constructor
    public function __construct ($member_memberid, $skill_skillid, $access)
    {
        $this->Member_Memberid = $member_memberid;
        $this->Skill_Skillid = $skill_skillid;
        $this->Access = $access;
    }

    //Methods
    //Accessors & Mutators
    public function getMembermemberid()
    {
        return $this->Member_Memberid;
    }
    public function setMembermemberid($member_memberid)
    {
        $this->Member_Memberid = $member_memberid;
    }
    public function getSkillskillid()
    {
        return $this->Skill_Skillid;
    }
    public function setSkillskillid($skill_skillid)
    {
        $this->Skill_Skillid = $skill_skillid;
    }
    public function getAccess()
    {
        return $this->Access;
    }
```

Figure 13: Member-Skills Entities Relationship Class Sample Code

III. Quality assurance

This section provides the quality assurance of ASDBMS by measuring the set of software metrics generated by PHPDepend from the deployed code. There are two charts: A-I chart that shows the inter package dependencies and Overview-Pyramid chart that shows a visual summary of the source code. The first chart shows the quality of the design in the terms of extensibility, reusability and maintainability. The second one is used to visualize the integrated system.

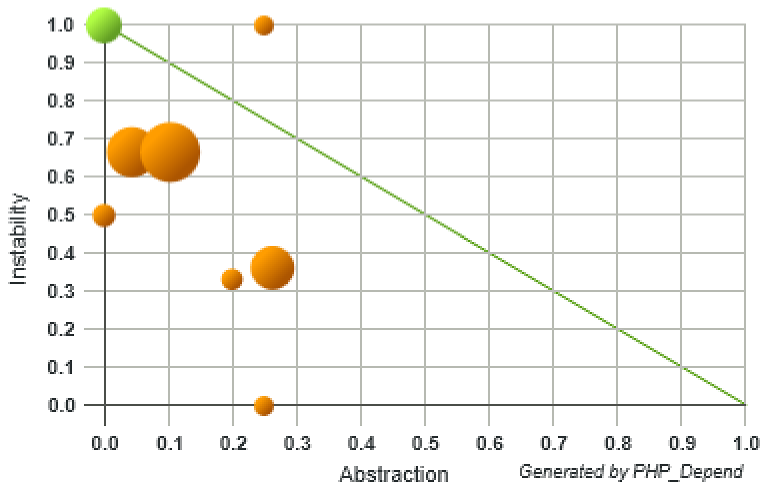


Figure 14: A-I chart

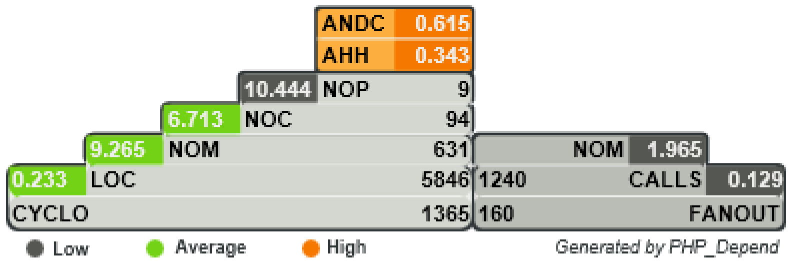


Figure 15: Overview-Pyramid chart

The charts show 30% of abstraction versus 60% of instability. This is a good trade off between inheritance and packages dependencies. In other words, ASDBMS design respects good OO practice in terms of extensibility, reusability, and maintainability by reducing the risk of breaks in multiple subsystems even with change in a single package.

Conclusion

Overall, this document demonstrates how the solutions to Comergence's business needs were engineered. It constitutes the design artifacts necessary to understand and maintain the developed system. It also benefits the stakeholder by showcasing the flexibility of the system especially how easily new features can be added to ASDBMS later on.