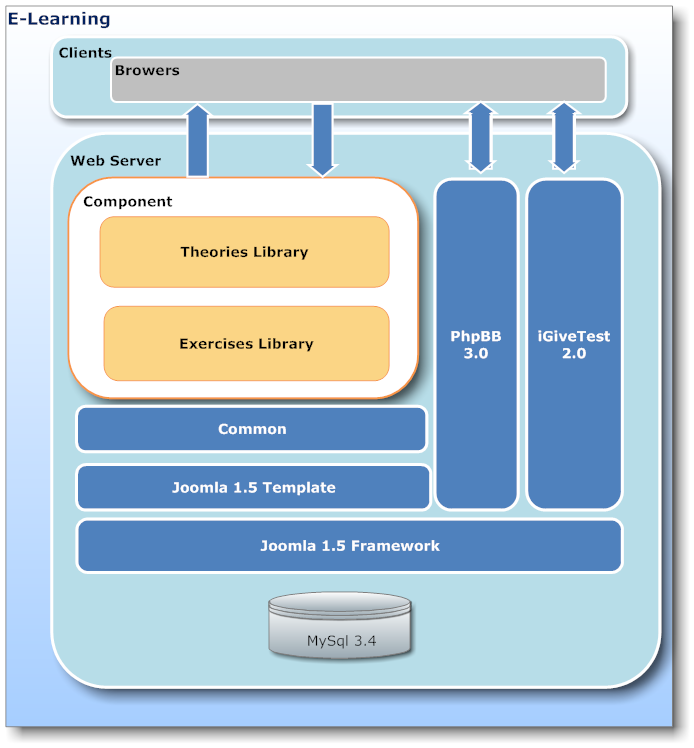
**Report No.4: Software Design Description (SDD)**

# 4.1. Design Overview

# 4.2. System Architectural Design

## 4.2.1 Choice of System Architecture

To approach the project, we choose the system architecture which is built as the diagram as below. This diagram will help us visually abstract the system and understand the key modules with their interaction in our E-Learning system.

**Figure 4.1:** E-Learning system architecture design

**Client:** the client layer is user, user will access to web server by browser. The layer will send requests on the server. Then, the server will receive and return request for user.

**Web server:** the layer is where process and returns request to the client. After receiving request, the controller on web server will control the model to get data from MySQL server. Then, the controller sends methods of model to update the view and return the browser on client. The layer includes 7 modules:

**+ Component:**

The layer includes 2 components: theories library and exercises library. *Theories library* will help users to study on the website. Content of theories are video, text which is compiled easy to understand and impression; users will study better and not boring. We based on **Bloom model** about learning method, each of theory have objective and concern questions. The questions usually are easy and have case study to help users to understand problems clearly. The next module is *exercises library*. The module brings out a large number of exercises to users to choose. Users can filter by subject, chapter, theory, and difficult.

**+ Common:**

The layer includes common definition, configuration and model objects. Components are developed will have structure look like the tree diagram:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | e-learning-website | | | |  |
|  | |-- | style | | |  |
|  | | | `-- style.css | | |  |
|  | |-- | | images | | |
|  | |-- | | components | | |
|  | | | | |-- com\_componentname | | |
|  | | | | | | models | |
|  | | | | | | `-- componentname.php | |
|  | | | | | | `-- index.html | |
|  | | | | |-- | views | |
|  | | | | | | `-- index.html | |
|  | | | | | | `-- componentname | |
|  | | | | | | |-- tmpl | |
|  | | | | | | | `-- default.php | |
|  | | | | | | | `-- index.html | |
|  | | | | | | |-- index.html | |
|  | | | | | | |-- view.html.php | |
|  | | | | |-- | controller.php | |
|  | | | | |-- | index.php | |
|  | | | | |-- | componentname.php | |
|  | |-- | | administrator | | |
|  | | | | |-- | components | |
|  | | | | | | |-- com\_componentname | |
|  | | | | | | `-- componentname.php | |
|  | | | | | | `-- componentname.xml | |
|  | | | | | | `-- install.sql | |
|  | | | | | | `-- uninstall.sql | |
|  | | | | | | `-- index.html | |

We will create “*index.html*” files in folders contain our code files. The file is forced in all of folders “**E-learning-website**” to increase security, avoid accessing administrator files. The file doesn’t have content, it only display white page.

Code *.css* to design components which we develop is placed in “**e-learning-website/style**” and image files will be put into Images folder.

Folder “**e-learning-website/components**”contain all of components of *Joomla* and our component. The components are categorized using prefix “com\_” and name of component. We choose Model-View-Controller model to develop the system. The schema of MVC code will be described below.

+ Model:

The classes that connect to database will be in “*componentname.php*” file. The file is placed in “**components/com\_componentname**”.The default “*componentname.php”* is as follow:

|  |  |  |  |
| --- | --- | --- | --- |
|  | *// application/models/applicationmodel.php*  jimport( ‘joomla.application.component.model’ );  class ComponentNameModelComponentName extends JModel | | |
|  | { |  | |
|  |  | function getModel() |  |
|  |  | { |  |
|  |  | $db =& JFactory::getDBO();  $query = “SELECT column\_name FROM table\_name”  $db ->setQuery( $query );  $result = $db ->loadResult();  return $result; | |
|  |  | } |  |
|  | } |  |  |

The class is extended *JModel* is an abstract class of *Joomla* framework. The class provides the basic functionality for concrete model objects in conjunction with *Joomla’s* MVC pattern.

+ View:

Files that concern display will be in Views folder. HTML code of component on front page is in *default.php* file. The file is placed in “**view/componentname/tmpl/**”. Besides, in “Views” folder, we use file “*view.html.php”* involves PHP code about display on the website.

The default “*view.html.php”* is as follow:

|  |  |  |  |
| --- | --- | --- | --- |
|  | *// application/views/* *applicationview/view.html.php*  jimport( ‘joomla.application.component.view’ );  class ComponentNameViewComponentName extends JView | | |
|  | { |  | |
|  |  | function display($tpl = null) | |
|  |  | { | |
|  |  | $model = & $this->getModel(); | |
|  |  | *// action body* |  |
|  |  | } |  |
|  | } |  |  |

The class is extended *JView* is an abstract class of *Joomla* framework*.* Creating tasks of view is very simple: It retrieves the data to be displayed and pushes it into the template. Data is pushed into the template using the *JView::assignRef* method. (Note: The key (the first argument) passed to the assignRef method cannot be preceded by an underscore i.e. *$this->assignRef('\_greeting',$greeting).* Doing so will cause the *assignRef* method to return false and your variable will not be pushed into the template.)

Our template is very simple: we only want to display the greeting that was passed in from the view.

This file is “**componentname/views/componentname/tmpl/default.php**”:

|  |
| --- |
| <?php defined('\_JEXEC') or die('Restricted access'); ?>  <h1><?php //echo $this->chapterName;?></h1> |

+ Controller:

Our component's action controllers contain our action workflow, and do the work of mapping our requests to the appropriate models and views. No data manipulation is required. All that needs to be done is the appropriate view loaded. We will have only one method in our controller: *display().* Most of the required functionality is built into the *JController* class, so all that we need to do is invoke the *JController::display()* method.

The code for the base controller “**componentname/controller.php**” is:

|  |
| --- |
| <?php  // No direct access  defined( '\_JEXEC' ) or die( 'Restricted access' );  jimport('joomla.application.component.controller');  class ComponentNameController extends JController  {  function display()  {  parent::display();  }  } |

The *JController* constructor will always register a display() task and unless otherwise specified (using the *registerDefaultTask()* method), it will set it as the default task.

This barebones *display()* method isn't really even necessary since all it does is invoke the parent constructor.

The *JController::display()* method will determine the name of the view and layout from the request and load that view and set the layout. When we create a menu item for our component, the menu manager will allow the administrator to select the view that they would like the menu link to display and to specify the layout. A view usually refers to a view of a certain set of data (i.e. a list of cars, a list of events, a single car, a single event). A layout is a way that that view is organized.

In “**administrator/components/com\_componentname**”folder, we create “*componentname.xml”* file to store information of component’s installation: folder tree structure, version, author… “*install.sql”* an “*uninstall.sql”* are files to store command statements into database when installing and uninstalling the component.

**+ PhpBB 3.0:**

We will use open sources: *iGiveTest* and *PhpBB3*. *PhpBB* is a free flat-forum bulletin board software solution that can be used to stay in touch with a group of people or can power your entire website. No other bulletin board software offers a greater complement of features, while maintaining efficiency and ease of use. Best of all, *phpBB* is **completely** **free**. We use *phpBB* version 3.0 to develop forum’s module.

**+ iGiveTest 2.0:**

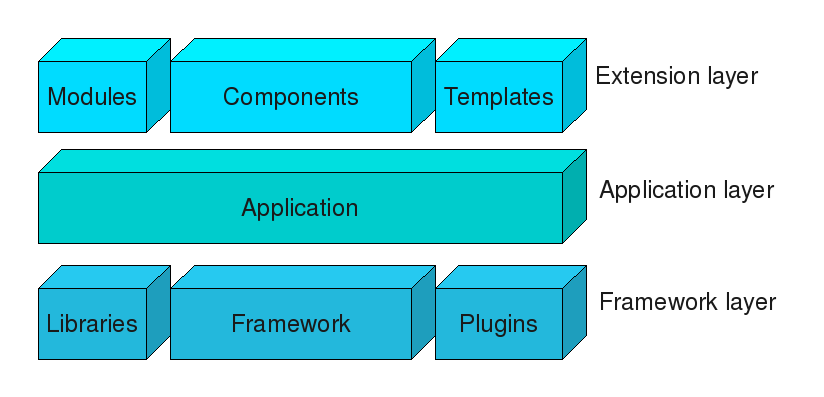
This open source is a comprehensive solution for creating, administering, and providing thorough analysis of tests on the Internet and Intranet. We use the software to develop test’s module and some other functions. The software will support us to create any tests, quizzes and assessments, ask any kind of question. Each test can present questions in random order. You can display one question per page or all questions on one page. Besides grading tests by points or by the number of correct answers, you can define your own grading scale (for example, the 5-grade system A, B, C, D, F, or any other system). Besides it allows you to view testing results for each user or group of users, at a glance and in detail (when the test was taken, how long it took, the answer to each question, points earned for each question, points for the entire test, and so on). Also, you can read and give or change marks for the essay answers.

**+ Joomla Framework:**

We use the framework to develop our components. We will build the framework uponto show the full power and versatility of the MVC design pattern in *Joomla*.

This is an important part of the *Joomla* architecture. It's based on modern object-oriented design patterns that make the *Joomla* core highly maintainable and easily extendable.

Third party developers benefit from the rich and easily accessible functionality that the *Joomla* Framework provides. On this page we'd like to provide you a reference of all classes and respective methods. The links will take you to further information about each class including, where possible, examples of use.

****

**Figure 4.2:** Architecture diagram showing 3-tier architecture of Joomla

*Joomla* is a three tiered system:-

* The top, Extensions layer, consists of [extensions](http://docs.joomla.org/Extension) to the *Joomla* [Framework](http://docs.joomla.org/Framework) and its applications:
  + - [Modules](http://docs.joomla.org/Module)
    - [Components](http://docs.joomla.org/Component)
    - [Templates](http://docs.joomla.org/Template)
* The middle, Application layer, consists of applications that extend the Framework *[JApplication](http://docs.joomla.org/JApplication" \o "JApplication)* class. There are three applications included in the Joomla distribution:
  + - [*JInstallation*](http://docs.joomla.org/JInstallation) is responsible for installing *Joomla* on a web server and is deleted after the installation procedure has been completed.
    - [*JAdministrator*](http://docs.joomla.org/index.php?title=JAdministrator&action=edit&redlink=1) is responsible for the back-end Administrator.
    - [*JSite*](http://docs.joomla.org/JSite) is responsible for the front-end of the website.
* The bottom, Framework layer, consists of:
  + - The Joomla [Framework](http://docs.joomla.org/Framework) itself, whose classes are listed below.
    - [Libraries](http://docs.joomla.org/Library) that are required by the [Framework](http://docs.joomla.org/Framework) or are installed for use by third-party developers.
    - [Plugins](http://docs.joomla.org/Plugin) extend the functionality available in the [Framework](http://docs.joomla.org/Framework).

**+ MySQL:**

We choose *MySql* is our database server because it is completely free. Besides, softwares which we choose as *Joomla*, *PhpBB* and *iGivetest* also use *MySQL*.

*MySQL* is the world's most popular open source database. Whether you are a fast growing web property, technology ISV or large enterprise, *MySQL* can cost-effectively help you deliver high performance, scalable database applications. *MySQL* is a popular choice of database for use in web applications, and is a central component of the widely used [LAMP](http://en.wikipedia.org/wiki/LAMP_(software_bundle)) open source web application software stack—LAMP is an acronym for "[Linux](http://en.wikipedia.org/wiki/Linux), [Apache](http://en.wikipedia.org/wiki/Apache_HTTP_Server), MySQL, [Perl](http://en.wikipedia.org/wiki/Perl)/[PHP](http://en.wikipedia.org/wiki/PHP)/[Python](http://en.wikipedia.org/wiki/Python_(programming_language))".

*MySQL* is an open source database management system and is used in some of the most frequently visited websites on the Internet, including [Flickr](http://en.wikipedia.org/wiki/Flickr) [Nokia.com](http://en.wikipedia.org/wiki/Nokia), [YouTube](http://en.wikipedia.org/wiki/YouTube).

[Free-software](http://en.wikipedia.org/wiki/Free_software)-open source projects that require a full-featured database management system often use *MySQL*. For commercial use, several paid editions are available, and offer additional functionality. Applications which use *MySQL* databases include: [TYPO3](http://en.wikipedia.org/wiki/TYPO3), [Joomla](http://en.wikipedia.org/wiki/Joomla" \o "Joomla), [WordPress](http://en.wikipedia.org/wiki/WordPress" \o "WordPress), [phpBB](http://en.wikipedia.org/wiki/PhpBB" \o "PhpBB), [MyBB](http://en.wikipedia.org/wiki/MyBB" \o "MyBB), [Drupal](http://en.wikipedia.org/wiki/Drupal)

## 4.2.2 Description of System Interface

All screens on the “E-Learning” system use the same format. Since the header, footer, navigation, and toolbar are consistent, the only place where the content will change is in the front page. The area is used to display content available only to the specific page. It will also be used to display a general application error message if the website is unavailable.

*Page Header*: It is displayed at the top of all pages. It contains logo, slogan and menu bar include a set of links about “Thư viện bài tập”, “Thư viện lý thuyết”, “Diễn đàn”, “Đề thi”, “Thành viên”. The header is standard on all “E-Learning” website.

*Navigation:* It displays relevant links to the categories of theory and question that exist on “E-Learning” system, and the affiliate links that are meant to add value to users.

*Page Footer:* It displays the following links:

* About Us
* Contact
* Policies
* Service
* Help

# 4.3. Component Diagram

# 4.4. Detailed Description of Components

## 4.4.1. Theory library

**4.4.1.1. Theory – User Interface Design**

**4.4.1.1.1. Layout**

**4.4.1.1.1.1. Layout – Screen Images**



**Figure 4.3:** “E-Learning” layout page

**4.4.1.1.1.2. Layout – Description of the User Interface**

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | Menu bar | Display menu bar which include links to other sites of the website |
| 2 | Body content | Display the content for specific request of user. |
| 3 | “Home page” panel | The panel contains list of categories in “E-Learning” system. |
| 4 | Specify category | Click on the category to access to other sites of the system. |
| 5 | Login text boxes | Allow user to login to the system by entering username and password to text boxes. |
| 6 | Login button | Click on the button to send request to the system. |
| 7 | Forgot password button | Click on the button to send request to get new password. |
| 8 | Forgot username button | Click on the button to send request to find username. |
| 9 | Create account button | Click on the button to register to the system. |
| 10 | New topic area | The area includes links to new topic. |

**4.4.1.1.2. Concern question**

**4.4.1.1.2.1. Concern question – Screen Images**

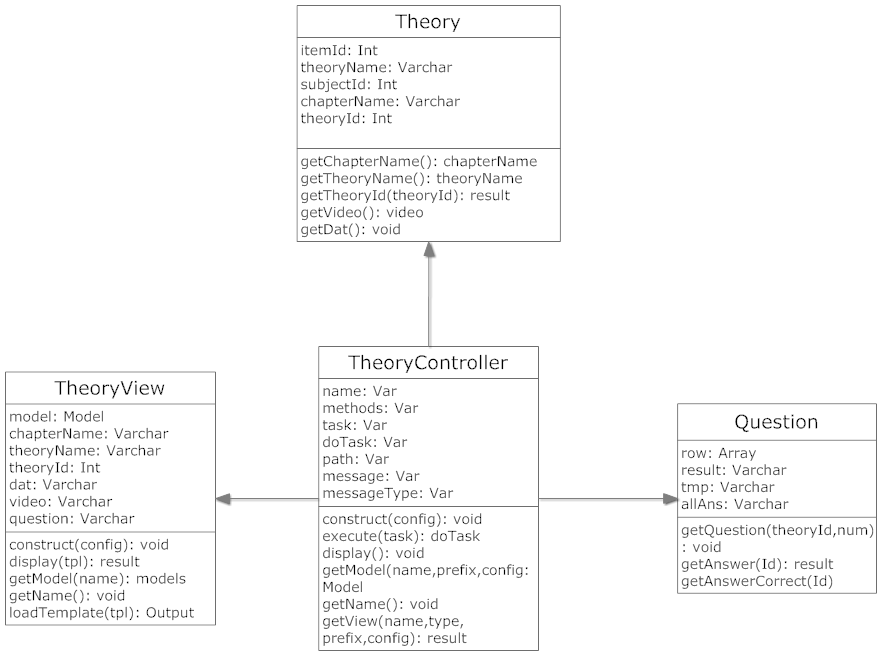


**Figure 4.4:** Question concern of theory panel

**4.4.1.1.2.2. Concern question – Description of the User Interface**

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | Question label | Display description of question concern of theory. |
| 2 | Question content | Show content of question. |
| 3 | Question answer | Show answers of the question |
| 4 | Case study | Show case study of the question |

**4.4.1.2. Theory – Class Diagram**



**Figure 4.5:** Theory’s class diagram

**4.4.1.3. Theory – Class Diagram Explanation**

**4.4.1.3.1. Theory class:**

**4.4.1.3.1.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Itemid | Int | ID of theory, it is set to be unique. |
| 2 | TheoryName | Varchar | Theory name. |
| 3 | Subjectid | Int | ID of subject which has the theory. |
| 4 | Chapter\_name | Varchar | Name of chapter which has the theory. |
| 5 | FileVideoPath | Varchar | Path of video file (if has) of theory |
| 6 | FileDatPath | Varchar | File keeps content of theory. |

**4.4.1.3.1.2. Methods:**

* **Method Get chapter**:

Purpose: get chapter of subject which user selected.

Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Subjectid | Int | ID of subject |
|  | <return> | None | None |



**Figure 4.6:** Sequence diagram for get chapter

* **Method Get theory**:

Purpose: get theory which user selected.

Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Chapter\_name | Varchar | Name of chapter |
|  | <return> | None | None |

 **Figure 4.7:** Sequence diagram for get theory

**4.4.1.3.2. Question class:**

**4.4.1.3.2.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Row | Array | Array of row to show question and answers. |
| 2 | Result | Var | Description of question which concern with theory. |
| 3 | Tmp | Var | Keep content of question. |
| 4 | AllAns | Var | Content of answers. |

**4.4.1.3.2.2 Operations:**

* **Method Get question**:
* Purpose: get question which concern of the theory user selected.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | QuestionID | Int | ID of question |
| 2 | num | Int | Number of question |
|  | <return> | None | None |



**Figure 4.8:** Sequence diagram for get question

* **Method Get answer**:
  + Purpose: get answers which correspond with question.
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | QuestionID | Int | ID of question |
|  | <return> | Result |  |



**Figure 4.9:** Sequence diagram for get answer

* **Method Get answer correct**:
  + Purpose: get answers is correct.
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Id | Int | ID of answer. |
|  | <return> | Result |  |

**4.4.1.3.3. TheoryView class:**

**4.4.1.3.3.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | model | Model | Instance of Model class. |
| 2 | chapterName | Varchar | Name of chapter which includes the theory. |
| 3 | theoryName | Varchar | Name of theory. |
| 4 | theoryId | Int | Id of theory. |
| 5 | Dat | Varchar | Content of theory. |
| 6 | Video | Varchar | Path of video file. |
| 7 | Question | Varchar | Content of question. |

**4.4.1.3.3.2. Methods:**

* **Method Construct:**
  + Purpose: set view name, layout, and charset used by the variable escaping functions.
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Config | Array | Array includes configuration of the system. |
|  | <return> | None | None |

* **Method Display** :
  + Purpose: Execute and display a template script, show chapter and content of theory.
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Tpl | Var | The name of the template files to parse. |
|  | <return> | Result |  |

* **Method GetModel**:
  + Purpose: method to get the model object.
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | The name of the model (optional) |
|  | <return> | Methods |  |

* **Method GetName**:
  + Purpose: method to get the name of the model.
  + Parameters & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
|  | <return> | Name |  |

* **Method LoadTemplate**:
  + Purpose: Load a template file -- first look in the templates folder for an override.
  + Parameters & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Tpl | Var | The name of the template source file. |
|  | <return> | Output | The output of the the template script. |

**4.4.1.3.4. TheoryController class:**

**4.4.1.3.4.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | Name of controller |
| 2 | Methods | Var | Array of class methods |
| 3 | Task | Var | Mapped task that was performed |
| 4 | DoTask | Var | Set of search directories for resources (views) |
| 5 | Path | Var | URL for redirection |
| 6 | Message | Var | Redirect message type |
| 7 | MessageType | Var | Section for the controller |

**4.4.1.3.4.2. Methods:**

* **Method construct:**
  + Purpose: recognized key values include “name”, “default task”, “model path” and “view path” (this list is not meant to be comprehensive).
  + Parameter & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Config | Array | Optional associative array of configuration setting |
|  | <return> | Void |  |

* **Method execute**:
  + Purpose: execute a task by triggering a method in the derived class.
  + Parameter & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Task | Var | Task to perform. If no matching task is found, the “default” task is executed, if defined. |
|  | <return> | retval |  |

* **Method display**:
  + Purpose: typical view method for MVC based architecture. This method is provided as a default implementation, in most cases we will need to override it in our own controllers.
  + Parameter & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
|  | <return> | Void |  |

* **Method getModel**:
  + Purpose: to get model to implement.
  + Parameter & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | Model name |
| 2 | Prefix | Var | Class prefix |
| 3 | Config | Array | Configuration array for model |
|  | <return> | Model | Object model |

* **Method getName**:
  + Purpose: method to get the controller name.
  + Parameter & Return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
|  | <return> | name | The name of the dispatcher |

* **Method getView**:
  + Purpose: method to get a reference to the current view and load it if necessary.
  + Parameter & Return:

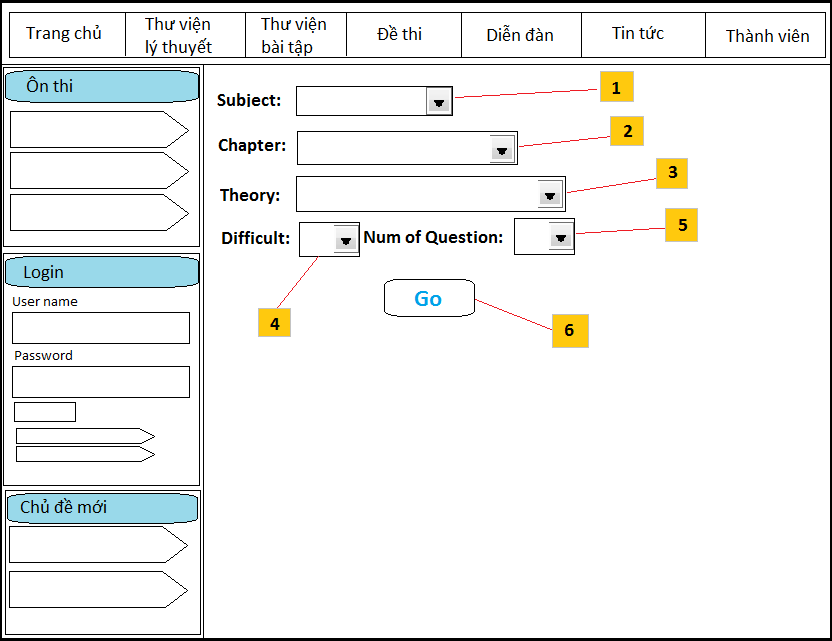
|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | The view name, optional, defaults to the controller name. |
| 2 | Type | Var | The view type |
| 3 | Prefix | Var | The class prefix |
| 4 | Config | Array | Configuration array for view |
|  | <return> | Result | Reference to the view or an error |

## 4.4.2. Exercise library

**4.4.2.1. Exercise – User Interface**

**4.4.2.1.1. Choice Question**

**4.4.2.1.1.1. Choice Question – Screen Images**



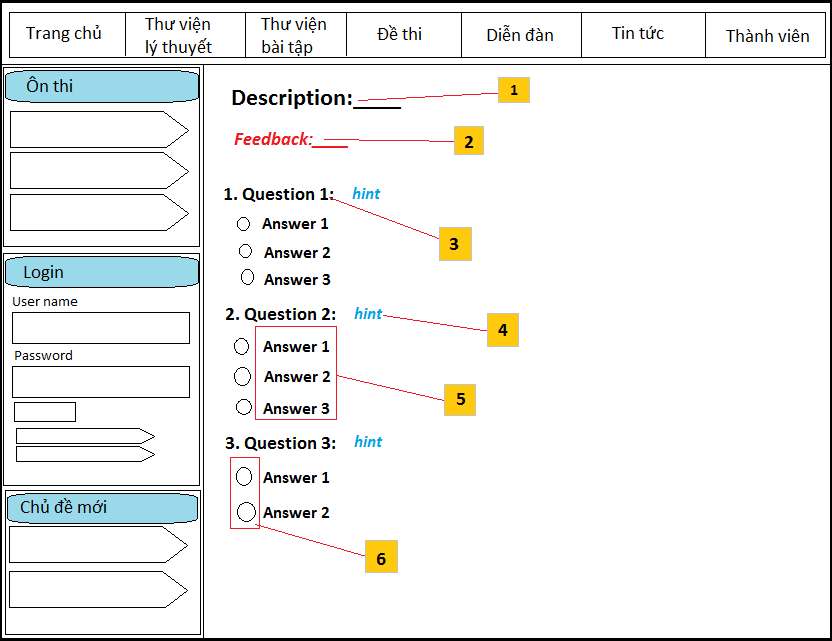
**Figure 4.10:** Choice question page

**4.4.2.1.1.1. Choice Question – Description of the User Interface**

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | List box subject | Allow user to choose subject. |
| 2 | List box chapter | Allow user to choose chapter of subject. |
| 3 | List box theory | Allow user to choose theory of chapter. |
| 4 | List box difficult | Allow user to choose difficult to do. |
| 5 | List box number of question | Allow user to choose number of question to do |
| 6 | Button Go | Click on button to start doing exercises. |

**4.4.2.1.2. Do exercise**

**4.4.2.1.2.1. Do exercise – Screen Images**

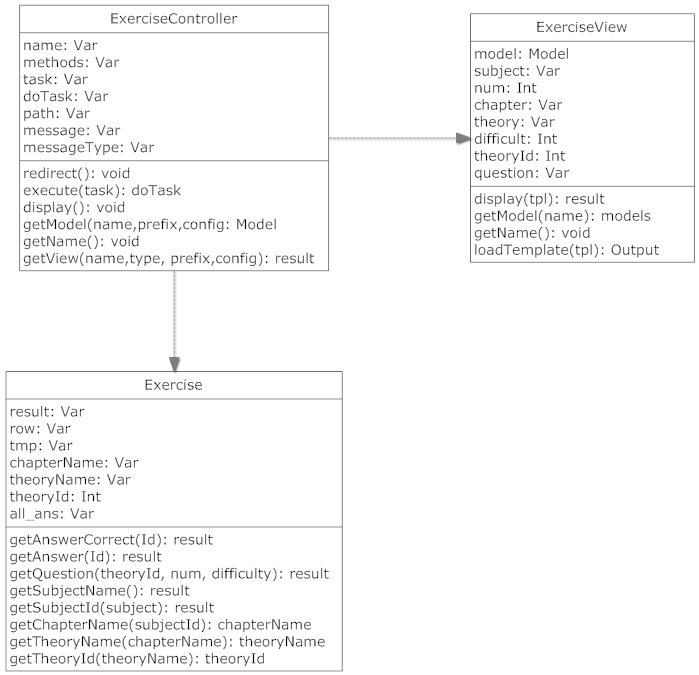


**Figure 4.11:** Do exercise page

**4.4.2.1.2.1. Do exercise – Description of the User Interface**

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | Description label | Show description of exercise includes: subject, chapter, theory, difficult which user choose. |
| 2 | Feedback label | Show feedback of the system about the exercise. |
| 3 | Content of question | Content of question |
| 4 | Button hint | Click on button to show case study following to the question. |
| 5 | Content of answers | Content of answers of question. |
| 6 | Check box answer | Click on check box to choose answers of user. |

**4.4.2.2. Exercise – Class Diagram**

****

**Figure 4.12:** Exercise’s class diagram

**4.4.2.3. Exercise – Class Diagram Explanation**

**4.4.2.3.1. Exercise class:**

**4.4.2.3.1.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Result | Var | Store content of questions and answers which are displayed on front page. |
| 2 | Row | Array | Array of record concern with each of question and answer. |
| 3 | Tmp | Var | Record of question and answers. |
| 4 | ChapterName | Var | Name of chapter |
| 5 | TheoryName | Var | Name of theory |
| 6 | TheoryId | Int | Id of theory |
| 7 | AllAns | Var | Content of answers of question. |

**4.4.2.3.1.2. Methods:**

* **Method getAnswerCorrect:**
* Purpose: get answer which is correct.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Id | Int | ID of question |
|  | <return> | Result |  |

* **Method getAnswer:**
* Purpose: get content of answer
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Id | Int | ID of question |
|  | <return> | Result |  |

* **Method getQuestion:**
* Purpose: get question by theory, number of questions and difficult.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | theoryId | Int | ID of theory |
| 2 | num | Int | Number of question |
| 3 | difficulty | Int | Difficulty of question. |
|  | <return> | Result |  |

* **Method getSubjectName**:
* Purpose: get name of subject.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 |  |  |  |
|  | <return> | Result |  |

* **Method getSubjectId**:
* Purpose: get ID of subject
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | subject | Var | Name of subject |
|  | <return> | result |  |

* **Method getChapterName**:
* Purpose: get name of chapter
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | subjectId | Int | ID of subject |
|  | <return> | chapterName |  |

* **Method getTheoryName**:
* Purpose: get name of theory
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | chapterName | Var | Name of chapter |
|  | <return> | theoryName |  |

* **Method getTheoryId**:
* Purpose: get ID of theory
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | theoryName | Var | Name of theory |
|  | <return> | theoryId |  |

**4.4.2.3.2. ExerciseController class:**

**4.4.2.3.2.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | Name of controller |
| 2 | Methos | Var | Array of class methods |
| 3 | Task | Var | Mapped task that was performed |
| 4 | doTask | Var | Set of search directories for resources (views) |
| 5 | Path | Var | URL for redirection |
| 6 | Message | Var | Redirect message type |
| 7 | messageType | Var | Section for the controller |

**4.4.2.3.2.2. Methods:**

* **Method redirect:**
  + - Purpose: Redirects the browser or returns false if no redirect is set.
    - Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 |  |  |  |
|  | <return> | Boolean |  |

* **Method execute:**
* Purpose: Execute a task by triggering a method in the derived class.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Task | Var | The task to perform. If no matching task is found, the default' task is executed, if defined. |
|  | <return> | retval |  |

* **Method display** :
  + - * Purpose: display view of content in front page.
      * Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 |  |  |  |
|  | <return> | None |  |

* **Method getModel**:
* Purpose: to get model to implement.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | Model name |
| 2 | Prefix | Var | Class prefix |
| 3 | Config | Array | Configuration array for model |
|  | <return> | Model | Object model |

* **Method getName:**
* Purpose: method to get the controller name.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 |  |  |  |
|  | <return> | name |  |

* **Method getView:**
* Purpose: method to get a reference to the current view and load it if necessary.
* Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Name | Var | The view name, optional, defaults to the controller name. |
| 2 | Type | Var | The view type |
| 3 | Prefix | Var | The class prefix |
| 4 | Config | Array | Configuration array for view |
|  | <return> | Result | Reference to the view or an error |

**4.4.2.3.3. ExerciseView class:**

**4.4.2.3.3.1 Attributes:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | Model | Model | The base path of the view |
| 2 | Subject | Var | Name of subject |
| 3 | Num | Int | Number of question |
| 4 | Chapter | Var | Name of chapter |
| 5 | Theory | Var | Name of theory |
| 6 | Difficult | Int | Difficulty of question |
| 7 | TheoryId | Int | ID of theory |
| 8 | Question | Var | Content of question |

4.4.2.3.3.2. **Methods**:

* **Method display**:
  + Purpose: display content of exercise site in fornt page
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | tpl | Var | Content of exercise |
|  | <return> | Result | Content of exercise site. |

* **Method getModel**:
  + Purpose: Method to get the model object
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | name | Var | The name of the model |
|  | <return> | name |  |

* **Method getName**:
  + Purpose: Method to get the view name
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 |  |  |  |
|  | <return> | name | The name of the model |

* **Method loadTemplate**:
  + Purpose: Load a template file -- first look in the templates folder for an override
  + Parameters & return:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Parameter | Type | Description |
| 1 | tpl | Var | The name of the template source file |
|  | <return> | output | The output of the template script. |

## 4.4.3. Open sources integration



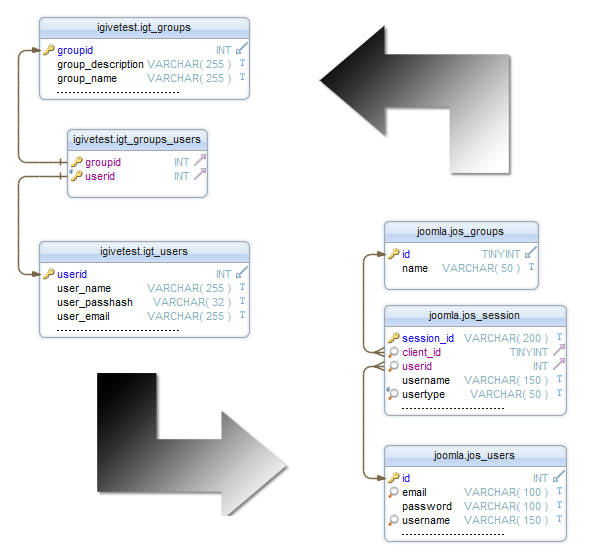
**Figure 4.13:** Synchronization Joomla, Phpbb and iGiveTest

**4.4.3.1. Synchronization databases**

**4.4.3.1.1 Joomla vs iGiveTest**

After analyzing database of *iGiveTest* and *Joomla*, we see they have same tables: users and groups. When installing *Joomla*, tables will have prefix “jos\_”, but *iGiveTest* we can choose any prefix for the open source. We need synchronize 2 open sources which use a common database with database of *iGiveTest* has prefix is “igt\_” and Database of *Joomla* has prefix is “jos\_”.

The pattern below shows table users and table groups of 2 open sources and the relationship of the tables:



**Figure 4.14:** Synchronization table users, groups of iGiveTest and Joomla

**Description of tables and columns and explanation:**

1. Table igivetest.igt\_users:

The table stores information of users. It has more column but we only describe some important column which we want to use.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Userid | INT | No | Unique identifier of user |
| 2 | User\_name | VARCHAR(255) | No | Username |
| 3 | User\_passhash | VARCHAR(32) | No | Password of user which is encrypted MD5 |
| 4 | User\_email | VARCHAR(255) | No | Email of user |

1. Table igivetest.igt\_groups\_users:

The table is joined by table users and table groups with 2 important columns: userid and groupid to set authority for users.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Userid | INT | No | Unique identifier of user |
| 2 | Groupid | INT | No | Unique identifier of group |

1. Table igivetest.igt\_groups:

The table stores information of user’s groups. For example: admin, user, guest, teacher…. The table has more columns but we only use the below column.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Groupid | INT | No | Unique identifier of user’s group |
| 2 | Group\_description | VARCHAR(255) | No | Description of group |
| 3 | Group\_name | VARCHAR(255) | No | Name of group |

1. Table joomla.jos\_groups:

The table stores access level of articles or sites. For example: all of users can view sites if they are at “public”, users have login to the website will view sites which are at “registered”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Id | TINYINT | No | Unique of access level’s group |
| 2 | Name | VARCHAR(50) | No | Name of access level’s group |

1. Table joomla.jos\_session:

The table stores session of users when access to the website.

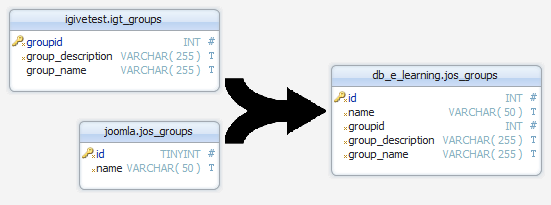
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Session\_id | VARCHAR(200) | No | Session when user access the website |
| 2 | Client\_id | TINYINT | No | Unique identifier of client correspond with groupid |
| 3 | Userid | INT | No | Unique identifier of user |
| 4 | Username | VARCHAR(150) | No | Username of user |
| 5 | Usertype | VARCHAR(50) | No | Type of user |

1. Table joomla.jos\_users:

The table stores information of user. It has more columns but we only describe some important column which we use.

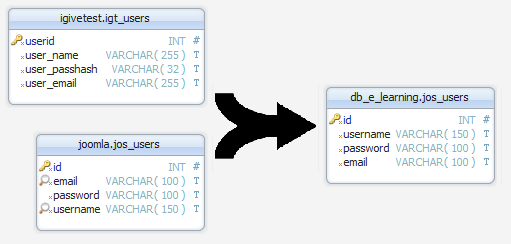
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Id | INT | No | Unique identifier of user |
| 2 | Email | VARCHAR(100) | No | Email of user |
| 3 | Password | VARCHAR(100) | No | Password |
| 4 | Username | VARCHAR(100) | No | Username |

We will merge 2 databases with a prefix “jos” to correspond with Joomla framework. There are common tables: groups and users. We will process as follow:



**Figure 4.15:** Merge table iGiveTest’s groups and table Joomla’s groups

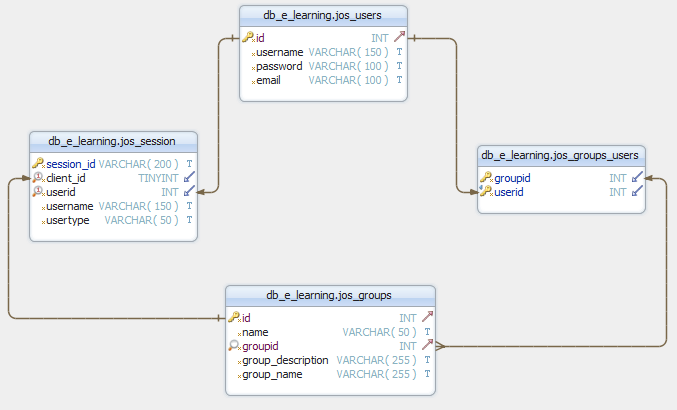
Because two tables don’t have columns which have the same name. We can merge them as above.



**Figure 4.16:** Merge table iGiveTest’s users and table Joomla’s users

The tables have columns which store the same information. The system only need use one of them. Because the system develops on Joomla framework, name of the columns should follow Joomla’s definition. In iGiveTest’s source code, we will edit variables: “*userid*” to “*id*”, “*user\_name*” to “*username*”, “*user\_passhash*” to “*password*” and “*user\_email*” to “*email*”. Other columns don’t change. Then, we merge the tables.

There is the result after merging the common tables of the database system about user:



**Figure 4.17:** Database schema of users and groups after merging

**4.4.3.1.2 Joomla vs Phpbb**

We will synchronize login/logout of *Joomla* and *Pbpbb* by an extension of *Joomla*. We will talk about it below.

**4.4.3.2. Implement tools.**

To synchronize *Joomla* and *Phpbb*, we will use extensions of *Joomla*: ***RokBridge***.

**What is RokBridge?**

*RokBridge* is a non-invasive, bi-directional bridge for *Joomla* and the forum platform *phpBB3*. By following a few simple sets, you can easily integrate the two platforms, providing registration, user syncing and syncing of login sessions.

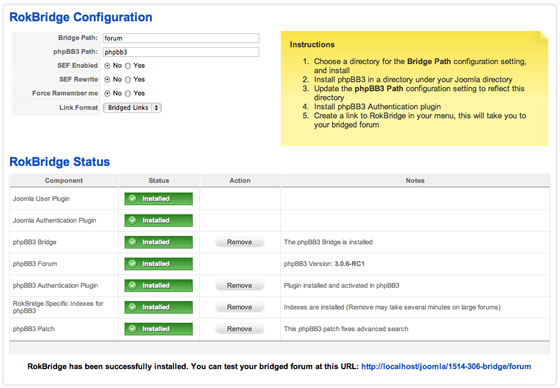
*RokBridge* has been developed with the help of key members of the *phpBB3* team and 100% GPL and Free.

**How to use?**

We use *RokBridge* to take care of the user synchronization login/logout synchronization from *Joomla* to *phpBB3*.

When a user creates an account via *phpBB3*, the user's password is stored in *phpBB3*. If they create an account in *Joomla*, the password is stored in *Joomla*. If they created an account in phpBB3 and try to change it in *Joomla*, it will update the password on *phpBB3* for them, however phpBB3 does not have a 'change password event', so if they create an account in *Joomla* and change the password in *phpBB3*, it doesn't know to update their *Joomla* password. Things can get a bit screwy. We suggest to only allowing users to change passwords in *Joomla*. To ensure this, we will disable the ability in phpBB3 via the user profile by going to the ACP -> System -> Module Management -> User Control Panel -> Profile, and then disabling Account Settings.

The interface of the extension which install successfully will display below:



**Figure 4.18:** RokBridge interface

**4.4.3.3. Integration modules.**

**4.4.3.3.1. Get password in Phpbb**

Because three open sources use different hashing algorithms, users can’t login in test site with their password. For example, a user registers to the website with username: “guest”, password: “guest”. The password will be hashed by *Phpbb* as: “$H$92gtO3cr4hj5gNY5jlvhJhJjqjT9Eq1”. With *iGiveTest*, it will be hashed: “df6f58808ebfd3e609c234cf2283a989”. So, the user only login to *Phpbb* but he can’t login to test site (*iGiveTest*).

So we need modify source code of *Phpbb* and *iGiveTest* to login to three open sources. We do tasks below:

+ Editing phpbb\_hash function:

The function is in “**e-learning-website/phpBB3/includes/functions.php**”. It’s used to hash password as a string. The string will be compared with user’s password stored in database. The algorithms of *Phpbb* as follow:

|  |
| --- |
| \*  \* *Hash the password in PhpBB*  \*  function phpbb\_hash($password)  {  for ($i = 0; $i < $count; $i += 16)  {  $random\_state = md5(unique\_id() . $random\_state);  $random .= pack('H\*', md5($random\_state));  }  $random = substr($random, 0, $count);  }    $hash = \_hash\_crypt\_private($password, \_hash\_gensalt\_private($random, $itoa64),  $itoa64);    if (strlen($hash) == 34)  {  return $hash;  } |
| } |

We will modify the function below:

|  |
| --- |
| \*  \* *Hash the password in PhpBB after modifying*  \*  function phpbb\_hash($password)  {  return md5($password);  } |

+ Editing signinUser function:

The file includes the function is placed in “**e-learning-website/exam/inc/functions.inc.php**”. We will add a query to get password in table *phpbb\_users* when users login to test. The query is below:

|  |
| --- |
| /\*  \* *To get password in table phpbb\_users*  \*/  $sql\_str = "SELECT \* FROM ".$srv\_settings['table\_prefix']."users  LEFT JOIN `phpbb\_users`  ON jos\_users.`username` = phpbb\_users.`username`  WHERE jos\_users.`username`=".$i\_username."  AND phpbb\_users.`user\_password` =".$i\_pass\_hash."  AND jos\_users.`user\_enabled`=1  AND (jos\_users.`user\_expiredate`=0  OR jos\_users.`user\_expiredate`>".$i\_time.")"; |

**4.4.3.3.2. Add authority for user.**

The system use iGiveTest’s open source to develop test site. When synchronizing open sources, we removed register’s module of *iGiveTest*. The module will auto set authority for user. The system only allow user to register by *Phpbb.*

When a user want to do a test, he need login to test site. At the first time to login, if successfully, the system will add authority for user to do a test and view result.

The system will check ID of user into table *jos\_groups\_users.* The table includes 2 columns: userid and groupid. Inside, groupid stores ID of authority of user. In test site, each of group will have different features which the system supports. *iGiveTest* defines authority of user below:

|  |  |
| --- | --- |
| Groupid | Description |
| 1 | Administrator |
| 2 | Teacher |
| 19 | User |
| 20 | Guest |

**Figure 4.19:** Description of groupid in iGiveTest

We will do tasks: getting ID of username and checking if ID exists in table jos\_groups\_users in singinUser function. The function is placed in “**e-learning-website/exam/inc/function.inc.php**”. We will develop code for the tasks as follow:

|  |
| --- |
| *\**  *\*get userid of username*  *\**  $sql\_str1 = "SELECT id FROM jos\_users WHERE username=".$i\_username;  $result1 = mysql\_query($sql\_str1);  while($row = mysql\_fetch\_array($result1))  {  $id = $row['id'];  } |

|  |
| --- |
| \*  \**check userid exist in jos\_group\_user*  *\*if not, adding groupid = ‘19’ for userid*  \*  $sql\_str = "SELECT \* FROM jos\_groups\_users WHERE id=".$id;  $result2 = mysql\_query($sql\_str);  while($row = mysql\_fetch\_array($result2))  {  $groupid = $row['groupid'];  }  $groupid = $i\_rSet1->fields['groupid'];  if ($groupid == "")  {    mysql\_query("INSERT INTO jos\_groups\_users (groupid, id)  VALUES ('19', $id)");  } |

When a user logins at the first time to test, the system will automatically set the value of *groupid* is ‘19’. It is equivalent of user.

**4.4.3.3.3. Logout the system.**

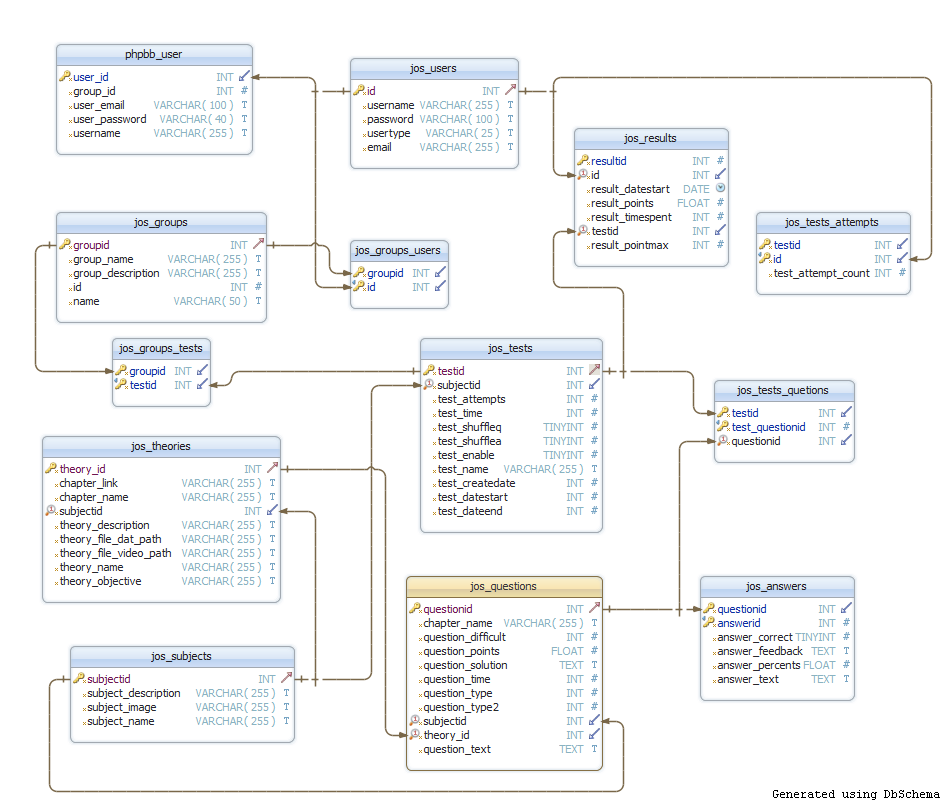
Because the system uses different open sources, each of them must require user to click on button “Sign Out” to logout the website. We develop a function to help user to click once button “Sign Out” to logout the system.

In file “*default.php*” is placed in “**e-learning-website/modules/mod\_login/tmpl/**”, we put the code below:

|  |
| --- |
| \*  \**Call to signout function of iGiveTest*  \*  echo "<script type=\"text/javascript\" src=\"exam/signout.php\"></script>"; |

# 4.5. Database Design or Data Structures

## 4.5.1 Detailed database design for “E-Learning” system:



**Figure 4.20:** Detailed database design for “E-Learning” system

## Table and columns description and explanation

1. Table phpbb\_users:

This table is built by PHPBB. We use the table to store information of user. When user register or edit privacy information, data will be stored in the table. It has 60 column but we only use 5 important columns below to develop the system:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | User\_id | INT | No | Unique identifier of user |
| 2 | Group\_id | INT | No | Unique identifier of user’s group |
| 3 | User\_email | VARCHAR(100) | No | Email of user. |
| 4 | User\_password | VARCHAR(40) | No | User’s password to login to the website. |
| 5 | username | VARCHAR(255) | No | Username of user to login to the website. |

1. Table jos\_users:

Because Joomla and iGiveTest also have database about user, we synchronized data from user’s table of Joomla with user’s table of iGiveTest to create the table. It has more column but we only describe some important column.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Id | INT | No | Unique identifier of user |
| 2 | Username | VARCHAR(255) | No | Username of user to login to the website. |
| 3 | Password | VARCHAR(100) | No | User’s password to login to the website. |
| 4 | Usertype | VARCHAR(25) | No | User’s group |
| 5 | Email | VARCHAR(255) | No | Email of user. |

1. Table jos\_results:

This table stores result about test of user. We use the table to count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Resultid | INT | No | Unique identifier of result |
| 2 | Testid | INT | No | Unique identifier of test |
| 3 | Id | INT | No | Unique identifier of user which have report of test result |
| 4 | Result\_datestart | INT | No | Date which result of test is started |
| 5 | Result\_timespend | INT | No | Time which user spend to test |
| 6 | Result\_points | FLOAT | No | Point of test of user |
| 7 | Result\_pointmax | FLOAT | No | Max point of test |

1. Table jos\_tests\_attempts:

This table stores number of attempts to take a test.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Testid | INT | No | Unique identifier of test |
| 2 | Userid | INT | No | Unique identifier of user which takes the test |
| 3 | Test\_attempt\_count | INT | No | Number of attempts to take the test |

1. Table jos\_groups:

This table stores information of authorities which correspond with user’s groups.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Groupid | INT | No | Unique identifier of iGiveTest’s group. |
| 2 | Group\_name | VARCHAR(255) | No | Name of iGiveTest’s group |
| 3 | Group\_description | VARCHAR(255) | No | Description of iGiveTest’s group |
| 4 | Id | INT | No | Unique identifier of Joomla’s user group |
| 5 | Name | VARCHAR(50) | No | Name of Joomla’s user group |

1. Table jos\_groups\_users:

This table is joined with groups table, users table by groupid and userid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Groupid | INT | No | Unique identifier of iGiveTest’s user group. |
| 2 | Userid | INT | No | Unique identifier of user |

1. Table jos\_groups\_tests:

This table is joined with groups table, tests table by groupid and testid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Groupid | INT | No | Unique identifier of iGiveTest’s user group. |
| 2 | Testid | INT | No | Unique identifier of test |

1. Table jos\_tests:

This table stores information of tests: test name, test time, date which test is started, shuffling question, number of user’s attempts…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Testid | INT | No | Unique identifier of test |
| 2 | Subjectid | INT | No | Unique identifier of subject |
| 3 | Test\_name | VARCHAR(255) | No | Name of test |
| 4 | Test\_time | INT | No | Time which test enable |
| 5 | Test\_datestart | INT | No | Date which test is started |
| 6 | Test\_dateend | INT | No | Date which test ends |
| 7 | Test\_shuffleq | TINYINT | No | Shuffling questions |
| 8 | Test\_shufflea | TINYINT | No | Shuffling answers |
| 9 | Test\_createdate | INT | No | Date which test are created |
| 10 | Test\_enable | TINYINT | No | State of test |
| 11 | Test\_attempts | INT | No | Number of attempts are allowed |

1. Table jos\_tests\_questions:

This table stores questions of tests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Test\_questionid | INT | No | Position of questions in a test |
| 2 | Testid | INT | No | Unique identifier of test |
| 3 | Test\_sectionid | INT | No | Section of questions in a test |
| 4 | questionid | INT | No | Unique identifier of question which is taken in question bank |

1. Table jos\_theories:

This table stores information of theories. It includes: theory name, theory description, objective, reference questions…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Theory\_id | INT | No | Unique identifier of theory. |
| 2 | Theory\_description | VARCHAR(255) | No | Description of theory |
| 3 | Theory\_name | VARCHAR(255) | No | Name of theory |
| 4 | Theory\_Objective | VARCHAR(255) | No | Objectives which theory bring out |
| 5 | Theory\_file\_dat\_path | VARCHAR(255) | No | Content of theory |
| 6 | Theory\_file\_video\_path | VARCHAR(255) | No | Path of theory source (video, text…) |
| 7 | Subjectid | INT | No | Unique identifier of subject of theory |
| 8 | Chapter\_link | VARCHAR(255) | No | Link to site which includes chapters |
| 9 | Chapter\_name | VARCHAR(255) | No | Name of chapter which includes the theory. |

1. Table jos\_subjects:

This table stores information of subjects.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Subjectid | INT | No | Unique identifier of subject |
| 2 | Subject\_description | VARCHAR(255) | No | Description of subject |
| 3 | Subject\_image | VARCHAR(255) | No | Link to subject image file |
| 4 | Subject\_name | VARCHAR(255) | No | Name of subject |

1. Table jos\_questions:

This table is considered as question bank. It include information of questions: question time, content of question, point of question…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Questionid | INT | No | Unique identifier of question |
| 2 | Subjectid | INT | No | Unique identifier of subject about test |
| 3 | Question\_time | INT | No | Time which question is finished |
| 4 | Question\_text | TEXT | No | Content of question |
| 5 | Question\_points | TEXT | No | Point of question |
| 6 | Question\_solution | TEXT | No | Solution of question |
| 7 | Question\_type | INT | No | Type of question |
| 8 | Question\_type2 | TINYINT | No | Allowing partially correct answers if Question\_Type is “Multiple Answer” |
| 9 | Theory\_id | INT | No | Unique identifier of theory which correspond with question. |
| 10 | Chapter\_name | VARCHAR(255) | No | Name of chapter |
| 11 | Question\_difficult | INT | No | Difficulty of question |

1. Table jos\_nswers:

This table stores answers of each question.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column name | Data type | Nullable | Description |
| 1 | Answerid | INT | No | Unique identifier of answer concern with each question |
| 2 | Questionid | INT | No | Unique identifier of question of answer |
| 3 | Answer\_text | TEXT | No | Content of answer |
| 4 | Answer\_feedback | TEXT | No | Feedbacks of answer (if have) |
| 5 | Answer\_correct | TINYINT | No | Answer is correct |
| 6 | Answer\_percents | FLOAT | No | Percentage correct of answer |

# 4.6. Other material (if any)