

VIETNAMESE STUDY SYSTEM FOR JAPANESE

Architecture Design

**Project Code: Veazy**

Document Code: Veazy\_ Architecture Design\_EN

**Ha Noi, 31/05/2016**

Record of change

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Effective Date | Changed Items | A\*M, D | Change Description | New Version |
| 07/07/2014 | Component diagram | M | Modified | v1.1 |
| 11/08/2014 | Sequence diagram | M | Modified | v1.2 |
| 16/08/2014 | Use-case view | M | Modified | v1.3 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1: Record of change

SIGNATURE PAGE

**ORIGINATOR:** Dao Thanh Tung 31/05/2016

Technical Leader

**REVIEWERS:** Nguyen Ngoc Minh 31/05/2016

Project Manager

**APPROVAL:**  Nguyen Van Sang 31/05/2016

Supervisor

TABLE OF CONTENTS

[1 Introduction 6](#_Toc457752307)

[1.1 Purpose 6](#_Toc457752308)

[1.2 Scope 6](#_Toc457752309)

[1.3 Definitions, Acronyms and Abbreviations 6](#_Toc457752310)

[1.4 References 6](#_Toc457752311)

[1.5 Overview 6](#_Toc457752312)

[2 Choice of Architecture design 7](#_Toc457752313)

[2.1 MVC Model 7](#_Toc457752314)

[**1.2.1** **MVC Model overview** 7](#_Toc457752315)

[2.2 Codeigniter 9](#_Toc457752316)

[3 Architectural Representation 10](#_Toc457752317)

[4 Architectural Goals and Constraints 11](#_Toc457752318)

[5 Use-Case View 12](#_Toc457752319)

[5.1 Guest 12](#_Toc457752320)

[**1.5.1** **Register account** 12](#_Toc457752321)

[**1.5.2** **Learning** 13](#_Toc457752322)

[**1.5.3** **Testing** 14](#_Toc457752323)

[**1.5.4** **Dictionary** 15](#_Toc457752324)

[5.2 Member 16](#_Toc457752325)

[**2.5.1** **Private Account Management** 16](#_Toc457752326)

[**2.5.2** **Testing** 17](#_Toc457752327)

[**2.5.3** **Content Reporting** 18](#_Toc457752328)

[5.3 Editor 19](#_Toc457752329)

[**3.5.1** **Private Account Management** 19](#_Toc457752330)

[**3.5.2** **Lesson Management** 20](#_Toc457752331)

[**3.5.3** **Question Management** 21](#_Toc457752332)

[5.4 Administrator 22](#_Toc457752333)

[**4.5.1** **Private Account Management** 22](#_Toc457752334)

[**4.5.2** **System Account Management** 23](#_Toc457752335)

[6 Logical View 24](#_Toc457752336)

[6.1 Overview 24](#_Toc457752337)

[6.2 Architecturally Significant Design Packages 25](#_Toc457752338)

[7 Process View 28](#_Toc457752339)

[8 Deployment view 30](#_Toc457752340)

[9 Quality 30](#_Toc457752341)

# Introduction

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

## Scope

The scope of this document is to depict the architecture of the Useful Japanese Dictionary for Vietnamese website created by UJD\_VN capstone project team.

## Definitions, Acronyms and Abbreviations

|  |  |  |
| --- | --- | --- |
| Acronym | Definition | Note |
| Veazy | Vietnamese Study System for Japanese |  |
| MVC | Model View Control |  |
| IDE | Integrated Development Environment |  |
| Q&A | Question and Answer |  |
| GUI | Graphic User Interface |  |

Table 2: Definitions

## References

* Veazy\_ SRS\_v1.4\_EN.docx
* Veazy \_Data Design\_v1.0\_EN.docx
* <http://en.wikipedia.org/wiki/Model-view-controller>

## Overview

The Software Architecture Document contains the following subsections:

* Section 1: Provide an overview of entire Software Architecture Document.
* Section 2: Choice of Architecture Design
* Section 3: Architectural Representation
* Section 4: Architectural Goals and Constraints
* Section 5: Use-Case view
* Section 6: Logical View
* Section 7: Process View
* Section 8: Deployment view.
* Section 9: Quality.

# Choice of Architecture design

## MVC Model

The purpose of UJD is developing a web application that supports users about Japanese. The system of UJD is structured based on MVC combined with layered architecture and Codeigniter framework

### **MVC Model overview**

MVC is a software architecture pattern that separates the representation of information from user’s interaction with it. The model consists of application data, business rules, logic and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view.

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application sometimes referred to as a layer—for example, model layer.



Figure 1: MVC Model

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them:

* **A controller:** can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).
* **A model:** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
* **A view:** requests from the model the information that it needs to generate an output representation to the user.

#### **Advantages and disadvantages of MVC model**

* Advantages:
* The MVC model demonstrates professionalism in programming and design analysis. It is divided into independent components to help develop applications faster, simpler, easier upgrades and maintenance.
* Many MVC vendor framework tool kits are available.
* Multiple views synchronized with same data model.
* Easy to change or plug in new interface views, allowing updating of interface views with new technologies without overhauling the rest of system.
* Very effective for deployment if graphic, programming and database development professionals are working in a team in a designed project.
* Disadvantages:
* For small projects that apply MVC model caused cumbersome, time consuming in development process.
* Time consuming to transits data between components.
* Not suitable for agent-oriented applications such as interactive mobile and robotics applications.
* Multiple pairs of controllers and views based on the same data model make data model change expensive.
* The division between the View and the Controller is not clear in some cases.

#### **The reason of choosing MVC model**

* Easily manage the complexity of application by dividing the application into three components: model, view and controller.
* Better support for test-driven development.
* It is good support for application built by project team has many developers and designers but still managed application features.
* UJD system is not complete system, now. We built the system that towards extensibility and maintainability in the future.

## Codeigniter

#### **Codeigniter overview**

CodeIgniter is an open source rapid development web application framework, for use in building dynamic web sites with PHP. CodeIgniter is loosely based on the popular MVC development pattern. While view and controller classes are a necessary part of development under CodeIgniter, models are optional.

#### **Advantages and disadvantages of Codeigniter**

Advantages:

* Designed according to the MVC Model
* CodeIgniter is a PHP framework have the fast speed
* CodeIgniter provides libraries cater to the most common tasks in web programming, such as database access, email, data checking, session management, image processing ... to the high function as XML-RPC, encryption, security ...
* Mechanism test data is closely, preventing XSS and SQL Injection of CodeIgniter helps minimize the security risk to the system.
* Disadvantages:
* No supports about some common module
* No supports about AJAX
* No support about **Event-Driven Programming**

#### **The reason of choosing Codeigniter**

* Consistent with UJD system
* Some modules need to reuse shared business logic layer.
* Designed according to the MVC Model

# Architectural Representation

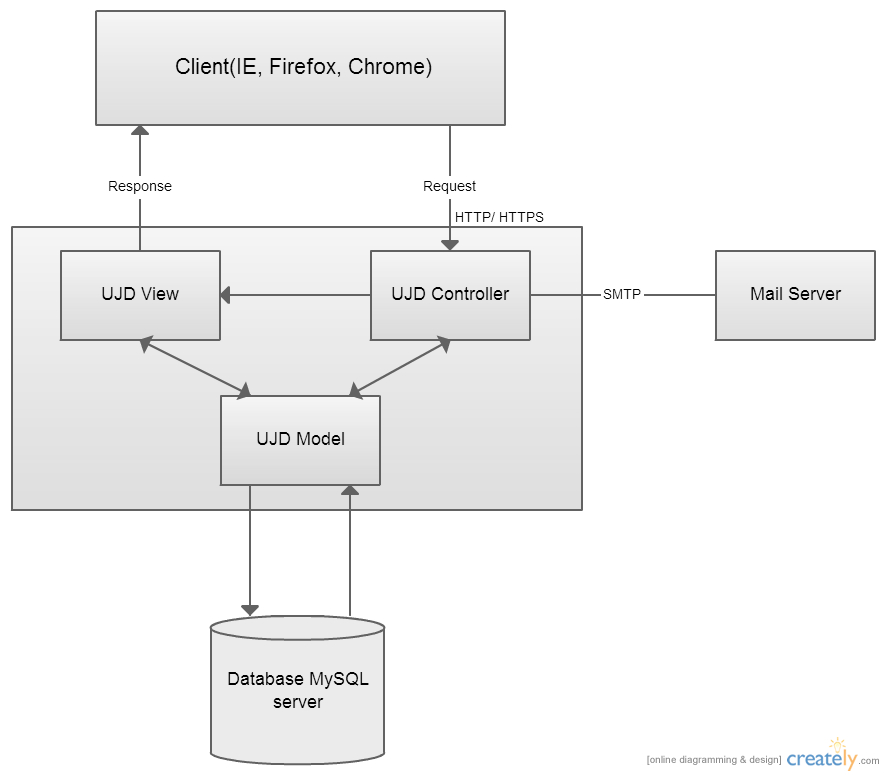


Figure 2: System overview

We follow MVC architecture to implement the UJD\_VN Project. MVC offers architectural benefits over standard JavaScript — it helps us write better-organized and therefore more maintainable code.

**Model** is where the application’s data objects are stored. A model object is in charge of encapsulating application state and one object could be related to other objects establishing a one-to-one or one-to-many relationship. The model object does not talk directly to a View, instead is made available to a controller, which accesses it when needed. When a model changes, typically it will notify its observers that a change has occurred. As with any data object it contains instance variables and getter/setter methods.

**View** is what is presented to the users and how users interact with the system. The view is expected to render the model in a meaningful way to the user. In UJD, the view is made with .cshtml file including css, JavaScript or jQuery, it sends user gestures to controller and allows controller to select view.

**Controller** is the decision maker and the glue between the model and view; it handles user actions and gestures, and responds to user events. For example, in CMS, when a user clicks the “Create” button to create a new contract, the controller for that action is invoked.

The controller will then make changes to the contract model. The view will then render the modified contract model to the display so that user can view the new contract he added in the contract list.

# Architectural Goals and Constraints

**Availability:** The application must be available 95% of time. Users can access to it everywhere from there .Web browser with internet connection.

Maintainability:

* Coding standards and naming conventions
  + Output of the project must include coding standards and naming conventions documentations. Implementation code must be easy to maintain.
  + All code must be clearly commented, including class, method documentations.
  + If some components are reused, the documentations of those components must also be included.
* Design
  + The design of the system must be loosely coupled that chances on some module will not affect others.
* Logging
  + All the errors should be logged, supporting for bug fixing and maintenance.
  + All strange or sensitive situations should also be logged.

**Usability:** Usability Requirements support the following from the perspective of its primary users:

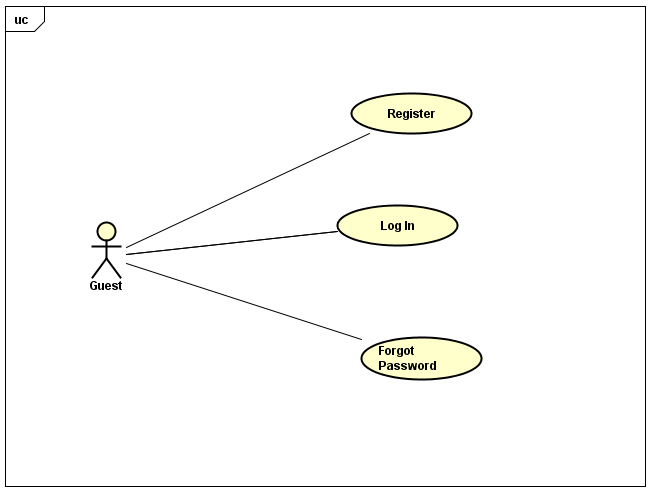
* *Efficiency of use***:** user can complete each function in less than 12 actions
* *Intuitiveness***:** all help/error messages are simple to understand; user can know exactly how to do each feature after one time using it.

**Capacity and scalability:** throughput, storage and growth requirements.

# Use-Case View

## Guest

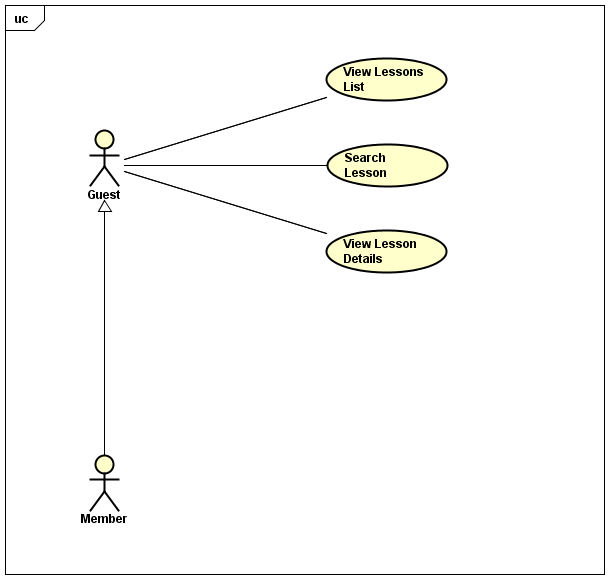
### **Register account**



*Figure 5‑1* ***Register account***

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Register | Guest | Registering new account to log into the system |
| 2 | Log In | Guest | Logging into the system |
| 3 | Forgot Password | Guest | Resetting user’s password |

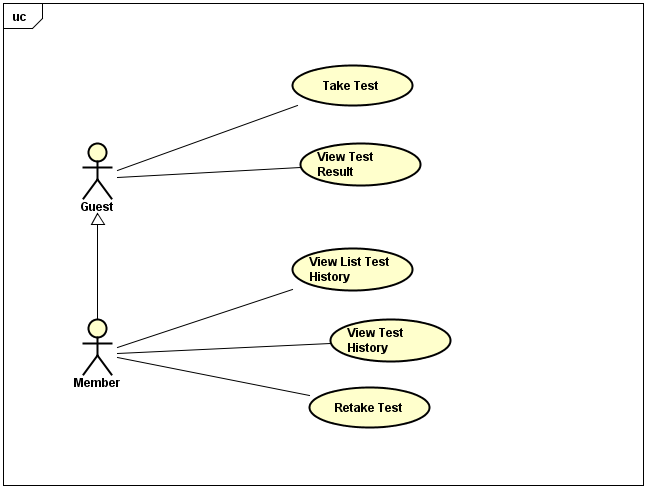
### **Learning**



*Figure 5‑2 Learning*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Lesson List | Guest, Member | Viewing lesson list of courses |
| 2 | Search Lesson | Guest, Member | Searching lesson |
| 3 | View Lesson Details | Guest, Member | Viewing detail of lesson |

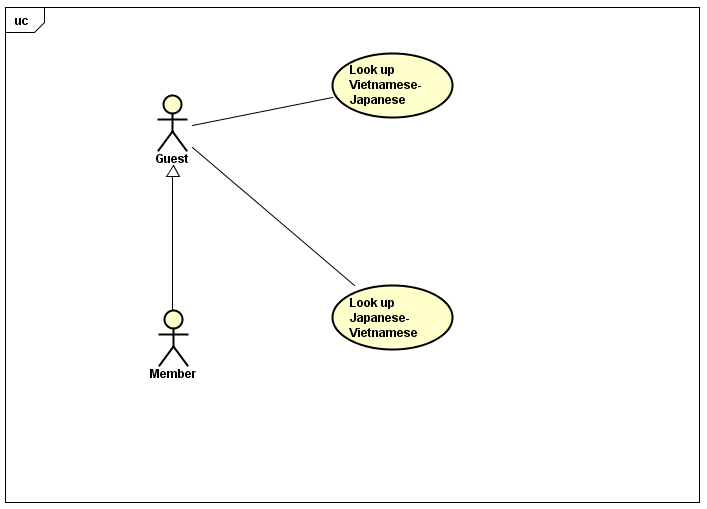
### **Testing**



*Figure 5‑3 Tesing*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Take Test | Guest, Member | Taking a multiple-choice test |
| 2 | View Test Result | Guest, Member | Viewing result of test |

### **Dictionary**

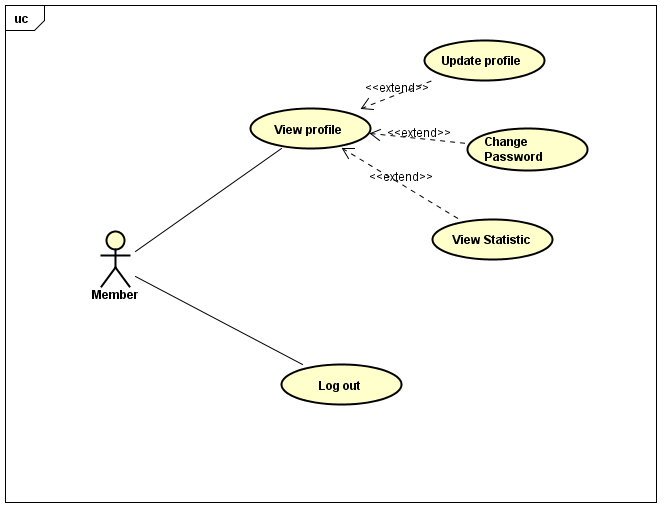


*Figure 5‑4 Dictionary*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Look up Vietnamese-Japanese | Guest, Member | Searching Japanese word through Vietnamese |
| 2 | Look up Japanese-Vietnamese | Guest, Member | Searching Vietnamese word through Japanese |

## Member

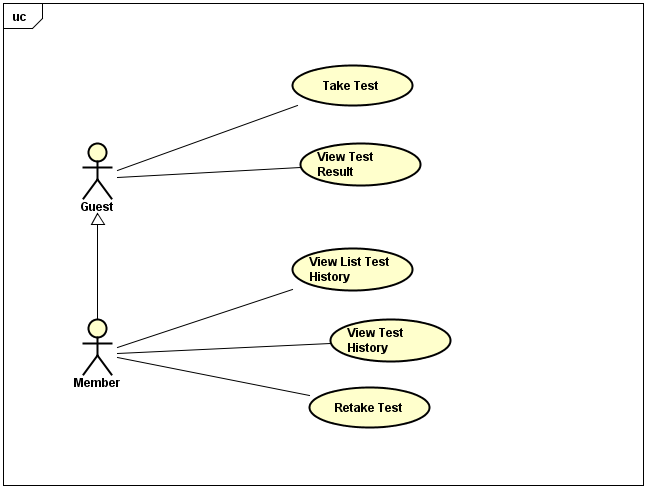
### **Private Account Management**



*Figure 5‑5 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Member | Viewing detail information of their profile |
| 2 | Update Profile | Member | Updating user’s profile |
| 3 | Change Password | Member | Changing user’s password |
| 4 | View Statistic | Member | Viewing statistics of user |
| 5 | Log Out | Member | Logging out of system |

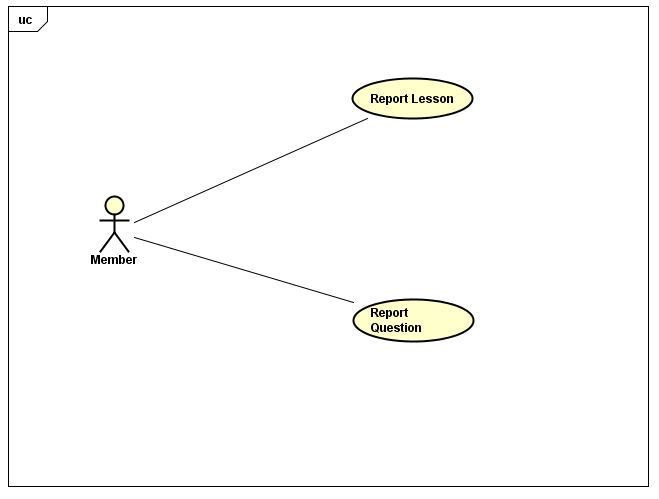
### **Testing**



*Figure 5‑6 Testing*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View List Test History | Member | Viewing list of test history |
| 2 | View Test History | Member | Viewing detail of test history |
| 3 | Retake Test | Member | Retaking a test in list test history |

### **Content Reporting**

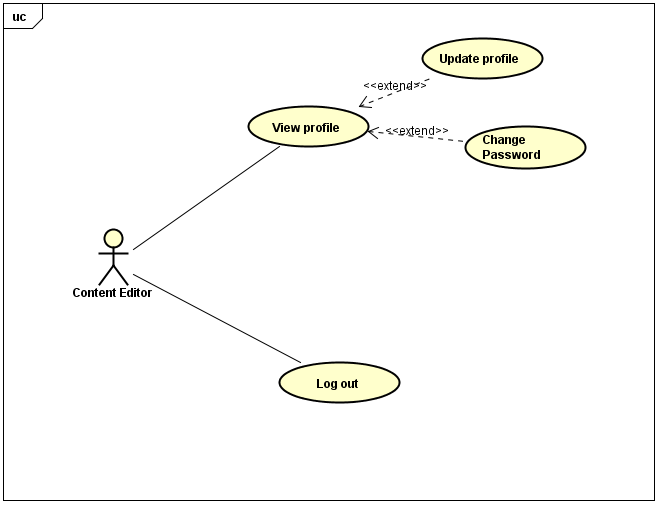


*Figure 5‑7 Content Reporting*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Report Lesson | Member | Reporting content of lessons that having wrong content |
| 2 | Report Question | Member | Reporting content of questions that having wrong content |

## Editor

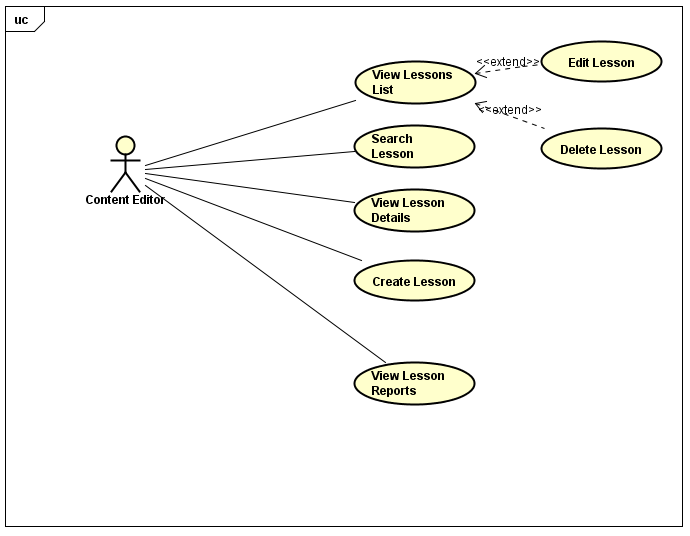
### **Private Account Management**



*Figure 5‑8 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Content Editor | Viewing detail information of their profile |
| 2 | Update Profile | Content Editor | Updating user’s profile |
| 3 | Change Password | Content Editor | Changing user’s password |
| 4 | Log Out | Content Editor | Logging out of system |

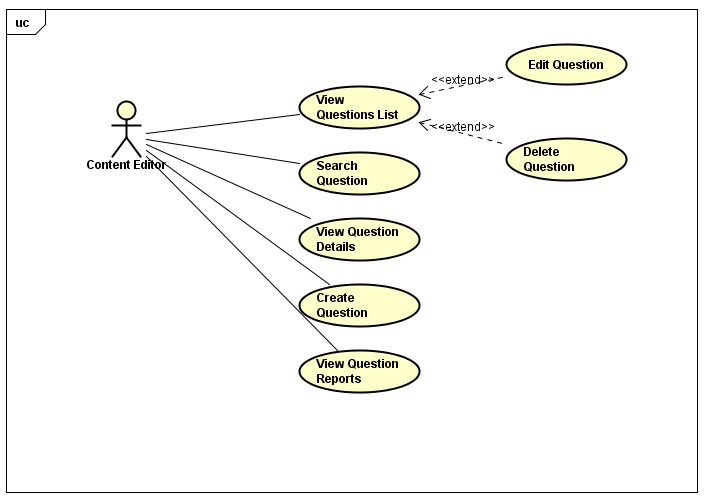
### **Lesson Management**



*Figure 5‑9 Lesson Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Lessons List | Content Editor | Viewing lesson list of courses |
| 2 | Search Lesson | Content Editor | Searching lesson |
| 3 | View Lesson Details | Content Editor | View detail of lesson |
| 4 | Edit Lesson | Content Editor | Editing content of lesson |
| 5 | Delete Lesson | Content Editor | Deleting lesson |
| 6 | Create Lesson | Content Editor | Creating new lesson of course |
| 7 | View Lesson Reports | Content Editor | Viewing list report about wrong content from member |

### **Question Management**

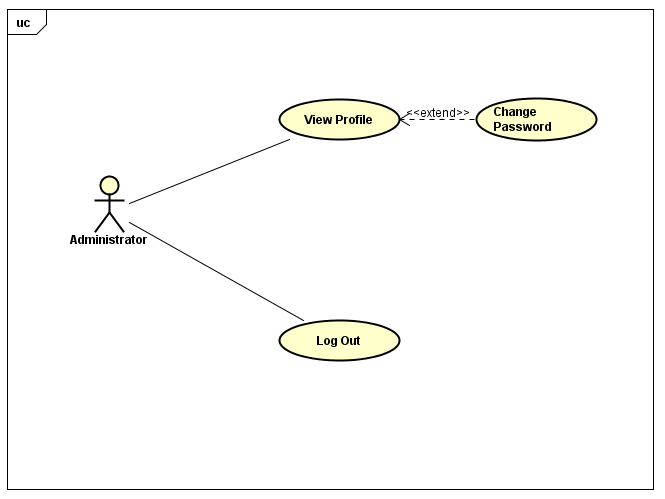


*Figure 5‑10 Question Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Questions List | Content Editor | Viewing question list |
| 2 | Search Question | Content Editor | Searching question |
| 3 | View Question Details | Content Editor | View detail of question |
| 4 | Edit Question | Content Editor | Editing content of question |
| 5 | Delete Question | Content Editor | Deleting question |
| 6 | Create Question | Content Editor | Creating new question of course |
| 7 | View Question Report | Content Editor | Viewing list report about wrong content from member |

## Administrator

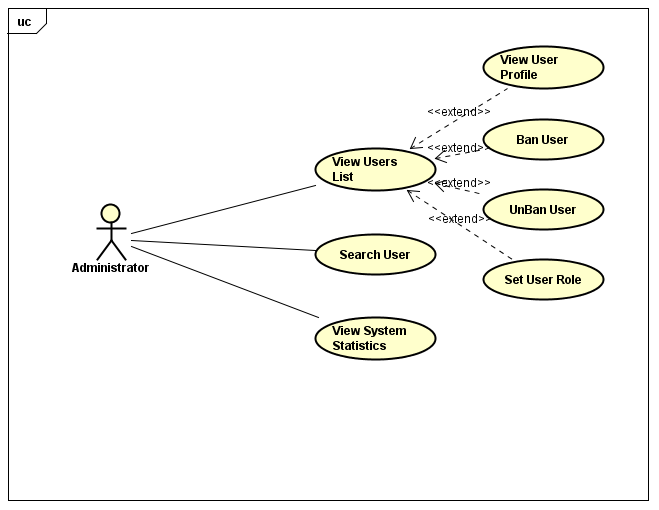
### **Private Account Management**



*Figure 5‑11 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Administrator | Viewing detail information of their profile |
| 2 | Change Password | Administrator | Changing password |
| 3 | Log Out | Administrator | Logging out of system |

### **System Account Management**



*Figure 5‑12 System Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Users List | Administrator | Viewing list of user |
| 2 | Search User | Administrator | Searching user |
| 3 | View User Profile | Administrator | Viewing detail information of user |
| 4 | Ban User | Administrator | Blocking user of system |
| 5 | Unban User | Administrator | Unblocking user |
| 6 | Set User Role | Administrator | Setting role for user |
| 7 | View System Statistics | Administrator | Viewing statistics of system |

# Logical View

## Overview

Logical View includes Package diagram and Class diagram. Package diagram describes the organization of packages and elements. Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system

* Controller contain the interface between
* Associated models
* Associated views
* The input devices (e.g., keyboard, pointing device, time).
* Send commands to the model to update the model's state.
* Model is:
  + the domain-specific software simulation
  + Or implementation of the application's central structure.
* View deal with everything graphical
* Requests data from their model
* Display the data

## Architecturally Significant Design Packages

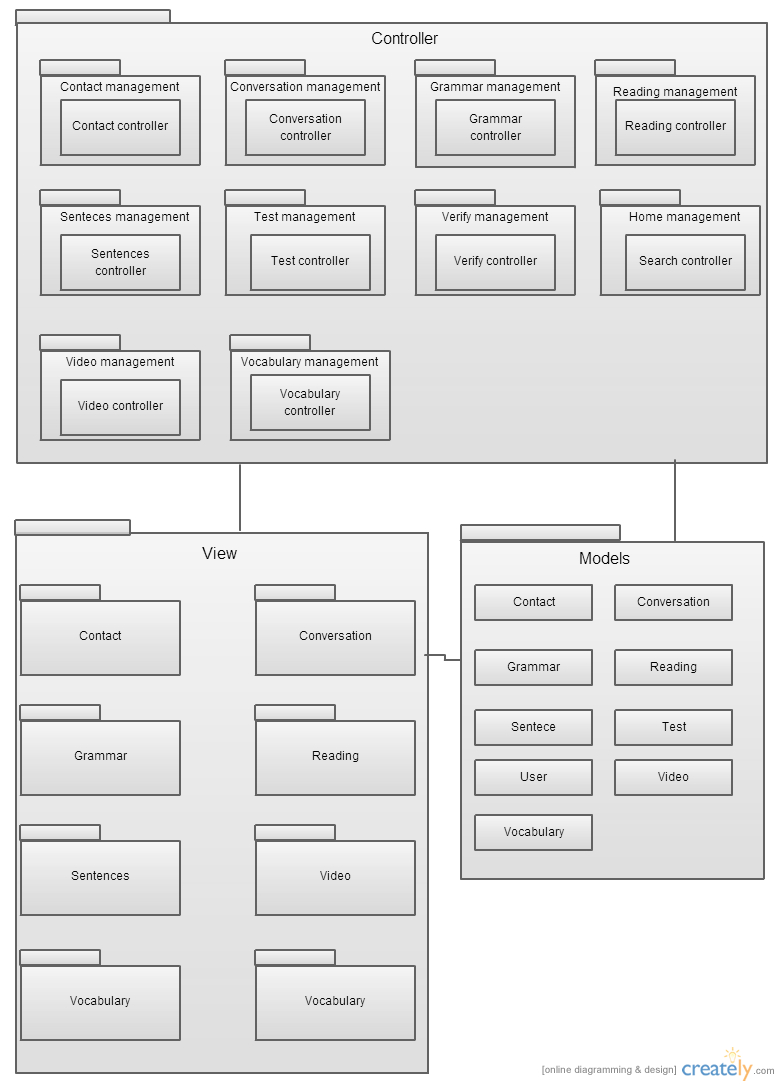


Figure 17: Package Diagram

* Model

|  |  |  |
| --- | --- | --- |
| No. | Controller class | Role |
| 1 | Contact | Description entity of Contact in database |
| 2 | Conversation | Description entity of Conversation in database |
| 3 | Grammar | Description entity of Grammar in database |
| 4 | Reading | Description entity of Reading in database |
| 5 | Sentence | Description entity of Sentence in database |
| 6 | Test | Description entity of Test in database |
| 7 | User | Description entity of User in database |
| 8 | Video | Description entity of Video in database |
| 9 | Vocabulary | Description entity of Vocabulary in database |

* Controller

|  |  |  |
| --- | --- | --- |
| No. | Controller class | Role |
| 1 | ContactController | * Receive request contact’s information from client. * Handle request from client and call method in Contact Model to get data from Database. * Respond data back to Contact View. |
| 2 | ConversationController | * Receive request conversation’s information from client. * Handle request from client and call method in Conversation Model to get data from Database. * Respond data back to Conversation View. |
| 3 | GrammarController | * Receive request grammar’s information from client. * Handle request from client and call method in Grammar Model to get data from Database. * Respond data back to Grammar View. |
| 4 | ReadingController | * Receive request reading’s information from client. * Handle request from client and call method in Reading Model to get data from Database. * Respond data back to Reading View. |
| 5 | SentenceController | * Receive request sentence’s information from client. * Handle request from client and call method in Sentence Model to get data from Database. * Respond data back to Sentence View. |
| 6 | TestController | * Receive request test’s information from client. * Handle request from client and call method in Test Model to get data from Database. * Respond data back to Test View. |
| 7 | VerifyController | * Receive request user’s information from client. * Handle request from client and call method in User Model to get data from Database. * Respond data back to Homepage View. |
| 8 | SearchController | * Receive request search’s information from client. * Handle request from client and call method in Search Model to get data from Database. * Respond data back to Search View. |
| 9 | VideoController | * Receive request video’s information from client. * Handle request from client and call method in Video Model to get data from Database. * Respond data back to Video View. |
| 10 | VocabularyController | * Receive request vocabulary’s information from client. * Handle request from client and call method in Vocabulary Model to get data from Database. * Respond data back to Vocabulary View. |

* View

Include many .php files

# Process View



Figure 18: Sequence diagram

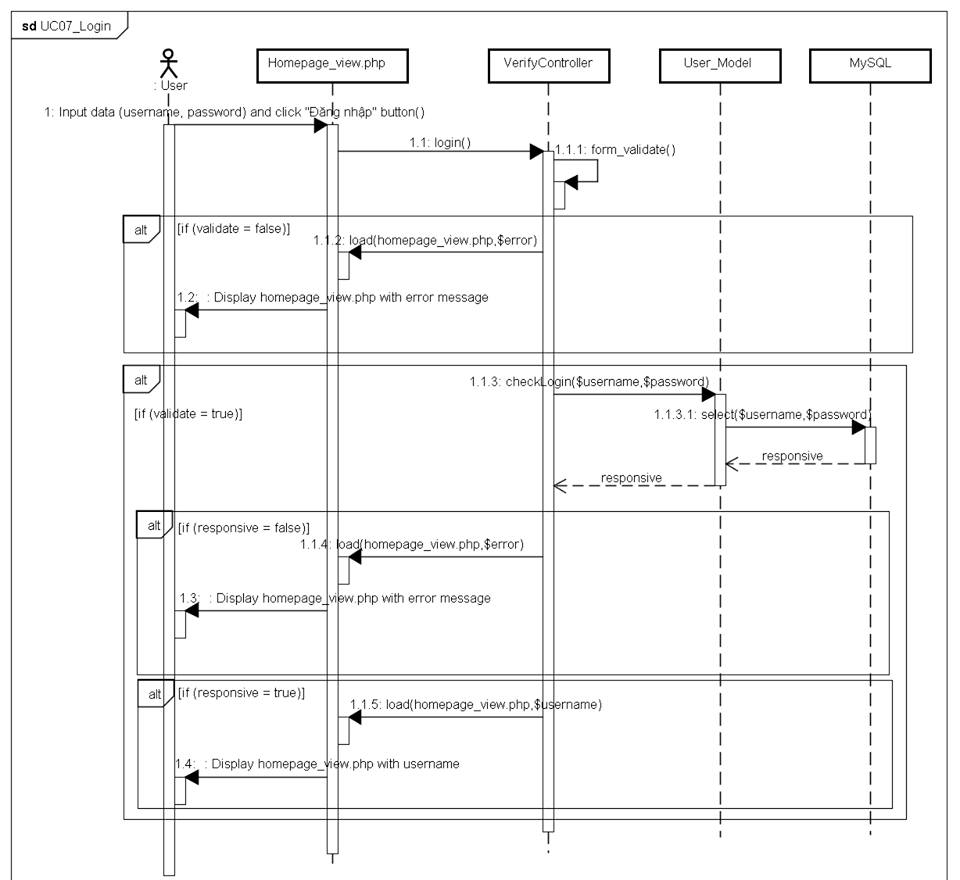


Figure 19: Login sequence

User input data(username, password) and click “Đăng nhập” button, user send a request login to verifyController class call function login()

If data entered validate is false, function login() will call function load(homepage\_view.php.$error) in verfifyController

If data enterd is validate is true, userModel will call function checkLogin($username,$password) in verifyController and send request select($username,$password) to MySQL. MySQL will response to ViewController

If responsive is false, function load(homepage\_view.php,$error) is called, display homepage\_view.php with error message to User

If responsive if tru, function load(homepage\_view.php,$username) is called, display homepage\_view.php with username

# Deployment view

Deployment view of website

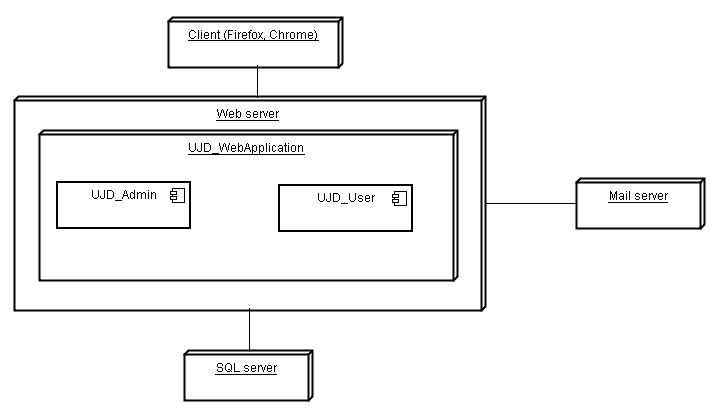


Figure 19: Deployment view

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| 1 | MySQL sever | MySQL server use to store system’s data. MySQL5.6.16 or higher |
| 2 | Mail server | Mail server use to interact with user’s email |
| 3 | Web server | Web server is host system’s website. |
| 4 | Client | Client is web browser to use system. Firefox 20, Chrome 36 or higher. |

# Quality

Reference to: “UJD\_VN\_Software requirement specification\_v1.2\_EN”