|  |
| --- |
| C:\Users\Tsubaki Yukino\Desktop\FlyAwayPlus Project\Capture.JPG |
| **FLY AWAY PLUS** |
| **Architecture Design** |
|  |
| |  |  |  | | --- | --- | --- | | **Fly Away Plus** | | | | **Group Members** | Hoàng Nghĩa Đức | SE02882 | | Dương Thanh Hải | SE02856 | | Nguyễn Minh Hoàng | SE02819 | | Trần Mạnh Hiếu | SE02778 | | Phan Tiến Lực | SE02923 | | Lê Minh Thúy | SE02881 | | **Supervisor** | Mr. Nguyễn Văn Sang | | | **Project code** | FAP | | |
|  |

**- Hanoi, 05/2015 –**

**RECORD OF CHANGE**

\*A-Added; M-Modified; D-Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Effective Date | Change Items | A\*M, D | Change Description | New Version |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

*Table 1: Record of change*

**SIGNATURE PAGE**

|  |  |  |
| --- | --- | --- |
| **ORIGINATOR** | Trần Mạnh Hiếu |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **REVIEWERS** | Lê Minh Thúy |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **APPROVAL** | Nguyễn Văn Sang |  |
|  | Supervisor |  |
|  |  |  |

Table of Contents

[**1.** **INTRODUCTION** 4](#_Toc420360201)

[1.1. Purpose 4](#_Toc420360202)

[1.2. Scope 4](#_Toc420360203)

[1.3. Definitions, Acronyms, Abbreviations 4](#_Toc420360204)

[1.4. References 4](#_Toc420360205)

[1.5. Overview 4](#_Toc420360206)

[**2.** **CHOICE OF ARCHITECTURE DESIGN** 4](#_Toc420360207)

[2.1. MVC Model 4](#_Toc420360208)

[2.1.1. MVC Model overview 4](#_Toc420360209)

[2.1.2. Advantages and disadvantages of MVC model 4](#_Toc420360210)

[2.1.3. The reason of choosing MVC Model 4](#_Toc420360211)

[2.2. Codeigniter 4](#_Toc420360212)

[2.2.1. Advantages and disadvantages of Codeigniter 4](#_Toc420360213)

[2.2.2. The reason of choosing Codeigniter 4](#_Toc420360214)

[**3.** **ARCHITECTURAL REPRESENTATION** 4](#_Toc420360215)

[**4.** **ARCHITECTURAL GOALS AND CONSTRAINTS** 4](#_Toc420360216)

[**5.** **USE-CASE VIEW** 4](#_Toc420360217)

[5.1. User module 4](#_Toc420360218)

[5.1.1. Common module 4](#_Toc420360219)

[**6.** **LOGICAL VIEW** 4](#_Toc420360220)

[6.1. Overview 4](#_Toc420360221)

# **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 FAP capstone project team.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| Acronym | Definition | Note |
| FAP | Fly Away Project |  |
| MVC | Model view control |  |
| IDE | Integrated development environment |  |
| Q&A | Question and answer |  |
| GUI | Graphic user interface |  |

## References

* FAP\_ Software Requirements Specification\_v1.0\_EN.docx
* FAP \_Data Design\_v1.0\_EN.docx
* Software Architecture Design Illuminated Book
* <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 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 FAP 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**

This application includes two parts:

* The first part is User module. User module includes member and visitor.
* Next part is Admin module. In admin side includes admin to manage vocabulary, grammar, video, test, listening, reading and managing member’s account.

## User module

### Common module

*Figure 3: Common module*

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
|  |  |  |  |
|  |  |  |  |

# **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