|  |
| --- |
|  |
| Capstone Project Document |

**DANDELION**

Report #2 – Architecture Design

|  |  |  |
| --- | --- | --- |
| **Dandelion** | | |
| **Group Members** | Vũ Công Chính | SE02585 |
| Lưu Ngọc Mạnh | SE02619 |
| Nguyễn Minh Huy | SE02723 |
| Dương Đức Anh | SE02700 |
| Vũ Ngọc Trung | SE02967 |
| Cao Thị Phương Mai | SE02908 |
| **Supervisor** | Mr. Nguyễn Văn Sang | |
| **Project code** | DDL | |

**- Hanoi, 09/2015 -**

# SIGNATURE PAGE

AUTHOR: Lưu Ngọc Mạnh 15/09/2015

Project Manager

REVIEWERS: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 16/09/2015

Team member

APPROVAL: Nguyễn Văn Sang --/--/--15

Supervisor

Record of change

\*A - Added M - Modified D – Deleted

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Effective Date | Changed Item | A,M,D | Change Description | Reason for Change | Rev. Number |
| 19/Sep/2015 | Create Architecture Design | A | First version | Create Architecture Design | 1.0 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

TABLE CONTENTS

[SIGNATURE PAGE 1](#_Toc430708079)

[1 PROJECT OVERVIEW 3](#_Toc430708080)

[1.1 Purpose 3](#_Toc430708081)

[1.2 Scope 3](#_Toc430708082)

[1.3 Definitions, Acronyms, Abbreviations 3](#_Toc430708083)

[1.4 References 3](#_Toc430708084)

[1.5 Overview 3](#_Toc430708085)

[2 CHOICE OF ARCHITECTURE DESIGN 4](#_Toc430708086)

[2.1 MVC Model 4](#_Toc430708087)

[2.1.1 MVC Model Overview 4](#_Toc430708088)

[2.1.2 Advantages and disadvantages of MVC Model 5](#_Toc430708089)

[2.1.3 The reasons for choosing MVC Model 5](#_Toc430708090)

[2.2 .Net Framework 6](#_Toc430708091)

[2.2.1 .Net Framework Overview 6](#_Toc430708092)

[2.2.2 Advantages and disadvantages of .Net Framework 6](#_Toc430708093)

[2.2.3 The reasons for choosing .Net Framework 6](#_Toc430708094)

[2.3 AngularJS 8](#_Toc430708095)

[2.3.1 AngularJS Overview 8](#_Toc430708096)

[2.3.2 Advantages and disadvantages of AngularJS 9](#_Toc430708097)

[2.3.3 The reason for choosing AngularJS 10](#_Toc430708098)

[3 ARCHITECTURAL REPRESENTATION 11](#_Toc430708099)

[4 ARCHITECTURAL GOALS AND CONSTRAINTS 12](#_Toc430708100)

[5 Use-case View 13](#_Toc430708101)

[6 Logical View 13](#_Toc430708102)

[6.1 Overview 13](#_Toc430708103)

[6.2 Architecturally Significant Design Packages 13](#_Toc430708104)

[7 Process view 13](#_Toc430708105)

[8 Deployment View 13](#_Toc430708106)

[9 Quality 13](#_Toc430708107)

# PROJECT OVERVIEW

## 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 Dandelion website created by DDL capstone project team.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| Acronym | Definition | Note |
| DDL | Dandelion |  |
| DB | Database |  |
| MVC | Model view control |  |
| IDE | Integrated development environment |  |
| Q&A | Question and answer |  |
| GUI | Graphic user interface |  |

Table 1‑1: Definitions and Acronyms

## References

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

## 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 DDL is developing as a crowdfunding website, where people can start their ideas and make it real. The system of FAP is structured based on MVC combined with layered architecture.

### MVC Model Overview

The **model-view-controller or MVC** is software architecture commonly used for creating web applications or software. In other words, it's a structure for web applications to follow in order to ensure efficiency and consistency. Many of the most popular frameworks use the MVC architecture, including ASP.NET, CodeIgniter, Zend, Django, and Ruby on Rails. At the same time, there are many web developers who don't use a coding framework yet still set up their applications to follow the MVC structure.

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 is sometimes referred to as a layer—for example, model layer.



Figure 2‑1: MVC Model

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

* **Controller:** The controller can be considered the "middle man" of the application. It works with the user, taking in data, and then working with the model to get the appropriate data or calculation, and then working with the view to show the response to the user.
* **Model:** A model is simply a representation of something we need to deal within our application. It is a "model" for something we must represent in code, such as a book, user, bank account, or whatever. The model is responsible for holding the functions and variables that are involved with whatever it's representing. You can think of a model's logic as the core concept to object oriented programming — models are just our "classes". However, don't let this confuse you as controllers are technically structured as classes as well.
* **View:** Finally, after the controller requests information from the model it sends it to a view. A view is just like the application's templating system — there might be a view for a certain type of page layout *(profile page)*, a mobile view, or a view for a particular theme/skin. A view will contain all of the markup, CSS, and etc. that you traditionally use with creating a static web page.

### Advantages and disadvantages of MVC Model

* Advantages:
* MVC separates system into components, which can be developed, maintained and upgraded individually without pausing system.
* Develop tools is useful and easy to use.
* Large of documentary sources.
* 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 reasons for choosing MVC Model

* MVC makes parts of system can be developed individually and simultaneously to reduce developing time.
* Better support for test-driven development.
* Tools is useful and documentary source is large makes MVC is easy to develop.
* DDL system is not complete system, now. We built the system that towards extensibility and maintainability in the future.

## .Net Framework

### .Net Framework Overview

.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages.

The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

* To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
* To provide a code-execution environment that minimizes software deployment and versioning conflicts.
* To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
* To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
* To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
* To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

### Advantages and disadvantages of .Net Framework

* Advantages
* Less time to produce product
* Less Complexity.
* Easily to access complex O.S functions
* Easily to build Data Oriented Project, support huge DB functions.
* Managed
* Support Both Windows and Web Application.
* Easy to create Dynamic sites.
* Disadvantages
* Not suitable for High End Application
* Low performance compare to C, C++.
* Unavailability of build in methods.
* .NET framework is free to download but Code Editor is costly.
* Only few O.S supports .NET.

### The reasons for choosing .Net Framework

* Consistent with DDL system.
* There are many plugins and resources which support creating a website using MVC model.
* Many members can use and have experiences using C# (a language of .Net Framework)

## AngularJS

### AngularJS Overview

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

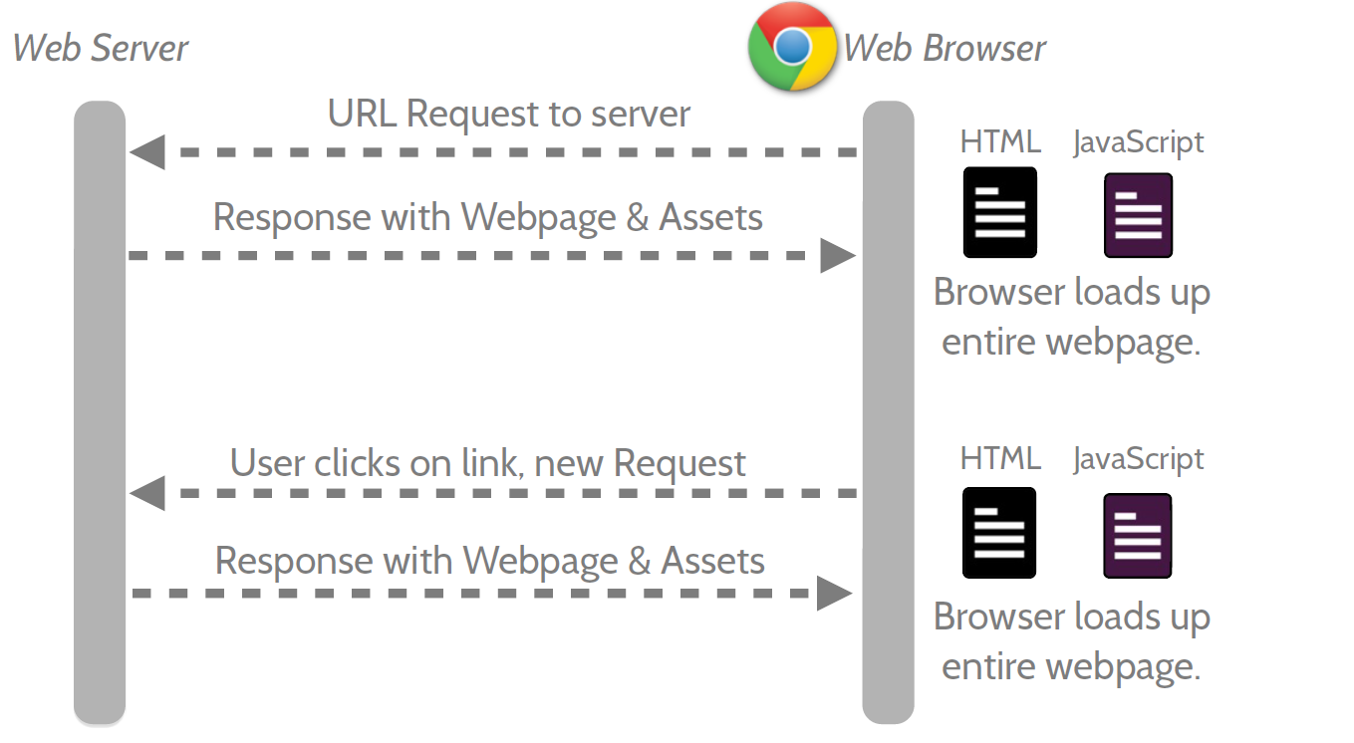


Figure 2‑2: Multi Page Application



Figure 2‑3: Single Page Application

* Features:
* AngularJS is a powerful JavaScript based development framework to create RICH Internet Application (RIA).
* AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC (Model View Controller) way.
* Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
* AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain.

* Core futures: following are most important core features of AngularJS:
* **Data-binding:** It is the automatic synchronization of data between model and view components.
* **Scope:** These are objects that refer to the model. They act as a glue between controller and view.
* **Controller:** These are JavaScript functions that are bound to a particular scope.
* **Services:** AngularJS come with several built-in services for example $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
* **Filters:** These select a subset of items from an array and returns a new array.
* **Directives:** Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ngBind, ngModel...)
* **Templates:** These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".
* **Routing:** It is concept of switching views.
* **Model View Whatever:** MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.
* **Deep Linking:** Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
* **Dependency Injection:** AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

### Advantages and disadvantages of AngularJS

* Advantages:
* AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
* AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience
* AngularJS code is unit testable.
* AngularJS uses dependency injection and make use of separation of concerns.
* AngularJS provides reusable components.
* With AngularJS, developer write less code and get more functionality.
* In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
* Disadvantages:
* **Not Secure:** Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
* **Not degradable:** If your application user disables JavaScript then user will just see the basic page and nothing more.

### The reason for choosing AngularJS

* AngularJS supports to create a website faster via Rest Api and MVC Model.
* Provide a more fluid user experience akin to a desktop application.
* All team members want to study a new language.

# ARCHITECTURAL REPRESENTATION

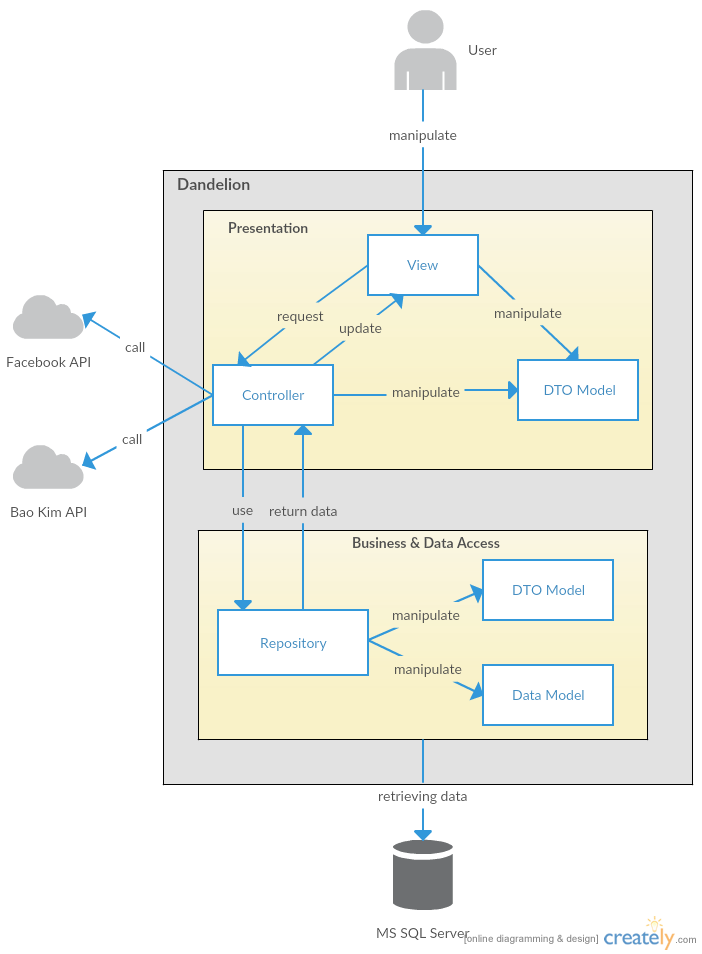


Figure 3‑1: System Overview

We follow MVC architecture to implement the DDL Project. MVC offers architectural benefits over standard Jquery and AngularJS — it helps us write better-organized and therefore more maintainable code.

* **DTO Model (data transfer object model)** is an object that defines how the data will be sent over the network to remove circular references from data model, Hide particular properties that clients are not supposed to view, omit some properties in order to reduce payload size, flatten object graphs that contain nested objects, to make them more convenient for clients, avoid “over-posting” vulnerabilities and decouple your service layer from your database layer.
* **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 DDL, the view is made with .cshtml file including css, AngularJS and 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.
* **Data 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.
* **Repository** is intermediate layer which used to separate the controller and the data access layer (database context). It queries the data source for data, maps it to DTO models, processes data and returns data to controller.

# 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:**
  + 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

# 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.
  + Repository:
    - Create queries to DB.
    - Process data.
    - Return to controller.

## Architecturally Significant Design Packages

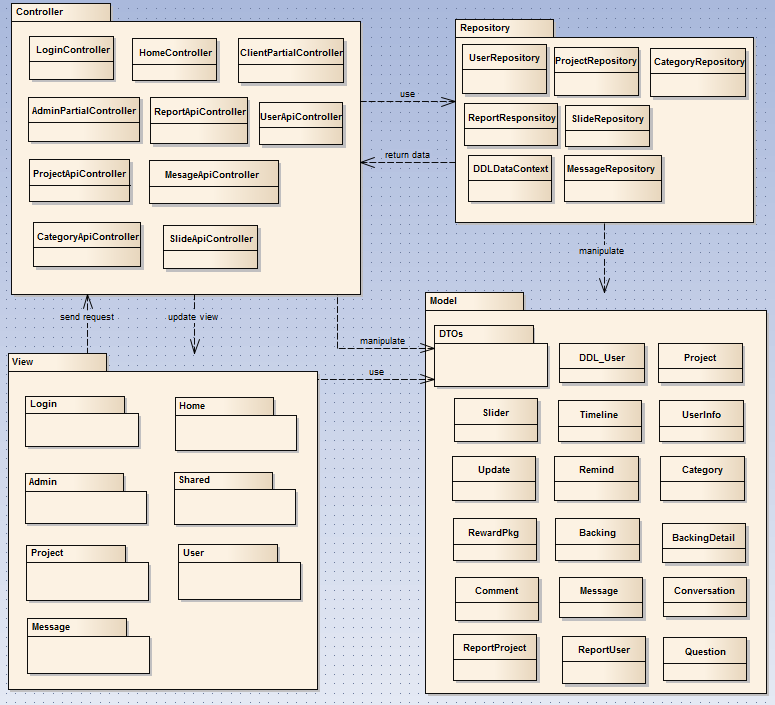


Figure 6‑1: Package Diagram

* **Model:**

|  |  |  |
| --- | --- | --- |
| No | Model class | Role |
|  | DDL\_User | Description entity of User in database |
|  | Project | Description entity of Project in database |
|  | Conversation | Description entity of Conversation in database |
|  | Update | Description entity of Update in database |
|  | Remind | Description entity of Remind in database |
|  | Category | Description entity of Category in database |
|  | Reward | Description entity of Reward in database |
|  | Backing | Description entity of Backing in database |
|  | BackingDetail | Description entity of BackingDetail in database |
|  | Comment | Description entity of Comment in database |
|  | QuestionAnswer | Description entity of QuestionAnswer in database |
|  | Message | Description entity of Message in database |
|  | ReportProject | Description entity of ReportProject in database |
|  | ReportUser | Description entity of ReportUser in database |
|  | Slider | Description entity of Slider in database |
|  | Timeline | Description entity of Timeline in database |
|  | Userinfo | Description entity of Userinfo in database |

Table 6‑1: Model list

* **Controller:**

|  |  |  |
| --- | --- | --- |
| No | Controller class | Role |
|  | LoginController | * Receive request login, logout from client. * Call method login, logout. * Respond login view and login, logout status. |
|  | HomeController | * Receive request to home page from client. * Handle request from client and call methods to get all data of home page. * Respond data back to Home View. |
|  | ClientParitialController | * Receive request get client partial views of from client. * Generate partial views and respond to client. |
|  | AdminParitialController | * Receive request get admin partial views of from client. * Generate partial views and respond to client. |
|  | UserApiController | * Receive requests (get, post, push, delete) about User from client. * Respond status, message and json data to client. |
|  | ProjectApiController | * Receive requests (get, post, push, delete) about Project from client. * Respond status, message and json data to client. |
|  | MessgaeApiController | * Receive requests (get, post, push, delete) about Message from client. * Respond status, message and json data to client. |
|  | SliderApiController | * Receive requests (get, post, push, delete) about Slider from client. * Respond status, message and json data to client. |
|  | ReportApiController | * Receive requests (get, post, push, delete) about Report User or Project from client. * Respond status, message and json data to client. |
|  | CategoryApiController | * Receive requests (get, post, push, delete) about Category from client. * Respond status, message and json data to client. |

Table 6‑2: Controller list

* **Repository:**

|  |  |  |
| --- | --- | --- |
| No | Repository class | Role |
|  | DDLDataContext | * Connect to DB, create queries to select, update, deletes data. |
|  | UserRepository | * Process user data and return to controller. |
|  | SlideRepository | * Process slide data and return to controller. |
|  | ReportResponsitoy | * Process report data and return to controller. |
|  | ProjectRepository | * Process project data and return to controller. |
|  | MessageRepository | * Process message data and return to controller. |
|  | CategoryRepository | * Process category data and return to controller. |

* **DTOs:**

Include many data transfer object classes.

* **View:**

Include many .cshtml file

# Process view

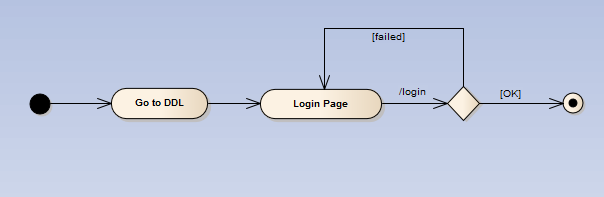


Figure 7‑1: Login activity diagram

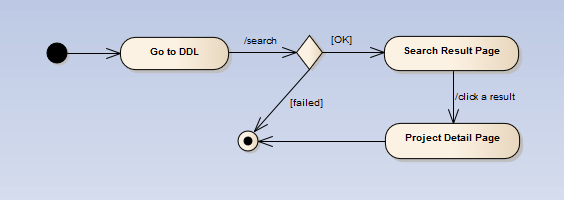


Figure 7‑2: Search Project activity digram

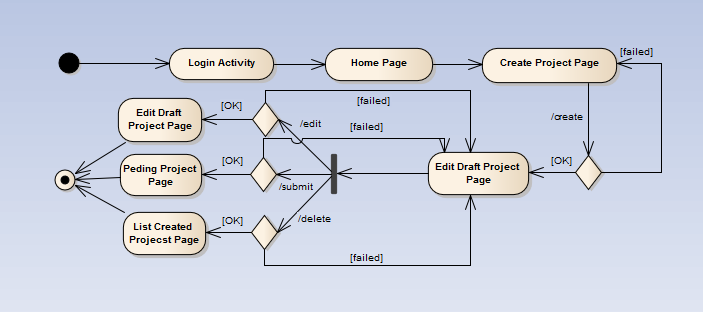


Figure 7‑3: Create Project activity diagram

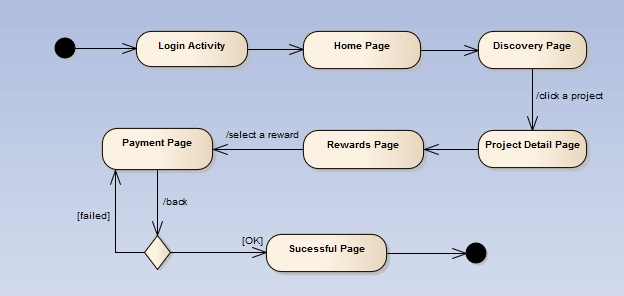


Figure 7‑4: Back Project activity diagram

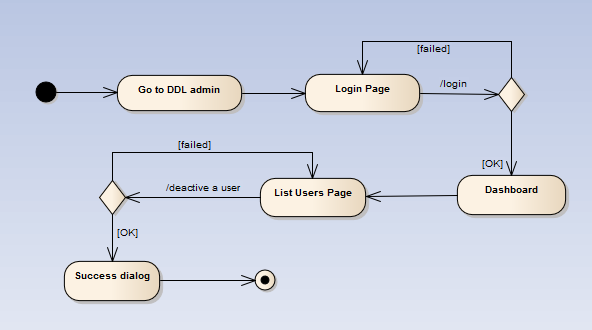


Figure 7‑5: De-active User activity diagram

# Deployment View

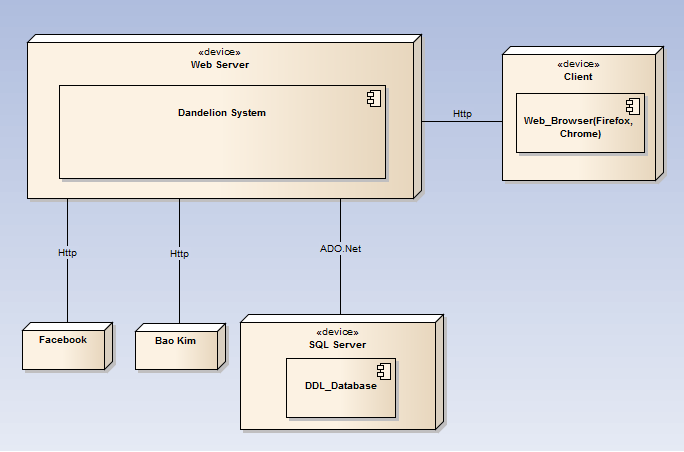


Figure 8‑1: Deployment Diagram

|  |  |  |
| --- | --- | --- |
| **No** | **Name** | **Description** |
| 1 | SQL Server | SQL Server use to store system’s data.  Using SQL Server 2010 or higher. |
| 2 | Facebook | Facebook Api server |
| 3 | Bao Kim | Bao Kim Api server |
| 4 | Client | Client is web browser to use system. Firefox 30, Chrome 40 or higher. |
| 5 | Web Server | Web server is hosted by Azure Web Service. |

**Table 8‑1:** Deployment Diagram Description

# Quality

Reference to: DDL\_Software requirement specification\_v1.0\_EN.docx