AALTO UNIVERSITY SCHOOL OF SCIENCE AND TECHNOLOGY

Faculty of Electronics, Communications and Automation

Hengzhi Liu

Re-engineering an Online User Interface over Web Application Framework and Ajax

Master's Thesis submitted for the degree of Master of Science in Technology.

Espoo, 17 May, 2010

Supervisor: Docent Timo O. Korhonen

Instructor: Docent Timo O. Korhonen

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY **Abstract of the Master's Thesis**

Author: Hengzhi Liu

Name of the thesis: Re-engineering an Online User Interface over Web Application Framework and Ajax

Date: **1**7 May, 2010

Number of pages: 13 + 76

Faculty: Faculty of Electronics, Communications and Automation
Professorship: S-72 Communications

Supervisor: Docent Timo Korhonen
Instructor: Docent Timo Korhonen

Abstract text:

Web development has changed dramatically since the introduction of the web application framework to this world. It lowers the barriers of entry to web programming, makes it possible for a single developer to construct powerful web applications in a relatively short time. However, in order to enjoy all benefits brought by the web application framework, traditional web development has to be adjusted at every phrase of the software engineering process. Furthermore, with the increasing popularity of Web 2.0 concepts and techniques, especially Ajax, questions like, for instance, whether a web application framework and Ajax may work together seamlessly or not, how the conflicts between them can be solved, and how to get the full benefits from their combination, all have to be addressed by web developers.

This research tried to answer these questions in a practical way, by re-engineering a static web site with the help of a web application framework and Ajax techniques. Concepts and methods of software engineering and project management were used and evaluated during the research. The reconstructed web site managed to be more user-centric which was similar as some well-known social web applications. To encourage site user interactivity with other users and generating more content, the site user interface (UI) was re-designed from scratch and implemented with Ajax being applied.

The re-engineered site with the new UI demonstrated the effectiveness of the web application framework and Ajax techniques. The new UI responds fast to user requests without the need for refreshing the page. An incentive mechanism works fine behind the compact UI to stimulate users to contribute more content. The new site has also a clearer code structure which, in turn, makes site maintenance much easier.

The research answered also the questions concerning the combination of the web application framework and Ajax techniques. A few conflicts, however, were discovered when trying to combine them together in practice. Some were caused by a defective API hidden in the antiquated module of the selected web application framework, some by the limitations of Ajax functions in the selected JavaScript library for compatibility across discordant web browsers. These problems that emerged were, however, solved quickly by modification of a few pieces of code depending on the specific situations. But in general, the combination of the web application framework and Ajax techniques performed quite well.

The experimental part was planned to support implementation with a moderate programming load such that the intended functionalities would still be fully supported. The re-engineered web site lacks support of some advanced Web 2.0 features such as own content tagging, RSS feeding, and IM integration, but the basic functionalities such as content sharing, content grading, and user authorization are supported. The actual service concept design was not within the scope of this study, this being an important reason why the re-engineered site has not yet gain increased popularity.

Keywords: Web application framework, Ajax, JavaScript library, User interface, Software engineering, Google App Engine, REST, JSON, Web 2.0

Preface

The appearance of the *web application framework* changes the way in which web sites are designed and implemented. With the help of such a powerful tool, web application development has been released from heavy and boring programming and testing into rapid and agile flashes of inspiration which has brought the fun back to code writing for web programmers.

I was interested in implementation of web sites about eleven years ago when I got to know the Internet and found out that it is possible to create and upload a web site to one of some free homepage hosting services. After that, I kept my eye on rapidly changing technologies of web development, and all related standards, concepts, languages, and tools. So, when Ruby on Rails finally emerged to the public in 2006, I was shocked just like most other web programmers in the world. The innovation of the web development environment since then has changed the web industry from many aspects.

I moved to Finland in 2004. After these years of living far from my home country, I always hope to do something to help my compatriots to communicate, because I believe that information sharing would increase the knowledge of each, and make life much easier. That is the motivation of this study and its experimental work.

I need to thank the existence of CSSA-Espoo Ry and Google at first. Without CSSA-Espoo Ry, I would not have such motivation to do this study. Without Google and its App engine platform, I would not be able to finish coding and deployment of such coding work.

I want to thank my supervisor Timo Korhonen, who helped me a lot for turning my code into this study paper, and Martin William, who helped me with the language checking of the final thesis manuscript.

I would like to thank my parents who are living in my home city in China now. Without their support, both financially and spiritually, I would not have managed to complete this study.

Special thanks to my wife, Shuang Gu, for always believing in and encouraging me, and for having the patience to take care of our new born daughter when I had to steal time to work on completing this Master’s thesis.

And also, to my lovely daughter, Yuxin.

Espoo, May 17, 2010

Hengzhi Liu

Table of Contents

List of Figures

List of Abbreviations

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
CRUD	Create, Read, Update, and Delete
DOM	Document Object Model
DRY	Don’t Repeat Yourself
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Services
IM	Instant Messaging
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAMP	Linux, Apache, MySQL, and PHP
MTV	Model-Template-View
MVC	Model-View-Controller
ORM	Object-Relational Mapping
OS	Operating System
REST	Representational State Transfer
RIA	Rich Internet Application
RSS	Really Simple Syndication / Rich Site Summary
SDK	Software Development Kit
SQL	Structured Query Language

SWOT	Strengths, Weaknesses, Opportunities, and Threats
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
YAML	YAML Ain’t Markup Language
XML	Extensible Markup Language

Key Concepts

Ajax	A group of interrelated web development techniques used on the client-side to create interactive web applications. [See Section 2.2]
Blog	A web communication tool in a form of a web site for publishing text together with images and links. Created with blogging software, for instance, Blogger.
Cloud computing	Internet-based development and use of computer technology. Instead of traditional client-server structure, cloud computing abstracts the details of control over technology infrastructure from users and sets up a new supplement, consumption, and delivery model based on the Internet. It typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet. [See Section 2.4.7]
Convention over Configuration	A software design philosophy and technique with a strategy of defaults over explicit configuration. [See Section 2.1.3]
Database management system	One of the standard web infrastructures on server side which controls the creation, maintenance, and the use of a database. This includes, for instance, MySQL, Oracle Databse, SQLite, and so on.
Don't Repeat Yourself	A principle of software development to reduce repetition of source code and all related information of software engineering. [See Section 2.1.2]
Facebook	The most popular social networking application. It enables e.g. creating your own profile, contacting friends, and sharing photos.
Google App Engine	A development environment for web applications and a hosting service provided by Google. It consists of a Python-based web framework and an un-relational data base, which is called

	Bigtable. For personal usage, it is free of charge to register and use with CPU and storage limitations. [See Section 2.4]
JavaScript library	A library of pre-written JavaScript controls which provides convenience to development of JavaScript-based applications, especially for Ajax and other web-centric technologies. [See Section 2.2]
jQuery	A fast and concise JavaScript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript. [See Section 2.2.2]
Language runtime	One of the standard web infrastructures on server side which explains and executes computer language scripts or binary executables. This includes, for instance, PHP's runtime, Python's, Ruby's, JVM (Java Virtual Machine), and so on.
Model-View-Controller	A software architectural pattern widely accepted in structural design of web application frameworks. [See Section 2.1.1]
Object-relational mapping	A programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. [See Section 2.3.4]
Operating System	One of the standard web infrastructures on server side which provides an interface between the hardware and other software. This includes for instance Unix, Linux, FreeBSD, Microsoft Windows, and so on.
Python	A general-purpose high-level programming language. Its design philosophy emphasizes code readability. Its use of indentation for block delimiters is unusual among popular programming languages. (Python Software Foundation, 1990-2010)
Representational State Transfer	A style of software architecture for web applications and other distributed hypermedia systems. A web application framework is

	RESTful by providing convenience to the application development following REST style. [See Section 2.1.4]
Rich internet application	Work more like desktop applications than traditional web applications, e.g. webmail. RIA's are enabled by rich technologies such as Ajax which lead into faster response times and more interactive GUI (graphical user interface) elements, e.g. always visible floating menus, and controls, e.g. the dragging of GUI elements.
RSS feed	A data syndication format RSS (Really Simple Syndication / Rich Site Summary) is used for collecting web content feeds, e.g. from blogs and news services. Feed reader programs such as Google Reader check lists of syndication feeds and display all the updated content on one page.
Social networking	Online communities which are formed with social networking applications, which offer ways of finding people with similar interests, communicating with others, and expressing oneself. Facebook is the most popular application.
Syndication	Presenting data from various web pages on a single page. The most common syndication format is RSS.
Tag	Tag is a descriptive keyword that is attached to a digital object such as a web page, an image, a blog entry, or a video. Tags are facilitating search functions and the arranging of digital objects.
Usability	Is a part of user experience that measures how easy and pleasurable a product or service is to use.
User experience	Positive user experience is achieved when service's features and design meet user's needs and expectations in a usable and pleasurable way.
Web 1.0	A concept in order to present traditional web technologies being used before Web 2.0 concept appearing. The web technologies of Web 1.0 include for instance HTML, XHTML and CSS.

Web 2.0	A concept for a collection of new technologies, applications, concepts, ideas, business strategies, and social trends in the web. Web 2.0 is more dynamic and more interactive than Web 1.0.
webapp	A simple web application framework provided by Google App Engine by default. [See Section 2.4]
Web application framework	A software framework that is designed to support the development of dynamic websites, web applications and web services. These include, for instance, Rails, Django, Spring, etc. [See Section 2.1]
Web server	One of the standard web infrastructures on server side which delivers contents using kinds of data transferring protocols over network. This includes for instance Apache, Microsoft IIS, Lighttpd, and so on.
Wiki	A web-based tool for creating, modifying, and deleting web content collaboratively. Web-based encyclopedia Wikipedia is the best known wiki.

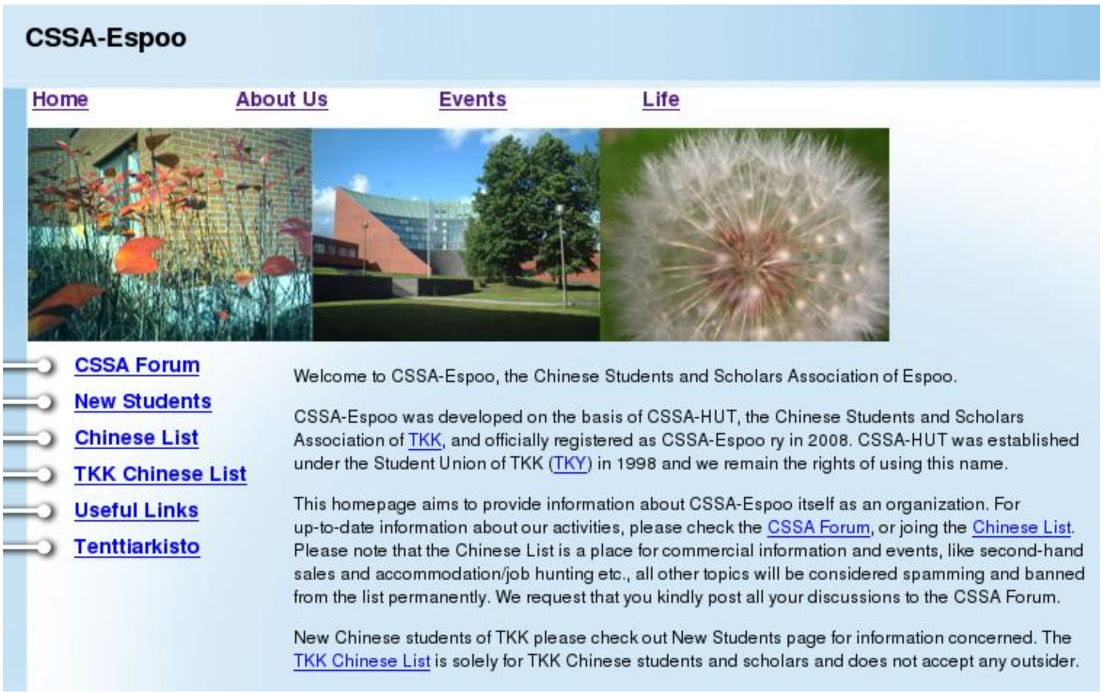
1. Introduction

Based on powerful dynamic-typing programming languages, the *web application framework* is changing the process of creating web sites. Using the combination of the web application framework with *Web 2.0* techniques, such as *Ajax*, helps web programmers write less code but achieve more than they managed to do, enhancing the *user experience* of websites at the same time.

1.1. Purpose of the Study

The purpose of this study is to find the changes the web application framework and Ajax techniques bring to software developers of web sites and their end-user experiences. In order to do the research in a practical way, a *web 1.0* based web site, CSSA-Espoo [See Figure 1.1:], was selected for the experimental part of the study. The experimental part re-engineered the user interface of the web site by combining the use of a web application framework and Ajax techniques. The study aims at finding out what kinds of benefits the web application framework and Ajax may offer to web developers, and how these benefits may consequently influence the user interface design and user experience of web sites.

The scope of the study includes discussion about some architecture styles and design principle which are implicated in the design of a web application framework, such as *MVC* [See Section 2.1.1], *DRY* [See Section 2.1.2], *Convention over configuration* [See Section 2.1.3], and *REST* [see Section 2.1.4]. When working on the experimental part using *Python*, *Google App Engine* was selected to be used as the hosting platform of the re-engineered web site, and thus *webapp*, a development framework provided by App Engine by default, is used to represent a web application framework. The reason for using Google App Engine was discussed also. In order to ease the coding complexity of Ajax-based web pages, a JavaScript library, *jQuery* [See Section 2.2.2 and Section 3.3.2], was introduced into the experimental part of the study. The conjunction and conflicts of the web application framework and Ajax techniques which were met in experimental part of the study are another focus of the discussion.



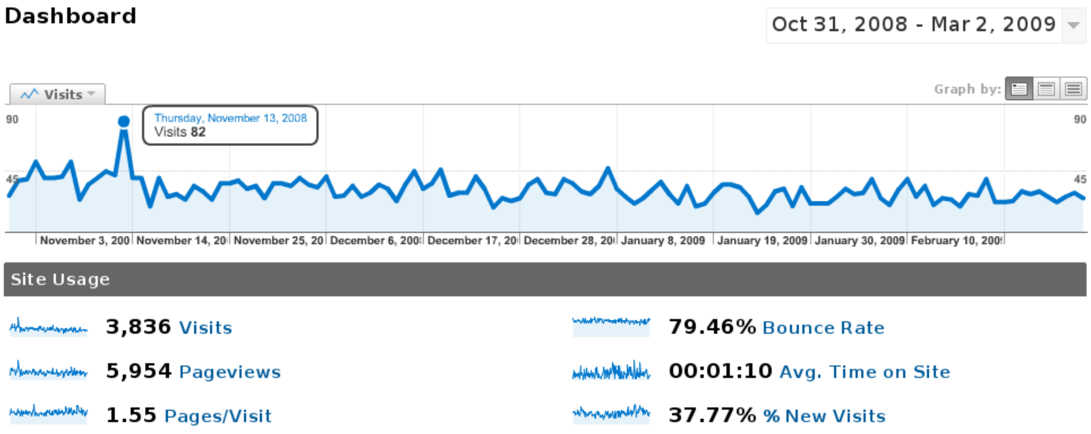
Welcome to CSSA-Espoo, the Chinese Students and Scholars Association of Espoo.

CSSA-Espoo was developed on the basis of CSSA-HUT, the Chinese Students and Scholars Association of TKK, and officially registered as CSSA-Espoo ry in 2008. CSSA-HUT was established under the Student Union of TKK (TKY) in 1998 and we remain the rights of using this name.

This homepage aims to provide information about CSSA-Espoo itself as an organization. For up-to-date information about our activities, please check the CSSA Forum, or joing the Chinese List. Please note that the Chinese List is a place for commercial information and events, like second-hand sales and accommodation/job hunting etc., all other topics will be considered spamming and banned from the list permanently. We request that you kindly post all your discussions to the CSSA Forum.

New Chinese students of TKK please check out New Students page for information concerned. The TKK Chinese List is solely for TKK Chinese students and scholars and does not accept any outsider.

Figure 1.1: Web site of CSSA-Espoo Ry (CSSA-Espoo, 2008) before the study began. It is a typical Web 1.0 site written mainly by HTML and CSS.



3,836 Visits

5,954 Pageviews

1.55 Pages/Visit

79.46% Bounce Rate

00:01:10 Avg. Time on Site

37.77% % New Visits

Figure 1.2: Daily site visits recorded by Google Analytics (Google, 2006). From the curve, the daily count of site visits averaged around 30-50 on most days.

The web site selected as the basis for the experimental part of the study, CSSA-Espoo, was created in 2004 and maintained lately by CSSA-Espoo Ry continuously with the purpose of information collecting and sharing, and activity organizing among Chinese students of Aalto University School of Science and Technology and other residents living in the Helsinki metropolitan area of Finland. Before the study began, the daily count of user visits of the web site remained quite low. [See Figure 1.2] Although maintainers of the site added an electronic mailing list (CSSA-Espoo, 2004) and an online discussion forum (CSSA-Espoo, 2008) as supplements lately in order to improve the convenience of information sharing, the situation did not improve noticeably.

Before the study began, all contents of the web site consist of only static documents and web pages created with Web 1.0 technologies [See Figure 1.1:]. By using traditional mark-up languages, such as HTML and CSS, which are popular and easy to learn and use, site creators and maintainers shortened the startup process for achieving necessary development skills, and kept the simplicity of document structure also which helped them for later textual modification. On the other hand, they abandoned possibilities to create interactive and responsive web interface for users at the same time. With the limitation of web 1.0 approaches, all information showing on the web site was static and unidirectional to each single user who would soon lose interest in revisiting the web site.

1.2. Research Questions

The research questions of this study concern web site design and programming changes that the web application framework and Ajax techniques bring to web development. The research questions are presented in order of importance.

1st Question: What are the benefits and drawbacks brought to web site developers in using the web application framework and Ajax?

This study concentrates on the new possibilities and challenges of web site implementation when using the web application framework and Ajax techniques in addition to traditional web 1.0 approaches. The main objectives are:

- What are the benefits in using the web application framework and Ajax?
- What adjustment is necessary in using the web application framework and Ajax?

Determining the benefits and conflicts brought by the web application framework and Ajax techniques helps web site developers to create more *user-centralized* web sites, and get more knowledge on how to bind these two powerful tools together without confusion.

2nd Question: How to design a web user interface to encourage users to generate additional content on the site?

The experimental part of the study focuses on design and implementation of a user interface which reflects a service concept design, and is expected to increase user experience and encourage user to generate new content and share it with the public through the site. Ajax techniques are used to achieve this goal in implementation. Therefore, this thesis focuses also on the following topics:

- What kinds of HTML elements of Ajax user interface are being used in a website design to attract users to visit the website frequently and to encourage them to interact with other users?
    - How does the web application framework and Ajax techniques influence the design of web user interface?
- Why a certain interface design is being used instead of other possible solutions?
    - What kinds of limitations and differences between the selected web site and other well-known social networking websites exist when considering user groups?

Determining the changes brought by new tools and techniques to user interface design helps web designers to find out better solutions for designing of user-friendly web sites and services.

3rd Question: What kinds of technologies are beneficial to accelerate web site implementation?

The architecture styles, programming guidelines and ready-to-use functions which are contained generally in the web application framework facilitate the implementation of web sites more rapidly than usual. Those general features provided by the web application framework to developers are also discussed in this study. The purpose is to find out:

- What kind of features that a web application framework has that will accelerate website implementation?
- How to bind a web application framework to Ajax techniques together?
- How to adjust the features so that these tools have for a practical implementation?

Determining the modern web development standards embedded naturally in the web application framework helps web site developers to understand web development tools better and to work with them more easily.

1.3. Structure of the Thesis

The structure of this Master's thesis is the following: the literature overview concentrates on determining the concepts of the web application framework [See Section 2.1] and Ajax techniques [See Section 2.2], as well as the characteristics and features provided by the practical development environment and hosting platform, Google App Engine [See Section 2.4]. The research chapter describes the research methods using in the experimental part of the study, and lists issues found, as well as considerations and possible solutions which emerged during the research. [See Section 3] The results of the research and code structure of the experimental part are located in the results chapter. [See Section 4] In the discussion chapter, the predetermined research questions are answered with further discussions around the UI design and recognizing of user group. [See Section 5] In the last chapter of the thesis, the conclusions of the research, defects and possible directions of future research are discussed. [See Section 6] The final user interface of the web site and source code are presented in the appendices.

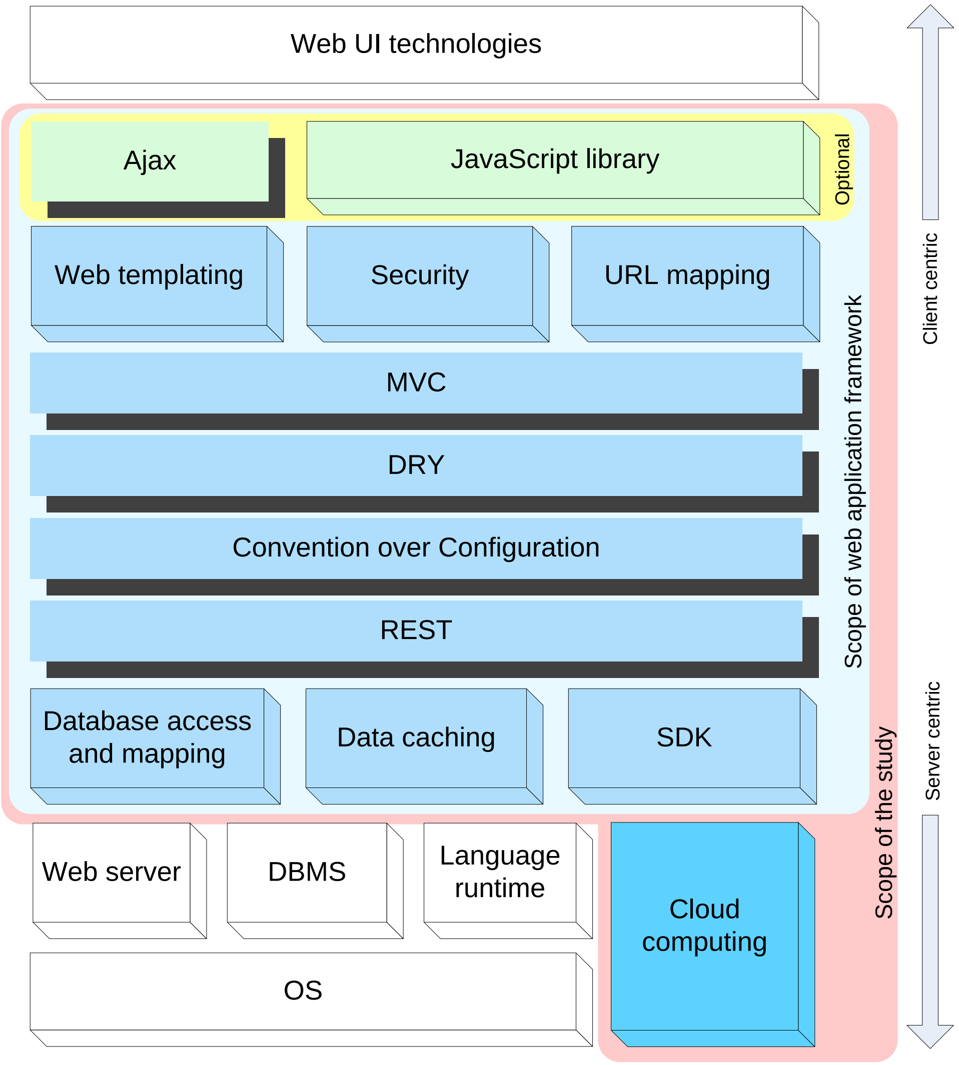


Figure 1.3: Scope of web application framework and the study (DocForge, 2010)

Explanation of the terms in the figure, from top to bottom and left to right:

Web UI technologies	Technologies for creating web user interface. These include for instance HTML, CSS, JavaScript, Flash, and so on.
Ajax	(Asynchronous JavaScript and XML) A group of interrelated web development techniques used on the client-side to create interactive web applications. [See Section 2.2]
JavaScript library	A library of pre-written JavaScript controls which provides convenience to development of JavaScript-based applications, especially for Ajax and other web-centric technologies. [See Section 2.2]

Web templating	A web application framework feature which used to separate content from presentation in web design, and for mass production of web documents. [See Section 2.3.1]
Security	A web application framework feature which provides authentication and authorization for a web server to identify the users of the application and restrict access to functions based on defined criteria. [See Section 2.3.2]
URL mapping	A feature of the a web application framework which allows more friendly URLs to be used to increase structure simplicity of web site and indexing accuracy of search engines. [See Section 2.3.3]
MVC	(Model-View-Controller) A software architectural pattern widely accepted in the structural design of a web application framework. [See Section 2.1.1]
DRY	(Don’t Repeat Yourself) A principle of software development to reduce repetition of source code and all related information of software engineering. [See Section 2.1.2]
Convention over Configuration	A software design philosophy and technique with a strategy of defaults over explicit configuration. [See Section 2.1.3]
REST	(Representational State Transfer) A style of software architecture for web applications and other distributed hypermedia systems. A web application framework is RESTful by providing convenience to the application development following REST style. [See Section 2.1.4]
Database access and mapping	A web application framework feature which provides a unified API to a database backend to enable web applications to work with a variety of databases without code changes, and

	allowing programmers to work with higher level concepts. [See Section 2.3.4]
Data caching	A web application framework feature which focuses on the caching of web documents and queried data in order to reduce bandwidth usage, server load, and perceived lag. [See Section 2.3.5]
SDK	(Software development toolkit) A set of development tools provided by a web application framework which commonly consist of, for instance, a light-weight web server, testing modules, debugging tools, deployment assistant, and so on. [See Section 2.3.6]
Web server	One of the standard web infrastructures on the server side which delivers contents using kinds of data transferring protocols over a network. This includes, for instance, Apache, Microsoft IIS, Lighttpd, and so on.
DBMS	(Database management system) One of the standard web infrastructures on the server side which controls the creation, maintenance, and the use of a database. This includes, for instance, MySQL, Oracle Database, SQLite, and so on.
Language runtime	One of the standard web infrastructures on the server side which explains and executes computer language scripts or binary executables. This includes, for example, PHP’s runtime, Python’s, Ruby’s, JVM (Java Virtual Machine), and so on.
OS	(Operating System) One of the standard web infrastructures on the server side which provides an interface between the hardware and other software. This includes, for instance, Unix, Linux, FreeBSD, Microsoft Windows, and so on.
Cloud computing	Internet-based development and use of computer technology. Instead of traditional client-server structure, cloud computing abstracts the details of control over technology infrastructure from users and sets up a new supplement, consumption, and

delivery model based on the Internet. It typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet. [See Section 2.4.7]

2. Literature Overview

In this chapter the concepts of the web application framework, Ajax techniques, and JavaScript library are introduced. The general features provided by a web application framework are also introduced in detail. In comparison with the web application framework, the hosting platform of the experimental part in this study, Google App Engine, and its important features which had been used in consideration for the coding practice are also discussed.

2.1. Definition of Web Application Framework

A web application framework is not only a web-based software development framework but also a reusable, skeletal, semi-complete modular platform. It is designed to support the development of dynamic websites, web applications and web services. Additionally, it is commonly allowed to be specialized to serve as a custom web server via HTTP(S) protocol. (Shan & Hua, 2006) The framework aims to relieve the overhead associated with common activities performed in web development. There are numerous web application frameworks available which associate different programming languages for each. These include, for instance, Rails (Rails, 2004-2010) framework which is built on top of Ruby language, Django (Django Software Foundation, 2005-2010) built on Python language, and Spring (SpringSource, 2004-2010) on Java.

Most web application frameworks were built-in modules to provide a range of services by following industrial standard architectural patterns and coding guide principles which had been defined and broadly used in software engineering. These software architecture patterns and coding principles include not only, for instance, MVC (Model-View-Controller) to gain the benefits from the isolation of business logic from the user interface and easier code maintenance, but also DRY (Don't Repeat Yourself), Convention over Configuration, and RESTful (Representational State Transfer regulated) as well. A web application framework will be in-complete when lacking of any of these characteristics. The following sections explain these concepts in detail.

2.1.1. Model-View-Controller

Model-View-Controller (MVC) is a software architectural pattern widely accepted in corporation software development. It divides the software system into three different layers which are in charge of interface, control logic, and data access separately. The pattern isolates the application logic for users from interaction and presentation with users. By using MVC architecture in software development, independence of development, testing and maintenance for each part are achievable. (Selfa, Carrillo, & Del Rocio Boone, 2006)

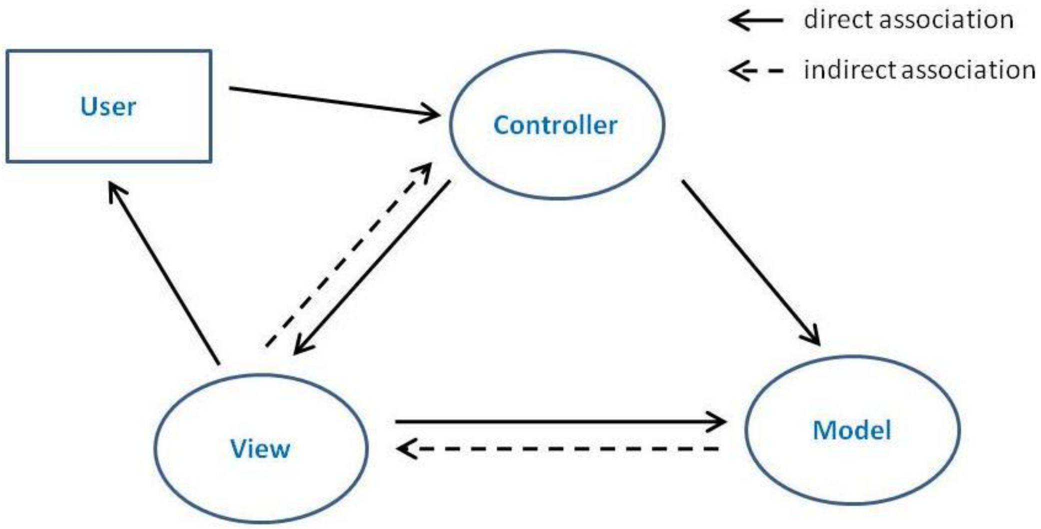


Figure 2.1: Model-View-Controller architecture (Selfa, Carrillo, & Del Rocio Boone, 2006)

Model

The model is the domain specific representation of the data upon which the application operates. Domain logic adds meaning to raw data. When a model changes its state, it notifies its associated views so they can refresh.

In a web application framework, the model of the structure usually means a consistent API which is able to access multiple data storage systems. Some aspirant frameworks implemented dynamic database-access API in the light of the ORM (*Object-relational mapping*) technique to store and retrieve data objects automatically and simply. The model is also the place to arrange caching above the database layer to achieve performance enhancement. Data integrity checks, such as validating relationships or confirming required fields, should also be dealt with in this layer.

View

The view is where the data, which was processed by the controller, gets output in the format requested by the user. The view renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. There should be no data processing in the view.

In web applications, a page of view typically consists of HTML and CSS, but can also be XML, JavaScript, JSON, or any other data formats.

Controller

The controller is the logic between the model and the view. Controllers contain methods which directly correlate to actions performed by the user. The controller's purpose is to respond to the action requested by the user, take any parameters the user has set, process the data, interact with the model, and then pass the requested data off to the view in final form.

In a web application framework, controller methods are typically short, small chunks of code that accomplish very specific tasks. A controller method receives a user request and decides what to do with it, handing it over to the model which contains business rules and requested data.

2.1.2. Do not Repeat Yourself

Do not repeat yourself (DRY) is a principle of software development to reduce repetition of source code and all related information of software engineering. The purpose of the DRY principle is to ensure every piece of knowledge in the development having a single, unambiguous, authoritative representation within a system. (Hunt, 2010) DRY can be applied quite widely to include database schemas, test plans, build system and even documentation. (Thomas, 2003) Modification of any single element of a system should not affect other elements when DRY is successfully applied.

By extending the requirement of avoiding code duplication to be more general, DRY is about avoiding multiple, and possibly diverging, ways to express every piece of knowledge, which includes, for example, logic, database schemas, and constants. Coding under the DRY principle helps software developers to understand the structure of large software systems faster and are thus easier to maintain.

2.1.3. Convention over Configuration

Convention over Configuration is a software design philosophy and technique with a strategy of defaults over explicit configuration. It seeks to apply defaults that can be implied from the code structure instead of requiring explicit code. The idea is to simplify development by decreasing the number of decisions that developers need to make during designing software behaviors for user interaction, thus allowing them to specify only the unconventional parts of the application and architecture. (Miller, 2009) Convention over configuration helps software development to gain simplicity without losing necessary flexibility. When the convention implemented by the tool matches one's desired behavior, the benefits of convention will be enjoyed without having to do the configuration work.

The approach of convention over configuration is used to make routine the design of software frameworks, which may poorly introduce multiple configuration files consisting of lots of settings. A large number of configuration files with too many parameters are often an indicator of an unnecessarily complex application design.

2.1.4. Representational State Transfer

Representational State Transfer (REST) is a style of software architecture for web applications and other distributed hypermedia systems. After the design of the HTTP/1.0 protocol, the REST architectural style was then developed in parallel with the HTTP/1.1 protocol. The World Wide Web is so far the largest known implementation of a system conforming to the REST architectural style. REST exemplifies how the web's architecture emerged by characterizing and constraining the macro-interactions of the four components of the web, namely originating servers, gateways, proxies and clients, without imposing limitations on the individual participants. As such, REST essentially governs the proper behavior of participants. (Fielding & Taylor, 2002)

RESTful architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource. REST has been applied to describe the desired web architecture, helped to identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the web successful.

A web framework application is considered as to be RESTful when it provides convenient tools for web developers to implement web services which follow the principle of REST. Some web application frameworks do not provide specific tools to help developers for implementing RESTful web services, but are also good to be the background of RESTful web services based on loose design and modular structure.

RESTful Web services are characterized by the uniform interface of their implementation. Typically a web user may interact with a server through a set of HTTP methods: GET; POST, PUT and DELETE. They act on resources accessing or sending representations according to the CRUD (Create, read, update, and delete) pattern. (Mazzetti, Nativi, & Bigagli, 2008)

2.2. Definition of Ajax and JavaScript Library

2.2.1. Ajax

Ajax (Asynchronous JavaScript and XML) is a group of interrelated web development techniques used to create interactive web applications on the client side. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. (Garrett, 2005)

Ajax is not a solo technology but a group of technologies instead. Ajax uses a combination of HTML and CSS to mark up and style information. The DOM (Document Object Model) is accessed with JavaScript to dynamically display and to allow the user to interact with the information presented. The use of Ajax techniques has led to an increase in interactive or dynamic interfaces on web pages. Despite the name of Ajax, the use of XML is not actually required for data interchange. Instead, JSON (*JavaScript Object Notation*) (JSON, 1999) is often used as an alternative format for data interchange, although other formats or plain text can also be used.

In many cases, related pages on a website consist of much content that is common between them. Using traditional methods that content would have to be reloaded on every request. By using Ajax, however, a web application can request only the content that needs to be updated, thus drastically reducing bandwidth usage and page load time. Additionally, the use of asynchronous requests allows the client's web browser UI to be more interactive and to respond quickly to user inputs, and sections of pages can also be reloaded individually. Users may perceive the application to be faster or more responsive even if the application has not changed on the server side. Furthermore, using Ajax reduces connections from the client to the server, since scripts and style sheets only have to be requested once.

Despite all of these benefits, in practical situations, Ajax interfaces are often harder to develop due to their dynamic nature when compared to static pages. To reduce the barriers to Ajax development for web developers, most modern *JavaScript Libraries* provide an *Ajax framework* and corresponding utilities.

2.2.2. JavaScript library

The JavaScript library is a set of pre-written JavaScript controls which provides convenience to the development of JavaScript-based applications, especially for Ajax and other web-centric technologies. It was developed to meet the expended demands for JavaScript with the rise of Web 2.0 concepts and their implementations.

By combining most of common operations invoked through JavaScript functions and common Ajax invoking functions, the JavaScript library allows developers to concentrate more upon the web user interface behaviors, without being disturbed by the compatibility issues caused by the client-side browser difference. For instance, the jQuery JavaScript library (jQuery, 2006) supports browsers which consist of Mozilla's Firefox version 2.0 and above, Microsoft's Internet Explorer version 6 and above, Apple's Safari version 3 and above, Opera's Opera version 9 and above, and Google's Chrome version 1 and above. (jQuery, 2010)

The JavaScript library is even chosen to be embedded into some web application frameworks as a component of them to convenient web developers further. For instance, one natively supported module of the web application framework Rails (Rails, 2004-2010) is Prototype (Prototype, 2006). Some other web application frameworks do not embed any JavaScript library, but are easy to work with one or several JavaScript libraries together depending on their loose design and modular structure. (Bennett, 2006)

2.3. Features of Web Application Framework

In client-server structure, a web application framework plays the role of the middleware [See Figure 1.3**:** Scope of web application framework and the study] receiving all requests from the client, querying data from database, and responding back to the client with the generated results, which are commonly rendered web pages. Built with the design patterns, development principles and architectural styles [see Section 2.1] by nature, most web application frameworks provide various features to simplify the work for web developers. These features aim to become helpers on a number of matters from rendering web pages, user authorization, fetching data from a database, to reducing the time consumed by a single query. (DocForge, 2010)

Although all features one web application framework provides are not usually implemented in another one, there are some features which are consistently applicable in most web application frameworks, which are used to identify if the design of a web application framework is mature or not. (Google, 2008) These general features include, for instance, web templating, security, URL mapping, data access and mapping, data caching, and easy-to-use SDK for developers, which are briefly described one by one in the sections below.

2.3.1. Web templating

In modern Web 2.0 sites, by using a web application framework, static web pages which are commonly found in Web 1.0 sites are usually replaced by dynamic web pages. A dynamic web page usually consists of a static HTML part and a dynamic part which generates HTML to a specific client and a context from modifiable code. The code generates HTML which is based commonly on variables in templates. In a template, variables from the programming language can be inserted without using language code, thereby removing the requirement for programming knowledge to make updates to the pages in a web site.

Syntax is made available to distinguish between HTML and variables. Many template engines support limited logic tags, for instance, IF and FOREACH. These are to be used only for decisions that need to be made for the presentation layer in order to keep a clean separation from the business logic layer, which is the Model part of the MVC pattern.

2.3.2. Security

User-centered design of modern Web 2.0 sites demands higher security requirements for web developers. Management of user accounts of a web site includes creating, saving and confirming user identities which are important for a server to recognize each user and provide personalized services. The security feature of a web application framework concentrates on helping web developers to simplify the process of building up necessary management tool for user accounts of web sites.

Some web application frameworks come with authentication and authorization frameworks which enable the web server to identify the users of the application, and restrict access to functions based on some defined criteria. Some web application frameworks provide not only role-based access to pages but a web-based interface for creating users and assigning them roles as well.

2.3.3. URL mapping

The URL mapping facility is the mechanism by which the framework interprets URLs. Some frameworks match the provided URL against pre-determined patterns using regular expressions while some others use *URL rewriting* to translate the provided URL into one that the underlying engine will recognize. Another technique is that of graph traversal where a URL is decomposed in steps that traverse an object graph of models and views.

A URL mapping system that uses pattern matching or URL rewriting allows more friendly URLs to be used, which increase the simplicity of web sites and allow better indexing by web search engines. The changed URLs which are easier to read provide search engines better information about structural layout of the site. A graph traversal approach also tends to result in creation of friendly URLs.

2.3.4. Database access and mapping

Many web application frameworks create a unified API to a database backend to enable web applications to work with a variety of databases without code changes, and allowing programmers to work with higher level concepts. Some object-oriented frameworks contain mapping tools to provide Object-relational mapping (ORM), which is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. By implementing ORM, programming languages used in a web application framework operate virtual objects instead of data from a database, which reduces the amount of code needed to be written by web developers. (Ambler, 2010) Transactional support and database migration tools are another two important features provided by some web application frameworks also.

2.3.5. Data caching

Data caching in web applications is the caching of web documents and queried data in order to reduce bandwidth usage, server load, and perceived lag. A web cache stores copies of documents and query results passing through it; subsequent requests may be satisfied from the cache if certain conditions are met. Some web application frameworks provide mechanisms for caching documents and queried results and bypassing various stages of the page’s preparation, such as database access or template interpretation.

2.3.6. Software development toolkit

Some web application frameworks also provide a built-in web server application and virtual hosting environment for web developers to run under the local environment for quick testing of the modification after instantly any code changes. The web server application built inside is commonly light-weight, fast to start up, and easy to debug with rich debugging features available.

Furthermore, some web application frameworks contain a JavaScript library to provide convenience for client-side UI development by web developers. [See Section 2.2.2] With the help of the JavaScript library, it turns out to be much easier for developers to manipulate DOM (Document object model) elements on HTML web pages, implement dynamic effects, and transfer client requests and server responses within Ajax-styled interfaces.

2.4. Introduction of Google App Engine

Google App Engine is a platform for developing and hosting web applications in Google's infrastructure. With the power of Google-managed servers and data centers, web applications implemented and deployed on App Engine are migrated with cloud computing technology seamlessly, which assures the stability and flexibility of its hosting service globally. (Google, 2008)

Google App Engine was designed and implemented to be programming language independent to attract web developers from different technical backgrounds. When it was first released as a beta version in April 2008, Google App Engine supports Python only. And one year later the support of Java was added. After the Java support was added in early 2009, theoretically, any programming language which is supported by the Java Virtual Machine (JVM) is available to be used for coding with App Engine nowadays.

Google App Engine was designed to work with most web application frameworks which are written in any one of its supported programming languages. For Python programmers, Google App Engine provides also a simple web application framework named as webapp, (Google, 2008) which inherited some advantages and characteristics from another powerful Python-based web application framework, Django. (Django Software Foundation, 2005-2010) With this built-in web application framework and other unique features, Google App Engine provides a complete, feature-rich environment for implementation and online publication of web applications.

The figure below [See Figure 2.2**:** Features provided by Google App Engine against Web Application Framework] shows the features provided by Google App Engine by default, which correspond with the standard functionalities provided by a web application framework mentioned and discussed in the previous section [See Section 2.3]. These features include a powerful template engine, Google Accounts, easy configuration, datastore, memcache, specific SDK, and the unique Cloud computing platform from the Google infrastructure, which are introduced one by one in the next sections.

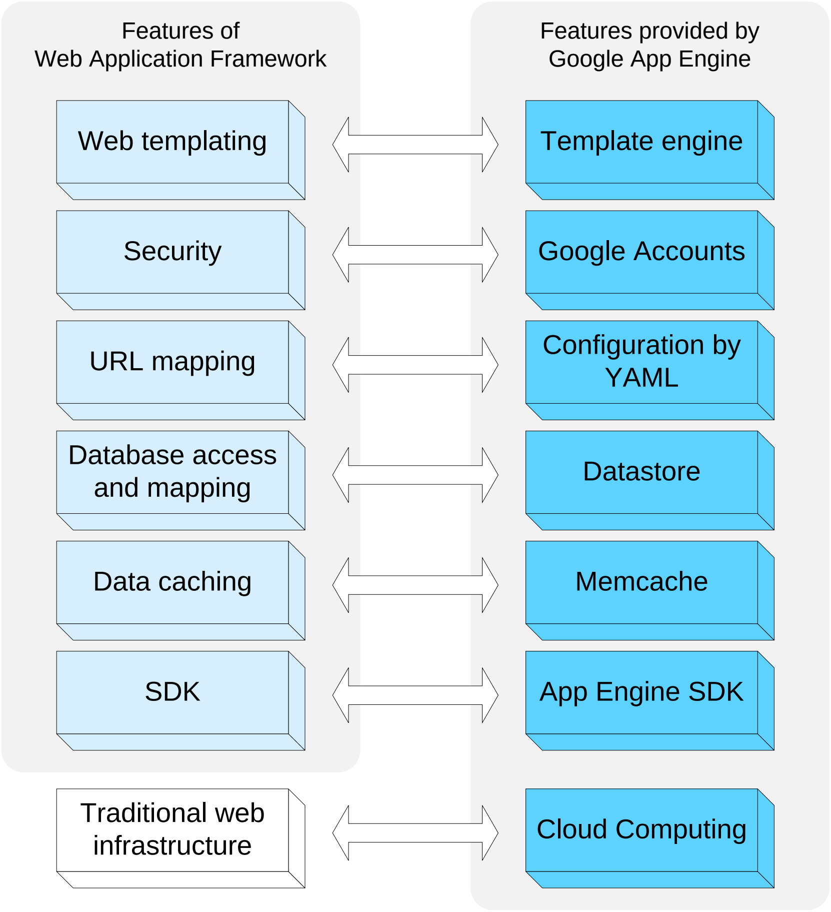


Figure 2.2: Features provided by Google App Engine against Web Application Framework (Google, 2008)

2.4.1. Template engine

The built-in web application framework, *webapp*, of Google App Engine inherited some advantages and characteristics from another Python-based web application framework, Django. Django owns many powerful features which help developers from each aspect of web development. The template system is one of the most powerful features Django has. Django maximizes the implantation of web page templates by adding a set of template language to its template engine in order to strike a balance between power and ease. It helps those who are used to working with HTML to feel comfortable when coding on text-based templates. (Django Software Foundation, 2005-2010)

Django's template language separates design, content, and Python code while at the same time remaining powerful, extensible and designer-friendly. Being proud of its modular concepts with template layer, the Django software foundation prefers to call its framework structure as MTV (Model-Template-View) in Django's documentation (Django Software Foundation, 2005-2010) instead of the standard name, MVC (Model-View-Controller) although, in fact, Django agreed to be a framework in line with MVC architecture pattern by its followers.

In App Engine, web developers who use webapp can conveniently invoke the API of Django's template engine after stating the import operation of the built-in template extension at the beginning of the handler code. (Google, 2010)

2.4.2. Google Accounts

Google Accounts is a service of App Engine platform which can be used to authenticate users of applications hosted by App Engine. By using the Google Accounts API, an application can detect whether a current user has signed in with Google's account, and redirect the user to Google Accounts sign-in page if not. The sign-in page will show a sign-in dialog to the user, or prompt him to create a new account. When a user is signed in to the application, the application can access the user's email address as well as a unique user ID. Google Accounts also helps an application detecting whether a current user is an administrator of it, making it easy to implement the administrator-only features of the application. (Google, 2010)

While Google Accounts integration is an optional feature of App Engine, it offers a robust, easy to use authentication mechanism with a mature set of features and millions of active users.

2.4.3. Configuration by YAML

App Engine routes requests to Python scripts based on the URL and mappings specified in the application's configuration file called *app.yaml*. This file describes which handler scripts should be used for which URLs. The URL path in a mapping is a regular expression which may contain regexp groupings to match parts of the URL. Patterns matched in groupings are passed to request handlers as arguments.

The syntax of the *app.yaml* file is YAML (YAML Ain't Markup Language), which is a human friendly data serialization standard for all programming languages. (Evans, 2004) With its concise design, it is especially suited for tasks where humans are likely to view or edit data structures, such as configuration files.

2.4.4. Datastore

The datastore of App Engine is a schemaless object datastore with a query engine and atomic transactions, providing robust scalable data storage for hosted web applications. With design considered web applications from the beginning, the datastore emphasizes performance of reading and querying operations to data. All queries are pre-indexed for fast results over very large data sets.

Unlike traditional databases, the datastore of App Engine uses a distributed architecture to manage scaling to very large data sets. An application hosted in App Engine can optimize how data is distributed by describing relationships between data objects and by defining indexes for queries.

The App Engine datastore is strongly consistent, but not a relational database. While its interface has many of the same features of traditional databases, the datastore's unique characteristics imply a different way of designing and managing data to take advantage of the ability to scale automatically.

2.4.5. Memcache

High performance scalable web applications often use a distributed in-memory data cache in front of, or in place of, robust persistent storage for some tasks. App Engine includes a memory cache service for this purpose. The App Engine's *Memcache* provides fast, transient distributed storage for caching the results of datastore queries and calculations.

One use of a memory cache is to speed up common datastore queries. The web application can cache the results in the memcache if many requests make the same query with the same parameters, or changes to the results do not need to appear on web pages right away. Subsequent requests can check the memcache and only perform the datastore query if the results are absent or expired. Session data, user preferences, and any other queries performed on most pages of a site are good candidates for caching.

2.4.6. App Engine SDK

The App Engine SDK includes a web server application which can be run on any developer's computer that simulates the application running in the App Engine runtime environment. While it is running, the web server watches for those changes the developer makes to the files, and reloads them if needed. For most kinds of changes, the developer can simply edit files and then reload the web page in his browser. The web server is needed to restart only when the module import caching needs to reset for the application to do dynamic imports.

Furthermore, the development web server of App Engine provides also simulations of App Engine datastore, Google Accounts, and many other features which can be found in the actual hosting platform. Such a powerful development environment helps web developers making the testing and debugging of their implementations far easier.

2.4.7. Cloud Computing

Beyond all of these standard features which should be possessed by a general-purpose web application framework, App Engine's Internet-based infrastructure, called Cloud computing, makes it a perfect platform for the development and hosting of web applications.

Cloud computing is a computing capability that provides an abstraction between the computing resource and its underlying technical architecture, enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. (Mell & Grance, 2009)

Instead of a traditional client-server structure, cloud computing abstracts the details of control over technology infrastructure from users and sets up a new supplement, consumption, and delivery model based on the Internet. Furthermore, it involves also dynamical scalability and resource virtualization as an Internet service.

3. Research

In this chapter, the research method which has been used in the experimental part of the study is discussed. Each step of the research process has been picked up and discussed separately with details of consideration, alternative solution, and final implementation. Some coding practices of the research issues are also presented in this chapter.

3.1. Research Method

The main purpose of the experimental part in this study is to re-define and re-construct an outdated web site to encourage users to contribute content, thus increasing user visits and information sharing among all users. To achieve this, the old structure and components of the site, which consist of static web pages, existing user interface, and the traditional LAMP server structure lying behind, [See Figure 1.1**:** Web site of CSSA-Espoo Ry before the study began. It is a typical Web 1.0 site written mainly by HTML and CSS.] were necessary to be re-defined and re-constructed. Creating such a public web site from the beginning is kind of similar to creating a software application under a desktop computing environment, as the standard concepts and work processes of industrial software engineering are appropriate also for creating a user-friendly web site.

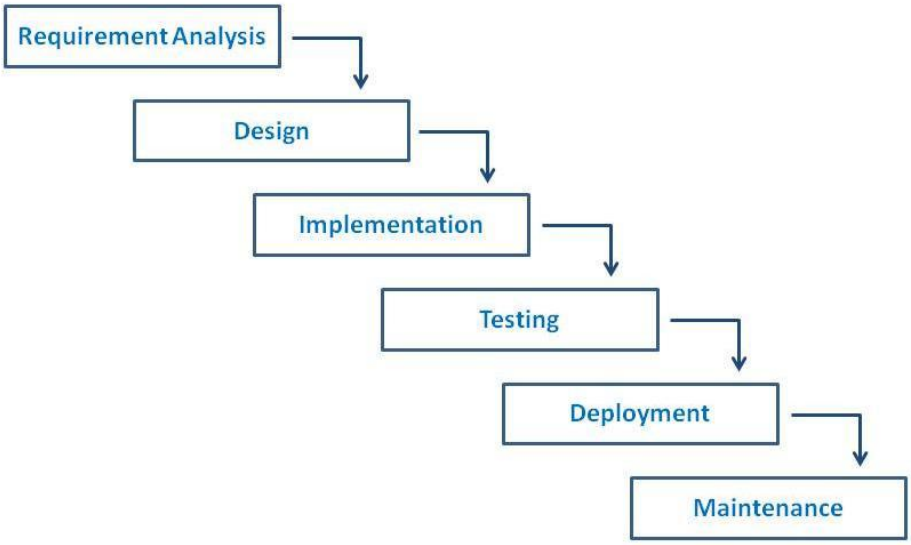


Figure 3.1: General activities of software development process represented in the waterfall model. There are other alternatives available.

According to this waterfall software development process, software development activities generally consist of requirement analysis, design, implementation, testing, deployment, and maintenance. [See Figure 3.1**:** General activities of software development process represented in the waterfall model. There are other alternatives available.] Correspondingly, the research method of the experimental part in this study also tries to follow this process, with some additional discussions about choice of hosting platform of the re-defined web site and choice of software development tools, as they are critical for building a software application located in the web environment.

3.2. Requirement Analysis

The web site CSSA-Espoo was created and maintained continuously by CSSA-Espoo Ry (Chinese Students and Scholars Association of Espoo) since the year 2004. CSSA-Espoo Ry is a non-profitable organization which serves Chinese students and residents in the Helsinki metropolitan area of Finland. The initial site structure and contents were established by several Chinese students from Aalto University School of Science and Technology some years ago to provide information to site visitors, who were mainly students and local residents from China. (CSSA-Espoo, 2009) According to the facts mentioned above, the main concerns of the requirement are listed below.

3.2.1. Functionality

The web site of CSSA-Espoo was supposed to be a channel for its users to submit, retrieve, and exchange helpful information. It should have a user-friendly web interface, and encourage users to contribute more content by providing a built-in incentive mechanism. It should be possible and convenient for users to manage all content submitted, and to give feedback to content provided by other users.

The web site should be accessible to the public without tedious authentication, meanwhile having enough security to protect user identities.

The re-defined web site should not be a replacement for any existing service managed by CSSA-Espoo Ry, for instance, the mailing list and discussion forum, but a necessary supplement.

3.2.2. Language on user interface

The majority of site users of CSSA-Espoo are Chinese who lives in a foreign country where their native language is hardly to be found useful most of the time. However, the electronic mailing list (CSSA-Espoo, 2008) provided by CSSA-Espoo Ry for information sharing does not accept Chinese characters because of its unchangeable encoding settings. Any Chinese character included in an e-mail body sending to the mailing list would be rendered garbled. To activate users to produce more content, the Chinese language should be considered as the first choice of user interface language of the re-engineered web site.

3.2.3. Service capacity

Distinct from a local software application which commonly serves one single user, service provided by a public web site has to provide enough capacity to process multiple user requests simultaneously. From the curve of daily user visits illustrated by Google Analytics, [See Figure 1.2**:** Daily site visits recorded by Google Analytics] the average number of daily visits to the old site is between 30 and 50. This helped with the estimation of the total amount of users possible for the re-defined site. As the amount of site users would rarely reach as high as a hundred, the design of site structure and request handling mechanisms should not be challenging.

3.2.4. Financial support

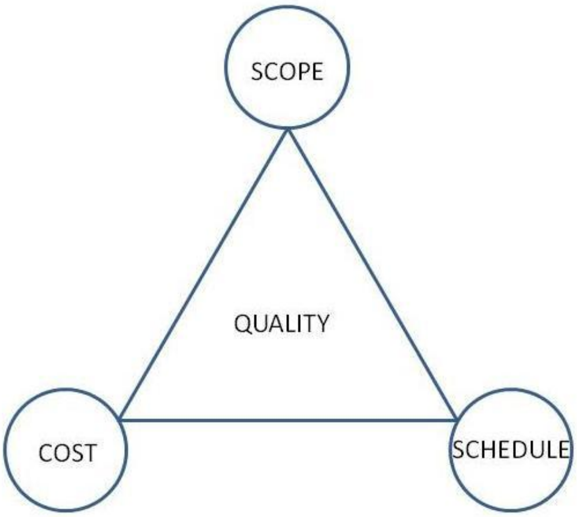


Figure 3.2: The Projects Management Triangle

A basic factor of the project requirement was a non-existent budget, considering the financial situation of such an organization, which demanded that the creation, hosting, and maintenance of the re-defined web site in this study should be all free of cost both for now and in the foreseeable future. This influences the choice of web development tools, hosting platform, complexity of site implementation, and related aspects to be considered.

3.2.5. Knowledge transfer

When the new site is created and deployed successfully by the end of the experimental part of the study, the maintenance work would probably be transferred to other one or several persons. In order to transfer both written code and knowledge lying behind smoothly, all code and related documents should be addressed in an accessible and secure location where is convenient for collaboration work among all developers and maintainers of the web site. Additionally, to decrease the difficulty of becoming familiar with the code for later coming developers, the whole structure and every piece of code of the re-defined web site should follow the KISS (Keep It Simple and Stupid) principle wherever possible.

3.3. Design

According to the results of the requirement analysis, some choices for coding and hosting practice have been made before starting the implementation. And the user interface of the new web site is also decided in this step of the research process.

3.3.1. Choice of hosting platform

The choice of hosting location is critical to every web site and online service, needing to be accessible from anywhere, and stable enough to be usable at any time. Depending on the programmable characteristic of the web service in this experimental part, the hosting platform needs to provide background database support, and an environment for running dynamic language scripts also.

The limitation of a non-existent budget [See Section 3.2.4] required that the selected web hosting platform needs to be completely free of charge for its services. Unfortunately, the majority of hosting services available on the Internet did not satisfy this requirement when considering dynamic language programming ability.

One possible choice is the web server service provided freely by TKY (Student Union of Aalto University School of Science and Technology). Based on a traditional LAMP structure, the hosting server provided by TKY allows a registered web site administrator to upload static web pages and PHP-based dynamic scripts, to construct the web site with Apache and a background MySQL database. The hosting server is maintained by TKY who has forbidden all possible online accesses from outside the university campus area, which means that the web site administrator has to be a student who registered in TKY, and he has to establish the connection to the server to do maintenance work from an inner IP address on the campus network. (The outdated CSSA-Espoo web site and the discussion forum were established on this server service.)

The main alternative, however, is the public online hosting service provided by Google App Engine. [See Section 2.4] As a feature-rich hosting platform, App Engine provides industrial scalability and is free of charge when the storage and page views per month do not exceed limited values. (Google, 2010) After balancing these values with the required server capability discussed in the requirement analysis [See Section 3.2.3], it has been agreed that the limited values for the free usage of App Engine service is sufficient for the experimental part of this study.

Compared with the network connecting limitation of TKY's web server, Google App Engine has better accessibility which allows anyone who wishes to help with the maintenance work of the re-designed online application. Furthermore, App Engine provides relatively complete documentation of integrated APIs and a feature-rich SDK for web developers, which was absent from the traditional LAMP-based web server provided by TKY.

After the measurement of benefits and limitations between these two possible solutions, Google App Engine was finally chosen to be the hosting platform of the re-engineered web site. As a matter of the fact, this decision influenced also the choice of web site development tools.

3.3.2. Choice of development tools

In order to investigate the benefits brought by a web application framework and Ajax techniques to web site development, the experimental part of the study will use a web application framework as a part of the development tools together with a JavaScript library which contains the necessary Ajax functionalities. All choices of development tools highlighted reducing the programming workload while minimizing the necessary time required learning how to use and maintain system.

Web application framework

Based on the choice of hosting platform, the built-in web application framework provided by Google App Engine, webapp, turned out to be the first, as well as the best, choice for the experimental part of this study. Webapp is simple but powerful enough to be used in the research work as it provides all general features a web application framework should have. [See Section 2.4] Since it is built-in to App Engine by default, it is not necessary for developers to be concerned about the binding work which is compulsory for any other web application framework to work with the App Engine platform.

JavaScript library

The JavaScript library is commonly used nowadays in web development on the Internet. It releases web developers from tedious JavaScript coding and debugging, and make it possible for beginners to implement the comprehensive behaviors of web page elements. Although webapp in Google App Engine does not contain a JavaScript library by default, there are many JavaScript libraries available with a free software license. As most of those libraries provide similar functionalities, one would be enough to be used in the research work.

Based on the survey of usage of common JavaScript libraries, jQuery is the most popular JavaScript library used on the Internet. (BuiltWith, 2010) It was also chosen to be used for the experimental part of the research.

Programming language

Based on the decision of using Google App Engine as hosting platform and webapp as the web application framework, Python turns out to be the best choice of working language for the programming part of the research. It is a remarkably powerful dynamic programming language that is used in a wide variety of application domains and available for all major operating systems. (Python Software Foundation, 1990-2010)

3.3.3. User interface

Based on the results from the requirement analysis, some basic web elements are selected to be used in the online user interface of the web site. These include a text input block for users to contribute content to the web site [See Figure 3.3: Web elements designed for user input], several uneditable text blocks for showing user-generated content on the page together with buttons to allow users to provide feedback to each text message contributed by other users [See Figure 3.4: Web elements designed for showing user input]. Considering the other standard functionalities necessary for the correct behaviors of the web site, some other buttons, for instance login, logout, turning to next and previous page, are also necessary to be presented on the web page.

In order to regulate the page layout easily, the maximum number of total characters contained in a text field should be limited to a fixed number in the text input field [See Figure 3.3: Web elements designed for user input]. With the limitation of character amount, the layout of the user interface could prevent chaos resulting from the irregular size of text blocks.

Besides the uneditable text field [See Figure 3.4: Web elements designed for showing user input], the buttons for users to provide feedback to messages created by other users should not always be clickable. These buttons should not be available when the message is created by the user himself. When the user has already provided his feedback to the message, only one of these two buttons should be available. The available button should turn around the user's feedback from "like" to "dislike", and vice versa.

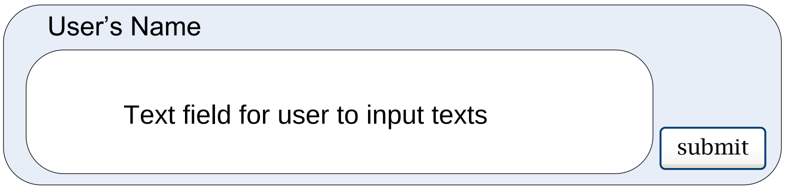


Figure 3.3: Web elements designed for user input

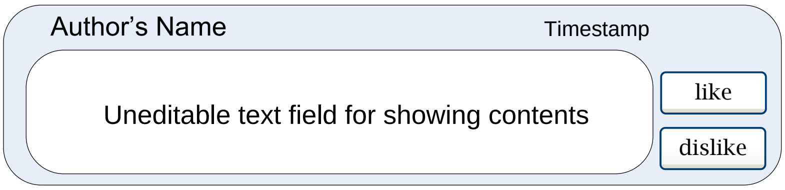


Figure 3.4: Web elements designed for showing user input

Ajax techniques would provide a good user experience when the user interacts with the page elements. After the "submit" button or one of the feedback buttons is pressed, the page should not try to refresh the whole layout. Instead, only those text blocks which have been changed in content would be re-rendered on the page.

3.3.4. Incentive mechanism

Beyond the design of user interface, a set of incentive mechanisms are also necessary to activate user participation of the re-engineered web site.

There should be two sets of ranking system existing on the web site. One set of ranking system is used to compare different text messages submitted by users. If a text message receives more positive feedback from users than another message, it should have higher ranking points and should be listed in front of the other. If two or more messages achieve the same ranking points from user feedback, the ranking order should be based on the timestamps of their creation. The later created message should be listed in front of the older one.

The other set of ranking system is used to compare the contribution among different users. Every user should get some ranking points when contributing new contents to the web site. The behavior of providing feedback to other users' contribution should be rewarded also no matter whether the feedback is positive or negative. In order to encourage users to contribute high-quality content, the message which receives more positive than negative feedback should add more ranking points to the author. Correspondingly, those messages which receive more negative feedback than positive feedback should decrease the ranking points of the author.

3.4. Implementation

The practical implementation work started after the decision of development tools had been made and the user interface had been drafted out. Unavoidably, there was some time consumed for the pre-study of necessary knowledge to become familiar with develop tools, which included the jQuery JavaScript library, webapp framework and database API provided by App Engine, and the Python language for coding practice. Fortunately, there were rich documents for web developers to learn these skills rapidly.

During the coding practice, several issues emerged when trying to combine the webapp framework with the jQuery library. These issues and solutions are described in the next sections.

3.4.1. Model-Template-View vs. Ajax

MVC (Model-View-Controller) is an architectural pattern commonly used in software engineering. The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each. [See Section 2.1.1] In spite of the widespread use of MVC frameworks in most software development environments, some web application frameworks, for instance, Django, named their framework structure as MTV (Model-Template-View), since the Template layer for rendering web pages in web application development is more important than in other development frameworks.

Inherited from the Django framework, the webapp framework built into Google App Engine implements its structure as MTV (Model-Template-View) also, which emphasizes the benefits brought by the powerful template layer. In the webapp framework, requests from clients are sent to the View layer to be handled. The View layer is responsible for invoking templates, filled variables with real data from the Model layer, and sending rendered pages back to the client side. The View layer is the key role inside the work flow, which is similar to the role that the Controller plays in the MVC (Model-View-Controller) pattern.

However, when asynchronous data transfer of Ajax techniques was introduced to this layer structure existing in the webapp framework, the absence of the Controller layer from the whole structure turns out to be an issue. This is because the asynchronous data transfer requests pure data as the results without formats or styles from templates. Using the View layer to handle both pure data and the combination of data with expressive content will leads to confusion. Such confusion prevented the View layer from being used as the backend for asynchronous data fetching.

3.4.2. jQuery vs. REST

The JavaScript library jQuery is a fast and concise JavaScript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. It is designed to achieve best performance with best compatibilities across all possible browsers on all platforms. (jQuery, 2006)

REST (Representational State Transfer) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The key point to follow the REST style is to interact with web servers by using four basic operational keywords, GET, POST, PUT and DELETE. While these keywords are a part of web standards, all browsers do not support all of these four keywords. This is a blocker which discouraged the web application framework or JavaScript library to implement REST concepts into the set of their standard features. Some web application frameworks solved this by using POST to simulate the behaviors of PUT and DELETE inside their structures. Some simply dropped the support for the PUT and DELETE keywords.

In jQuery's partial API of Ajax controls, the keywords PUT and DELETE are unfortunately dropped to achieve the best compatibilities across common browser applications in the market. As the coding part of this experimental research tried to follow the REST principle all the way, using PUT and DELETE keywords for data transferring became an issue when exerting jQuery's Ajax API. Fortunately, jQuery provides a powerful plug-in system, which allows users to create additional functions to enhance its standard abilities.

3.4.3. Ajax over JSON

Asynchronous data transfer, which is the core idea of Ajax techniques, is useful for the client side to send a request to and receive a response from the server side without the need for a page refreshing operation. [See Section 2.2.1] However, not as its name pointed out, Ajax techniques are, in fact, not necessary to work with XML (Extensible Markup Language). Data presenting in XML format is quite heavy if the amount of transferring data is not large when considering the redundancy brought by the beginning and ending tags of XML. In such a situation, JSON turned out to be a better data format to carry these data in the transferring process among servers and client applications.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write, and for machines to parse and generate, which make JSON an ideal data-interchange language. By using the default data format built-in to JavaScript, JSON simplified the boundary symbols, which used to separate and identify data, much more than the markup labels used by XML, therefore shrinking the redundancy of data presence to the lowest level. (JSON, 1999)

The design of the re-engineered web site for the research requires only a small set of data to be transferred at a time across the server side and client pages. As the experimental part of the research focuses on improving the user interface and user experience, JSON turned out to be the better data format for the data transferring operation when implementing Ajax behaviors.

However, during the implementation, one critical function for operating JSON-formatted data, *simplejson.dumps()*, which was provided by the webapp framework but inherited directly from Django version 0.96, was found to contain a software bug for processing UTF-8 characters. The bug will lead the function to generate an incorrect series of data during transferring, which could not be recognized by browser applications which support Unicode text format. An observation later found out, however, that the bug was actually occurring in the earlier version of the Django API only, and has been fixed in new versions later than version number 0.96. The framework webapp tried to use the buggy version of Django API by default, and thus introduced this error into the App Engine environment.

3.5. Testing

In the process of software engineering, testing commonly happens after the implementation work has finished according to the design. In practical programming work, the testing actually happens together with implementation from time to time. It is quite difficult to specify in which time period the work of testing separated from implementation. This is more obvious in agile-styled development.

As many tiny issues, for instance coding syntax errors, were found actually during implementation and solved easily, they will not be mentioned here. Instead, the issues found during integration testing which were so serious that the whole structure of the research work had to be considered once again because of those are discussed in the next sections.

3.5.1. Browser compatibility

Compatibility testing is always a critical step for any web application development process because differences among browser applications always exist. As browser applications are software products produced by different commercial companies, each company implements web browser standards in an independent way. The browser’s parser of CSS is one example of these conflicts.

CSS (Cascading Style Sheets) is commonly used on web sites as a basic standard to regulate presenting styles of text and pictures in web pages. W3C (W3C, 2010) has published the CSS standards from version 1 to 3, to reflect the demand of web design requirement. (W3C, 2010) However, all browsers do not follow these de facto standards. Some grammars were replaced by their unique implementation out of standard. Some features were even not implemented at all.

In the design of the experimental part of this research, CSS would be used heavily to present text and page styles of the whole web site. Some UI effects are based on the CSS parser and operations, which include round-corner blocks and half-transparent blocks. During the testing phrase, it was observed that although the majority of modern browser applications, for instance Mozilla Firefox, Google Chrome, and Apple Safari supported these CSS effects created for the experimental part of work, Microsoft Internet Explorer seemed do not.

For the half-transparent effect of text blocks, Internet Explorer provided a differently named CSS attribute to achieve the similar results. But for the round-corner-block effect, this browser application totally blocked the possibility of implementing it in a CSS way.

3.5.2. Invoking API of JavaScript library

Another compatibility issue which emerged in the testing phrase of the engineering process was generated by the usage of the invoking API of JavaScript libraries provided by Google.

This set of APIs was named by Google as the AJAX Libraries API. It is a content distribution network and loading architecture for the most popular, Open Source JavaScript libraries. (Google, 2010) It provides a simple method, *google.load* (Google, 2010), for JavaScript programs to launch the necessary JavaScript libraries easily. One of these important JavaScript libraries is jQuery, which was used in the implementation heavily for all interactive behaviors presented to users and asynchronous data transfer in the background.

By using Google's AJAX Libraries API, these JavaScript libraries which are commonly used on web sites get the benefits by an offline cache mechanism provided by most kinds of browser applications. When a browser application is requested by a user operation to fetch a JavaScript library file from a remote address, it will check if the file has been downloaded into the local cache before trying to download it a second time. Providing a unified web location to store the most famous JavaScript libraries in Google's infrastructure undoubtedly saves the downloading time of all of the users who visit these web sites, thus benefiting the web sites by increased loading speed from the client user's point of view.

Unfortunately, from the observation during testing of this research, Microsoft Internet Explorer has a proven lack of support for this set of APIs. It blocked the attempt to run the unified invoking function provided by Google, which was used to specify the version number and related options in order to invoke a correct JavaScript library file. This behavior was observed from Microsoft Internet Explorer version 6, 7, and 8, which led the user to see only incorrectly rendered pages of web sites. Instead, other commonly used web browser applications, for instance Mozilla Firefox, Google Chrome, and Apple Safari, loaded the JavaScript library correctly and rendered web pages quite well.

3.6. Deployment

The deployment step in software engineering means to deliver the finished coding results into the real working environment. For web applications, this means setting up the server-side services and getting those ready to serve client users. In web application development, deployment could be more difficult than local software applications since web service has to be a combination of services by the HTTP server, database server, language runtime, and other relative services. The configuration and adjustment of all these interconnected services are time-consuming and error-prone.

Fortunately, Google App Engine provides a powerful SDK for web developers to deploy the tested application onto the remote server. The SDK is responsible for detection of any file changes since the last time when new site content was submitted, and uploads the changed files automatically for the user.

Google App Engine supports also debugging on the remote server directly. After the coding work has been deployed onto the App Engine platform, App Engine will record all warnings and errors happening during the serving time, which are easy to access by web developers. This helps web developers to easily trace hidden issues happening only in the real environment.

3.7. Maintenance

Maintenance is the last step of the software engineering process. It commonly consists of fixing bugs found after deployment, adding new features onto existing applications, and other code modification to fit the user's requirement changes. Because the experimental part of the study aimed to create a web site with new styled user interface to encourage users' interaction, maintenance was certainly important for following the feedback of users and upgrading the user interface and functionalities.

In order to help other developers to become familiar with the code and the whole structure of the experimental part of the study as quickly as possible so that they could continue the maintenance work later after this study, all parts of the research work have been saved in the source code and uploaded to a public online version control system. The usage of a version control system helps several developers to share source code changes and prevent the risk of data loss or undesired operations resulting from human mistakes. [See Appendix C]

4. Results

The results of the study are presented here which consist of both the final user interface and code implementation. Related to the interface design, consideration of how to increase the users' visiting rate and, thus the generated mechanism of a user rating system built-in to the final application, is also introduced. For the coding presentation, the results of the research of conflicts among Google App Engine platform, jQuery, and related coding guidelines are presented and discussed.

4.1. Decision of Design

This section described decisions made during the design phrase of the research process, which included the final presence of the user interface, the usage of dynamic effects on page UI elements, and the incentive mechanism adopted for stimulating user participation.

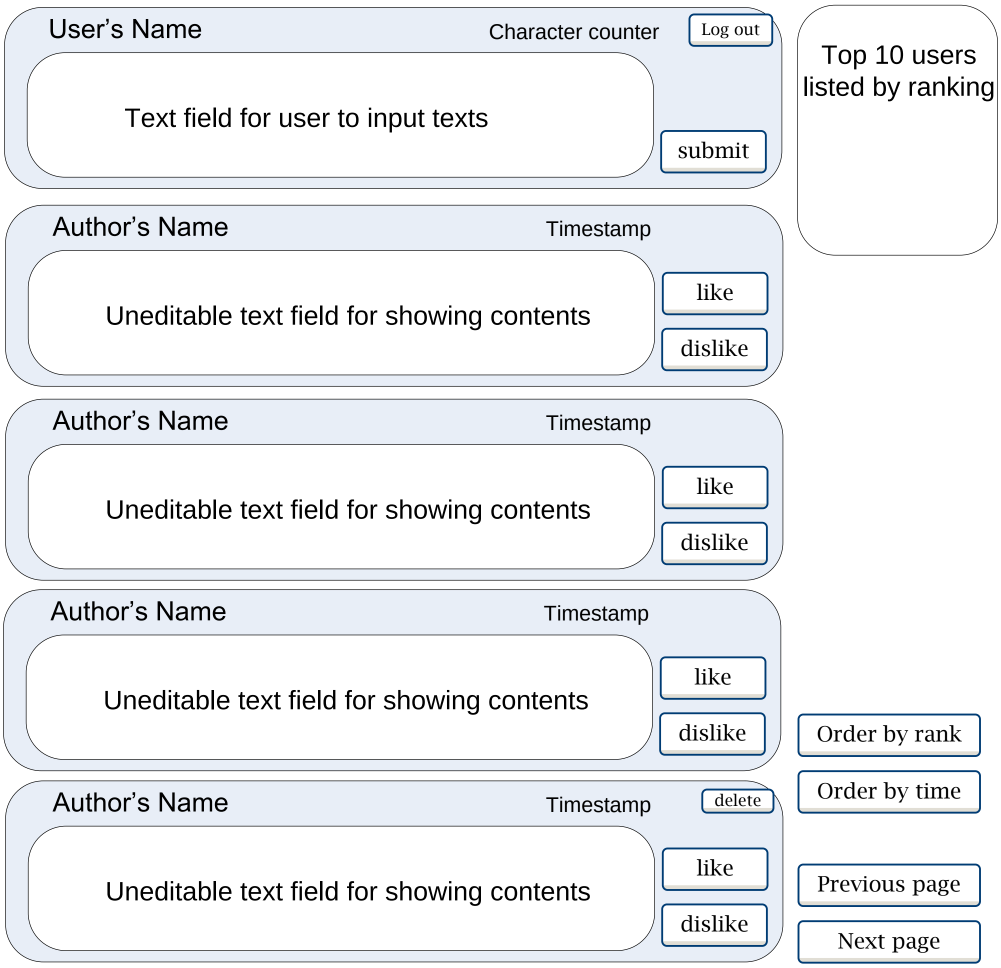

User's Name
Character counter
Log out
Text field for user to input texts
submit
Top 10 users listed by ranking
Author's Name
Timestamp
Uneditable text field for showing contents
like
dislike
Author's Name
Timestamp
Uneditable text field for showing contents
like
dislike
Author's Name
Timestamp
Uneditable text field for showing contents
like
dislike
Order by rank
Order by time
Author's Name
Timestamp
delete
Uneditable text field for showing contents
like
dislike
Previous page
Next page

Figure 4.1: Finalized design of user interface [See Appendix A]

4.1.1. User interface

In order to simplify the functionalities of the web application and the development work of the coding part, the experimental part of the study was designed as a single-page web application, which means the user interface was unified all over the functional steps across the usage. [See Figure 4.1]

One benefit of keeping the application as a single-page application is the avoidance of users' reloading operation of the web page. All of the data transfer is done by Ajax when users interact with the application. And the UI will change according to the data modifications made by the users' data requests.

Another benefit of using single-page design is the unified user experience across all users' operation behaviors. The UI keeps the same all the time from when to user starting browsing the web site until leaving it. This helps users become familiar with the application faster at the first visit without the need for any form of tutorial, and to maintain familiarity when they come back later.

Furthermore, the unification of the UI helps the implementation work to be done with relative ease, because the UI design does not change when the data transfer state changes. For an experimental part of this study, such a decision may save a lot of time otherwise spent on coding work.

4.1.2. Interactive UI elements

Although this web application has a single-page UI, the implementation tries to add some interactive UI elements to enhance the user experience and provide more functions in a more convenient method. So, as a result, some JavaScript based user-interactive-driven DOM (Document Object Model) behaviors were added to the UI.

One of the examples of these is the button behavior. When a user moves his cursor over to a button of the application, the button slides a bit to the right side smoothly, and shows a word as a hint of the functionality it has, while the floating cursor turns to a finger shape at the same time. All these changes of the behaviors help users a lot to recognize the usage of the UI elements of the application. Instead, if a button widget is not clickable, the widget will not be interactive with the cursor, nor the cursor shape will change.

Another example of such interactive UI elements is the button used for the rating system. As a normal button of the application, users can click the button to rate a message when the button is clickable. But according to the background logic judgment, the button might turn to be unable to click, to avoid a user to rate a message twice or to rate one produced by himself. In such a context, the button’s background color turns to dark, and the cursor hovering on it will not change to a finger shape, but remain the same as a normal one.

4.1.3. User ranking system

Based on the rating system designed for arranging the order of all messages published on the application, a ranking calculation of each registered user of the web application is also running in the background. Every user will get some points when he submits a new message to the application, and when the message is labeled as “like” or “dislike” by any one or several other users, the author of the message will be rewarded or punished by the overall rating of the messaged received.

Additionally, a user will also get points when he points out his “like” or “dislike” to a message created by another user. Every user has only one vote for each message which is however allowed to be changed to opposite later after voted. When a message is later deleted by its author, the points other users got by voting on the message will be removed also.

Based on these points achieved by each user of the application, the ranking order of all users is easy to be defined. By showing the names of several top ranking users, the application tries to stimulate users to create more useful messages and also give feedback to every message created by others, and also competing with other users to reach the highest rank.

4.2. Solution of Implementation

This section summarizes all issues and conflicts found out in the implementation and testing phrases of the research process, and describes the corresponding solutions which were decided and implemented in the finished coding practice. Each issue mentioned below has been described in the Research chapter in detail.

4.2.1. Model-Template-View vs. Ajax

This issue arose because the MTV structure of the webapp framework was not suitable for Ajax techniques to be used easily. [See Section 3.4.1]

As a solution, in the final code structure of this experimental part of the research, the Controller layer was added into the code structure, dealing with the fetching requests of all asynchronous data, and sending pure data results back to the client side in JSON (JavaScript Object Notation) format. This new pattern combined the benefits from MVC and MTV altogether, and solved the issue of using Ajax techniques together with the webapp framework.

4.2.2. jQuery vs. REST

This issue arose because the jQuery JavaScript library removed the support of two standard HTTP methods, PUT and DELETE, in order to increase its compatibility across browser applications. [See Section 3.4.2]

To fix this issue to allow jQuery to support all four standard keywords of the HTTP protocol, a patch was created and put into the final application code. The patch consists of two self-defined functions which were implemented to simulate the behaviors of PUT and DELETE keywords. Getting these two HTTP methods back to work made the implementation based on the REST principle being possible. As a drawback, the compatibility across different kinds of browsers of this re-engineered web site would be affected. The browser applications those do not support PUT and DELETE keywords would not render the web site correctly.

4.2.3. Ajax over JSON

This issue arose because the default API set in the webapp framework introduced a bug to the built-in function which was necessary for operating JSON-formatted data. [See Section 3.4.3]

To fix this issue in the final implementation, some lines of code were added to the header part of every executable script file in the experimental part of the work. These lines of code requested the webapp framework to load the latest Django API (version 1.1 instead of version 0.96 by default) before invoking any function borrowed from Django directly. This ensures JSON-formatted data being correctly analyzed and operated by the webapp framework and client-side browser applications.

4.2.4. Browser compatibility

This issue arose because the browser application, Microsoft Internet Explorer, did not support part of standard CSS effects used in the user interface implementation. [See Section 3.5.1]

In order to decrease the workload for enhancing the compatibility of the web site, the experimental part of the research did not try everything to perfect the user experience for users who used Microsoft Internet Explorer for browsing. It has been observed during the testing period that the lack of specific modification of code for Internet Explorer did not, however, greatly affect user experience. There existed also the expectation that the W3C's

CSS standards will be accepted and implemented across all browser applications in the world in the near future.

4.2.5. Invoking API of JavaScript library

This issue arose because the browser application, Microsoft Internet Explorer, did not support the API provided by Google for invoking JavaScript libraries. [See Section 3.5.2]

The temporary “work-round” solution of this issue was to avoid using the invoking API for JavaScript libraries provided by Google, instead using the direct URI in the declaration of the JavaScript library, which is jQuery in the research work. In such a way, the jQuery library will be ensured to be pre-fetched by all browser applications. But by pointing out a direct web link explicitly, we lost the flexibility of using the API to fetch the library file dynamically.

5. Discussion

In this chapter, based on the research work and results of the experimental part of the study, an attempt has been made to answer the questions generated in the *Introduction* chapter [See Section 1.2]. Some further comparison between the final work and some common online social network services has been discussed in this chapter too.

5.1. Answering Research Questions

In the beginning of this thesis, research questions were defined [See Section 1.2]. Based on the conducted research and the results presented in the *Research* chapter [See Section 3] and the *Results* chapter [See Section 4], the research questions will now be reviewed.

1st Question: What are benefits and drawbacks brought to web site developers in using a web application framework and Ajax?

From the practice of the re-engineering work for the experimental part of the study, the benefits of using a web application framework are obvious to web developers. The usage of the web application framework helps the web site development on most aspects which significantly reduced the workload of web developers.

By using a web application framework, web developers may concentrate on the user interface design and background control mechanism without worrying about the redundant and complex coding tricks. With helps from the kinds of features provided by a web application framework, web developers are released from error-prone steps of configuration and adjustment of the developing environment and deployment, which were a heavy workload for site administrators who have to work with traditional Web 1.0 sites.

Furthermore, the design patterns and software architectures hidden in the nature of a web application framework encourage web developers to think and organize their codes with systematic senses. By design and implementation on an industrial level of thoughts, the web application development benefits not only the maintenance work of created web sites in the future, but also scalabilities and internal exchange of information across the whole Internet.

The usage of Ajax techniques enhances the online user interface a lot. Ajax provides possibilities to web UI designers to increase user experience on originally static web pages which stimulates users to participate in the online interactions. The asynchronous data transferring decreases the size of a single data flow every time between the server and the client, which decreases the requirement of network capacity and enhances the client user experience.

On the other side, bringing a web application framework and Ajax techniques into the web application development process will also, naturally, introduce new challenges to web development. Every web application framework needs web developers at some time to become familiar with before starting work with it. Some web application frameworks lack enough flexibility to be adjusted for specific site design requirements. Some web application frameworks are not mature enough to remove all defects and bugs of built-in functions.

As a combination of several different web technologies, Ajax techniques would be treated as a bigger challenge by web developers to grasp and use with ease. The executable scripts written in JavaScript are difficult for developers to debug if some errors which are not obvious happen. This is especially true when the web application has been deployed into a real environment. It has been observed that even experienced web developers who work with Ajax for years would use a lot of time to trace an error occurring in Ajax code stacks.

The practice of combining a web application framework and Ajax together brought some conflicts when the implementation started. Some web application frameworks are simply not optimized for an Ajax-based site structure, which is used for transferring pure data without style information between the server and client. These web application frameworks will put web developers into trouble for rearrangement of data flow in the structural layers of the web application framework.

2nd Question: How to design a web user interface to encourage users to generate additional content on the site?

By designing an online user interface in practice of the experimental part of the study, some principles of web user interface design had been found out.

One good practice is to concentrate on functionalities and content. A user of online applications would lose their interests in a user interface if there was not enough content to attract their attentions. An appealing web page should provide rich useful contents to visitors instead of colorful but empty text blocks or curves.

Another good practice is to keep the UI pattern being concise and easy to grasp. Users of online applications prefer to spend less time on learning the operations needed for using the web elements in the user interface. Complex layouts or complicated operation steps would scare away users rather than encourage their active participation.

The usage of an incentive mechanism in the implementation practice of the experimental part of the study has been observed to be useful for increasing the level of user participation. The design of a user ranking system in this study might be too simple to be migrated to another web application, but the idea behind it is valuable for encouraging new user to contribute more content to user-centric web sites.

3rd Question: What kinds of technologies are beneficial to accelerate web site implementation?

During the whole process of the research, a lot of concepts and technologies related to web development were mentioned and discussed in this study. It has been proved that the more concepts and technologies were acknowledged by the developers, the better design and implementation of the web site would be generated, which means greater scalabilities and compatibilities across multiple client browsing environments.

For the server-side structures and construction, some concepts and principles, for instance, MVC [See Section 2.1.1], which presents MTV [See Section 3.4.1] also, ORM [See Section 2.3.4], DRY [See Section 2.1.2], Convention over Configuration [See Section 2.1.3], and REST [See Section 2.1.4] were emphasized and discussed during the introduction of web application frameworks.

For the client-side structures and construction, some concepts and principles, for instance, Ajax [See Section 2.2.1], JavaScript library [See Section 2.2.2], YAML [See Section 2.4.3], and JSON [See Section 3.4.3] were emphasized and discussed during the description of research and implementation processes.

In the real practice of the implementation work, the API provided by the Google App Engine platform [See Section 2.4], functions provided by jQuery JavaScript library [See Section 3.4.2], and language grammars of the Python language were also necessary for web developers to become familiar with.

5.2. Benchmarks

The benchmark section tries to compare the re-engineered web site with other styles of web applications which provide different functionalities in order to find out the user group differences.

In the second part, the web site was compared with some famous web sites on the Internet. It would be interesting and valuable to compare the results of the experimental part of the study with common social network applications on the Internet. The comparison helps to find out the advantages and disadvantages of the design and implementation, and provides a reference for possible future work for bug fixing or adding new features.

5.2.1. Comparison with other web services

Here the differences between the web application in the experimental part of this study and some widely deployed web services are presented. The discussion concentrates on the user interactive method, UI design and related technical issues.

Web 1.0 site

From a technical point of view, all of the existing web sites which consist mainly of static web pages are Web 1.0 sites. The majority of the documentation pages of these web sites are written purely with HTML, thus are not able to interactive with behaviors of users.

Those web sites implemented purely by HTML and CSS are good for documentation but lose the possibilities to attract users from an interactive UI, which makes the sites not suitable for presenting personalized content or any context-sensible content.

The requirement of the user participation on the UI and background incentive mechanism prevents the usage of Web 1.0 concepts in the experimental part of the study.

Electronic mailing list

An electronic mailing list is a special usage of email that allows for widespread distribution of information to many Internet users. It is similar to a traditional mailing list which is a list of names and addresses used by an individual or an organization to send material to multiple recipients, but with an additional reflector, which is a single e-mail address that, when designated as the recipient of a message, will send a copy of that message to all of the subscribers.

Electronic mailing lists are usually fully or partially automated through the use of special mailing list software and a reflector address that are set up on a server capable of receiving email. Incoming messages sent to the reflector address are processed by the software and, depending on their content, are acted upon internally (in the case of messages containing commands directed at the software itself) or are distributed to all e-mail addresses subscribed to the mailing list. Depending on the software, additional addresses may be set up for the purpose of sending commands.

The characteristics of an electronic mailing list service decide whether one user of the service will receive almost all messages sent by any of the users included in the mailing list without exception no matter if it is useful to him or not. This creates quite a significant amount of junk mail inside the inbox of the email account if the user is not actively cleaning his mailbox frequently every day. Additionally, it is possible also that some of the useful information will get lost during the spreading, which is inevitable as a volume of junk mail is ever increasing.

These characteristics of the electronic mailing list decide it is not suitable to be used to collect or classify information based on some set of rules, nor to be used to stimulate its users to interact on the service framework.

Internet forum

An Internet forum is an online discussion site mainly focusing on managing user-generated content. It is originated as the modern equivalent of a traditional bulletin board. People participating in an Internet forum may cultivate social bonds and interest groups for a topic made from the discussions.

An Internet forum requires the member to visit the website and check regularly for new posts. The strict user identity checking process and the difficulty to find out specific topics decreases the service usability of Internet forums. Moreover, the complex deployment and configuration are other negative factors of deploying an Internet forum widely.

Wiki

A Wiki is a website that allows the easy creation and editing of any number of interlinked web pages via a web browser using a simplified markup language or a WYSIWYG (What you see is what you get) text editor. Unlike conventional forums, Wikis typically allow all users to edit all content, including each other's messages. Wikis also allow the creation of other content outside the talk pages.

The syntaxes of Wikis are quite different across kinds of Wiki engines available on the Internet. User needs time to learn the syntax of one wiki for contribute contents to it.

Wikis commonly do not provide any incentive mechanism to encourage users to contribute, because it is not possible to identify a user when allowing a user to be active anonymously without asking for user registration.

Blog

A blog is a type of web sites, usually maintained by an individual with regular entries of commentary, descriptions of events, or other material such as graphics or video. Entries are commonly displayed in reverse-chronological order.

Blogging is good for personal expression but not for group discussion. A blog is commonly authorized by a single user instead of a group of attendees. A new comer to a group blog would need authorization from the administrator of the blog to join, which may be time-consuming.

Instant messaging

Instant messaging is a form of real-time direct text-based communication between two or more people using personal computers or other devices, along with shared software clients. The user's text is conveyed over a network, such as the Internet. More advanced instant messaging software clients also allow enhanced modes of communication, such as live voice or video calling.

Although the group chat in an instant messaging application is available, it is not suitable for information sharing activity also. The information fades away quickly as time goes by. And it is difficult to search with keywords which are not obvious in a chat record with a long time span.

Social network service

A social network service focuses on building and reflecting on social networks or social relations among people, for example, who share interests and activities. A social network service essentially consists of a representation of each user, his social links, and a variety of additional services. Most social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging.

A web service can be called a social network service when it is designed with user-centered concepts. User authorization and identity checking for personalized service should be one compulsory characteristic of such a web service. Considering of this, the workout of the experimental part of the study could be considered as a social network service also.

5.2.2. Comparison with other social network services

Here we pick up several famous Internet services and compare the functionalities they have with the experimental part of the study. By comparing, we will know what unique features that the experimental part of the study has, and also the competence possibility.

Facebook

Facebook is a social networking website that is operated and privately owned by Facebook, Inc. Since September 2006, anyone over the age of 13 with a valid e-mail address can become a Facebook user. Facebook's target audience is more for an adult demographic than a youth demographic. Users can add friends and send them messages, and update their personal profiles to notify friends about themselves. Additionally, users can join networks organized by workplace, school, or college. The web site's name stems from the colloquial name of books given to students at the start of the academic year by university administrations in the US with the intention of helping students to get to know each other better. (Facebook, 2010)

Facebook aims to be the one-stop social center which provides comprehensive functionalities to its users. The public accessible API helps it generating thousands of handy applications based on its huge user group. However, the rich features provided by Facebook sometimes may confuse new coming users and influence user experience in some degree.

Digg

Digg is a social news website made for people to discover and share content from anywhere on the Internet, by submitting links and stories, and voting and commenting on the submitted links and stories. Voting stories up and down is the site's cornerstone function, respectively called digging and burying. Many stories get submitted every day, but only the most *dugg* stories appear on the front page. Digg's popularity has prompted the creation of other social networking sites with story submission and voting systems. (Digg Inc., 2010)

The experimental part of the study borrowed the voting system idea of Digg. It has been proved that this voting system is suitable for group users to provide feedback to a single topic. On the other side, Digg allows users to submit comments for feedback. But the experimental part of the study does not because it needs more workload on the interface design, and would make the implementation being more complex.

Twitter

Twitter is a social networking and micro-blogging service that enables its users to send and read messages known as tweets. Tweets are text-based posts of up to 140 characters displayed on the author's profile page and delivered to the author's subscribers who are known as followers. Senders can restrict delivery to those in their circle of friends or, by default, allow open access. (Twitter, 2010)

The experimental part of the study borrowed the idea of 140 character limitation of a submitted test message from Twitter. This limitation of Twitter was come from the limitation of a short message service of mobile industry, which helps Twitter to be used on mobile platforms. The limitation in the experimental part of the study made the re-engineered web service possible also to be extended to use on mobile devices in the future.

Reddit

Reddit is a social news website owned by Conde Nast Digital on which users can post links to content on the Internet. Other users may then vote the posted links up or down, causing them to become more or less prominent on the Reddit home page. The site has discussion areas in which users may discuss the posted links and vote for or against others' comments. When there are enough votes against a given comment, it will not be displayed by default, although a reader can display it through a link or preference. Users who submit articles which other users like and subsequently “vote up” receive “karma” points as a reward for submitting articles those other users consider interesting. (Conde Nast Digital, 2010)

The experimental part of the study borrowed the idea of giving bonus points as a reward for submitting new contents to the web site from Reddit. The bonus points are used to give ranking to users, which were used to encourage users to contribute more to the site.

6. Conclusion

One of the main purposes of this study was to find out the benefits which a web application framework and Ajax techniques may bring to online service development. The business possibility of the re-engineered web application was presented here. At the end of the chapter, the challenges of the study are discussed and the research process will be evaluated. The possible future work is also discussed.

6.1. Research Evaluation

SWOT Analysis is a strategic planning method used to evaluate the Strengths, Weaknesses, Opportunities, and Threats involved in a project or in a business venture. It involves specifying the objective of the business venture or project and identifying the internal and external factors that are favorable and unfavorable to achieving that objective. SWOT is an ideal way to measure business demand and indentify the possibility of making profits from the market. By using SWOT, the business opportunity is easier to be understood. Here is a paragraph which describes the SWOT analysis results.

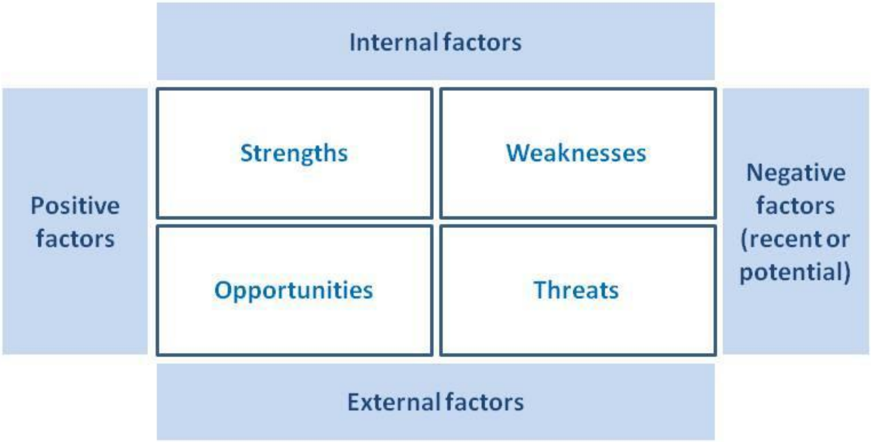


Figure 6.1: The SWOT model (CIPD, 2008)

Here the results of the experimental part of the study will be analyzed for the business prospects by using the SWOT analysis methods. The purpose is to find out the potentials to migrate the application structure to serve more users and to achieve commercial benefits if possible. The listed analysis results are based on the SWOT analysis template. (Chapman, 1995-2010)

Strengths

Advantages of proposition: Ajax-styled user interface and user incentive mechanism.

Capabilities: Concise code structure. Ease of maintenance.

Competitive advantages: Free distribution of source code. Platform of Cloud computing.

Unique selling points: Google infrastructure. Free distribution of source code.

Resources, assets, people: Single developer product. Free development tools.

Experience, knowledge, data: DVCS in use. Rich documentation from the Internet.

Financial reserves, likely returns: Possible online advertisements.

Marketing – reach, distribution, awareness: Available online now.

Innovative aspects: Concise user interface. Coding less with the help from development tools.

Location and geographical: Accessible online from anywhere of the world.

Price, value, quality: Free access and free usage. Ease for maintenance.

Accreditations, qualifications, certifications: Similar online sites proved successful.

Processes, systems, IT, communications: Everything is online. Ease for maintenance.

Cultural, attitudinal, behavioral: Ease for localization. Ease for extending user amount.

Management cover, succession: Single person product. Ease of knowledge transfer.

Weaknesses

Disadvantages of proposition: Simple functionalities. Some functionality is missing.

Gaps in capabilities: Lots of work to do to add new features.

Lack of competitive strength: Some functionality is missing.

Reputation, presence and reach: No one has noticed its existence so far.

Financials: No budget available at all. No income would be available in the near future.

Own known vulnerabilities: Some functionality is missing.

Timescales, deadlines and pressures: Single person product relies on a personal schedule.

Cash flow, start-up cash-drain: No cash at all.

Continuity, supply chain robustness: Depends on the interests from users.

Effects on core activities, distraction: Browser compatibility would be a trouble.

Reliability of data, plan predictability: online users lose interests faster.

Morale, commitment, leadership: Single person product.

Accreditations, etc: Too many failure examples of Web 2.0 sites.

Processes and systems, etc: Dependence on accessibility to Google servers.

Management cover, succession: Knowledge was not shared in team. Too much knowledge to learn for a newbie.

Opportunities

Market developments: More functionality could be added.

Competitors' vulnerabilities: Higher budgets. More payment for human resources.

Industry or lifestyle trends: User-centered concepts will collect user interests.

Technology development and innovation: Users enjoy responding to new ideas.

Global influences: Accessible from anywhere in the world.

New markets, vertical, horizontal: More functionality to ease life of oversea users.

Geographical, export, import: Accessible from anywhere in the world.

New unique selling points: Google's cloud computing platform.

Business and product development: Possible to adhere to Google Adsence.

Information and research: Possible to add more functionality ready on App Engine.

Partnerships, agencies, distribution: Combined with Google service. Google Account is the only requirement for new user registration.

Volumes, production, economies: Free of charge for hosting and code upgrading.

Threats

Political effects: Google services may not be accessible in specific geographic locations.

Legislative effects: Content filter might be necessary.

Environmental effects: None.

IT developments: Experienced web developer is necessary for maintenance.

Competitor intentions –various: Social network services may offer the same functionality.

Market demand: UI usability from mobile devices.

New technologies, services, ideas: Maintenance work might be a lot to catch up with changes.

Vital contracts and partners: Google App Engine might be not free any more on some day.

Sustaining internal capabilities: Another experienced web developer is needed.

Obstacles faced: Users refuse to try something new.

Insurmountable weaknesses: lack of support for mobile SMS access.

Loss of key staff: If there has no another experienced web developer coming.

Sustainable financial backing: Google Adsence might not work.

6.2. Challenge of the study

During the preparation period of this research, it took a very long time for grasping the necessary concepts and knowledge for exerting development tools well enough. Since web development technologies evolve rapidly, web developers have to learn new concepts and coding practices fast, which is relatively difficult for an amateur who can only do this study work in spare time.

The web application framework is a new concept for web development. After Rails was published in the year 2004, the power of the web application frameworks was recognized by the web industry little by little. As a result, more and more general-purposed web application frameworks appeared based on different kinds of languages. Every web application framework has its own advantages and drawbacks when compared to others. It is hard to tell whether a web application framework is better to be used on one occasion than on another without getting familiar with and accumulating enough practical experiences on both of them. And the evolution of web application frameworks themselves will never stop. The Rails framework is going to release its third major version, Rails 3.0, and the Django framework has released the schedule for its next minor version update, Django 1.2.

The rapid upgrading of the structures influences the stability of the APIs of these frameworks. Some functions disappeared or were renamed after a minor update of the framework. Some ready-made web applications did not work anymore when the dependent web application framework was upgraded without careful detection of possible conflicts. Even the tutorials would turn out to be useless when a new minor version upgrade appeared.

The initial idea of this research was generated before Google released its App Engine platform. It was a big surprise to find out there was such a free yet powerful service provided by a commercial company to be a practical field for web development. To start working with App Engine, necessary knowledge of Python language has to be learned. The documentation of App Engine API was another task for web developers to study in detail. These study processes were done together with the endless upgrading of the App Engine SDK from the core team of the App Engine platform.

6.3. Possible future work

As the experimental part of the study concentrated on identifying the web development changes brought by the web application framework and Ajax techniques with moderate programming load, some advanced features which are commonly available in social network services are absent from the re-engineered web site. This section listed some of these features in order to provide a reference for possible future development of the site.

Content tagging

Every piece of text information contributed by a user to the re-engineered web site of the study should be classified by one or several labels, which are commonly called tags in Web 2.0 sites. In online services, a tag is a non-hierarchical keyword or term assigned to a piece of information, for instance an internet bookmark, digital image, or computer file. These kinds of keywords help to describe a set of information and allow it to be found easily by browsing or searching. Tagging is an important feature of many Web 2.0 services. (Yahoo! Inc., 2010)

Some online services provide automatic tagging functionality to the information generated by user which helps users to create tags easily. Furthermore, a tag cloud showing on the web site would help user to quickly recognize the hot topics highlighted by the public.

RSS feeding

RSS (Really Simple Syndication) is a family of web feed formats used to publish frequently updated works, for instance blog entries, news, audio, and video, in a standardized format. An RSS feed is a document consisting of summarized text, publishing dates, author's information, etc. (Harvard Law, 2003) Adding RSS feeding ability to the experimental part of the study would provide an important access for user to get to know the latest changes happened on the site.

IM integration

IM (instant messaging) is a form of real-time text-based chat between two or more people shared software clients across network. IM software applications are widely used nowadays in desktop computing environments and even mobile devices. Web applications hosted on Google App Engine platform can send and receive instant messages to and from users of Google Talk service. (Google, 2010) By integrating IM ability into the re-engineered web site, users who are using Google Talk applications on a computer desktop would receive the latest changes made to the web site immediately without the need to go to the online address of the site. Furthermore, it is also a convenient way for users to contribute new content to the web site by using the IM client application directly without browsing the online web site.

UI optimization for mobile device

Recently, there are more and more users browsing online services using mobile devices. Mobile browsers are commonly optimized to display web content effectively on small screens of portable devices. The major difference between the screen sizes of a desktop PC and a mobile handset requires the UI design being reorganized to satisfy both situations.

In addition to the change of layout, the behaviors of dynamic page elements and background Ajax techniques are also necessary to be revised in order to decrease the lag of the mobile browser as well as the power consumption of mobile device.

Bibliography

Ambler, S. W. (2010). *Mapping Objects to Relational Databases: O/R Mapping In Detail.* Retrieved April 30, 2010, from www.agiledata.org: Techniques for Successful Evolutionary / Agile Database Development: http://www.agiledata.org/essays/mappingObjects.html

Bennett, J. (2006, July 2). *Django and Ajax.* Retrieved April 26, 2010, from b-list.org: http://www.b-list.org/weblog/2006/jul/02/django-and-ajax/

BuiltWith. (2010). *JavaScript Usage Statistics.* Retrieved May 1, 2010, from Web and Internet Technology Usage Statistics: http://trends.builtwith.com/javascript

Chapman, A. (1995-2010). *SWOT Analysis.* Retrieved May 3, 2010, from businessballs.com: http://www.businessballs.com/swotanalysisfreetemplate.htm

CIPD. (2008, March). *SWOT analysis.* Retrieved May 3, 2010, from cipd.co.uk: http://www.cipd.co.uk/subjects/corpstrtgy/general/swot-analysis.htm

Conde Nast Digital. (2010). *reddit.com: what's now online!* Retrieved May 3, 2010, from reddit.com: http://www.reddit.com/

CSSA-Espoo. (2009). *About us.* Retrieved May 1, 2010, from CSSA-Espoo ry: http://sites.google.com/site/cssa-espoo/Home

CSSA-Espoo. (2008). *Chinese-list -- Mail list for Chinese in Aalto.* Retrieved May 1, 2010, from CSSA-Espoo mailing list: http://list.ayy.fi/mailman/listinfo/chinese-list/

CSSA-Espoo. (2004). *Chinese-list Info Page*. Retrieved April 25, 2010, from http://list.ayy.fi/mailman/listinfo/chinese-list/

CSSA-Espoo. (2008). *Cssa-Espoo - outdated version*. Retrieved April 18, 2010, from http://cssa.ayy.fi/backgrd.old/

CSSA-Espoo. (2008). *CSSA-Espoo ry forum Index page*. Retrieved April 25, 2010, from http://cssa.ayy.fi/phpBB2/index.php

Digg Inc. (2010). *Digg - The Latest News Headlines, Videos and Images*. Retrieved May 3, 2010, from digg.com: http://digg.com/

Django Software Foundation. (2005-2010). *Django | The Web framework for perfectionists with deadlines*. Retrieved April 18, 2010, from http://www.djangoproject.com/

Django Software Foundation. (2005-2010). *FAQ: Django...How come you don't use the standard names?* Retrieved April 30, 2010, from Django documentation: http://docs.djangoproject.com/en/dev/faq/general/

Django Software Foundation. (2005-2010). *The Django template language.* Retrieved April 30, 2010, from Django documentation: http://docs.djangoproject.com/en/dev/topics/templates/

DocForge. (2010, April 18). *Web application framework.* Retrieved April 26, 2010, from DocForge: http://docforge.com/wiki/Web_application_framework

Evans, C. C. (2004). *The Official YAML Web Site*. Retrieved April 18, 2010, from http://www.yaml.org/

Facebook. (2010). *Welcome to Facebook*. Retrieved May 3, 2010, from Facebook: http://www.facebook.com/

Fielding, R. T., & Taylor, R. N. (2002). Principled Design of the Modern Web Architecture. In R. T. Fielding, & R. N. Taylor, *Principled Design of the Modern Web Architecture* (pp. 115-150). ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002.

Garrett, J. J. (2005, February 18). *Ajax: A New Approach to Web Applications.* Retrieved April 26, 2010, from adaptive path: http://www.adaptivepath.com/ideas/essays/archives/000385.php

Google. (2010). *Developer's Guide - Google AJAX APIs.* Retrieved May 5, 2010, from Google Code: http://code.google.com/apis/ajax/documentation/

Google. (2010). *Google Accounts Python API Overview - Google App Engine.* Retrieved April 30, 2010, from Google Code: http://code.google.com/appengine/docs/python/users/overview.html

Google. (2010). *Google AJAX Libraries API.* Retrieved May 3, 2010, from Google Code: http://code.google.com/apis/ajaxlibs/

Google. (2006). *Google Analytics | Official Website*. Retrieved April 18, 2010, from http://www.google.com/analytics/

Google. (2008). *Google App Engine*. Retrieved April 18, 2010, from Google Code: http://code.google.com/appengine/

Google. (2010). *Project Hosting on Google Code*. Retrieved May 3, 2010, from Google Code: http://code.google.com/projecthosting/

Google. (2010). *Quotas - Google App Engine.* Retrieved May 1, 2010, from Google Code: http://code.google.com/appengine/docs/quotas.html

Google. (2008). *The webapp Framework - Google App Engine*. Retrieved April 18, 2010, from Google Code: http://code.google.com/appengine/docs/python/tools/webapp/

Google. (2010). *Using Templates - Google App Engine.* Retrieved April 30, 2010, from Google Code: http://code.google.com/appengine/docs/python/gettingstarted/templates.html

Google. (2010). *XMPP Python API - Google App Engine.* Retrieved May 3, 2010, from Google Code: http://code.google.com/appengine/docs/python/xmpp/

Harvard Law. (2003, July 15). *RSS 2.0 Specification (RSS 2.0 at Harvard Law).* Retrieved May 3, 2010, from Berkman Center For Internet & Society at Harvard University: http://cyber.law.harvard.edu/rss/rss.html

Hunt, A. (2010, 1 13). *Dont Repeat Yourself*. Retrieved April 18, 2010, from c2.com: http://c2.com/cgi/wiki?DontRepeatYourself

jQuery. (2010). *Browser Compatibility.* Retrieved April 26, 2010, from jQuery JavaScript Library: http://docs.jquery.com/Browser_Compatibility

jQuery. (2006). *jQuery: The Write Less, Do More, JavaScript Library*. Retrieved April 18, 2010, from http://jquery.com/

JSON. (1999). *Introducing JSON*. Retrieved April 25, 2010, from http://www.json.org/

Mazzetti, P., Nativi, S., & Bigagli, L. (2008). Integration of REST style and AJAX technologies to build Web applications; an example of framework for Location-Based-Services. *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on* (pp. 1 - 6). Univ. of Florence Prato, Florence: IEEE Conferences .

Mell, P., & Grance, T. (2009, October 7). *The NIST Definition of Cloud Computing.* Retrieved April 18, 2010, from NIST.gov - Computer Security Division - Computer Security Resource Center: http://csrc.nist.gov/groups/SNS/cloud-computing/index.html

Miller, J. (2009, February). *Design For Convention Over Configuration*. (Microsoft) Retrieved April 18, 2010, from MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/dd419655.aspx

Prototype. (2006). *Prototype JavaScript Framework: ease development of dynamic web applications*. Retrieved April 25, 2010, from http://www.prototypejs.org/

Python Software Foundation. (1990-2010). *Python Programming Language -- Official Website*. Retrieved April 18, 2010, from http://www.python.org/

Rails. (2004-2010). *Ruby on Rails*. Retrieved April 18, 2010, from http://rubyonrails.org/

Selfa, D., Carrillo, M., & Del Rocio Boone, M. (2006). A Database and Web Application Based on MVC Architecture. *Electronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference on* (pp. 48 - 48). Benemerita Universidad Autónoma de Puebla, Mexico: IEEE Conferences.

Shan, T. C., & Hua, W. W. (2006). Taxonomy of Java Web Application Frameworks. *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on* (pp. 378 - 385). Shanghai: IEEE Conferences.

SpringSource. (2004-2010). *SpringSource.org*. Retrieved April 18, 2010, from http://www.springsource.org/

Thomas, D. (2003, March 10). *Orthogonality and the DRY Principle*. (i. b. Venners, Editor) Retrieved April 18, 2010, from artima developer: http://www.artima.com/intv/dry.html

Twitter. (2010). *Twitter - Discover what’s happening right now, anywhere in the world*. Retrieved May 3, 2010, from twitter.com: http://twitter.com/

W3C. (2010). *Cascading Style Sheets.* Retrieved May 3, 2010, from W3C.org: http://www.w3.org/Style/CSS/

W3C. (2010). *World Wide Web Consortium*. Retrieved May 3, 2010, from http://www.w3.org/

Yahoo! Inc. (2010). *Help: Tags*. Retrieved May 3, 2010, from Flickr: http://www.flickr.com/help/tags/

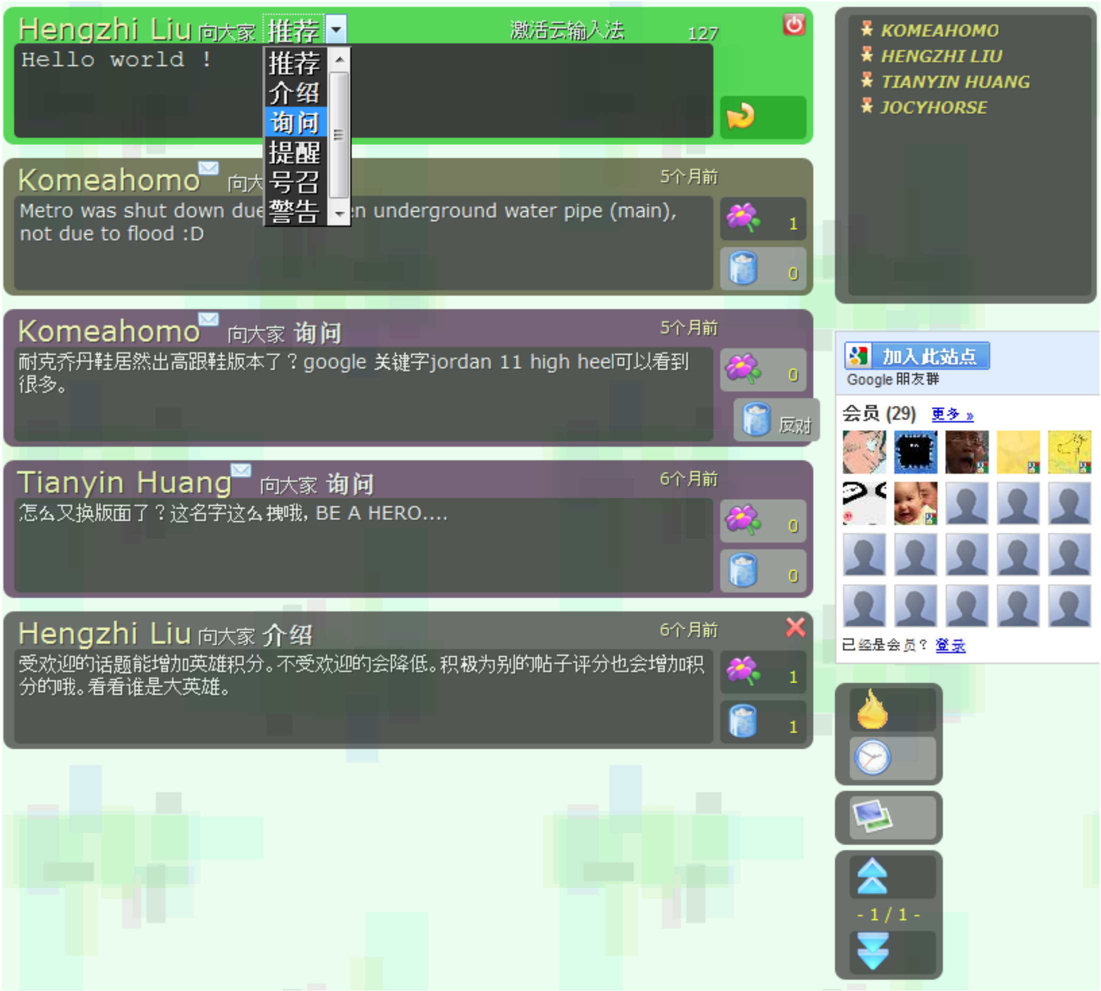
Appendices

Appendix A: Screenshot of the Re-engineered User Interface

This screenshot was captured on May 5, 2010 from the URL below. The browser application is Mozilla Firefox version 3.5.7 running on Microsoft Windows Vista operating system.

http://cssa.appspot.com/

The language used in the user interface is Chinese. [See Section 3.2.2] The explanation of corresponding page elements is available in Section 4.1.1.



Hello world !

Metro was shut down due ... n underground water pipe (main), not due to flood :D

耐克乔丹鞋居然出高跟鞋版本了？google 关键字jordan 11 high heel可以看到很多。

怎么又换版面了？这名字这么拽哦，BE A HERO....

受欢迎的话题能增加英雄积分。不受欢迎的会降低。积极为别的帖子评分也会增加积分的哦。看看谁是大英雄。

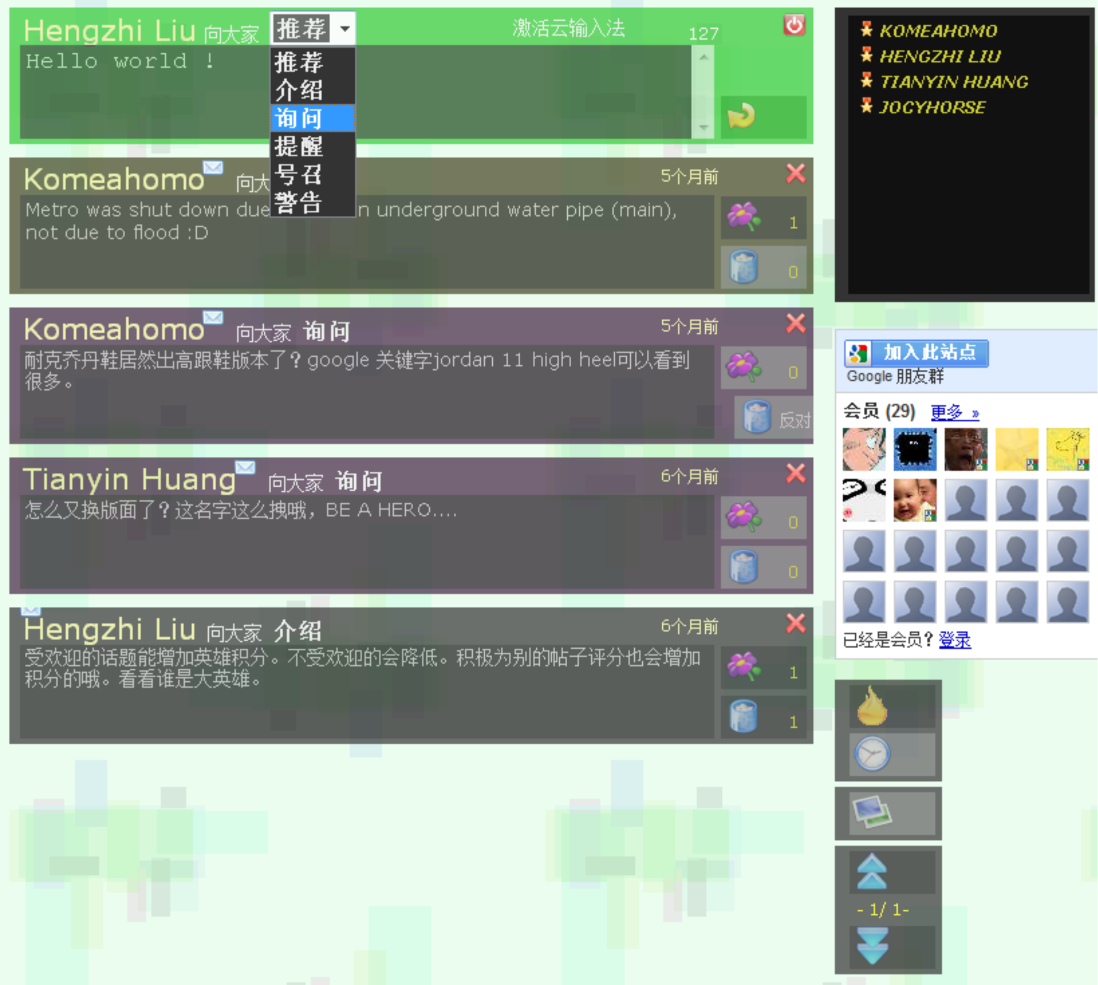
Figure 0.1: Screenshot of re-engineered web site on Mozilla Firefox 3.5.7

Appendix B: Screenshot of the Re-engineered User Interface (by IE)

This screenshot was captured on May 5, 2010 from the URL below. The browser application is Microsoft Internet Explorer version 7.0 running on Microsoft Windows Vista operating system. This screenshot explains the compatibility issue existing in different browser applications. [See Section 3.5.1]

http://cssa.appspot.com/

The language used in the user interface is Chinese. [See Section 3.2.2] The explanation of corresponding page elements is available in Section 4.1.1.



Hello world !

Komeahomo 向大 5个月前
Metro was shut down due n underground water pipe (main), not due to flood :D
1
0

Komeahomo 向大家 询问 5个月前
耐克乔丹鞋居然出高跟鞋版本了？google 关键字jordan 11 high heel可以看到很多。
0
反对

Tianyin Huang 向大家 询问 6个月前
怎么又换版面了？这名字这么拽哦，BE A HERO....
0
0

Hengzhi Liu 向大家 介绍 6个月前
受欢迎的话题能增加英雄积分。不受欢迎的会降低。积极为别的帖子评分也会增加积分的哦。看看谁是大英雄。
1
1

Figure 0.2: Screenshot of re-engineered web site on Microsoft Internet Explorer 7

Appendix C: Source Code of Experimental Part

The source code of the experimental part of the study has been uploaded and managed by the online version control service provided freely by Google. (Google, 2010) The code resources and all pictures used on the user interface are located in the URL below. The whole research work is released under GNU GPL v2.

http://code.google.com/p/meixi/source/browse/#hg/ishare

Here in the appendix all source code files are not listed. Only the important source files are listed for providing reference to the research evidence. Other unlisted source code files can be accessed from the URL above.

File: app.yaml

```
application: cssa
version: 1
runtime: python
api_version: 1

handlers:
- url: /style
  static_dir: style

- url: /icons
  static_dir: icons

- url: /favicon.ico
  static_files: icons/favicon.ico
  upload: icons/favicon.ico

- url: /log(in|out)
  script: auth.py

- url: /hero
  script: heroview.py

- url: /(pagecount|list|read/.+)
  script: controller.py

- url: /(create|(pos|neg|delete)/.+)
  script: controller.py
  login: required

- url: /.*
  script: mainview.py
```

File: auth.py

```
import django_ver_guard

from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from model import Hero

class Login(webapp.RequestHandler):
  def get(self):
    user = users.get_current_user()
    if user:
      Hero().getOrCreate(user)
      self.redirect('/')
    else:
      self.redirect(users.create_login_url(self.request.uri))

class Logout(webapp.RequestHandler):
  def get(self):
    if users.get_current_user():
      self.redirect(users.create_logout_url(self.request.uri))
    else:
      self.redirect('/')

application = webapp.WSGIApplication([('/login', Login),
                                       ('/logout', Logout)],
                                      debug=True)

def main():
  run_wsgi_app(application)

if __name__ == "__main__":
  main()
```

File: controller.py

```
import django_ver_guard

from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from django.utils import simplejson
from model import *
import helper as hp

class PageCount(webapp.RequestHandler):
  def get(self):
    count = Story.all().count()
    total = 1 + (count - 1) / hp.PAGESIZE
    self.response.headers['Content-Type'] = 'application/json'
    self.response.out.write(simplejson.dumps(total))

class StoryList(webapp.RequestHandler):
  def get(self):
    order = self.request.get('order')
    if not order: # prevent empty string
      order = 'hot'

    page = self.request.get('page')
    if not page: # prevent empty string
      page = 1
    else:
      page = int(page)

    offset = hp.PAGESIZE * (page - 1)
    if order == 'time':
      stories = Story.all().order('-created')\
                          .fetch(hp.PAGESIZE, offset)
    else:  # should be 'hot' then
      stories = Story.all().order('-score')\
                          .order('-created')\
                          .fetch(hp.PAGESIZE, offset)
    keys = [str(x.key()) for x in stories]

    self.response.headers['Content-Type'] = 'application/json'
    self.response.out.write(simplejson.dumps(keys))

class StoryController(webapp.RequestHandler):
  def get(self):
    key = self.request.path.strip('/').split('/')[1]
    try:
      story = db.get(db.Key(key))
    except db.Error:
      self.redirect('/')
      return

    place = self.request.get('place')
    if not place:
      self.redirect('/')
      return

    block = dict(key = key,
                 author = story.author.name(),
                 time = story.pastTime(),
                 topic = story.topic,
                 pushword = story.pushword(),
                 bgcolor = story.bgcolor(),
                 count_pos = Vote().countBy(story),
                 count_neg = Vote().countBy(story, False),
                 email = story.author.user.email(),
                 place = place)

    user = users.get_current_user()
    if user:
      if user == story.author.user:
        opt = [True, True, True]
      else:
        vote = Vote().getBy(story, user)
        if vote:
          opt = [False, vote.pos, not vote.pos]
        else:
          opt = [False, False, False]
    else: # not a user
      opt = [False, True, True]

    block.update(deletable = opt[0],
                 lock_pos = opt[1],
                 lock_neg = opt[2])

    self.response.headers['Content-Type'] = 'application/json'
    self.response.out.write(simplejson.dumps(block))

  def post(self):
    user = users.get_current_user()
    # no cgi.escape needed as jQuery did this
    topic = self.request.get('topic')
    topic = ' '.join([x.strip() for x in topic.splitlines()])
    word_id = self.request.get('word_id')

    if user and topic and word_id:
      Story().create(topic, user, word_id)

  def put(self):
    pos, key = self.request.path.strip('/').split('/')
    try:
      story = Story.get(db.Key(key))
    except db.Error:
      self.redirect('/')
      return

    user = users.get_current_user()
    if user and story and user != story.author.user:
      # check if the user has already given a value
      vote = Vote().getBy(story, user)
      if not vote:
        Vote().create(story, user, (pos == 'pos'))
      elif vote.pos != (pos == 'pos'):
        vote.update()

  def delete(self):
    key = self.request.path.strip('/').split('/')[1]
    try:
      story = Story.get(db.Key(key))
    except db.Error:
      self.redirect('/')
      return

    user = users.get_current_user()
    if user and story and user == story.author.user:
      Vote().removeBy(story) # remove its values
      story.remove()

application = webapp.WSGIApplication([('/pagecount', PageCount),
                                      ('/list', StoryList),
                                      ('/create', StoryController),
                                      ('/read/.+', StoryController),
                                      ('/pos/.+', StoryController),
                                      ('/neg/.+', StoryController),
                                      ('/delete/.+', StoryController)],
                                     debug=True)

def main():
  run_wsgi_app(application)

if __name__ == "__main__":
  main()
```

File: django_ver_guard.py

```
# There is a bug in simplejson of Django 0.96 to process utf-8
# so that latest version of Django is needed.
import os

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
from google.appengine.dist import use_library
use_library('django', '1.1')
```

File: Helper.py

```python
# coding=utf-8
from datetime import datetime

CSS_PATH = (
'/style/main.css',
)
JS_PATH = (
# IE does not like google jsapi
# 'http://www.google.com/jsapi',
# '/style/jquery_loader.js',
'http://jqueryjs.googlecode.com/files/jquery-1.3.2.min.js',
'/style/jquery_restful_patch.js',
'/style/textlimit.js',
'/style/main.js'
)
BUTTON_DATA = {
'hot' : '热评',
'time': '最新',
'wall': '壁纸',
'prev': '上页',
'next': '下页'
}
PUSHWORD_PREFIX = '向大家'
PUSHWORD = {
'1': ('推荐', '#442'),
'2': ('询问', '#424'),
'3': ('介绍', '#333'),
'4': ('号召', '#244'),
'5': ('提醒', '#422'),
'6': ('警告', '#411')
}
PAGESIZE = 7
HEROSIZE = 10
SPV = 50  # Story's Positive Value
SEV = 10  # Story's Even Value
SNV = -20  # Story's Negative Value
VV = 1   # Vote's Value

def pastTime(time):
  diff = datetime.utcnow() - time
  days, seconds = diff.days, diff.seconds
  if days:
    years = days / 365
    if years: return '%d年前' % years
    months = days / 30
    if months: return '%d个月前' % months
    weeks = days / 7
    if weeks: return '%d周前' % weeks
    if days == 1: return '昨天'
    if days == 2: return '前天'
    return '%d天前' % days
  else:
    hours = seconds / 3600
    if hours: return '%d小时前' % hours
    minutes = seconds / 60
    if minutes >= 30: return '半小时前'
    if minutes: return '%d分钟前' % minutes
    return '刚刚'

def alterName(user):
  name = user.nickname().split('@')[0]
  return ' '.join([x.strip('1234567890').capitalize() for x in name.split('.')])
```

File: heroview.py

```
import django_ver_guard

import os
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext.webapp import template
from model import Hero
import helper as hp

class HeroView(webapp.RequestHandler):
  def get(self):
    heroes = Hero.all().filter('score >', 0)\
                       .order('-score')\
                       .order('created')\
                       .fetch(hp.HEROSIZE)
    values = {
      'heroes': [x.name() for x in heroes],
    }
    path = os.path.join(os.path.dirname(__file__),
                        'templates/block_hero.html')
    self.response.out.write(template.render(path, values))

application = webapp.WSGIApplication([('/hero', HeroView)],
                                     debug=True)

def main():
  run_wsgi_app(application)

if __name__ == "__main__":
  main()
```

File: index.yaml

```
indexes:

- kind: Story
  properties:
  - name: score
    direction: desc
  - name: created
    direction: desc

- kind: Hero
  properties:
  - name: score
    direction: desc
  - name: created
```

File: mainview.py

```
import django_ver_guard

import os
from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext.webapp import template
import helper as hp

class MainView(webapp.RequestHandler):
  def get(self):
    user = users.get_current_user()
    username = hp.alterName(user) if user else None
    topblock = 'entry' if user else 'login'

    button = {}
    path = os.path.join(os.path.dirname(__file__),
                        'templates/block_button.html')
    for id, info in hp.BUTTON_DATA.items():
      values = {
        'id': id,
        'info': info,
      }
      button[id] = template.render(path, values)

    values = {
      'csspath': hp.CSS_PATH,
      'jspath': hp.JS_PATH,
      'pushword_prefix': hp.PUSHWORD_PREFIX,
      'pushword': hp.PUSHWORD,
      'block_top': 'block_%s.html' % topblock,
      'username': username,
      'total': range(hp.PAGESIZE),
      'button': button,
    }

    path = os.path.join(os.path.dirname(__file__), 'templates/main.html')
    self.response.out.write(template.render(path, values))

class Redirect(webapp.RequestHandler):
  def get(self):
    self.redirect('/')

application = webapp.WSGIApplication([('/', MainView),
                                      ('/.*', Redirect)],
                                     debug=True)

def main():
  run_wsgi_app(application)

if __name__ == "__main__":
  main()
```

File: model.py

```
from google.appengine.ext import db
import helper as hp

class Hero(db.Model):
  user = db.UserProperty()
  score = db.IntegerProperty(default=0)
  created = db.DateTimeProperty(auto_now_add=True)

  def getOrCreate(self, u):
    hero = self.all().filter('user = ', u).get()
    if not hero:
      hero = Hero(user=u)
      hero.put()
    return hero

  def name(self):
    return hp.alterName(self.user)

  def addToScore(self, v):
    self.score += v
    self.put()

class Story(db.Model):
  topic = db.StringProperty()
  author = db.ReferenceProperty(Hero)
  word_id = db.StringProperty()
  score = db.IntegerProperty(default=0)
  created = db.DateTimeProperty(auto_now_add=True)

  def create(self, t, u, w):
    h = Hero().getOrCreate(u)
    s = Story(topic=t, author=h, word_id=w)
    s.put()
    h.addToScore(hp.SEV)

  def addToScore(self, pos):
    self.changeScore(pos, 1)

  def changeScore(self, pos, point=2):
    self.author.addToScore(-self.value())
    self.score += (-point, point)[pos]
    self.put()
    self.author.addToScore(self.value())

  def remove(self):
    self.author.addToScore(-self.value())
    self.delete()

  def value(self):
    return (hp.SPV, (hp.SEV, hp.SNV)[self.score < 0])[self.score <= 0]

  def pastTime(self):
    return hp.pastTime(self.created)

  def pushword(self):
    return hp.PUSHWORD[self.word_id][0]

  def bgcolor(self):
    return hp.PUSHWORD[self.word_id][1]

class Vote(db.Model):
  story = db.ReferenceProperty(Story)
  author = db.ReferenceProperty(Hero)
  pos = db.BooleanProperty()
  created = db.DateTimeProperty(auto_now_add=True)

  def create(self, s, u, pos):
    h = Hero().getOrCreate(u)
    v = Vote(story=s, author=h, pos=pos)
    v.put()
    s.addToScore(pos)
    h.addToScore(hp.VV)

  def getBy(self, s, u):
    h = Hero().getOrCreate(u)
    return self.all().filter('story = ', s).filter('author = ', h).get()

  def countBy(self, s, pos=True):
    return self.all().filter('story = ', s).filter('pos = ', pos).count()

  def update(self):
    self.pos = not self.pos
    self.put()
    self.story.changeScore(self.pos)

  def removeBy(self, s):
    votes = self.all().filter('story = ', s)
    for v in votes:
      v.author.addToScore(-hp.VV)
      v.delete()
```

File: style/jquery_restful_patch.js

```
/* Extend jQuery with functions for PUT and DELETE requests. */

function _ajax_request(url, data, callback, type, method) {
    if (jQuery.isFunction(data)) {
        callback = data;
        data = {};
    }
    return jQuery.ajax({
        type: method,
        url: url,
        data: data,
        success: callback,
        dataType: type
        });
}

jQuery.extend({
    put: function(url, data, callback, type) {
        return _ajax_request(url, data, callback, type, 'PUT');
    },
    delete_: function(url, data, callback, type) {
        return _ajax_request(url, data, callback, type, 'DELETE');
    }
});
```

File: style/main.js

```
jQuery.fn.extend({
  renew: function() {
    return this.unbind("click").removeClass("finger lock");
  },
  awake: function() {
    return this.addClass("finger");
  },
  lock: function() {
    return this.addClass("lock").css("right", "5px");
  }
});

$(function() {
  var TEXT_LIMIT = 140;
  var ENTER_KEY = 13;

  $("textarea:first").textlimit("#entry_counter", TEXT_LIMIT);
  $("textarea:first").keypress( function(e){ return e.keyCode != ENTER_KEY; });

  $("a").attr("target", "_blank");
  $("#login").click( function(){ window.location="/login"; });
  $("#logout").click( function(){ window.location="/logout"; });

  $(".button").live("mouseover", function() {
    $(this).children(".value_info").hide();
    $(this).children(".button_info").show();
  }).live("mouseout", function() {
    $(this).children(".button_info").hide();
    $(this).children(".value_info").show();
  });

  $(".smartlink").live("click", function() {
    var link = $(this).text()
    var prefix = link.search(/ftp|https?/) === -1 ? "http://" : "";
    window.open(prefix + link);
  });

  $(".button.finger").live("mouseover", function() {
    $(this).stop().animate({right: -5}, 100);
  }).live("mouseout", function() {
    $(this).stop().animate({right: 5}, 100);
  });

  $("#wall").awake().click( function() { setWallpaper(); });
  $("#entry_im_button").awake().click( function() { loadCim(); });

  $("#submit").click( function() {
    var topic = $("textarea:first").val();
    var word_id = $("select:first").val();
    $("textarea:first").val("");
    $("#entry_counter").text(TEXT_LIMIT);
    $.post("/create",
           {"topic": topic, "word_id": word_id},
           function(){
             listBy("time", 1);
             updateHero();
           }
    );
  });

  $(".story_block").hide();
  setWallpaper();
  listBy();
});

function listBy(order, page) {
  if(!order && !page)
    updateHero();

  if (order)
    this.session_order = order;
  else if (this.session_order)
    order = this.session_order;
  else {
    order = "hot";
    this.session_order = order;
  }

  if (page)
    this.session_page = page;
  else if (this.session_page)
    page = this.session_page;
  else {
    page = 1;
    this.session_page = page;
  }

  $("#" + order).renew().lock();
  var the_other = (order === "hot") ? "time" : "hot";
  $("#" + the_other).renew().awake().click( function(){ listBy(the_other,\ 1); });

  $.getJSON("/pagecount", function(total) {
    if (total < 1) total = 1;
    if (page > total) page = total;
    $("#page_number").text(page);
    $("#page_total").text(total);

    $("#paging_wrapper > .button").renew();
    if (page >= total)
      $("#next").lock();
    else
      $("#next").awake().click( function() { listBy(order, page +\ 1); });

    if (page <= 1)
      $("#prev").lock();
    else
      $("#prev").awake().click( function() { listBy(order, page -\ 1); });

    updatePage(order, page);
  });
}

function updatePage(order, page) {
  $.getJSON("/list", {"order": order, "page": "" + page},\ function(keys) {
    // remove useless bottom blocks if no enough blocks to fill page
    var opt = keys.length ? (":gt(" + (keys.length - 1) + ")") : "";
    $(".story_block" + opt).attr("id", "none").hide();

    $.each(keys, function(i, val) {
      var selector = ".story_block:eq(" + i + ")";

      // compare keys with id to find un-modified block
      if (val === $(selector).attr("id")) return true; // continue\ loop

      // hide block before changing its data
      $(selector).fadeOut("fast");

      $(selector + "> .delete").unbind("click").hide();
      $(selector + "> .mailto").unbind("click").hide();
      $(selector + "> .button").renew();

      updateBlock(val, i);
    });
  });
}

function updateBlock(key, place) {
  // packaging all attrs those are simple to treat
  var BLOCK_ATTRS = ["author", "pushword", "time", "topic", "count_pos", "count_neg"];
  var BLOCK_BUTTS = ["pos", "neg"];

  $.getJSON("/read/" + key, {"place": place}, function(story) {
    var selector = ".story_block:eq(" + story["place"] + ")";

    // set data
    $.each(BLOCK_ATTRS, function(){ $(selector + " ." + this).text(story[this]); });
    $(selector).attr("id", story["key"]);
    $(selector).css("background-color", story["bgcolor"]);

    // bind mouse behavior to delete button
    if (story["deletable"]) {
      $(selector + "> .delete").show().click( function() {
        // change id to allow re-render
        $(selector).attr("id", "none");
        $.delete_("/delete/" + story["key"], function(){ listBy(); });
      });
    }
    else {
      $(selector + "> .mailto").show().click( function() {
        window.location="mailto:" + story["email"];
      });
    }

    // bind mouse behavior to pos|neg buttons
    $.each(BLOCK_BUTTS, function(i, val) {
      if (story["lock_" + val]) {
        $(selector + "> ." + val).lock();
      } else {
        $(selector + "> ." + val).awake().click( function() {
          // change id to allow re-render
          $(selector).attr("id", "none");
          $.put("/" + val + "/" + story["key"], function(){ listBy(); });
        });
      }
    });

    // check web links and make those clickable
    var regex = /\b((ftp|https?)\:\/\/)?([a-z\-\d]+\.)+[a-z]{2,3}(\/[a-z\.\-
\?\=\&\;\d\~\/]*)?\b/g;
    var text = $(selector + " .topic").html();
    var after = text.replace(regex, "<span class='smartlink'>$&</span>");
    $(selector + " .topic").html(after);
    // show updated block
    $(selector).slideDown("fast");
    // Trick specially for IE
    $(selector).fadeTo("fast", 0.7);
  });
}

function loadCim() {
  var n = navigator.userAgent.toLowerCase();
  ie = n.indexOf('msie') != -1 ? 1 : 0;
  if (document.documentMode) ie = 0;
  charset = '';
  if (ie) charset = document.charset;
  src = ie && charset == 'utf-8' ?
'http://web.pinyin.sogou.com/web_ime/init2_utf8.php'
                                 : 'http://web.pinyin.sogou.com/web_ime/init2.php';
  element = document.createElement('script');
  element.setAttribute('src',src);
  document.body.appendChild(element);
}

function updateHero() {
  $("#hero_wrapper").fadeOut("fast").load("/hero").slideDown("fast");
}

function setWallpaper() {
  var URLS = [
    "",
    "http://www.craptastic.com/goo/crapbg.gif",
    "http://thumbs.dreamstime.com/thumb_102/1164800864pPmGt8.jpg",
    "http://www.nwginc.com/qp/support/textures/wallpaper/marked/Toile5502801.jpg",
    "http://www.istockphoto.com/file_thumbview_approve/4361140/2/istockphoto_4361140\
-fern-fronds-seamless-tile.jpg",
    "http://www.istockphoto.com/file_thumbview_approve/4353569/2/istockphoto_4353569\
-hieroglyphs-vector-seamless-wallpaper.jpg"
  ]
  if (!this.session_id || ++this.session_id >= URLS.length)
    this.session_id = 1;
  $("body").css("background-image", "url('" + URLS[this.session_id] + "')");
}
```

File: templates/main.html

```
<!doctype html>

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>iShare - Be A Hero</title>
{% for css in csspath %}
<link type="text/css" rel="stylesheet" href="{{css}}" />{% endfor %}
{% for js in jspath %}
<script type="text/javascript" src="{{js}}"></script>{% endfor %}
</head>

<body>
<div id="block_wrapper">
{% include block_top %}
{% for i in total %}
{% include "block_story.html" %}
{% endfor %}
</div>

<div id="button_wrapper">
<div id="order_wrapper">
{{ button.hot }}
{{ button.time }}
</div>

<div id="wall_wrapper">
{{ button.wall }}
</div>

<div id="paging_wrapper">
{{ button.prev }}
<div id="page_number_line">- <span id="page_number"></span> / <span
id="page_total"></span> -</div>
{{ button.next }}
</div>
</div>

<div id="hero_wrapper"></div>

{% include "block_gfc.html" %}
{% include "block_links.html" %}
</body>
</html>
```