**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

Capstone Project Document

Smart Buy

|  |  |
| --- | --- |
| **Group 1** | |
| **Group member** | Doan Ho Anh Triet – Team Leader – SE60763  Huynh Thanh Viet – Team Member - SE60666 (Dropped out)  Dang Huu Hoang – Team Member - 60486  Tran Trung Dung – Team Member - 60236 |
| **Supervisor** | Mr. Kieu Trong Khanh |
| **Ext. Supervisor** | N/A |
| **Capstone Project code** | SmartB |

-Ho Chi Minh City, 01/2014-

*This page is intentionally left blank*

***ACKNOWLEDGEMENTS***

We wish to thank various people for their contribution to this project: Our teachers for their advice and participation in the final review, our friends for their valuable technical support.

Special thanks should be given to Mr. Kieu Trong Khanh, our research supervisor for his professional guidance and the useful, constructive recommendations throughout the course of this project.

# Table of Contents

[Table of Contents 4](#_Toc385581080)

[List of Tables 6](#_Toc385581081)

[List of Figures 7](#_Toc385581082)

[Definitions, Acronyms, and Abbreviations 8](#_Toc385581083)

[A. Software Project Management Plan 9](#_Toc385581084)

[1. Problem Definition 9](#_Toc385581085)

[1.1 Name of this Capstone Project 9](#_Toc385581086)

[1.2 Problem Abstract 9](#_Toc385581087)

[1.3 Project Overview 9](#_Toc385581088)

[2. Project organization 10](#_Toc385581089)

[2.1 Software Process Model 10](#_Toc385581090)

[2.2 Roles and responsibilities 11](#_Toc385581091)

[2.3 Tools and Techniques 12](#_Toc385581092)

[3. Project Management Plan 13](#_Toc385581093)

[B. Software Requirement Specification 14](#_Toc385581094)

[1. User Requirement Specification 14](#_Toc385581095)

[1.1 Guest Requirement 14](#_Toc385581096)

[1.2 Member Requirement 14](#_Toc385581097)

[1.3 Staff Requirement 14](#_Toc385581098)

[1.4 Admin Requirement 14](#_Toc385581099)

[1.5 System Requirement 14](#_Toc385581100)

[2. System Overview Use Case 14](#_Toc385581101)

[3. Main Flow 15](#_Toc385581102)

[3.1 Guest and Member 15](#_Toc385581103)

[3.2 System and Staff 15](#_Toc385581104)

[4. Entity Relationship Diagram 17](#_Toc385581105)

[C. Software Design Description 19](#_Toc385581106)

[1. Design Overview 19](#_Toc385581107)

[2. System Architectural Design 20](#_Toc385581108)

[2.1 Web Architecture 20](#_Toc385581109)

[2.2 Mobile Architecture 20](#_Toc385581110)

[3. Component Diagram 21](#_Toc385581111)

[4. Detailed Description of Components 22](#_Toc385581112)

[4.1 Class Diagram 22](#_Toc385581113)

[4.2 Class Diagram Explanation 23](#_Toc385581114)

[5. Algorithms 27](#_Toc385581115)

[5.1 Dynamic Programming 27](#_Toc385581116)

[5.2 String Comparison 29](#_Toc385581117)

[D. Appendix 31](#_Toc385581118)

# List of Tables

[Table 2: Hardware Requirement for Server 10](#_Toc385581119)

[Table 3: Hardware Requirement for Mobile App 10](#_Toc385581120)

[Table 4: Roles and Responsibility Details 12](#_Toc385581121)

[Table 5: Iteration 13](#_Toc385581122)

[Table 32: Entity Data dictionary - Describe content of all entities 18](#_Toc385581123)

# List of Figures

[Figure 1: Agile Development Model 11](#_Toc385581133)

[Figure 2: System Overview Use Case 15](#_Toc385581134)

[Figure 3: Guest and Member main flow 15](#_Toc385581135)

[Figure 4: System and Staff main flow 16](#_Toc385581136)

[Figure 5: Conceptual Diagram 17](#_Toc385581137)

[Figure 6: MVC Architecture 20](#_Toc385581138)

[Figure 7: Mobile Architecture 20](#_Toc385581139)

[Figure 8: Component Diagram 21](#_Toc385581140)

[Figure 9: Class Diagram 22](#_Toc385581141)

# Definitions, Acronyms, and Abbreviations

|  |  |  |
| --- | --- | --- |
| **No.** | **Abbreviation & Acronym** | **Definition** |
| 1 | SmartB | Smart Buy |
| 2 | OS | Operation System |
| 3 | Admin | Administrator |
| 4 | API | Application Programming Interface |
| 5 | HTTP | Hyper Text Transfer Protocol |

# Software Project Management Plan

## Problem Definition

### Name of this Capstone Project

Smart Buy (SmartB).

### Problem Abstract

People go shopping every day. They buy clothes, books, stationeries, etc… and most importantly - food. They often buy them at their familiar markets, or the nearest one. Today, together with the growing use of smart phones, we should find a smarter way to do our daily jobs, such as shopping. To achieve this, we have to resolve the most important problem of shopping: price. How can we know that the price of a product is not too high? Or somewhere else has better price? Our system will, therefore, help users to find the price of a product in the market, keep track of the price fluctuation, suggest the best way to buy a list of products, etc…

### Project Overview

#### Problem Definition

The problem arises when people want to buy food for their meal, but they do not know how much it is.

* If they are at home, they have to go to many websites to estimate the price, because the price is not the same in many websites.
* If they are at the market, they only know the price which the seller tells them. They have no idea about the price at other places.

#### The Proposed System

The system is intended for use by those with a smart phone or a laptop/computer with Internet connection. The system will have the following functions:

##### Web Application

* Admins can manage accounts in the system.
* System can parse data from many websites at specific time.
* Staff can manage markets, products, parsers, configure system, and train machine.
* Member can save product history, ask for the best buying way, and propose product price.
* Guest can search product to view its price range, add products to cart, view report.

##### Mobile Application

* Users can request to view the price, ask for the best buying way, and propose product price.

#### Boundaries of the System

* The system can be used by every people with a smart phone or a laptop/computer with Internet connection.
* The system is **not intended** for managing these aspects:

+ Managing product quality.

+ Managing your expense.

+ Managing nutritional ingredients of your meals.

* The language of the system is Vietnamese.
* The complete product includes:

+ The website, for staff and user.

+ Mobile Application for user.

+ All the process document involved.

#### Development Environment

##### Hardware requirements

**For Server**

|  |  |  |
| --- | --- | --- |
| Windows | Minimum Requirements | Recommended |
| Internet Connection | Cable, Wifi (4 Mbps) | Cable, Wifi (8 Mbps) |
| Operating System | XP, Vista, 7, 8 | XP, Vista, 7, 8 |
| Computer Processor | Intel® Core 2 Duo | Intel® Core(TM) i5 CPU , M 460 @ 2.53GHz |
| Computer Memory | 1GB RAM | 3GB or more |

Table 2: Hardware Requirement for Server

**For Mobile Application**

|  |  |  |
| --- | --- | --- |
| Mobile | Minimum Requirements | Recommended |
| Internet Connection | Wifi (2Mbps) | Wifi (4Mbps) |
| Operating System | Android 4.2 | Android 4.2 |
| Hardware | Touchscreen | Touchscreen |
| Memory | 512 MB or more | 1 GB or more |

Table 3: Hardware Requirement for Mobile App

##### Software requirements

* Microsoft Windows 7 Ultimate: operating system and platform for development.
* SQL Server 2008 Enterprise R2: used to create and manage the database for system.
* StarUML: used to create models and diagrams.
* Skype: used for communication and meeting.
* Visual Studio 2012: used to implement website and web service.
* Eclipse Juno 4.4, Android SDK 22.0.5, ADT 22.0.5 & JDK 7u25: used to implement mobile application.
* Google Code & TortoiseSVN: used for source control.

## Project organization

### Software Process Model

Project is developed under agile model.



Figure 1: Agile Development Model

For more information: <http://www.indicthreads.com/1439/quick-introduction-to-agile-software-development/>

(Owner: IndicThreads.com. Online Software Developer Magazine and Conferences)

### Roles and responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Full name** | **Role in Group** | **Responsibilities** |
| **1** | Kieu Trong Khanh | Project manager | * Specify user requirement * Control the development process * Give out technique and business analysis support |
| **2** | Doan Ho Anh Triet | Team Leader, BA, DEV, Tester | * Managing process * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **3** | Huynh Thanh Viet | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **4** | Dang Huu Hoang | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **5** | Tran Trung Dung | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |

Table 4: Roles and Responsibility Details

### Tools and Techniques

- Front-end technologies: HTML5, CSS3, JavaScript, jQuery, AJAX.

- Back-end:

* Website: ASP.NET MVC4 + Entity Framework 5.
* Web Service: Web API.
* Mobile App: Android - Java.
* Scheduler: Quartz.
* Parse data from Excel file: OLEDB.NET.

- Web Server: Microsoft IIS.

- Database Management System: MS SQL Server 2008 Enterprise R2.

## Project Management Plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Phase**  **/Iteration** | **Description** | **Deliverables** | **Resource needed** | **Dependencies and Constrains** | **Risks** |
| **Preliminary Investigation or Analysis** | - Study similar existing systems.  -Identify and clarify requirements for the system in general. | -Introduction of proposed system.  -Main functions.  -Project Iteration Plan. | 30 man-days | N/A | Project may  not be feasible  for developing  because lack of technologies  and/or data |
| **Data management** | - Parse data from websites.  - Input data manually.  - Import data from excel files. | - Data management service. | 30 man-days | N/A | Lack of experience.  The implemented parsers are not the best.  Lack of test data |
| **Main user’s functions** | - Let user update price for current day.  - User can search a product price. | - Main user’s functions on web and mobile. | 30 man-days | Depend on “Data management”. | Lack of experience.  Not have a clear understanding about business process. |
| **Suggestion algorithm** | - Build algorithm to calculate the best way to buy a list of products. | - Suggestion service.  - User now can ask for the best way to buy a list of products. | 20 man-days | Depends on “Data management”. | The implemented algorithm is not the best.  Lack of test data. Lack of experience on making and deploying web service. |
| **Market and User Account management** | - Manage markets in the system.  - Manage user accounts in the system | - Market management system.  - User account management system. | 15 man-days | N/A | Lack of experience.  Not have a clear understanding about business process. |

Table 5: Iteration

# Software Requirement Specification

## User Requirement Specification

### Guest Requirement

Guest is a person who doesn’t have access to the system. Guest can use some functions in the system. To use all functions, guest must login. These are some functions guest can use:

* Register, Login.
* Search product.
* Add product to cart.

### Member Requirement

Member is guest who uses his account to login to the system. Member can use additional functions, such as:

* Propose product price.
* Ask for system suggestion
* Save product history.

### Staff Requirement

Staff is the person who manages markets and products. Staff can use these functions:

* Manage markets.
* Manage products.
* Manage parsers.
* Configure system.

### Admin Requirement

Admin is the person who manages the system. Admin can use these functions:

* Manage user.

### System Requirement

System is also an actor, run in the background to keep the system working. System can do functions:

* Parse data.

## System Overview Use Case

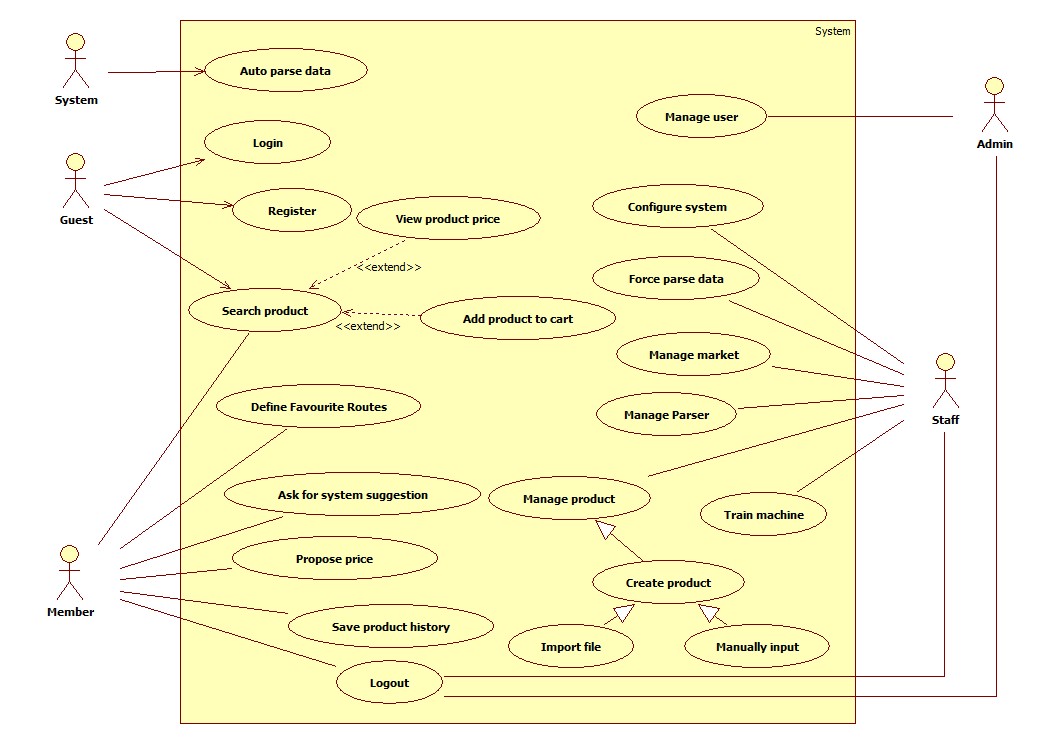


Figure 2: System Overview Use Case

## Main Flow

### Guest and Member

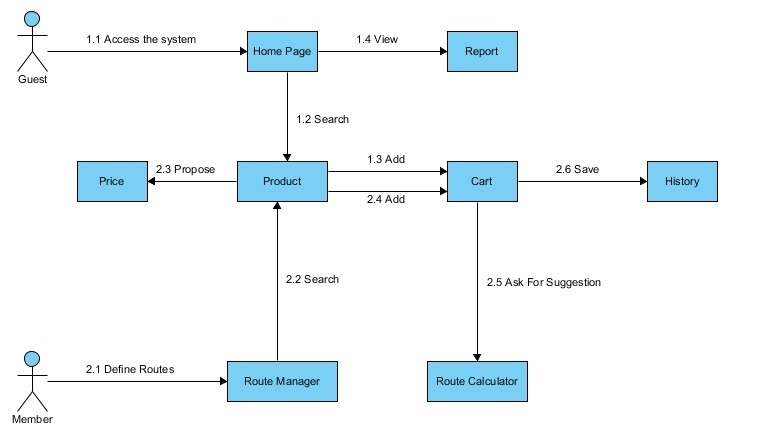


Figure 3: Guest and Member main flow

### System and Staff

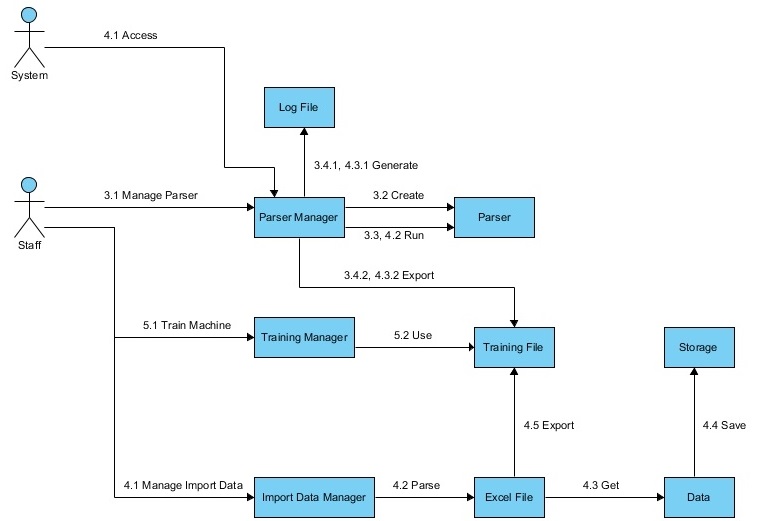


Figure 4: System and Staff main flow

## Entity Relationship Diagram

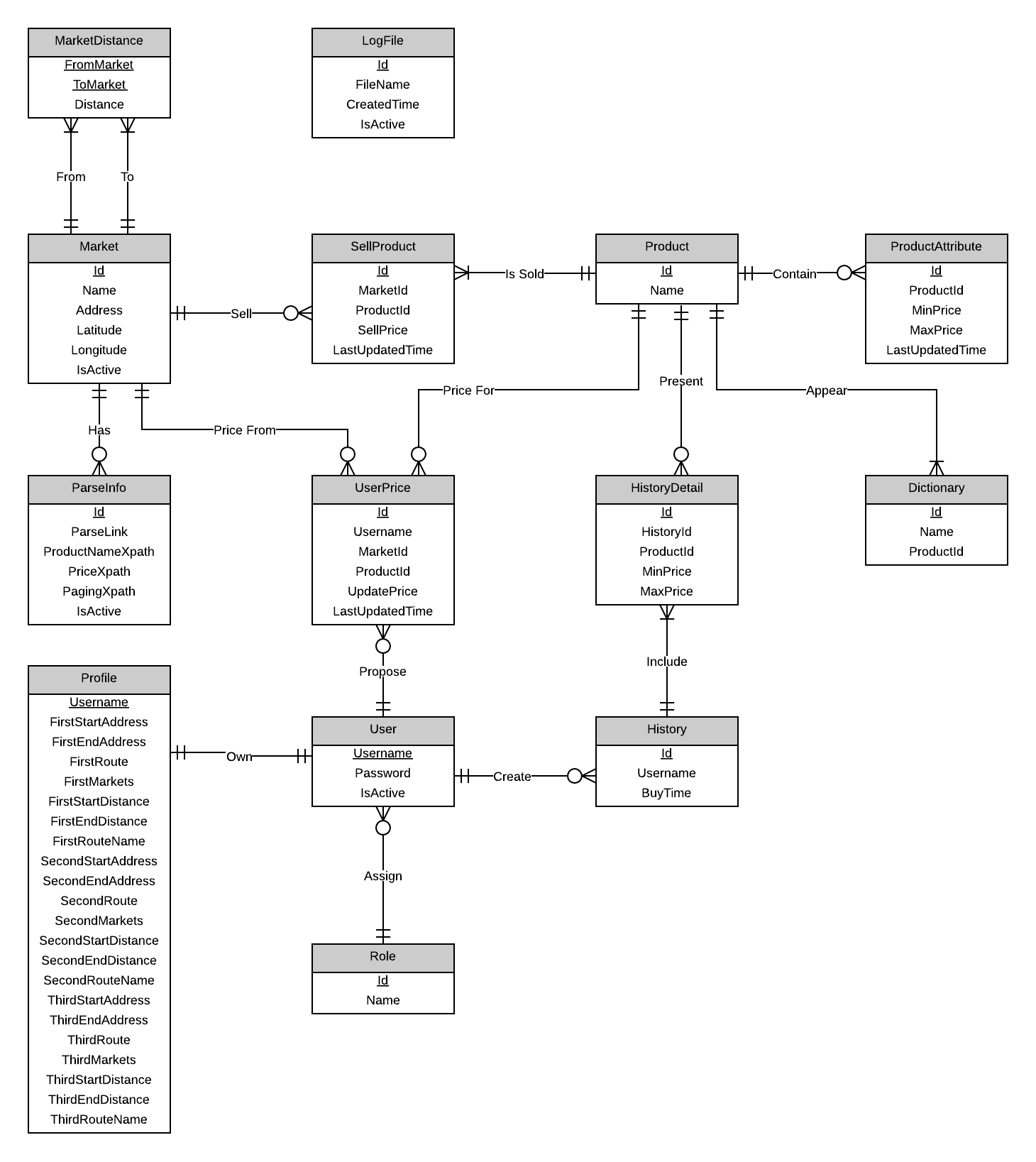


Figure 5: Conceptual Diagram

|  |  |
| --- | --- |
| Entity Name | Description |
| Market | Describe all markets in the system. Market may exist in real life or just virtual. |
| Parse Info | Describe all needed data to parse a website. |
| Product | Describe all products in the system. |
| Product Attribute | Describe attributes of each product. |
| Log File | Describe all log files of the system. Log file is generated after parsers finish running. |
| User | Describe all accounts in the system. |
| Role | Describe all roles in the system. |
| Profile | Describe all user profiles in the system. Each user has only one profile. |

Table 32: Entity Data dictionary - Describe content of all entities

# Software Design Description

## Design Overview

* This document describes the technical and user interface design of Smart Buy System using mobile device. It includes the architectural design, the detailed design of common functions and business functions and the design of database model.
* The architectural design describes the overall architecture of the system and the architecture of each main component and subsystem.
* The detailed design describes static and dynamic structure for each component and functions. It includes class diagrams, class explanations and sequence diagrams for each use cases.
* The database design describes the relationships between entities and details of each entity.
* Document overview:
* Section 2: gives an overall description of the system architecture design.
* Section 3: gives component diagrams that describe the connection and integration of the system.
* Section 4: gives the detail design description which includes class diagram, class explanation, and sequence diagram to details the application functions.
* Section 5: describe an ERD with logical diagram.

## System Architectural Design

### Web Architecture

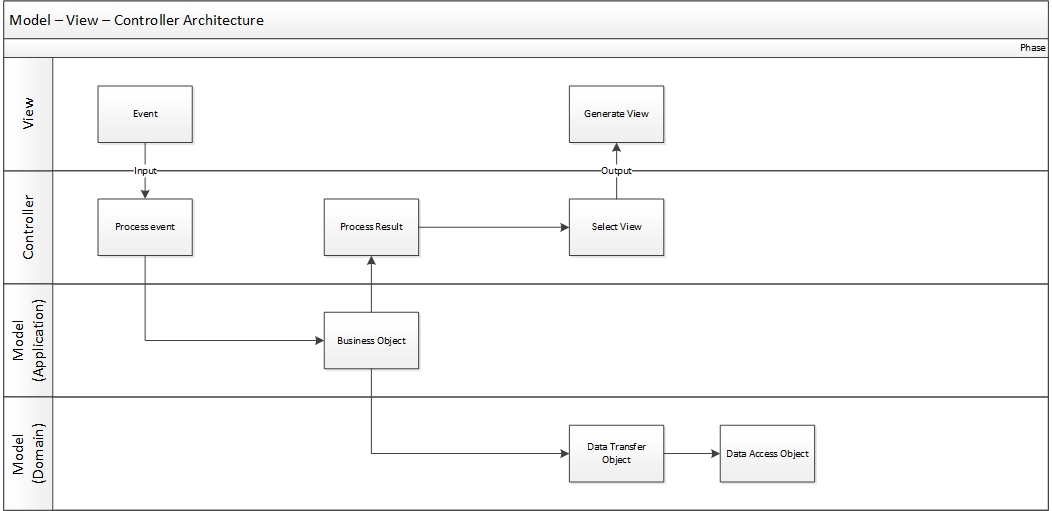


Figure 6: MVC Architecture

* **Model** is the part of the application that handles the logic for the application data. Often model objects retrieve data (and store data) from a database.
* **View** is the parts of the application that handles the display of the data. Most often the views are created from the model data.
* **Controller** is the part of the application that handles user interaction. Typically controllers read data from a view, control user input, and send input data to the model.

### Mobile Architecture

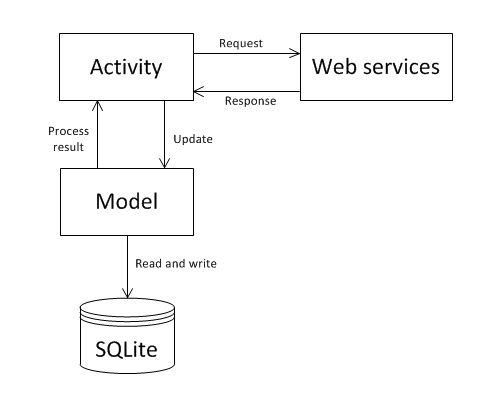


Figure 7: Mobile Architecture

## Component Diagram

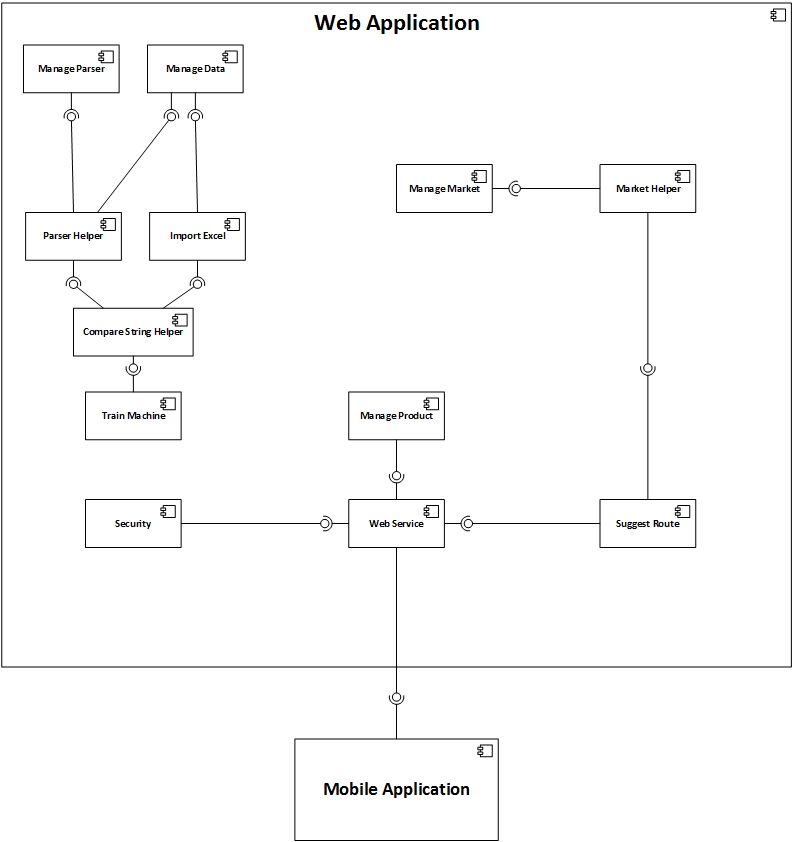


Figure 8: Component Diagram

## Detailed Description of Components

### Class Diagram

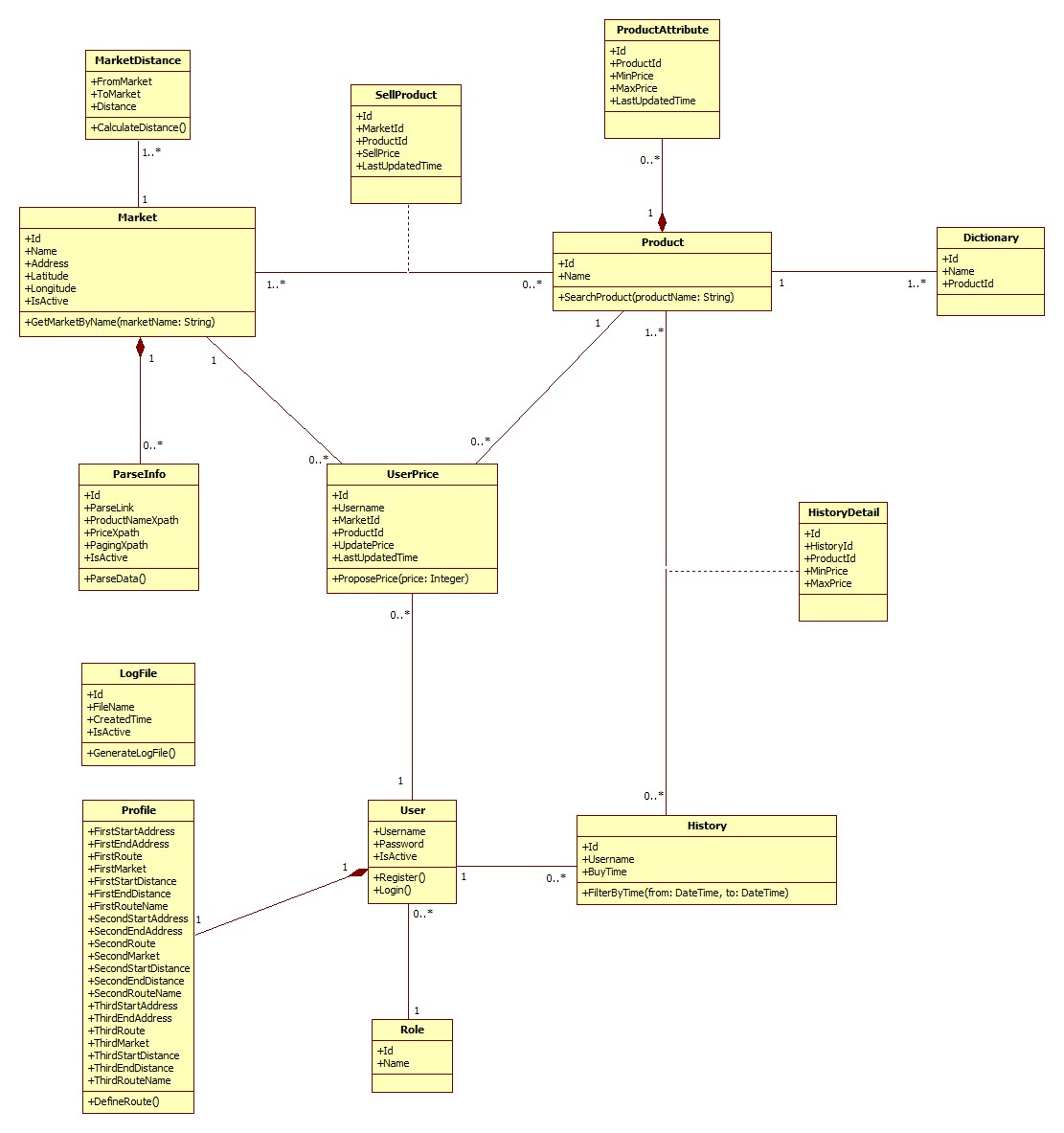


Figure 9: Class Diagram

### Class Diagram Explanation

#### Market

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each market |
| Name | String | Public | Name of market |
| Address | String | Public | Address of market |
| Longitude | Double | Public | Longitude of market |
| IsActive | Boolean | Public | Status of market |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| GetMarketByName | List of markets | Public | Find market by its name |

#### Parse Info

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each item |
| ParseLink | String | Public | Link used to parse |
| ProductNameXpath | String | Public | Xpath to get the name of products |
| PriceXpath | String | Public | Xpath to get the price of products |
| PagingXpath | String | Public | Xpath to get the pagination part |
| IsActive | Boolean | Public | Status of info |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| ParseData | Void | Public | Run parser to get data |

#### Sell Product

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each item |
| MarketId | Integer | Public | Id of market |
| ProductId | Integer | Public | Id of product |
| SellPrice | Integer | Public | That market sells that product at this price |
| LastUpdatedTime | DateTime | Public | The time when the price is updated. |

#### Product

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each product |
| Name | Integer | Public | Product name |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| SearchProduct | List of products | Public | Search product by its name |

#### Product Attribute

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each item |
| ProductId | Integer | Public | Id of product |
| MinPrice | Integer | Public | Min price of this product |
| MaxPrice | Integer | Public | Max price of this product |
| LastUpdatedTime | DateTime | Public | The time when this information is updated. |

#### User Price

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each item |
| Username | String | Public | Username of member who propose price |
| MarketId | Integer | Public | Id of market |
| ProductId | Integer | Public | Id of product |
| UpdatePrice | Integer | Public | Proposed price |
| LastUpdatedTime | DateTime | Public | The time when the price is proposed. |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| ProposePrice | Void | Public | Propose price for the system |

#### User

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Username | String | Public | Username for each user |
| Password | String | Public | User’s password |
| IsActive | Boolean | Public | Status of this account |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| Register | Void | Public | For new user to register |
| Login | Boolean | Public | Used to log in the system |

#### History

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each history |
| Username | String | Public | Owner of this history |
| BuyTime | DateTime | Public | The time when owner create this history |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| FilterByTime | List of history | Public | Filter history list by the range of time |

#### History Detail

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each item |
| HistoryId | Integer | Public | Id of history |
| ProductId | Integer | Public | Id of product |
| MinPrice | Integer | Public | Min price of that product at that time |

#### Role

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each role |
| Name | String | Public | Role name |

#### Log File

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique identifier of each file |
| FileName | String | Public | File name |
| CreatedTime | DateTime | Public | The time when this file is created |
| IsActive | Boolean | Public | Status of this file |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return type** | **Visibility** | **Description** |
| GenerateLogFile | Boolean | Public | Generate log file for each time system runs the parser |

#### Dictionary

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| Id | Integer | Public | Unique id of each item |
| Name | String | Public | The name of the product |
| ProductId | Integer | Public | The identifier of that product |

#### Market Distance

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| FromMarket | Integer | Public | Unique id of a market |
| ToMarket | Integer | Public | Unique id of a market |
| Distance | Integer | Public | Distance between 2 markets |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return Type** | **Visibility** | **Description** |
| CalculateDistance | Integer | Public | Calculate distance between 2 markets |

#### Profile

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| FirstStartAddress | String | Public | Start address of the first route |
| FirstEndAddress | String | Public | End address of the first route |
| FirstRoute | String | Public | First favourite route |
| FirstMarket | String | Public | List of nearby market id separated by comma |
| FirstStartDistance | String | Public | List of distance from the first start point to all nearby markets, separated by comma |
| FirstEndDistance | String | Public | List of distance from the first end point to all nearby markets, separated by comma |
| FirstRouteName | String | Public | Name of the first route |
| SecondStartAddress | String | Public | Start address of the second route |
| SecondEndAddress | String | Public | End address of the Second route |
| SecondRoute | String | Public | Second favourite route |
| SecondMarket | String | Public | List of nearby market id separated by comma |
| SecondStartDistance | String | Public | List of distance from the second start point to all nearby markets, separated by comma |
| SecondEndDistance | String | Public | List of distance from the second end point to all nearby markets, separated by comma |
| SecondRouteName | String | Public | Name of the second route |
| ThirdStartAddress | String | Public | Start address of the third route |
| ThirdEndAddress | String | Public | End address of the third route |
| ThirdRoute | String | Public | Third favourite route |
| ThirdMarket | String | Public | List of nearby market id separated by comma |
| ThirdStartDistance | String | Public | List of distance from the third start point to all nearby markets, separated by comma |
| ThirdEndDistance | String | Public | List of distance from the third end point to all nearby markets, separated by comma |
| ThirdRouteName | String | Public | Name of the third route |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Return Type** | **Visibility** | **Description** |
| DefineRoute | Void | Public | Define favourite routes |

## Algorithms

### Dynamic Programming

#### Definition

In [mathematics](http://en.wikipedia.org/wiki/Mathematics), [computer science](http://en.wikipedia.org/wiki/Computer_science), [economics](http://en.wikipedia.org/wiki/Economics), and [bioinformatics](http://en.wikipedia.org/wiki/Bioinformatics), dynamic programming is a method for solving complex problems by breaking them down into simpler sub-problems. It is applicable to problems exhibiting the properties of [overlapping sub-problems](http://en.wikipedia.org/wiki/Overlapping_subproblem) and [optimal substructure](http://en.wikipedia.org/wiki/Optimal_substructure). When applicable, the method takes far less time than naive methods that don't take advantage of the sub-problem overlap (like [depth-first search](http://en.wikipedia.org/wiki/Depth-first_search)).

References: <http://en.wikipedia.org/wiki/Dynamic_programming>

#### Define Problem

In our system, there is a function called “Ask for System Suggestion”. This function gives users the best way to buy their list of products based on their cart and the markets around their defined route. Here we can describe this problem as below:

**Given a two-dimensional array named Price, size means market sells product at price. Find a way to buy all products at the minimum cost.**

**The output should be in this format**:

* **Buy <product name> at <market name>.**
* **Buy <product name> at <market name>.**
* **Buy <product name> at <market name>.**
* **…**

#### Solution

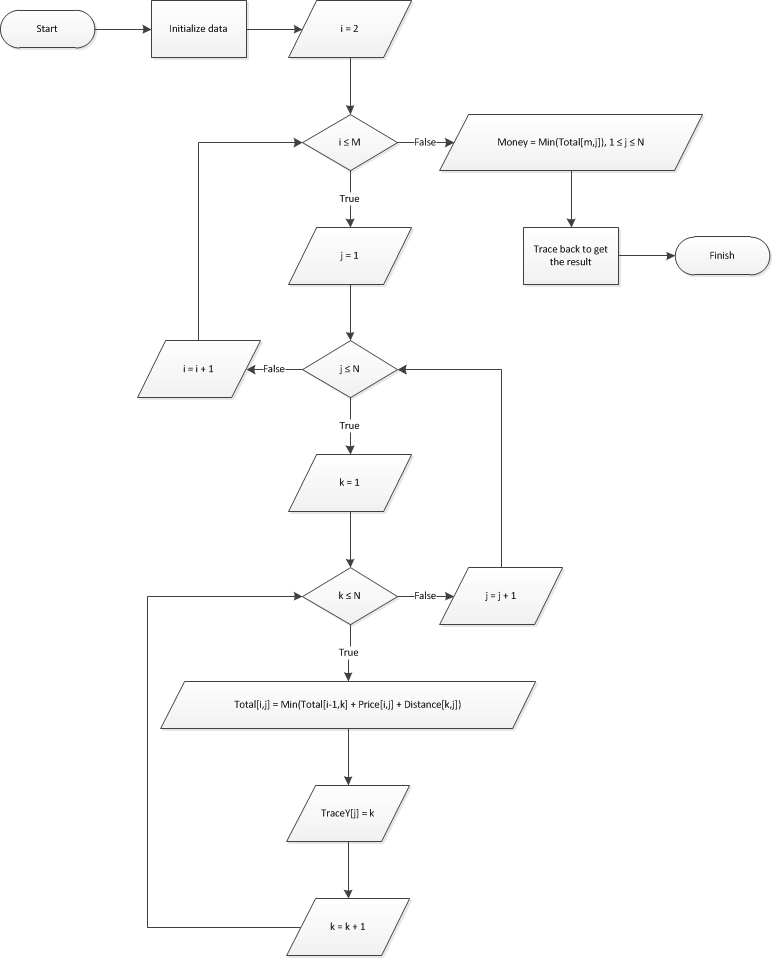
To solve this problem, we should follow these steps:

* Call **M** is the number of total products; **N** is the number of total markets.
* Create a two-dimensional array named **Total** size . is the sum of money user have to pay when the buying process is at item in market .
* Create a two-dimensional array named **Distance** size . is the distance from market *i* to market *j*.
* Create a one-dimensional array named **TraceY**, size **N**. means we buy the product before product *i* at market .
* Create a one-dimensional array named **DistanceA**, size **N**. is the distance from start point to market *i*.
* Create a one-dimensional array named **DistanceB**, size **N**. is the distance from end point to market *i*.
* Initialize all elements in **Total** with a very big number.
* Set the first row of **Total**:
* Use this regression formula to calculate the rest of **Total**:
* Set
* The minimum value at the final row of **Total** is the answer.
* Trace back to find the complete answer.

#### Complexity

* To traverse through all elements in **Total** array, we go through steps.
* With each element, we traverse its previous row. Each row has elements.
* In total, the complexity of this algorithm is

#### Flowchart



### String Comparison

#### Define Problem

Given 2 strings. Calculate their matching percent.

#### Requirement

* Robustness to changes of word order: two strings which contain the same words, but in a different order, should be recognised as being similar.
* Language Independence - the algorithm should work not only in English, but in many different languages.

#### Solution

* If a string contains many words, break it into a list of words.
* For each word, we find out how many adjacent character pairs are contained in it.
* Create a function *pairs(s)* which returns a list of adjacent character pairs of string *s*.
* Then, we use below formula to calculate matching percent.

#### Example

Calculate the matching percent of 2 strings: France and French.

* Upper case 2 strings:
  + France FRANCE.
  + French FRENCH.
* Break string into list of adjacent character pairs:
  + FRANCE
  + FRENCH
* Calculate its matching percent.

# Appendix

1. Dynamic Programming: <http://en.wikipedia.org/wiki/Dynamic_programming>
2. String Comparison: <http://www.catalysoft.com/articles/StrikeAMatch.html>
3. Google API: <https://developers.google.com/maps/documentation/javascript/tutorial>