MyAlcoholFreeWine Web Application

Author: Duan, Hongyi
Email: hod8@aber.ac.uk
Reference: 159011454

Monday 7th December 2015

SE31520/CHM5820 Assignment 2015 - 2016
Module: CHM5820 - Enterprise Systems Development

Contents

List of Figures

1 Introduction

1.1 Summary

This documentation consists of specifications which provide details of the functionality implemented from the CHM5820 first assignment.

In this report, these parts have be described:

1. The introduction of the MAF application includes requirements, analysis and implementations with the architecture of UML diagrams.
2. The design of RESTful supplier web services and the rationale includes how the decision made.
3. The test strategy covers the models and controllers of the MAF Rails application and a result of test table.
4. A critical evaluation and self-assessment.

1.2 System Description

In this system, customer can connect with the MAF web, make some operations and get web page view reply. Each supplier web service has a database (I use file here) and should be able to load or save data into file. When supplier web runs, MAF web application will transfers data through RESTful requests with web service.

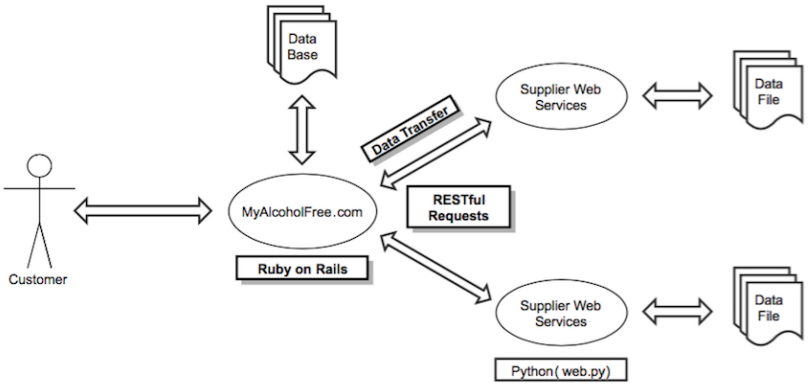

Figure 1: Architecture of the MAF System

Figure 1 shows how the system works and interact with each other between MAF web application and supplier web services.

By using RESTful requests, MAF will obtain information for wines and retail supplier can get customers' order details. The Rails web application - MyAlocoholFreeWine contains one database for each model. When the application runs, it has abilities to add/delete/edit data in the database as requirements.

2 Requirements Analysis

The MyAlcoholFreeWine web application provides a resource of wines rendered by different retailer supplier web services that allows customers to find cheapest non-alcohol wines. The details description of specification requirements will be discussed in the next three subsections.

2.1 MAF Web Application

When a customer visit the MyAlcoholFreeWine.com, these specified functions will be given to the user:

1. In order to get enough information, functionality of Browse, search and display details non-alcohol wines should be provided to the customers. Browse or search wines only provides wine information of picture, description, size, price and company. Functionality of display details will show all information of the wine customer wants to know more.

2. There should be basket or cart for customers. After picking the wine they are interested in, users can add the wine products to their shopping cart. The cart should be able to display all the wines when a guest want to check products what he has chosen.

3. Although guests can view the wine products in the MAF shop, they can not checkout and pay for the products. Only a real user can check out and fill a deliver form. If a customer want to checkout, they will be requested to input email and password to login. After that, the retailer supplier will receive the order, and then they can package the wines and send to users.

4. According to the requirement of third function, Login/Logout and registration would be necessary. Customer would able to allow to registration as a real user at home page or when guest checks out. Login and logout functions are provided for users at any moments when they browse the MyAlcoholFreeWine.com pages.

Other information on how to complete the program will be discussed in the MAF Application section.

2.2 Supplier Web Service

As the requirements from the assignment document hand-out before, the most important part of supplier web service need to be implemented is that the service must present a RESTful interface that receives and delivers data.

The following requirements for supplier web service is necessary:

- The architectural style of web service application should be REST stands for Representational State Transfer.
- The web service can read and refresh data from a data file, so that MAF web application can get wine data from it by using HTTP verb - GET.
- Once an order is made by customer, MAF web would send POST request to remote service.

The details of decisions and implementations of web service should be cross-referenced as part of the Supplier Web Service section.

2.3 Testing

As for me, testing is an integral part of application development. I want to make sure that program can behaves like I expected. Rails makes test easy by producing skeleton test or using some test 'gem'.

In this web application, the test strategy was taken and built by the me. Some of models in the MAF application were to use inbuilt Rails testing. Otherwise, for all the controllers, I use a gem called 'Cucumber' - a better way in testing controller and behavior.

More details and results of tests can be found in the Test Specification section.

3 MAF Application

This section includes 3 UML diagrams referred to my MAF web application and how to Implement some important functionality.

3.1 UML Diagram

3.1.1 MVC Structure

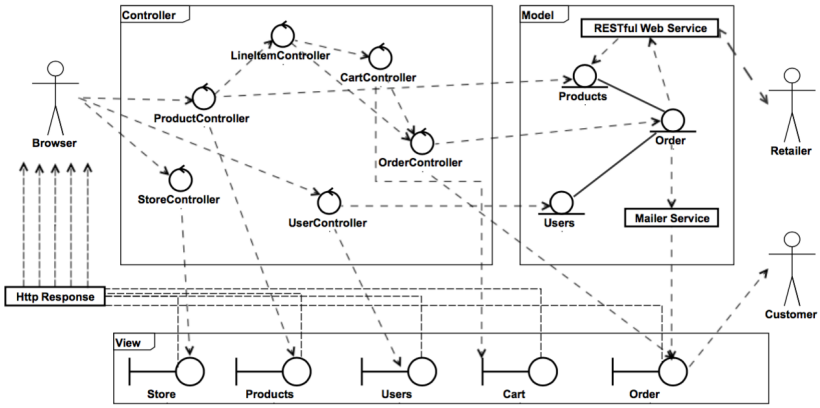

Figure 2: UML Diagram of MVC Structure

The diagram above illustrates the Model-View-Controller (MVC) structure used in the application. It divides MAF application into three interconnected parts and lead me to complete each part in turn.

3.1.2 User Case

Here shows two images of main user cases, each user case contains two user - guest and real use. For the users who uses MAF web system, different limits of authority are given by different user identities. In some cases, users share many of the same capabilities. But some functions set up are only provided for special user.

Figure 3 below shows the Registration user case. Guests can sign up to be a new user, then they can access functions like login, logout or edit profile. That says they are authorised to use more functions (e.g. Checkout).

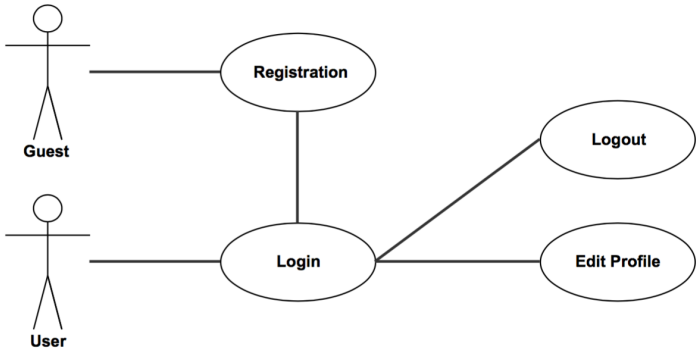

Figure 3: User Case - New User

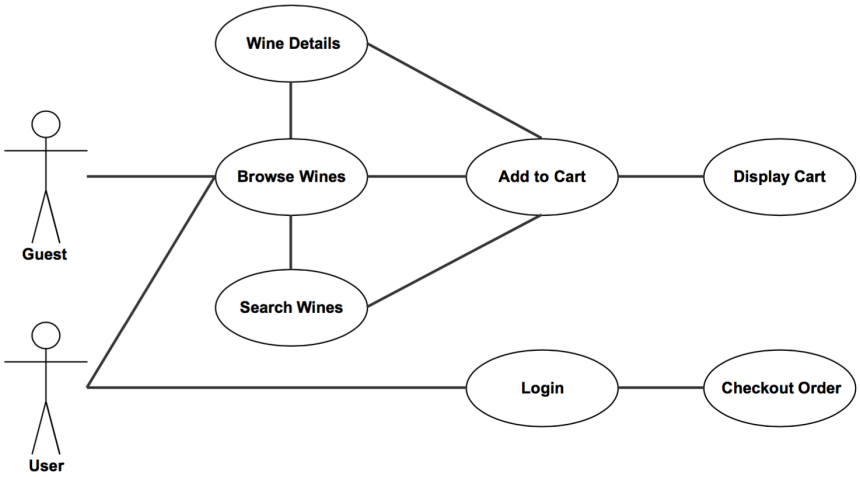

Figure 4: User Case - Order and Checkout

Figure 4 above give the user case of Order and Checkout. As I have said before, guests and users share most of functions like Browse Wine, Search, Add to Cart, etc. Only user are authorised to checkout after they have logged in. In this user case, it shows clearly that there are 3 ways provided for user to add wine products to the cart. So when users use this application to browse, search or check details of wine, there should exist a button to meet demands.

3.1.3 Data Model Diagram

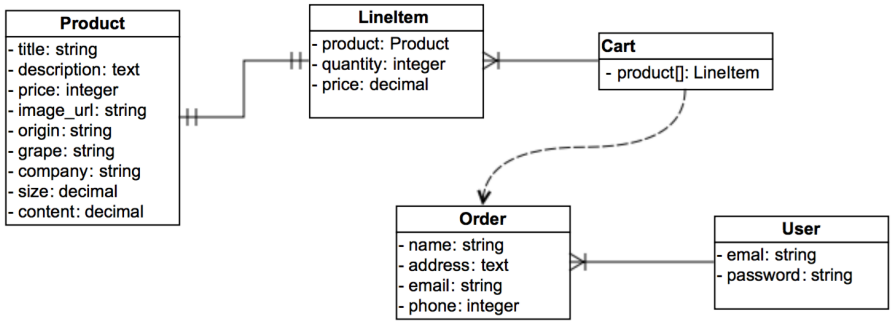

Figure 5: Current Data Model

Figure 5 shows the current data model includes the relationships and dependencies among the data. There is one to one relationship between Product and LineItem. The reason for create a data LineItem is to connect wine Product with Cart, the quantity attribution should not be in the Product model nor Cart model. Also one to many relationships should be used between Cart and LineItem model, User and Order model. One more things should be notice is that, every time, when user check out successfully, the data in current Cart will destroy and create a new data of Order entity.

3.2 Implement Process

The implementation of MAF web application can not be described in all sides. Follows show some basic process of implementation and some parts I want to talk about:

1. Rails framwork for ruby is a so powerful and easy for developer to implement a RESTful web service. Most parts of application can be generated automatically by itself.

2. The designs of controller and model I have shown in this section. Follow the UML diagrams and make good use of rails can create database and connection between controllers quickly.

3. As for the User model and controller, I use a ruby gem 'device' which can satisfy all the demands of user function part includes login/logout, registration, authorisation and etc.

4. I spent most of time in View design and optimization. Before starting developing, I draw a page flow as a guide like Figure 6 below. Detailed demos and final result can be found in the screencast.

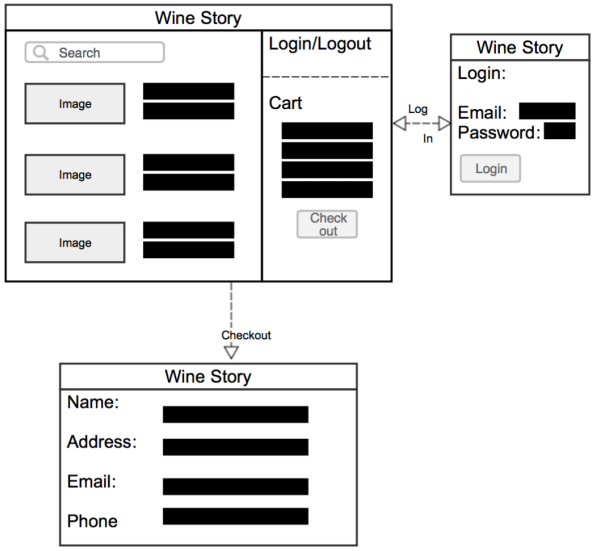

Figure 6: Initial Page Flow

4 Supplier Web Service

The requirements of functionality and standard of supplier web service I have illustrated clearly in the Requirements Analysis section. The further introduction of rationale design and implement will be discussed in the next two subsections.

4.1 Analysis and Rationale

In view of requirements, these design ideologies were used:

1. Without UI expectations and to meet basic needs, also because I have used ruby in MAF application. I decide to build this web service in python. The reason is that python can be more suitable and fast than Java, php or other language with the limited feature needs.

2. Because the data to be transmitted is less and it is easier to gain high transmission speed, I choose Json as data interchange format. The requests of small data and frequent exchange can be completely satisfied.

3. The most important part in the web service is data exchange between two applications. Next points below show the rationale for designs:

   (a) Wines and orders are the data needed to be transmitted. MAF application can get wines from supplier web services and vice versa for orders.

(b) Each web service has a data file to store wines. When the web service get the requests from MAF users, it will read data file and send it back to users as respond.

(c) The Figure 7 shows obviously how the data be exchanged in this system. So only the HTTP verbs - GET and POST need to used.

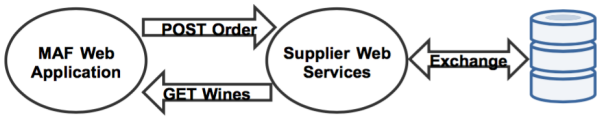

Figure 7: Data Exchange

4.2 RESTful Implementation

At the beginning, I need to decide when and how to transfer data. Although the methods of request are similar, there are still have different strategies to implement RESTFful architectural: MAF can send GET request to supplier for wines products. On the other hand, suppliers also can POST wine data to MAF if they have new product. The same situation also exit in the transfer of orders data as well.

When users browse my website, I hope that they can get most accurate and up-to-date product information. This though make me define a method connected with 'index', named 'load-product' in the 'store-controller'. MAF will send GET request to supplier for receiving newest wine data and display to users when they refresh or go to the HOME page. As for orders data, order data will POST to supplier automatically when the order has been confirmed. The data will be saved in the 'order-data' file

The next step is to build supplier web system. I write a small program to write wines data into 'wine-data' file. The program of web service can read data in this file. I choose python to make a website, and web.py provides the code to make this easier. For example, at the beginning and type 'import web.py' to imports this module. Tell python that URL structure is 'urls = ('/wines', 'index')' and, set class and define GET functions and then tell web.py to start serving web pages.

5 Test Specification

5.1 Strategy

The test strategy taken by the author of this application can be divided into three parts - use the inbuilt test, gem test, UI test. According to the requirements and self-demand test:

1. The way of testing will include inbuilt Rails tests and cucumber test.
2. For some models, validates, uniqueness and format tests are necessary.
3. There will be at least 3 or 4 cucumber tests for each controller.
4. After all tests have passed, start rails service and test the view, links and buttons of website.

5.2 Model Unit Test

Only the wine product model and order model have their own attributions, so in this part I only test these two models by using fixtures, the database independent written in .yml files. There is one YAML file per model and I create one object in each .yml file.

In my model unit test, the validates of all attributes, format of emails and uniqueness of title for Product model and Order model (without title) all have passed.

5.3 Cucumber and UI Test

I use Cucumber to test most of functions in controllers. Basic functions like index/create/delete of each controller will be tested. And more, some special requirement functions will also be included here. For example: search, checkout, login/logout and etc. But some simple methods like calculate total price won't be tested here. The result of cucumber is:

16 scenarios (16 passed)
61 steps (61 passed)
0m0.608s

The UI test (functional test) is easy but necessary. This is the double check that the buttons, links, display wines and other functions can run totally correctly as I want. The details of functional test can can be found in screencast. And all the results of Model, Functional and most of cucumber test will be given in the Test Table section.

5.4 System Test Table

MyAlcoholFreeWine System Test Table

Requirement	Description	Input	Expected Output	Pass/Fail
FR1	**Functional Test:** Customer can browse non-alcoholic wines	n/a	All wines will display	**P**
FR1	**Model Test:** All attributions should be valid	Using Fixtures: PD1	Can see product_test pass	**P**
FR1	**Model Test:** Title should be uniqueness / Image has special format	Using Fixtures: PD1 / PD2	Can see error of title and image	**P**
FR1	**Cucumber Test:** Product can be created/edit/ delete	Use Factory Girl	All scenarios and steps are green	**P**
FR2	**Cucumber Test:** Search function can be used from all data	Table new wines: red, blue, red blue, white	Can get result: red and red blue wine.	**P**
FR2	**Functional Test:** Customer can search wines that matched	Text Input: "Red" / "Rose"	Only wines' title are 'Red' / 'Rose' will display	**P**
FR3	**Functional Test:** Customer can display details	Click wine image	All the information of this wine will display	**P**
FR4	**Cucumber Test:** Once click add wine button, a line_item will be created. A new cart will display.	Feature: LineItem1, LineItem 2 and Cart	The scenario of 'Create LineItem', 'Index LineItem' and 'Create Cart' will be green.	**P**
FR4	**Functional Test:** Customer can add one wine to cart when they browser wines	Click 'add to cart' button	One wine will be added to the cart	**P**
FR4	**Functional Test:** Customer can choose quantity	Go to detail page click quantity choice button	The quantity of wines will be added to cart	**P**
FR5	**Cucumber Test:** Once click add wine button, a cart will be created.	Feature: Cart, LineItem	The LineItem can be see in this Cart.	**P**
FR5	**Cucumber Test:** Once click empty button, all line_items will be deleted	Feature: Cart, LineItem 3	The LineItem3 will destroy and 0 items in cart	**P**
FR5	**Function Test:** Customer can empty cart	Click 'empty cart' button	All wines in the cart will be disappeared	**P**
FR5	**Functional Test:** Cart will display on the right	n/a	A cart with wines will always show on the right	**P**
FR6a	**Cucumber Test:** Order can be created/index	Feature: Order1, Order2 and Order3	The scenario of 'Create Order' and 'Index Order' will be green.	**P**
FR6a	**Cucumber Test:** Once one order is created, the cart will be destroy	Feature: Cart, Order3	The scenario of 'Create Order' and 'Empty Cart' will be green.	**P**

Figure 8: Test Table 1

5.5 System Test Table

Requirement	Description	Input	Expected Output	Pass/Fail
FR6a	**Functional Test:** Checkout with login	Click checkout button	Customer will go to checkout page.	P
FR6a	**Model Test:** All attributions filled should be valid	Using Fixtures: OD1	Can see order_test pass	P
FR6a	**Model Test:** Email have special format	Using Fixtures: OD2	Can see error of email	P
FR6b	**Functional Test:** Customer need to login before check out	Do not login and click 'checkout' button	Customer will go to the login page.	P
FR7	**Cucumber Test:** Customer can use login/logout and registration function.	n/a	All scenarios in user.feature will be green	P
FR7	**Functional Test:** Customer can registration.	Click 'registration' link	Go to registration page and fill the users' details	P
FR7	**Functional Test:** Customer can login	Click 'login' link	Login page and ask to fill email and password	P
FR8	**Functional Test:** Users can logout	Click 'logout' link	Can see 'welcome' disappeared.	P

Figure 9: Test Table 2

6 Critical Evaluation

6.1 Overview

In this section, I will talk about scores of accurate self-evaluation, explain the reasons, problems I faced and Technologies Used.

6.2 Marking

6.2.1 Reason and Indication

- Screencast: In the screencast, the MAF application and two web services can be run well. I will show the all functions one by one. I want to make the screencast look good but the time is one minutes longer. So I can not get full score in this part.
- Documentation: I have spent lots of time in writing report and drawing figures. I have discussed all sections that should be noted and I think

I have done my best. But I am a foreign student, some sentences or grammars might not be very accurate.

- MAF application: This part is the one I concentrated on most. All the basic requirements are completed and can exchange data with service successfully. The problems are that the added functionality means added complexity, so I do not add more utility functions. One more thing, the function of page I do not add to this application. I totally forget it until I prepared to hand in the assignment. I do not have enough time to finish this part. If I do, I need to modify controller, search function ,view and cucumber test. So I only write here and change the score. This is the limit of high marks.

- Supplier web service: Totally understand RESTful and make a good implementations in another language. But only use GET and POST http requests and do not make a UI design although this is not requirement, it is still not perfect for me.

- Testing: The tests I made is more than requirements by using Rails test and cucumber. But not cover all areas.

- Evaluation: Have a right of evaluation of the advantages and disadvantages of my system.

- Flair: I have a mail service in my application. When users have finished order, the details of products will send to user's email. Second is Development to Heroku. The problem is 'sqlite3' is not allowed and I have to use 'pg' to replace 'sqlite3' in Heroku. The CSA example application worked well. But I do not want change anything in this application, I will use rails new -d postgresql as database next time if I want to push my application to heroku. Although I think the UI design and functions were implemented well. Low scores in this part.

6.2.2 Table

Part	Score
Screencast showing MAF and web services	9 (10)
Design as reported in the documentation	14 (20)
Implementation: MAF application	20 (25)
Implementation: Supplier web service	12 (15)
Test	12 (15)
Evaluation	5 (5)
Flair	3 (10)
Total	75 (100)

6.3 Other Information

6.3.1 Assignment Problems

1. Problems exist with RESTful architecture and ruby on rails for person never got hands on it at the beginning. After lectures and searching on Google make me understand better. I have to refer to the book Agile Web Development with Rail 4, this really help me a lot and I use the greet idea that create a 'LineItem' module to connect cart and product. This make program more flexible and provide good extensibility in the future development.

2. Ajax is still a big problem. I have followed lectures and some tutorials, but my web application still can not use Ajax. Actually I was thinking use Ajax for Cart and Search part but failed. I will focus more on this in the future.

3. MAF can GET new wine products from other web services and add them to database. The problem is these wines will still exist when is disconnected or only delete it in web service database. The wines disappear only when I run 'rake db:seed'. I know that this part is not required in this assignment and the solution is use DELETE http request. I just want to talk about this defect here.

4. I think I can not describe clearly what I want to express in the report, particularly when I try to write some professional sentence or opinions under academic requirements.

5. When user want to choose the wines' quantity, I set 10 digital (1 - 10) button as the quantity indicator. My opinion is that this function can make user choose quantity more convenient than addition/subtraction button or a field to input quantity number. The is the different way of implement function between my MAF and requirement.

6. Now, I can make good use of Rails to develop a simple web. But there are still lots of gems and technologies needs to learn about Ruby on Rails and RESTful web service.

6.3.2 Basic

- Technologies Used: Ruby 2.2.3, Rails 4.2.4, python 2.7, ShareLateX.
- References: Ruby, S. Thomas, D. and Hansen, H.H., 2013, Agile Web Development with Rails 4
- Total words: 2676