My Alcohol Free Wine

Daniel Clark

1 Introduction

This document will provide an overview of the Architecture, Rationales, Design and Testing of this implementation of the MyAlcoholFreeWine website, and it's defined Suppliers. From this point onwards MyAlcoholFreeWine will simply be referred to as MAF. The MAF is a price comparison site for My Alcohol Free Wines.

2 Architecture and Rationales of MAF

Due to the inherent nature of Rails, it logically made sense for the MAF application to use the Model View Controller design pattern. Rails makes using MVC really easy by providing generators for each part and convenience methods for each one.

2.1 UI Design

My initial UI design can be seen in the files UIDesign.svg and UIDesign.png. It gives a rough overview of what I was planning to do with my UI. These plans changed quite quickly however because for some reason my rails would not even generate default CSS for me, the files had only 2 lines of comments in. With that in mind I tried to implement as close as I could without CSS. I considered something like Bootstrap but in my opinion it adds a lot of design complexity with not that much gain. Following through the functional requirements and the marking grid there is no mention that user interface will have any bearing on marks for this work as the only mention is for a responsive design for mobile, which is flare marks. The design also changed slightly because my initial idea of having the basket on every page changed when the Gem I planned to use was built around having the basket in a separate view.

2.2 Database Design

My Database design can be seen in the files DBDesign.svg and DBDesign.png. It gives an overview to what my database structure looks like underneath. IT describes how the different models in my MAF application link together and what datatypes each piece of data is.

2.3 Software Design

The rough design for my MVC design pattern can be seen in SWDesign.svg and SWDesign.png. They outline which Models, Views and Controllers I use. In my design the user can also edit their account details

3 Architecture of Suppliers

I chose to develop the wine supplier applications in Python. This was mainly because I had experience with creating web servers in Python using the BaseHTTPServer.

I chose my endpoints to: /wines - GET - Returns a JSON list of JSON wine objects /wines/0 - GET - Returns the JSON wine object of ID '0' (or any ID specified) /wines/0/name - GET - Returns the Name of the wine with ID '0' (or any ID specified) /orders - POST - Adds the posted order to the suppliers orders, and returns an ID number, a list of the ordered wines, and a list of any wines which couldn't be ordered (because they don't exist or are out of stock).

The simplicity of the task meant that my supplier server only needed to deal with HTTP GET requests, for the getting of the available wines, and HTTP POST requests, for the ordering of wines. If I had more time I would have implemented HTTP DELETE to remove an order and HTTP PATCH to modify part of an order, e.g. the delivery address.

4 Conformance to Functional Requirements

FR1 (Browse non-alcoholic wines) has been fully implemented to the extent that all the pieces of data outlined in the specification can be viewed as part of a paginated list of all the wines available. The only slight adjustment to that is the images, however that is purely because the URLs for the images (which are stored in a JSON file on the supplier side and are easy to change) don't point to anything. Those URLs can be changed on the supplier and the change will propagate into the MAF within a minute thanks to the Whenever Gem integrating with Cron to supply a minutely update of the MAF database from the available suppliers.

FR2 (Search) has been fully implemented such that any search term can be entered and the site will display a paginated list of any items which match the search term in any field.

FR3 (Display Detail) is also fully implemented except with the same issue regarding images as described above.

FR4 (Add to Basket) has been implemented mainly thanks to the "acts_as_shopping_cart" gem which provides a good framework for building the basket.

FR5 (Display Shopping Basket) was implemented again thanks to the ease of use of "acts_as_shopping_cart".

FR6a (Checkout With Log-In) was almost completed however the checkout functionality does everything except actually post to the suppliers. This was purely due to time constraints. Most of the functionality for logging in was implemented with use of the "Devise" gem which provides an excellent authentication solution.

FR6b (Checkout Without Log-In) was easily completed thanks again to "Devise".

FR7 (Login/Logout and Registration) was, after a reasonable amount of work modifying the "Devise" gem to do what I needed, completed entirely.

5 Testing

My test table can be found in test_table.md My main strategy for my tests was to first test that the routing for the internal API is working correctly, and then go on to test the functionality.

The tests in my Test Table have been implemented using Ruby Tests and can be found in "ruby/test/controllers".

6 Self Evaluation

6.1 Screencast

The screencast for this covers all important aspects of the user interface exhibiting all functional requirements which have been implemented within the 5 minute period. After the 5 minute period there is an extra minute of review going over the features again just in case but as it is just a review, feel free to stop at 5 minutes. As I have covered every part of the specification for the screencast, I would give myself 10/10 for this section.

6.2 Design

My design for this project is probably quite lacking. This is mainly because of the nature developing software in Rails seems to work. It's based a lot around the use of Gems, which perform the functionality you need it to, and you're not necessarily aware of how it does it. Obviously Rails enforces good use of the MVC design pattern so the aspects of my program which fit into that have been documented as such in the diagrams. However they do not represent the system in much of a complete manner. The hardest bit of this section was trying to represent all the different parts of the system which had been generated/created. I'd give myself a mark of 12/20 for this section.

6.3 Implementation of the MAF

I think my implementation of the MAF site fits the functional requirements well. There is only one minor sections missed, like displaying images correctly, and one rather major

part missed, which is the posting of orders to the suppliers, but that section is still mostly implemented. The hardest part of this section was the sheer number of small/simple parts which needed to be added, none particularly difficult, but there was a lot to it. Another difficult part for me was getting the actual price comparison part of the MAF working. My code quality is good however it is very lacking in comments. For this section I'd give myself 20/25.

6.4 Implementation of Suppliers

I think my implementation of the wine suppliers fits the functional requirements well. The only place it falls short is that the Ordering endpoints have not been tested as the MAF application was not able to send POST requests to the suppliers. Endpoints involving each individual piece of information about a wine were also implemented, for example /wines/0/name which allows for easier management when only part of the data has changed. The hardest part of this was trying to get trying to get the order POST api working, which I believe I managed in the end, just not in time to integrate it with the MAF. Again comments are not particularly good. I'd give myself a percentage mark of 12/15.

6.5 Testing

I think that the tests I have written are a good start to a test framework, but there are very few. This is actually because there are very few controllers at this point, and each one doesn't do particularly much. The hardest part of this was learning the syntax of how the tests work. I think for the Testing section I would get 7/15.

6.6 Evaluation

I have critiqued every part of the marking grid and given my feedback on each section, so I think for this section I should get 5/5.

6.7 Flair

I added a few bits of extra functionality, like the ability to edit a users account (email, address, password, etc). I also have the necessary code on the supplier side to support SSL. These together I think could give me a flair mark of 2/10.

References

[1] Manu S Ajith, *acts_as_shopping_cart_app*, (2014), GitHub repository, https://github.com/crowdint/acts_as_shopping_cart_app/