SE31520: 'MyAlcoholFreeWine.com'

Due on Monday, December 7, 2015

James Euesden - jee22

Contents

Introduction

I had to do this [1]

MAF Architecture

Design

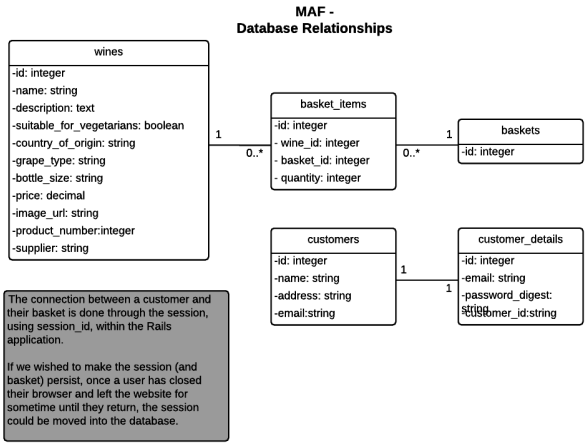

Figure 1: The intended design of the database and relations between tables in the MAF application.

Justify why you built it this way What is a session? Basket Id and Customer Id for use in login and basket while they are there How do you get the Wines? Wines stored with supplier - Supplier in config for if it was deployed, but could also be in Database. Which is better? Thin Controllers and models - Keep logic seperate, DRY principles?

MVC

```
1 can have code here
```

Web Service Architecure

Simple web service has two RESTful routes, Orders and Wines. Get Wines with /wines.json, and send Orders with /orders. As RESTful resources, we don't need to direct to a route like 'send_order' or 'get_wines'. Just making a GET request on the wines.json returns us the Wines (since last updated, based on headers), and sending a POST to /orders with the json data for a new order automatically routes to 'create' in the Orders controller. Just adds the order as a record into the database.

Test Strategy

System Testing

Include lots of screenshots along with Test Strategy Table

Unit Test for models and controllers

Demonstrate implementation of own Unit Tests for models and controllers (show test executions passing, show test names)

Self-Evaluation

Mark Breakdown

Screencast

Showed 2 Web Service Suppliers running, showed MAF running with all functionality discussed in this report. Expanded on this to show tests passing, Search in depth, etc Mark: /10%

Design

Design diagrams for both how I expected MVC to look and also for my database ideas, along with improvements. Explanation for my designs present and meeting the requirements, backed up through screencast. Documentation quality is okay, could be more in-depth, and design could be improved, but I'm new to Rails and need more experience. Mark: /20%

Implementation: MAF

Application runs, and meets the requirements (except quantity indicator when adding to basket). Rails controllers make calls to models, and views receive the data from the controllers in order to display it. Code is commented, attributions to code where used and identifier names speak for themselves. Mark: /25%

Implementation: Web Service

Web Services run, code has comments and names are good. RESTful resources, wines and orders, can call them and the routing sends the request to the controller through the type of request. Mark: /15%

Testing

Unit tests for each controller, more than just the ones generated by rails, and tests for the models, validations and own methods too. System test table matches assumed customer expectations. Results are discussed and based on these what could be taken forward if this prototype were to be fully implemented Mark: /15%

Evaluation

Full evaluation of report, MAF and web service, with summary of evaluation below the mark breakdown, with what was learned, difficult and easy. Mark: /5%

Flair

Implementation of Search using Solr with Sunspot to search partial words from all sections, editing the schema to change the gram of the search. Calling the WebService asynchronously with SuckerPunch, scheduling it to update with rufus-scheduler for hands off updating without interrupting the customer as they browse the website. Mark: /10%

Total

Mark: /100%

Summary

I think I met all of the functional requirements as requested (bar quantity indicator in FRwhatever), and the resources as RESTful and data is represented in the Rails way. I know there are areas for improvement (security, clean up, persistent sessions and baskets, better tests, cucumber tests, etc), but as far as the time allowed, my understanding and learning of Rails and this assignment tasking me to just create a prototype, I am happy with the resulting program, and happy I could learn more about Rails and Ruby, how to implement an MVC pattern, talking between web services and even implement external features, like learning configurations of Solr for my search. I found the most difficult part to be implementing routing, and getting my links in the views to exectue the correct controller actions as I wanted (for the more complex tasks, such as removing the last item from a basket, logging in and sending the webservice order). Before this assignment I had little working knowledge of rails outside of the workshops and a small amount of experience with routing, and doing this assignment helped me get my head around routing and how it works, and has helped me feel comfortable going forward in the future to building more Rails apps when I want to make a web application.

References

[1] Chris Loftus, "MyAlcoholFreeWine.com",SE31520/CHM5820 Assignment 2015-16, October 27 2015