Testing Spec
Project:
Cafeteria Management System:
Resolve

T-RISE
Rendani Dau (13381467)
Elana Kuun (12029522)
Semaka Malapane (13081129)
Antonia Michael (13014171)
Isabel Nel (13070305)

July 13, 2015

Contents

1 Introduction **4**

2 Vision **4**

3 Background **4**

3.1 Test Plan 5
3.1.1 Introduction 5
3.1.2 Technologies used for testing 5
3.1.3 How to run the unit tests 5
3.2 Testing approach 6
3.3 Tests conducted/ Test coverage 6
3.3.1 Register Module 6
3.3.2 Profile Module 6
3.3.3 Authentication Module 7
3.4 Test Case Description 8
3.4.1 Test Items 8
3.4.2 Input Specification 8
3.4.3 Output Specification 8
3.4.4 Environmental Needs 8
3.4.5 Special procedural requirements/rules 8
3.4.6 Intercase dependancies 8
3.5 Test Summary Report 8
3.5.1 Summary 8
3.5.2 Variances 8
3.5.3 Comprehensive assessment 8
3.5.4 Summary of results 8
3.5.5 Evaluation 8
3.5.6 Summary of activities 8

Document Title	Testing Documentation
Document Identification	Document 0.0.2
Author	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael
Version	0.0.2
Document Status	Second Version - contains tests for settings, password, superuser, and authentication controller, and models

Version	Date	Summary	Authors
0.0.1	29 May 2015	First draft contains first two use cases	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel,
0.0.2	9 July 2015	Second draft adding Testing for models and controllers for settings, password, superuser and authentication	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel

1 Introduction

This document contains the testing for the Resolve Cafeteria Management System that will be created for Software Engineering (COS 301) at the University of Pretoria 2015, by the group T-RISE. In this document we will thoroughly discuss and layout the project's testing to provide a clear view of the system as a whole. An agile approach is being followed which involves an interactive and incremential method of managing and designing the system.

2 Vision

The vision of this project is to implement a flexible, pluggable, fully functional software application that will be maintainable, with detailed supporting documentation and an instruction manual for the Cafeteria Management System. This system will assist in managing the cafeteria's inventory/stock, executing orders from the cafeteria, generating bills and sending these to the appropriate parties and facilitating payments for access cards (or the use of unique access card numbers).

3 Background

As specified in the project proposal document from Resolve - the cafeteria is currently cash only and does not accept bank cards or electronic payments. This makes it inconvenient for employees as they have to carry around cash if they want to purchase anything from the cafeteria. Hence, this is equivalent to purchasing from an external food outlet where they can also pay with their preferred method of payment. The employees have to hence use up fuel and time and lastly this does not bring in the maximum amount of income to the cafeteria, hindering its growth and improvement.

Resolve is therefore looking for a means to accept payments from employees for the canteen using their employee access cards or access card numbers, with an amount being deducted from their salary at the end of the month.

Resolve proposed the Cafeteria Management System to assist with this problem. After our first meeting with the client, they brought to our attention that at times the cafeteria does not even have enough stock to provide some of the menu items, thus the managing of inventory or stock will also be part of the system. The system will also predict what inventory/stock needs to be bought for the next week in order to avoid such a problem. At

the end of each month, the bill for the month will be sent to either payroll or to the employee. This option is configurable from the user's profile. The employee can also set a spending limit for each month for control purposes. The system will have its own maximum, such that users cannot set a limit that exeeds this.

3.1 Test Plan

3.1.1 Introduction

The scope of the testing: This document will be used be the Team t-rise to elaborate and substantiate the unit tests conducted for the various use cases/ modules of the Resolve Cafeteria Management System. Unit tests using mocks and Integration tests using actual data from the system have been conducted and will be explained in the document below. Tests for the server side functionality as well as the client side functionality were executed.

3.1.2 Technologies used for testing

Testing was done during the construction of the various functions to ensure that the code is working at all times, and that only fully working functions are used in other functions going forward. PhantomJS was used to run the Mocha, Karma and Jasmine tests automatically, hence the tests for the different files could be run consecutively without being manually executed seperately. The reason the testing frameworks were used was due to the simple set up that it required and due to its compatibility with PhantomJs. In addition the syntax of these was simple and intuitive. Karma in particular is the official Angular Js test runner, hence it was used. Mocha, is based on node.js and hence was used for this reason.

3.1.3 How to run the unit tests

To execute the tests one must first ensure that the Mongo database is running. After this, one opens a seperate terminal inside the Cafeteria Management System. Then sudo npm tes is run which executes PhantomJs to run the tests automatically. The output will indicate whether the tests are passing or failing as well as the different descriptions for each test. Server side and client side tests are executed.

3.2 Testing approach

The unit tests written for the Authentication module were done to test the functions inside authentication.client.controller.js and the superuser.client.controller.js file. The Register/ Sign up functionality for the register module resides in the authentication.client.controller.js file. The unit tests written for the Manage Profile module were done to test the settings.client.controller.js and the password.client.controller.js files, which contained the code to implement the manage profile module.

The functions tested for the Authentication module include signin(), located in the authentication.client.controller.js file, assignRoles(), setSystemWideLimit(), and setCanteenName() located in the superuser.client.controller.js file.

The function signup() was tested for the Register module module, located in the authentication.client.controller.js file.

The functions changeUserPassword() from the settings.client.controller.js and askForPasswordReset() from password.client.controller.js were tested. resetUserPassword() was not tested due to the fact that each time a user clicks Forgot Password, an email is sent to the user

3.3 Tests conducted/ Test coverage

Test for the controllers:

3.3.1 Register Module

1.) Sign up:
Pre condtions: User entered valid credentials, Username/ Employee Id does not exist on the system
Post conditions: User is signed up to use the system
Test Case 1 - the function should register user with correct data: Pass
Test Case 2 - should fail to register with duplicate Username: Pass

3.3.2 Profile Module

1.) Change Password Pre condtions: Valid password entered
Post conditions: Password has changed
Test Case 1: should let user change their password if a valid one was entered: Pass
Test Case 2: should send an error message if new password is too short : Pass
Test Case 3: should send an error message if the current password is incorrect:

Pass
Test Case 4: should send an error message if the passwords do not match: Pass

3.3.3 Authentication Module

1.) Sign in :
Pre condtions: Valid credentials have been entered, the user exists on the system
Post conditions: User has been signed in
Test Case 1: should login with a correct user and password: Pass
Test Case 2: should fail to log user in if nothing has been entered: Pass
Test Case 3: should fail to log user in with wrong credentials: Pass
2.) Assign Roles -
Pre conditions: User id entered by superuser exists in the system
Post conditions: Role has been assigned to user by superuser
Test Case 1: should let superuser assign roles because the user id entered exists on the system: Pass
Test Case 2: should not let superuser assign roles because user id entered does not exist on the system: Pass
3.) Set system wide limit -
Pre conditions: Limit is valid
Post conditions: Limit has been set
Test Case 1: should allow superuser to set limit: Pass
4.) Set canteen name- Pre conditions: N/a
Post conditions: Canteen name set successfully
Test Case 1: should let user set canteen name: Pass

3.4 Test Case Description

3.4.1 Test Items

3.4.2 Input Specification

3.4.3 Output Specification

3.4.4 Environmental Needs

3.4.5 Special procedural requirements/rules

3.4.6 Intercase dependancies

3.5 Test Summary Report

3.5.1 Summary

3.5.2 Variances

3.5.3 Comprehensive assessment

3.5.4 Summary of results

3.5.5 Evaluation

3.5.6 Summary of activities

//////////////commented out for now

Testing for the two main use cases discussed in this document has been done using Mocha for unit testing. We have tested the various functions identified for the two different use cases. For placing an order, the functions that are to be created are checkProductAvailability, checkLimits, isLoggedIn and generateBill.

Once the user has selected items to be purchased, the system needs to check that the user has logged in, so that using the employeeId the system will know which user's profile needs to be edited. The system has to then check whether the products are in stock and whether the total price of the items selected stays within the user's set limit. The bill is then generated, stating the items purchased and the cost.

These functions will return strings indicating whether the desired operation associated with them was successfully carried out or not. Mock Json objects consisting of the different parameters required for each function have been passed to the function in the tests. If the functions in the placeOrder.js file pass all these tests, the file will be able to be integrated into the rest of the system.

Due to the fact that an agile approach is being used, these unit tests might

need to be edited if the functional requirements are edited along the way. Screenshots of some of the unit tests that have been created are displayed below. The entire unit test file can be found in the github repository in the unit test folder.
The second use case is manage profile, and the functions that have thus far been identified for this use case are editProfile, changePassword, changeEmail, setLimit, editRecipients, displayBill, viewHistory and generateFavourites.
The system must hence allow the logged in user to change their various settings, as well as allow the user to view the day's bill as well as the account history.
A sample of the code from the unit testing file is displayed below.