Testing Document

Cafeteria Management System: Resolve Solution Partners (Pty) Limited

Client: Gareth Botha and Jaco Pieterse

T-RISE

Rendani Dau (13381467)

Elana Kuun (12029522)

Semaka Malapane (13081129)

Antonia Michael (13014171)

Isabel Nel (13070305)

September 25, 2015

Contents

Document Title	Testing Documentation
Document Identification	Document 0.0.6
Author	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael
Version	0.0.6
Document Status	Sixth Version - contains added tests for menu items controller and reporting as well as updates to other tests

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

3.1 The current situation/ problems the client currently experience

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 is it inconvenient for employees as they have to carry around cash if they want to purchase anything from the cafeteria. Employees may choose to go to an external food outlet where they can pay with their preferred method of payment, which uses time and fuel. Thus, this means the cafeteria does not achieve the maximum amount of income which hinders its growth and improvement.
A problem with the cafeteria itself is that certain meal items are hardly in stock due to either lack of ingredients to make the meal or under estimating the quantity of the meal item required.

3.2 How the aforementioned problems will be alleviated by the CMS

The Cafeteria Management System will provide a means to accept payments from employees, at 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. The option of cash payments ,however, will not be discarded. At the end of each month, the bill for the month will be sent to either payroll, to the employee, or to both. This option is thus configurable from the user's profile. The employee can also set a spending limit for each month. There will also be a system wide limit that users cannot exceed.
The system will predict which inventory items needs to be bought for the next week in order to avoid the "out of stock" situation described above. The system will also enforce that the cafeteria manager

4 Test Plan

4.1 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.

4.2 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 separately. 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.

4.3 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 separate terminal inside the Cafeteria Management System. Then sudo npm test 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.

4.4 Testing approach - Client Side

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/ signup functionality also 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() and signup(), located in the authentication.client.controller.js file, assignRoles(), setSystemWideLimit(), and setCanteenName() located in the superuser.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

4.5 Testing approach - Server Side

The various functions from each module were tested on the server side, using the save() function to test if the items were able to be saved on the database. For the unit tests, mock objects were created, storing data under the different table column headings.
The pre and post conditions, test cases and the result of the tests will now be discussed.

4.6 Tests conducted/ Test coverage - Client Side

Test for the controllers:

Profile Module - settings.client.controller.js

The tests can be found in settings.client.controller.test.js 1.) Change Password Pre conditions: Valid password entered
Post conditions: Password has changed
Test Case 1: $scope.changeUserPassword() should let user change their password if a valid one was entered: Pass
Test Case 2: $scope.changeUserPassword() should send an error message if new password is too short : Pass
Test Case 3: $scope.changeUserPassword() should send an error message if the current password is incorrect: Pass
Test Case 4: $scope.changeUserPassword() should send an error message if the passwords do not match: Pass

Authentication Module - authentication.client.controller.js

The tests can be found in authentication.client.controller.test.js 1.) Sign in :
Pre conditions: Valid credentials have been entered, the user exists on the system
Post conditions: User has been signed in
Test Case 1:$scope.signin() should login with a correct user and password: Pass
Test Case 2: $scope.signin() should fail to log user in if nothing has been entered: Pass
Test Case 3: $scope.signin() should fail to log user in with wrong credentials: Pass
2.) Sign up:
Pre conditions: 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 - $scope.signup() should register user with correct data: Pass
Test Case 2 - $scope.signup() should fail to register with duplicate Username: Pass

Manage System Module - superuser.client.controller.js

1.) 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: $scope.assignRoles() should let superuser assign roles because the user id entered exists on the system: Pass

Test Case 2: $scope.assignRoles() should not let superuser assign roles because user id entered does not exist on the system: Pass
Test Case 3: $scope.assignRoles() should redirect if superuser role is assigned: Pass

2.) Set system wide limit -
Pre conditions: Limit is valid
Post conditions: Limit has been set
Test Case 1: $scope.setSystemWideLimit() should allow superuser to set limit: Pass

3.) Set canteen name-
Pre conditions: N/a
Post conditions: Canteen name set successfully
Test Case 1: $scope.setCanteenName() should let user set canteen name: Pass

4.) Change Employee ID-
Pre conditions: New employeeID doesn't exist currently, new employeeID field is not blank
Post conditions: Employee ID updated
Test Case 1: $scope.changeEmployeeID()should not change employeeID if the user is not in the database: Pass
Test Case 2: $scope.changeEmployeeID() should change employee ID: Pass

5.) Search Employees
Test Case 1: $scope.searchEmployee() should search for employee by ID: Pass
Test Case 2: $scope.searchEmployee() should not search for employee if emp id blank: Pass
Test Case 3: $scope.searchEmployee() should not search for employee by ID: Pass

Manage Cafeteria Module - menuItems.client.controller.js

1.)Update Menu Items
Pre conditions: information entered is valid
Post conditions: menu item updated
Test Case 1: should update menu item: Pass
Test Case 2: should not update menu item: Pass

2.) Search Menu items
Pre conditions: menu item exists in the database
Post conditions: menu item found
Test Case 1: $scope.updateMenuItem() should update menu item: Pass
Test Case 2: $scope.updateMenuItem()should not update menu item: Pass
Test Case 3: $scope.searchMenu() should search and find menu item: Pass
Test Case 4: $scope.searchMenu() should search and NOT find menu item: Pass

3.) Add Menu item
Pre conditions: menu item does not exist
Post conditions: menu item added
Test Case 1: $scope.createMenuItem() should create menu item: Pass
Test Case 2: $scope.createMenuItem() should fail to create menu item: Pass

4.) Load menu items
Pre conditions: menu items must exist in the database
Post conditions: menu items will be loaded
Test Case 1: $scope.loadMenuItems() should load menu items: Pass

5.) Create a menu category
Pre conditions: Category does not exist
Post conditions: Created successfully, added to database
Test Case 1: $scope.createMenuCatagory() should create category: Pass
Test Case 2: $scope.createMenuCatagory() should not create category: Pass

6.) Load Menu Categories
Pre conditions: One/ more than one menu category exists in the database
Post conditions : Categories have been loaded and are displayed on navigation bar Test Case 1: $scope.loadMenuCategories should load categories: Pass
Test Case 2: $scope.loadMenuCategories should fail to load categories: Pass

7.)Check Stock
Pre conditions: There are menu items and inventory items saved
Post conditions: The inventory needed for the menu item is either enough to be used for the quantity of items specifed or not. If not it is marked as out of stock
Test Case 1: $scope.checkStock should successfully perform a check: Pass

8.) Delete Menu Items

Pre conditions : Menu item exists in the database
Post conditions: Menu item successfully deleted or error message returned
Test Case 1: $scope.deleteMenuItem should delete menu item
Test Case 2: $scope.deleteMenuItem should NOT delete menu item

Reporting module- finance.client.controller.js

The tests for this file can be found in finance.client.controller.tests.js. Pre Conditions: User has been found, user ID exits in the database
Post Conditions: Report has been downloaded in pdf format
Test Case 1: $scope.generateReport should not generate report if missing fields: Pass

Place order Module - orders.client.controller.js, cashier.client.controller.js , finance.client.controller.js

1. The first set of tests can be found in orders.client.controller.test.js.
   Pre Conditions: A valid user has logged in. The user has sufficient funds to account for the order placed.
   Post Conditions: The order is sent through to the cashier.
   Test Case 1: $scope.placeOrder() should not allow order to be placed: Pass
   Test Case 2: $scope.placeOrder() should allow order to be placed: Pass

2. The second set of tests can be found in cashier.client.controller.test.js.
   Pre Conditions: The user facilitating the orders is the cashier and is hence on the process orders page that is only accessible to the cashier
   Post Conditions: The order has been marked as ready or closed.The user is notified when order is ready
   Test Case 1: $scope.markAsReady() should let user know order is ready: Pass
   Test Case 2: $scope.markAsReady() should not let user know order is ready when incorrect parameters sent to function: Pass
   Test Case 4: $scope.markAsPaid() should mark the order as paid successfully: Pass
   Test Case 5: $scope.markAsPaid() should mark the order as collected successfully: Pass
   Test Case 6: $scope.getOrders() should get a list of orders where status is open: Pass

Test Case 7: $scope.getOrders() should not get a list of orders where status is closed: Pass

3. The third set of tests can be found in finance.client.controller.test.js.
   Pre Conditions: The user facilitating the orders is the finance manager and is hence on the process orders page that is only accessible to the finance manager.
   The employee ID has been entered and is valid
   Post Conditions: The bill corresponding to the Employee ID entered is generated
   Test Case 1: $scope.getUserOrders() should get users orders: Pass
   Test Case 2: $scope.getUserOrders() should not get users orders: Pass

4.7 Tests conducted/ Test coverage - Server Side

Authentication Module - user.server.model.js

The tests for this file can be found in user.server.model.test.js. Pre Conditions: The fields are all filled in with valid credentials, The system starts off with no users
Post Conditions: The user has been saved in the database / Error message sent
Test Case 1: should begin with no users: Pass
Test Case 2: should be able to save without problems: Pass
Test Case 3: should fail to save an existing user again: Pass
Test Case 4: should be able to show an error when trying to save without first name: Pass
Test Case 5: should be able to show an error when trying to save without last name: Pass
Test Case 6: should be able to show an error when the password is too short: Pass
Test Case 7: should be able to show an error when the passwords do not match: Pass
Test Case 8: should show an error when the email does not contain an @ sign and the email is in correct format: Pass

Manage Inventory Module - inventory.server.model.js

The tests for this file can be found in inventory.server.model.test.js.
Pre Conditions: The inventory item does not exist currently in the database, all the fields have been filled in with valid information, the system does not have inventory stored at the start
Post Conditions: The inventory item has been added to the database/ Error message sent
Test Case 1: should begin with no inventory : Pass
Test Case 2: should be able to save without problems : Pass
Test Case 3: should fail to save an existing inventory item again : Pass
Test Case 4: should be able to show an error when trying to save without inventory name : Pass
Test Case 5: should be able to show an error when trying to save without quantity : Pass
Test Case 6: should be able to show an error when trying to save with incorrect unit : Pass

Place Orders Module - menuItem.server.model.js

The tests can be found in menuItem.server.model.test.js.
Pre Conditions: There are no items in the menu database to start off with, All required fields are filled in with valid information
Post Conditions: The menu item has been saved to the database/ Error message sent
Test Case 1: should begin with no menu items: Pass
Test Case 2: should be able to save without problems: Pass
Test Case 3: should fail to save an existing menu item again: Pass
Test Case 4: should be able to show an error when trying to save without menu item name: Pass
Test Case 5: should be able to show an error when trying to save without description: Pass
Test Case 6: should be able to show an error when trying to save without price: Pass
Test Case 7: should be able to show an error when trying to save without category: Pass
Test Case 8: should be able to show an error when trying to save with incorrect category: Pass
Test Case 9: should be able to show an error when trying to save without ingredients: Pass

Database Storage Test - test.server.model.js

The tests for this file can be found in test.server.model.test.js. Pre Conditions: Name has been filled in
Post Conditions: Item saved to database/ Error message sent: Pass
Test Case 1: should be able to save without problems: Pass
Test Case 2: should be able to show an error when try to save without name: Pass

Authentication module- test.server.routes.js

The tests for this file can be found in test.server.routes.test.js. Pre Conditions: User has logged on, User has signed in
Post Conditions: Test instance added/deleted/updated/ list of items retrieved or Error message sent
Test Case 1: should be able to save Test instance if logged in: Pass
Test Case 2: should not be able to save Test instance if not logged in: Pass
Test Case 3: should not be able to save Test instance if no name is provided: Pass
Test Case 4: should be able to update Test instance if signed in: Pass
Test Case 5: should be able to get a list of Tests if not signed in: Pass
Test Case 6: should be able to get a single Test if not signed in: Pass
Test Case 7: should be able to delete Test instance if signed in: Pass
Test Case 8: should not be able to delete Test instance if not signed in: Pass

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 client side controllers for settings, password, superuser and authentication	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel
0.0.3	20 July 2015	Third draft adding Testing for models and controllers for inventory, placedOrders register and authentication	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel
0.0.4	23 July 2015	Fourth draft adding Testing for models and controllers for manage cafeteria and manage system	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel
0.0.5	25 August 2015	Fifth draft adding Testing for orders, finance and cashier controllers and updated spec	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel
0.0.6	24 September 2015	Sixth draft adding Testing for reporting, and and menu items controllers and updates	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel