

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 22, 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 0.0.3
Author	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael
Version	0.0.3
Document Status	Third Version - contains tests for inventory, placedOrders, registration 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 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

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 incremental 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 exceeds this.

4 Test Plan

4.1 Introduction

The scope of the testing: This document will be used by the Team 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/ 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

4.5 Testing approach - Server Side

4.6 Tests conducted/ Test coverage - Client Side

Test for the controllers:

4.6.1 Register Module - `authentication.client.controller.js`

The tests can be found in `authentication.client.controller.test.js` 1.) 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 - the function should register user with correct data: Pass

Test Case 2 - should fail to register with duplicate Username: Pass

4.6.2 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: 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

4.6.3 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: 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

4.7 Tests conducted/ Test coverage - Server Side

4.7.1 Register and 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

4.7.2 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

4.7.3 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

4.7.4 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

4.7.5 Register and 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

