Testing Spec
Project:
Cafeteria Management System:
Reslove

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

August 28, 2015

Contents

Document Title	Design Requirements Document
Document Identification	Document 0.0.2
Author	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael, Isabel Nel
Version	0.0.2
Document Status	Second Version - added the function templates for the client controllers

Version	**Date**	**Summary**	**Authors**
0.0.1	29 May 2015	First draft contains first two use cases template methods and declarations	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel,
0.0.2	27 August 2015	Second draft adding all the client and controllers for the updated system	Rendani Dau, Elana Kuun, Semaka Malapane, Antonia Michael Isabel Nel

1 Introduction

This document contains the functional requirements specification, architecture requirements and 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 design requirements to provide a clear view of the system as a whole. An agile method is being followed so the following document focusses on the PlaceOrder and ManageProfile modules.

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.

4 Standards and conventions

4.1 Design standards

The diagrams are designed and created using UML. The main use case of the system is decomposed into components.

¡¡¡¡¡¡¡ HEAD =======

5 Authentication

5.1 something.client.controller.js

1. Declaration: angular.module('users').controller.......

2. Methods:

   -

6 Manage Profile

¿¿¿¿¿¿¿ origin/phase4

6.1 superuser.client.controller.js

1. Declaration:
   angular.module('users').controller('superuserController', ['$scope', '$http', '$location', '$window', 'Users', 'Authentication', function($scope, $http, $location, $window, Users, Authentication) { }

2. Methods:

   - Assign roles
     $scope.assignRoles = function(isValid) {}

Usage: Superuser can assign cashier, cafeteria manager, finance manager and admin roles

- Assign roles admin role
  $scope.assignRolesAdminRole = function(isValid) { }

  Usage: Admin user also has access to the assign roles functionality and serves as a back up superuser

- Change employee ID
  $scope.changeEmployeeID = function(Boolean isValid) {}

  Usage: The superuser can change the employee ID of the users if the user signed up with the incorrect ID or if the company changes the IDs.

- Remove employee
  $scope.removeEmployee = function(Boolean isValid) {}

  Usage: The superuser is also able to remove users from the system, due to resignation or dismissal for example

- Search employee
  $scope.searchEmployee = function(Boolean isValid) { }

  Usage: This function is used to retrieve employees from the system to be able to change their employe ID or remove them from the system

- Search employee ID
  $scope.searchEmployeeID = function(row) {}

  Usage: This function is used to retrieve employee IDs

- Set system wide limit
  $scope.setSystemWideLimit = function(Boolean isValid){ }

  Usage: This is used by the superuser to set the maximum monthly spending limit for all the users of the system.

- Set canteen name
  $scope.setCanteenName = function(Boolean isValid){ }

  Usage: The canteen name is configurable from the superuser's branding settings page.

- Check user
  $scope.checkUser = function(){}

  Usage: This is a security function that checks to make sure that the authorized superuser is accessing the page and if this is not the case, the user will be redirected to the home page

- Load employees
  $scope.loadEmployees = function(){}

  Usage: This function is used to dynamically populate the drop down menu for the change employee ID functionality

¡¡¡¡¡¡¡ HEAD =======

6.2 cashier.client.controller.js

1. Declaration:
   angular.module('users').controller('cashierController', ['$scope', '$http', '$stateParams', '$location', 'Authentication', function($scope, $http, $stateParams, $location, Authentication) {}

2. Methods:

   - Get orders
     $scope.getOrders = function(){}

     Usage: Function to obtain the list of orders that an order placed from the menu

   - Mark as ready
     $scope.markAsReady = function(username, orderNumber){}

     Usage: Function that the cashier uses to send a notification to the user notifying the user that their order is ready. The function also changes the status in the model from open to ready.

   - Mark as collected
     $scope.markAsCollected = function(username, itemName,orderNumber){}

     Usage: On the click of the 'Order collected' button on the cashier page, this function will be called. It changes the status of the order in the model from ready/open to closed.

- Mark as paid
  $scope.markAsPaid = function(username, itemName, orderNumber){}

  Usage: On the click of the ’Paid and collected’ button on the cashier page, this function will be called. It changes the status of the order in the model from ready/open to closed.
- Check user
  $scope.checkUser = function(){}

  Usage: This is a security function that checks to make sure that the authorized cashier is accessing the page and if this is not the case, the user will be redirected to the home page

6.3 cashier.client.controller.js

1. Declaration: angular.module(’users’).controller(’FinanceController’, [’$scope’, ’$http’, ’$location’, ’$stateParams’, ’Authentication’, function($scope, $http, $location, $stateParams, Authentication) {}
2. Methods:

   - Get user orders
     $scope.getUserOrders = function(){}

     Usage: Function to obtain the orders placed by the users via adding to plate from the menu page and proceeding to place that order. The finance manager will enter the employee ID of the user in the textbox displayed on the page and the orders placed by that user will be displayed on that page.
   - Check user
     $scope.checkUser = function(){}

     Usage: This is a security function that checks to make sure that the authorized cashier is accessing the page and if this is not the case, the user will be redirected to the home page

7 Manage System

8 Manage Cafeteria

9 Manage Inventory

¿¿¿¿¿¿¿ origin/phase4

10 Place Order

10.1 orders.client.controller.js

1. Declaration: angular.module('orders').controller('OrdersController', ['$scope', '$rootScope','$http', '$stateParams', '$location', '$cookies', 'Authentication', 'Orders', function($scope, $rootScope, $http, $stateParams, $location, $cookies, Authentication, Orders) {}

2. Methods:

   - Change item quantity
     $scope.quantityChange = function(){}

     Usage: Function to increase or decrease the number of items ordered. It is used when a user wants to order multiple of the same item. It then also increases or decreases the total price accordingly.
   - Change or add preferences
     $scope.prefChange = function(){}

     Usage: Function to add preferences. The user can specify how they prefer their order to be prepared.
   - Change the total
     $scope.subTotal = function(){}

     Usage: Function to calculate the total. This function calculates the total that will be displayed for the user at the bottom - so they can know how much their order is going to cost.
   - Place order
     $scope.placeOrder = function(){}

Usage: Function to place the order. If the user is not signed in, the function redirects them to the signin page. If the user is signed in, the function checks the order total against the user's available balance. If the user has enough funds in their account the order is placed. Otherwise a pop-up message tells the user they have insufficient funds in their account and that if they proceed with placing the order they will have to use cash - if the user agrees to use cash the order is placed otherwise nothing is done.

- Remove item from plate
  $scope.removeFromPlate = function(){}

  Usage: Function to remove an item from the plate. If the user decides they don't want an item anymore, they can remove it from the plate (before the order is placed).

10.2 orders.server.controller.js

1. Methods:

   - Place order
     exports.placeOrder = function(req, res){}

     Usage: Function to place the order. It adds the order to the orders table if the user is successful in placing the order, otherwise it generates the appropriate error message.
   - Mark the order as ready
     exports.markAsReady = function(req, res){}

     Usage: Function to mark the order as ready. The cashier can mark the order as ready when the items are done being prepared and ready to be collected. This changes the status of the order to ready and sends the client a notification to come fetch their order.
   - Send Email
     function sendEmail(uname, orderNum){}

     Usage: Function to send the user an email. This function is called in markAsReady - it sends the user an email letting them know their order (with the given orderNumber) is ready to be collected.

- Mark order as paid
  exports.markAsPaid = function(req, res){}

  Usage: Function to mark the order as paid and collected. This function marks the order as closed in the orders table and it removes it from the process orders page.
- Mark order as collected
  exports.markAsCollected = function(req, res){}

  Usage: Function to mark the order as paid and collected. This function marks the order as closed in the orders table and it removes it from the process orders page. It also subtracts the total of the order from the user's limit because the client only collected the order and didn't pay. This can only be used when the user has sufficient funds in their account.
- Get a list of orders
  exports.getOrderList = function(req, res){}

  Usage: Function to get a list of all the orders that are open and ready. This is used to display the orders that are still being processed and have not yet been collected on the process orders page (accessed by the cashier).

10.3 Manage Profile

var manageProfileObj = require('./manageProfile'); extends CMS
Constructors: manageProfile.exports=function(jsonObj) -usage: creates instances of this component
Methods:

- changePassword exports.changePassword =function(jsonObj) where jsonObj contains employeeID, password, newPassword

- changeEmail exports.changeEmail = function(jsonObj) where jsonObj contains employeeId, email

- setLimit exports.setLimit = function(jsonObj) where jsonObj contains employeeId,newLimit

- displayBill exports.displayBill = function(jsonObj) where jsonObj contains employeeId

- viewHistory exports.viewHistory = function(jsonObj)where jsonObj contains employeeId
- generateFavourites exports.generateFavourites = function(jsonObj) where jsonObj contains employeeId