Functional Requirements Document Specification Project: Cafeteria Management System: Resolve

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

https://github.com/toniamichael94/MainProjectCOS301

May 29, 2015

Contents

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 functional requirements to provide a clear view of the system as a whole. An agile approach is being followed, hence the main use cases that this document will be focussing on are placing orders and managing a user profile. These will be explored in quite some detail. The agile method 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, placing orders made by employees, generating bills, and sending the appropriate information to the right parties.

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 have cash on hand if they want to purchase anything from the cafeteria. Employees might choose to buy somewhere else where they can use another form of payment. The employees have to use fuel and time, and 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. The amount spent at the cafeteria can then be deducted from their salary at the end of the month.

After our first meeting with the client, they brought to our attention that at times the cafeteria does not have enough stock to provide some of the menu items, therefore the managing of inventory and 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 shortcomings. At the end of each month, the bill for that 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. The system will also have a maximum limit that users cannot exceed.

4 Functional Requirements and Application Design

In this section we will discuss the functional requirements.

4.1 Use Cases

Below is a list of all the use cases we have identified:

- Authentication
- Register
- Manage Profile
- Place Order
- Notify
- Manage Inventory
- Report

4.2 Use Case Prioritization

Below the use cases mentioned above will be catagorized as critical, important or nice to have.

4.2.1 Critical

- Authentication
  This is a critical use case due to the fact that you cannot send through any order if you are not logged in. In such a case you will only be able to view the menu.

- Register
  This is a critical use case because you can not log into the system if you have not been registered on the system. Furthermore, if you are not logged in, you can not place orders.
- Place Order
  This is critical due to the fact that the main functionality of the system revolves around the ability to place orders at the cafeteria as well as other activities relating to that.

4.2.2 Important

- Manage Profile
  This use case is considered important because the user must be able edit their profile by resetting their personal spending limits and changing their email and passwords. The user must also be able to view his/her account history and current bill. It is crucial that the user is able to configure spending limits according to their own preferences as well as keep track of monthly purchases.
- Manage Inventory
  This use case is considered important, because it deals with incrementing stock that has been added and decrementing stock when it is purchased or expired. It also deals with the cafeteria manager marking orders as completed.
- Reporting
  This use case is considered important because users must be able to print a billing report. They should also have the option of sending their montlhly bill to payroll where it will be deducted from their salaries. A history of all the orders that have taken place on the system can also be printed.

4.2.3 Nice to have

- Notify
  This is classified as nice to have as it includes notifying a user when their order is ready for collection, as well as other notifications, such as notifying the cashier or cafeteria manager that they are low on certain inventory. The system is not dependant on this functionality.

5 Modular System

The system will be built using a modular approach to allow more modules to be added at a later stage. This will also provide for pluggability and integratibility of the system.

5.1 High level use case diagram of the CMS

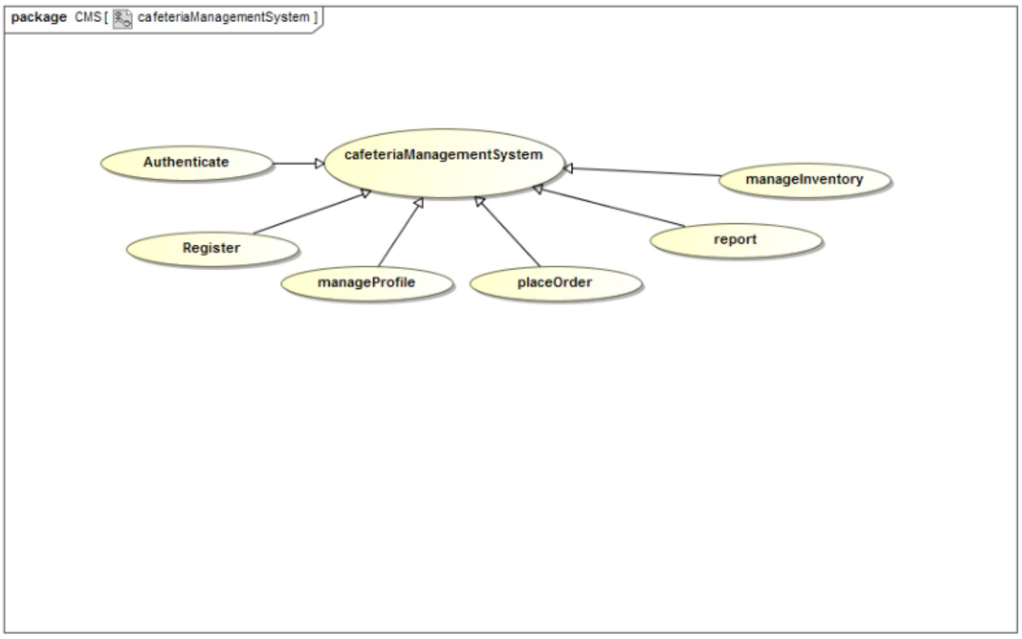
package CMS [cafeteriaManagementSystem]
cafeteriaManagementSystem
Authenticate
Register
manageProfile
placeOrder
report
manageInventory

Figure 1: Cafeteria Management System Use Case

The core of the system is a cafeteria management system that will provide functionality such as allowing users to place orders once they have registered and logged on to the system. Different types of users will have different privileges.

5.2 Place Orders Module

The main functionality the system serves to provide, is to allow the user to use their access card number to purchase food items from the cafeteria via the system. This use case diagram indicates the functionality around placing orders, such as CheckProductAvailability and checkLimits, which are the use cases of this module.

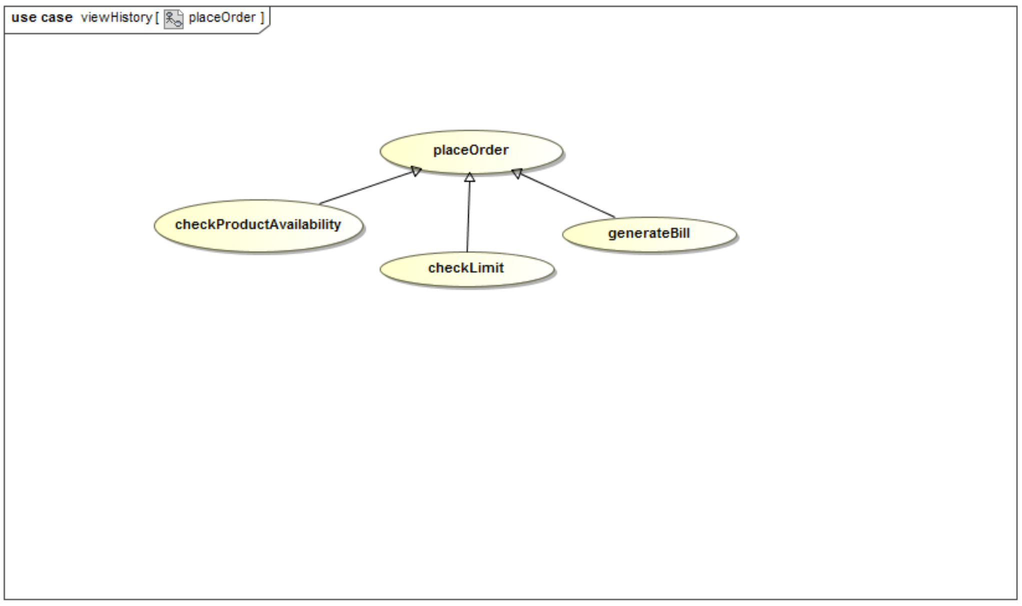

Figure 2: The use case for placing an order

5.2.1 CheckProductAvailability

The service contract and activity diagram for CheckProductAvailability follow. CheckProductAvailability falls under the use case for Place Orders (refer to page ..... to view this use case diagram). The system will check whether the product that the user has selected to purchase is currently in stock.

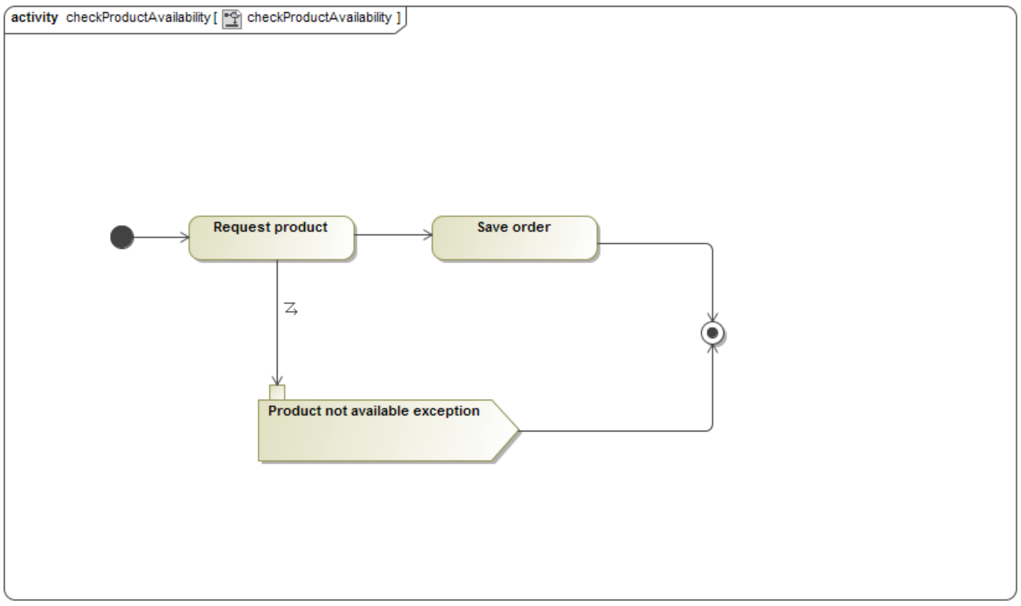
activity checkProductAvailability [checkProductAvailability]
Request product
Save order
Product not available exception

Figure 3: The activity diagram for checking product availibilty

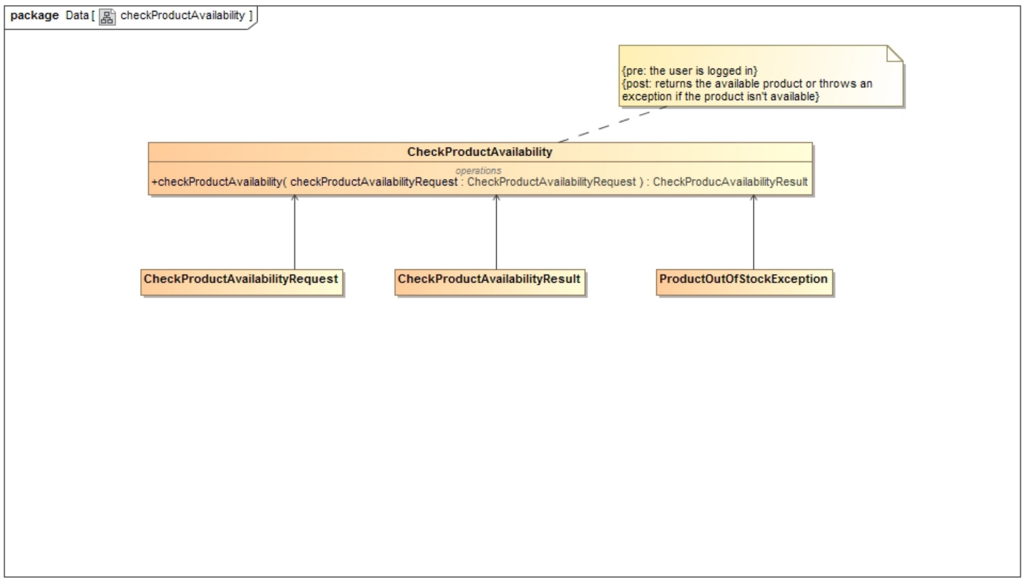
package Data [checkProductAvailability]
{pre: the user is logged in}
{post: returns the available product or throws an exception if the product isn't available}
CheckProductAvailability
operations
+checkProductAvailability(checkProductAvailabilityRequest : CheckProductAvailabilityRequest) : CheckProducAvailabilityResult
CheckProductAvailabilityRequest
CheckProductAvailabilityResult
ProductOutOfStockException

Figure 4: The service contract for checking product availabilty

5.2.2 checkLimit

The service contract and activity diagram for checkLimit follow. checkLimit falls under the use case for Place Orders (refer to page ..... to view this use case diagram). The system will be able to view the user's personal limit to make sure that the total of the bill is not larger than the users spending limit.

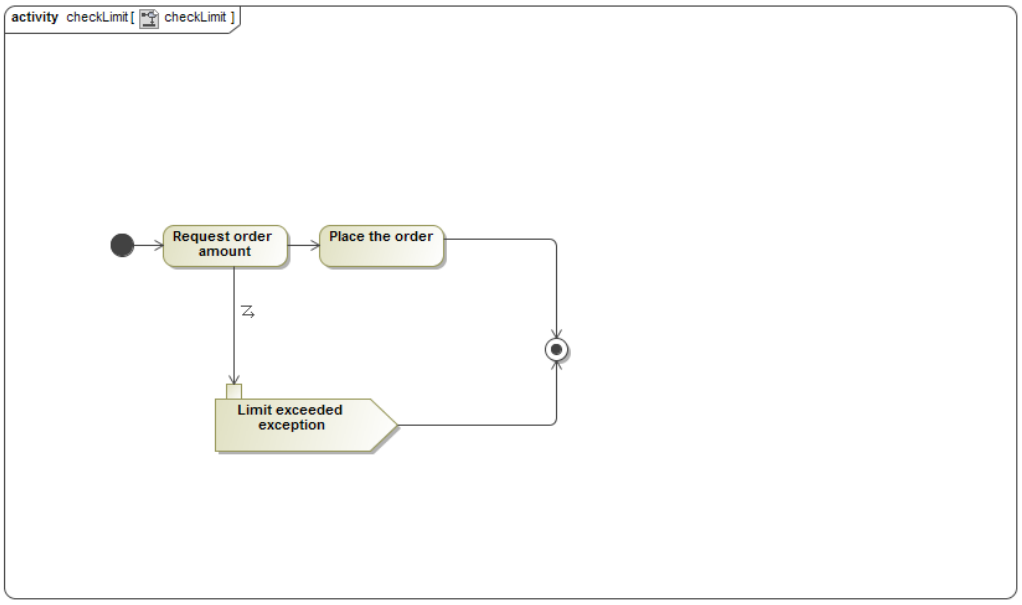

Figure 5: The activity diagram for checking limits

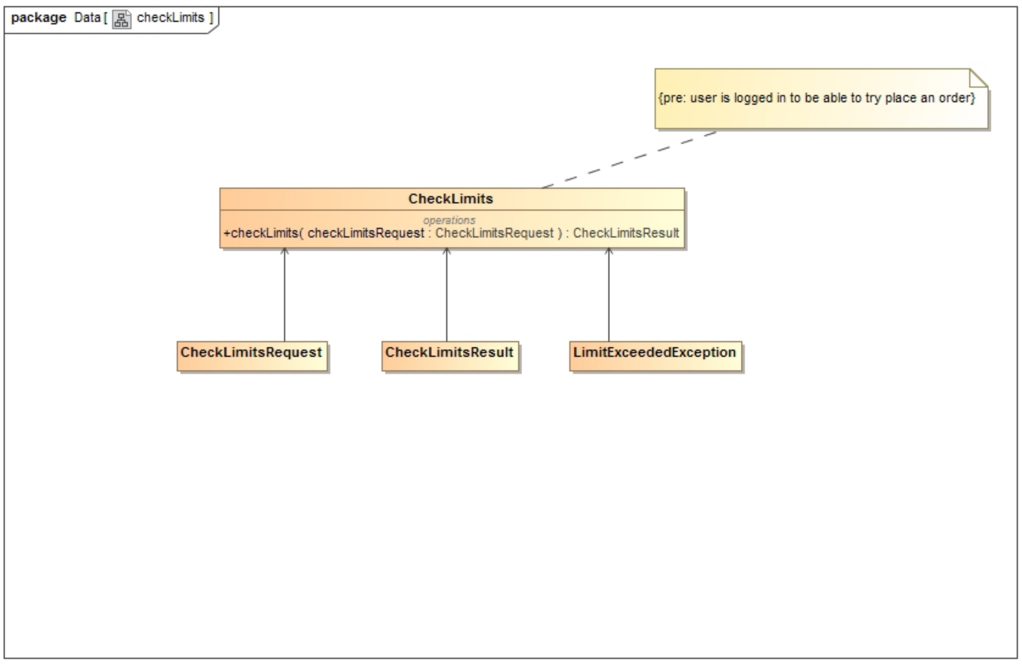
package Data [checkLimits]
{pre: user is logged in to be able to try place an order}
CheckLimits
operations
+checkLimits(checkLimitsRequest : CheckLimitsRequest) : CheckLimitsResult
CheckLimitsRequest
CheckLimitsResult
LimitExceededException

Figure 6: The service contract for checking limits

5.3 Manage Profile Module

The user will be allowed to customize various settings such as resetting his/her personal limit, changing password and email, displaying the bill and viewing favourites. The following use case diagram indicates the above mentioned functionality around managing the profile. Hence, it has the use cases generateFavourite and edit profile.

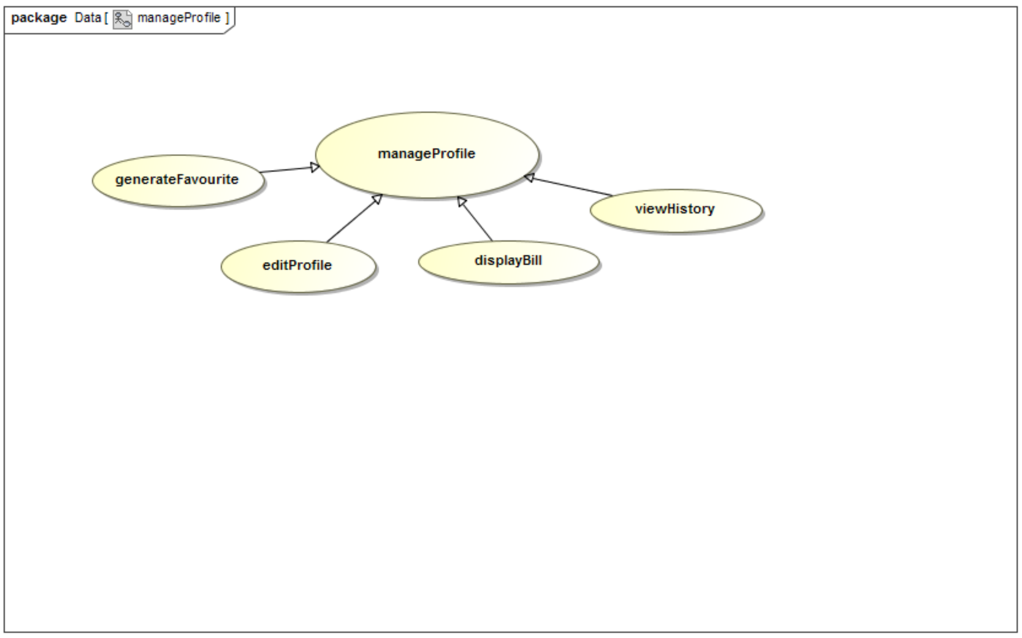

Figure 7: The use case diagram for managing profile

5.3.1 generateFavourite

The service contract and activity diagram for generateFavourite follow. generateFavourite falls under the use case for Manage Profile (refer to page ..... to view this use case diagram). The user will be able to view their most popular purchases and these will be based on the most recurring item in the user's account history.

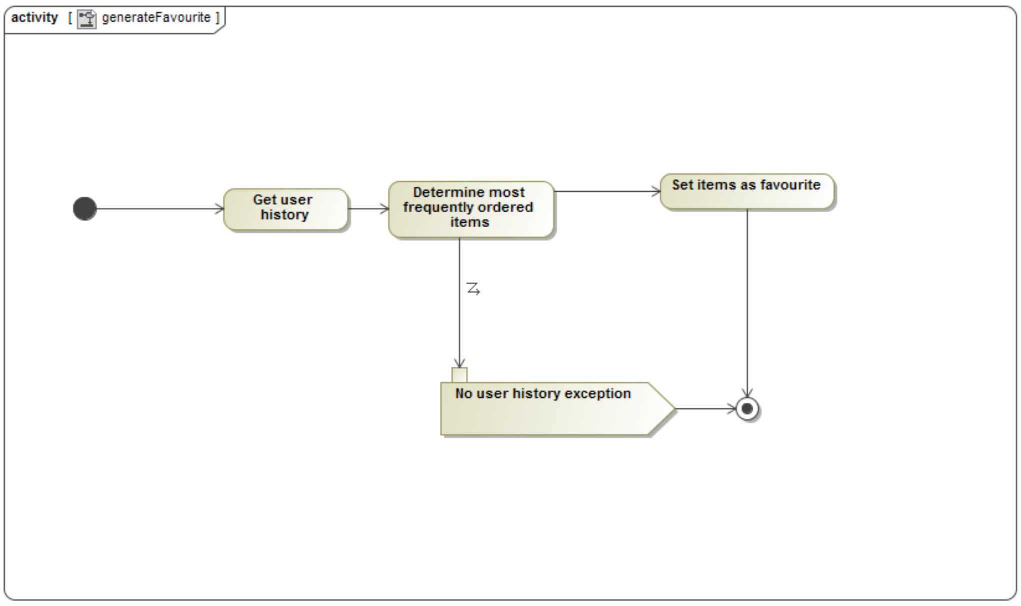
activity [generateFavourite]
Get user history
Determine most frequently ordered items
Set items as favourite
No user history exception

Figure 8: The activity diagram for generating favourites

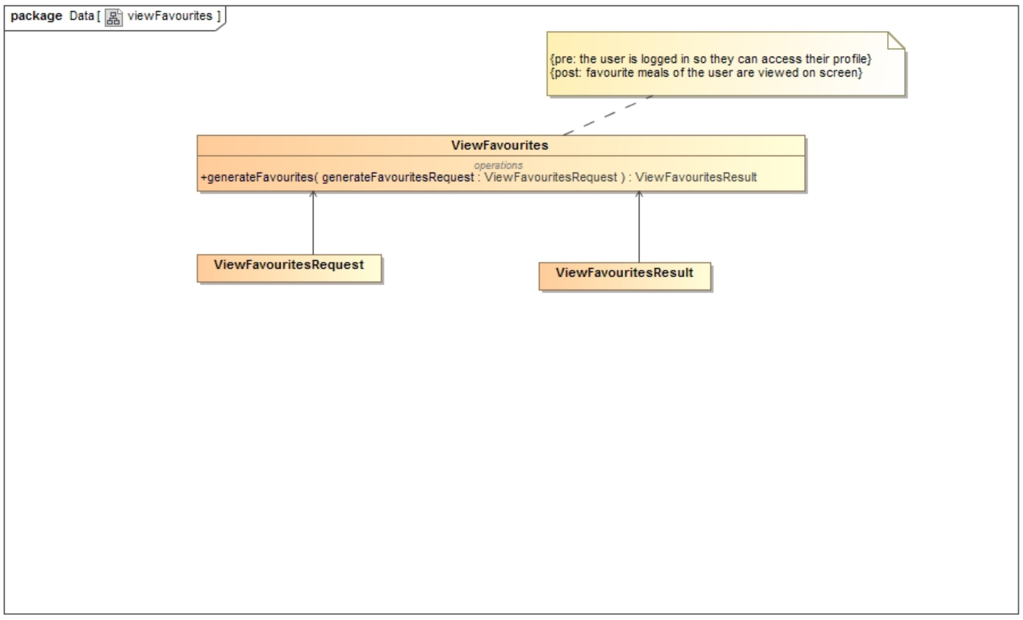
package Data [viewFavourites]
{pre: the user is logged in so they can access their profile}
{post: favourite meals of the user are viewed on screen}
ViewFavourites
operations
+generateFavourites(generateFavouritesRequest : ViewFavouritesRequest) : ViewFavouritesResult
ViewFavouritesRequest
ViewFavouritesResult

Figure 9: The service contract for generating favourites

5.4 editProfile

Edit profile includes the user changing his/her password and email address as well as changing his/her spending limit. Hence, the following use cases, changePassword, changeEmail and setLimit, are used in the following use case diagram. To view where this use case fits into the system, refer to figure ....

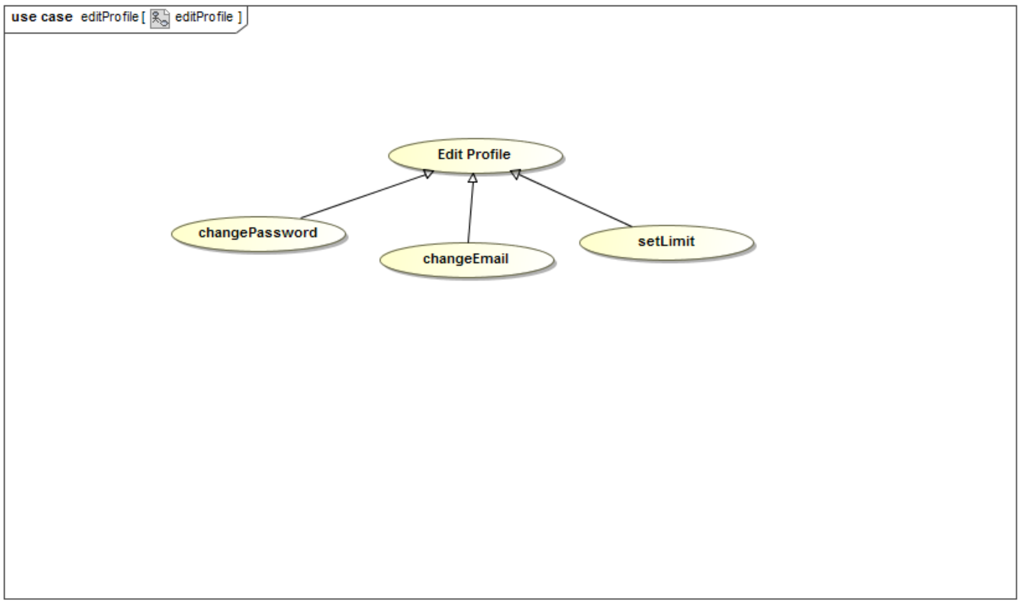

Figure 10: The use case diagram for editing profile

5.4.1 changePassword

The service contract and activity diagram for changePassword follow. changePassword falls under the use case for Edit Profile (refer to page ..... to view this use case diagram). The user will be able to edit their password.

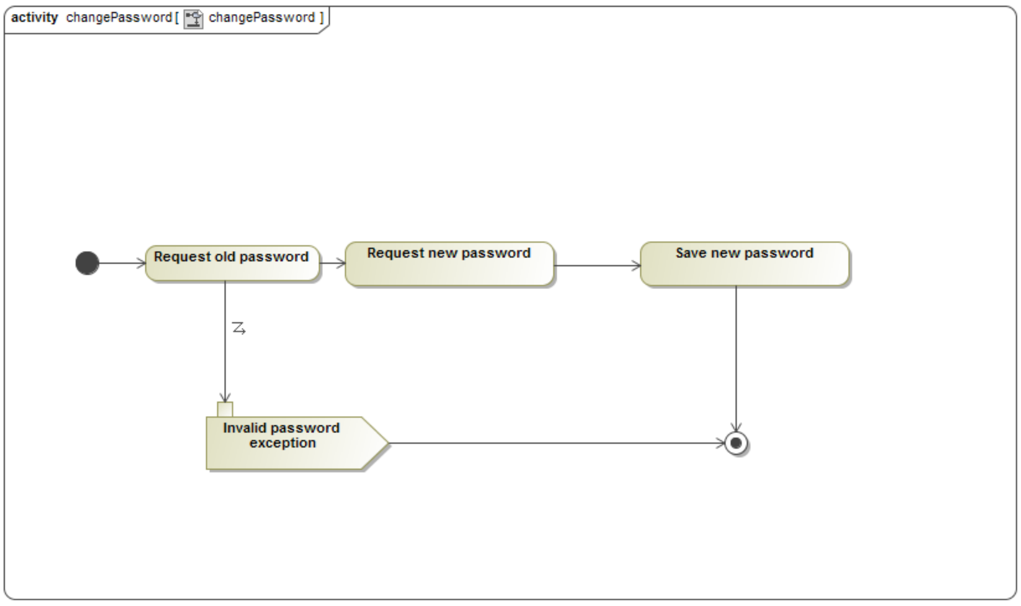
activity changePassword [changePassword]
Request old password
Request new password
Save new password
Invalid password exception

Figure 11: The activity diagram for changing password

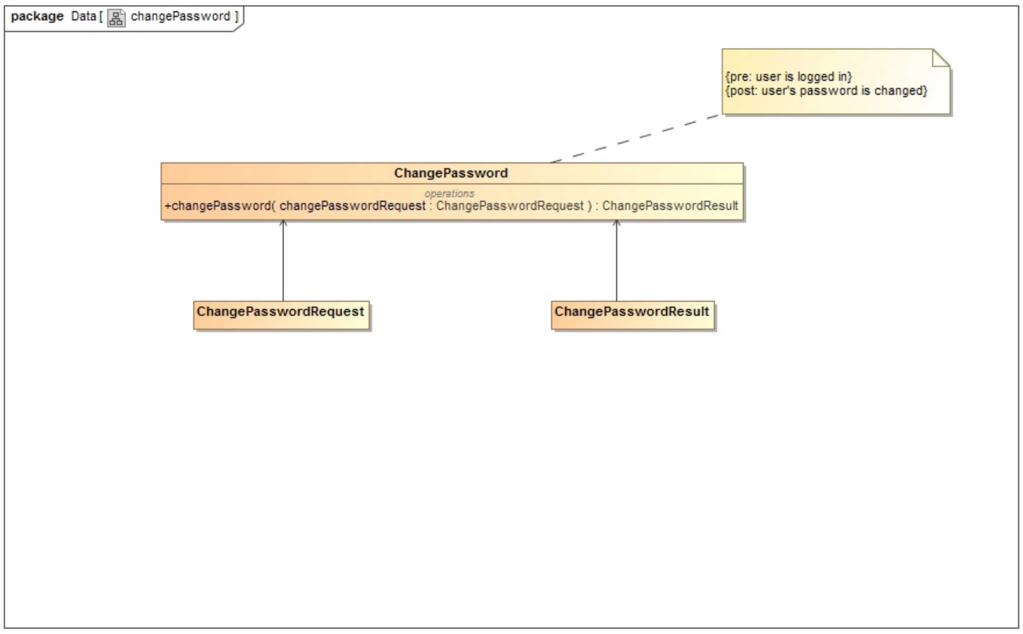
package Data [changePassword]
{pre: user is logged in}
{post: user's password is changed}
ChangePassword
operations
+changePassword(changePasswordRequest : ChangePasswordRequest) : ChangePasswordResult
ChangePasswordRequest
ChangePasswordResult

Figure 12: The service contract for changing password

5.4.2 changeEmail

The service contract and activity diagram for changeEmail follow. changeEmail falls under the use case for Edit Profile (refer to page ..... to view this use case diagram). The user will be able to edit their email address.

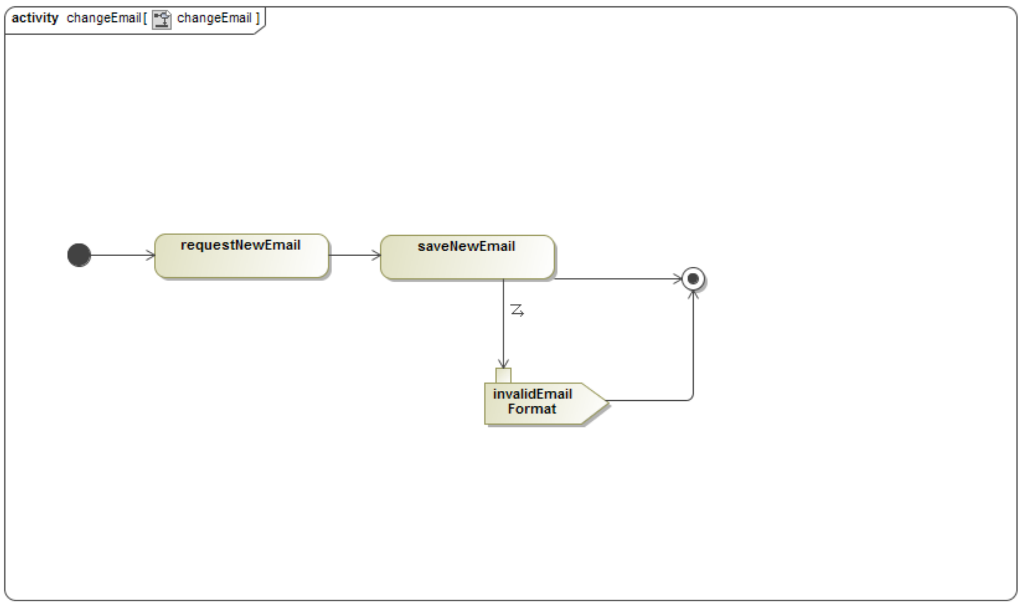

Figure 13: The activity diagram for changing email address

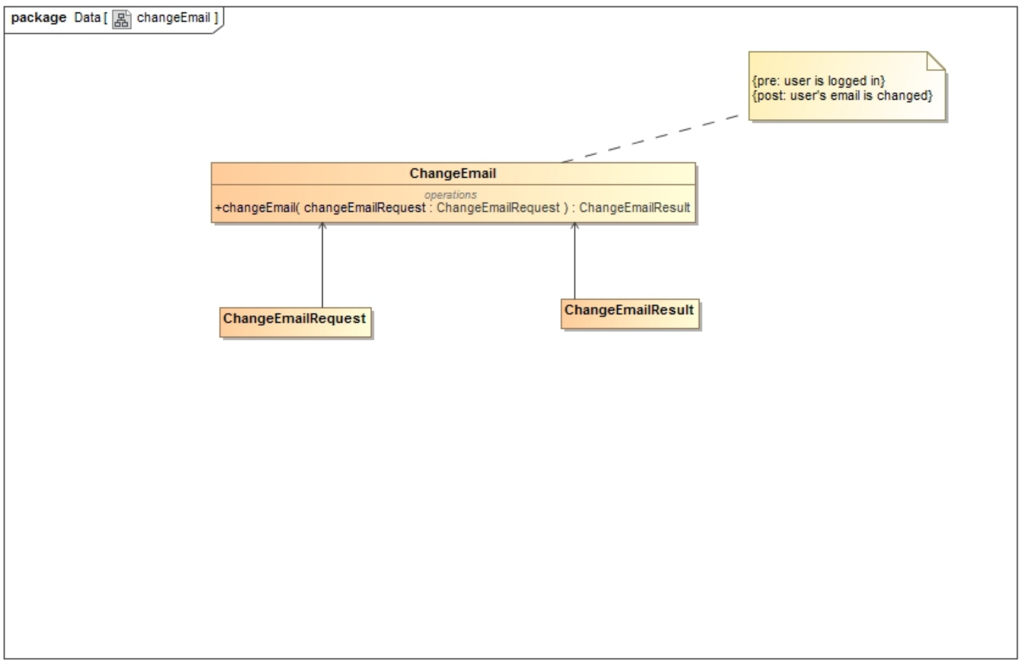
package Data [changeEmail]
{pre: user is logged in}
{post: user's email is changed}
ChangeEmail
operations
+changeEmail(changeEmailRequest : ChangeEmailRequest) : ChangeEmailResult
ChangeEmailRequest
ChangeEmailResult

Figure 14: The service contract for changing email address

5.4.3 changeLimit

The service contract and activity diagram for changeLimit follow. changeLimit falls under the use case for Edit Profile (refer to page ..... to view this use case diagram). The user will be able to change their personal spending limit on their profile, however, this must not exeed the system's limit.

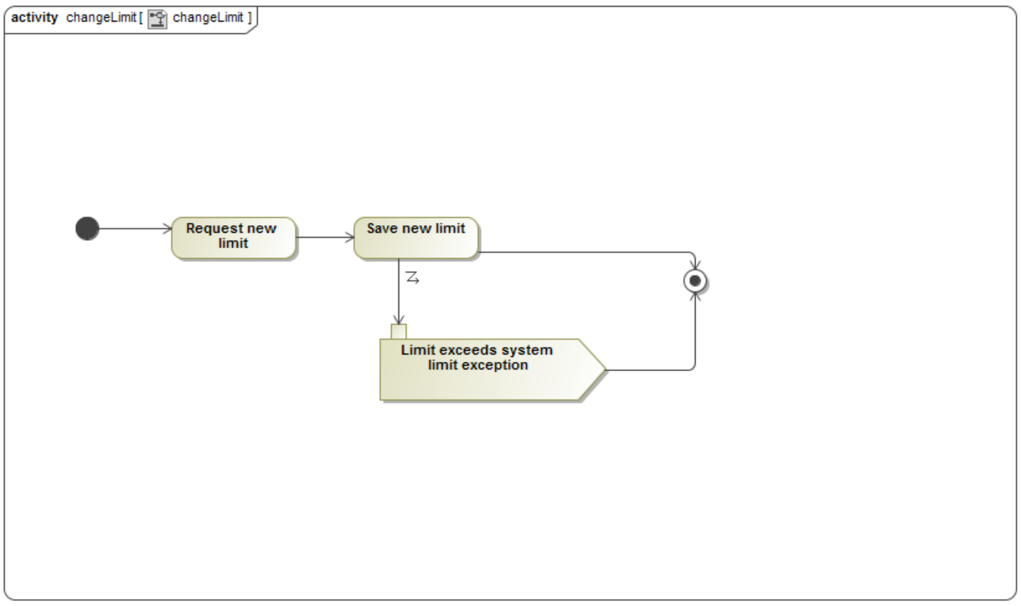
activity changeLimit [changeLimit]
Request new limit
Save new limit
Limit exceeds system limit exception

Figure 15: The activity diagram for setting limit

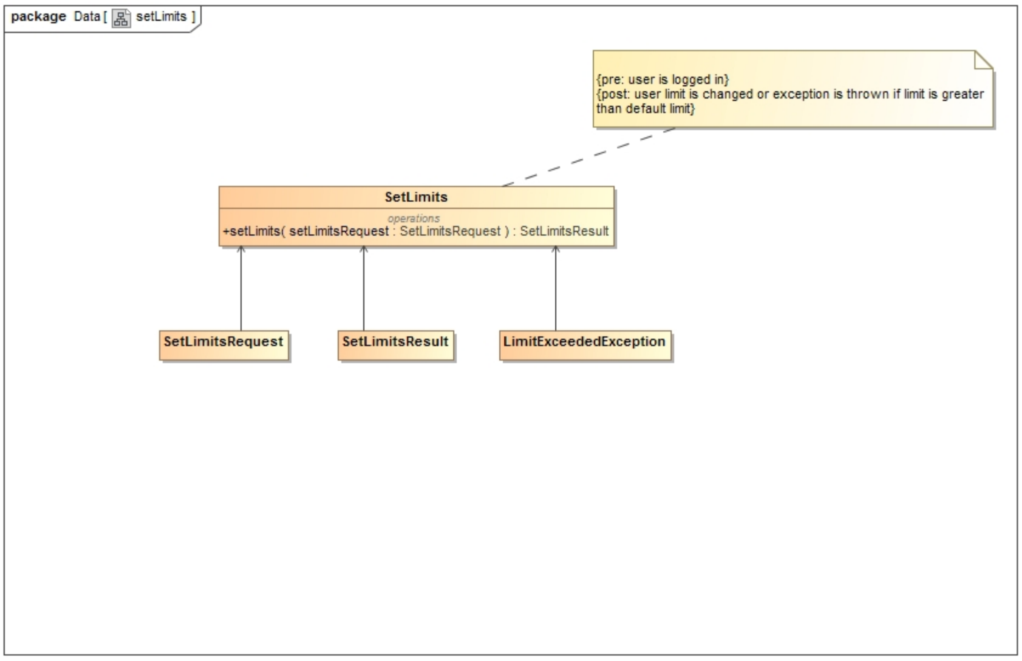
package Data [setLimits]
{pre: user is logged in}
{post: user limit is changed or exception is thrown if limit is greater than default limit}
SetLimits
operations
+setLimits(setLimitsRequest : SetLimitsRequest) : SetLimitsResult
SetLimitsRequest
SetLimitsResult
LimitExceededException

Figure 16: The service contract for setting limit

6 Comment

Due to the fact that an agile approach was followed, the main two modules of the system were explored in this document. As time progresses, the rest of the diagrams will be added and the existing ones will be subject to change.