# Project Use Case Analysis

*11/30/2020*

## *Semester Project (Deliverable-IV)*

SE 6329: Object Oriented Software Engineering

Instructor: R. Z. Wenkstern

TA: Dongcheng Li

*Maytanee Wethington, Shanjida Khatun, Ajey Subrahmanya, Meng-Hsiang Chang*

# *Table of Contents*

# 1. Domain Model

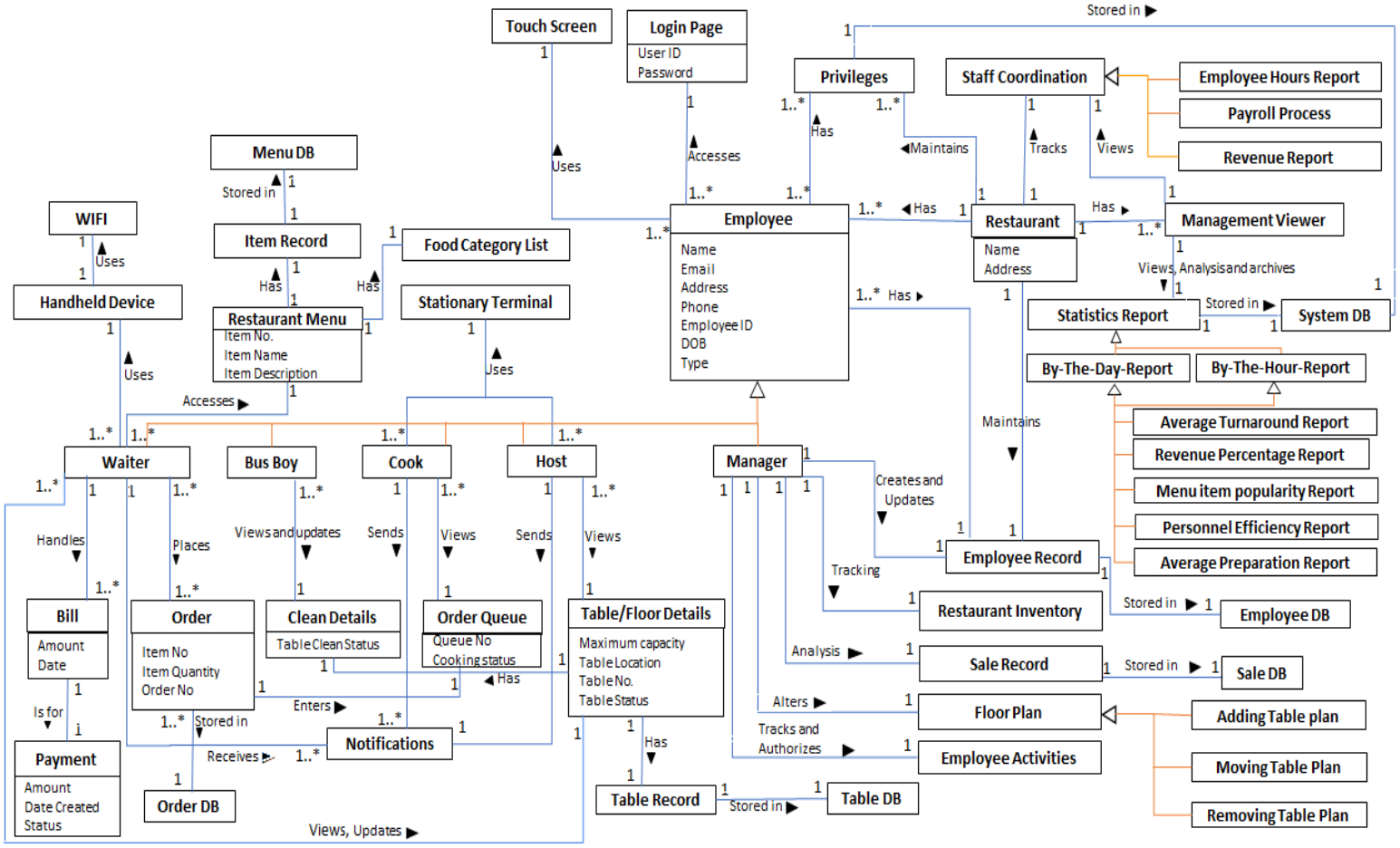
Version 3

Date: 11/26/2020

***Revision History***

| Version - description | Date | Description | Author |
|---|---|---|---|
| Inception draft | 10/26/2020 | First draft. To be refined primarily later | Dream Team |
| Elaboration 1 draft 1 | 10/27/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 2 | 10/29/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 3 | 11/6/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 4 | 11/26/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 5 | 11/30/2020 | Refined draft. | Dream Team |

## 1.1 Domain Model



# 2. Use Case Model

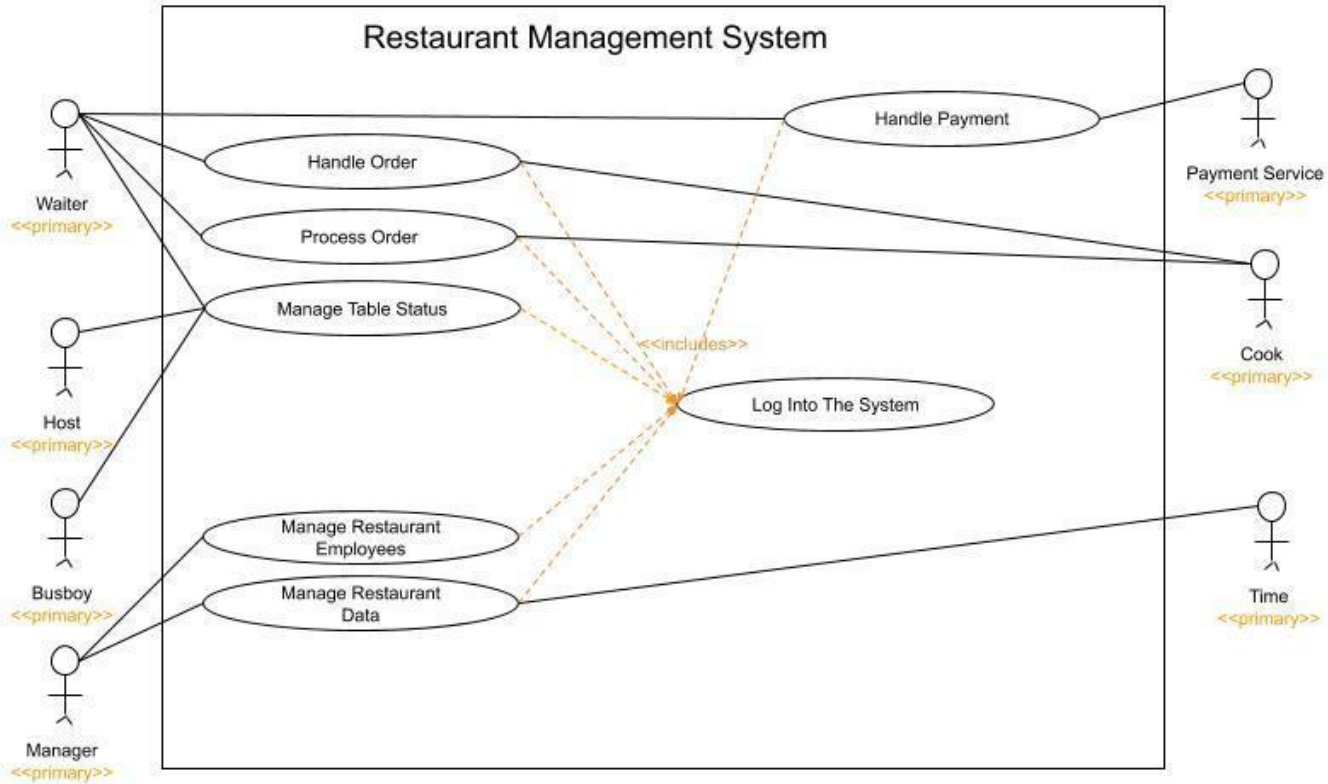
Version 4

Date: 11/26/2020

***Revision History***

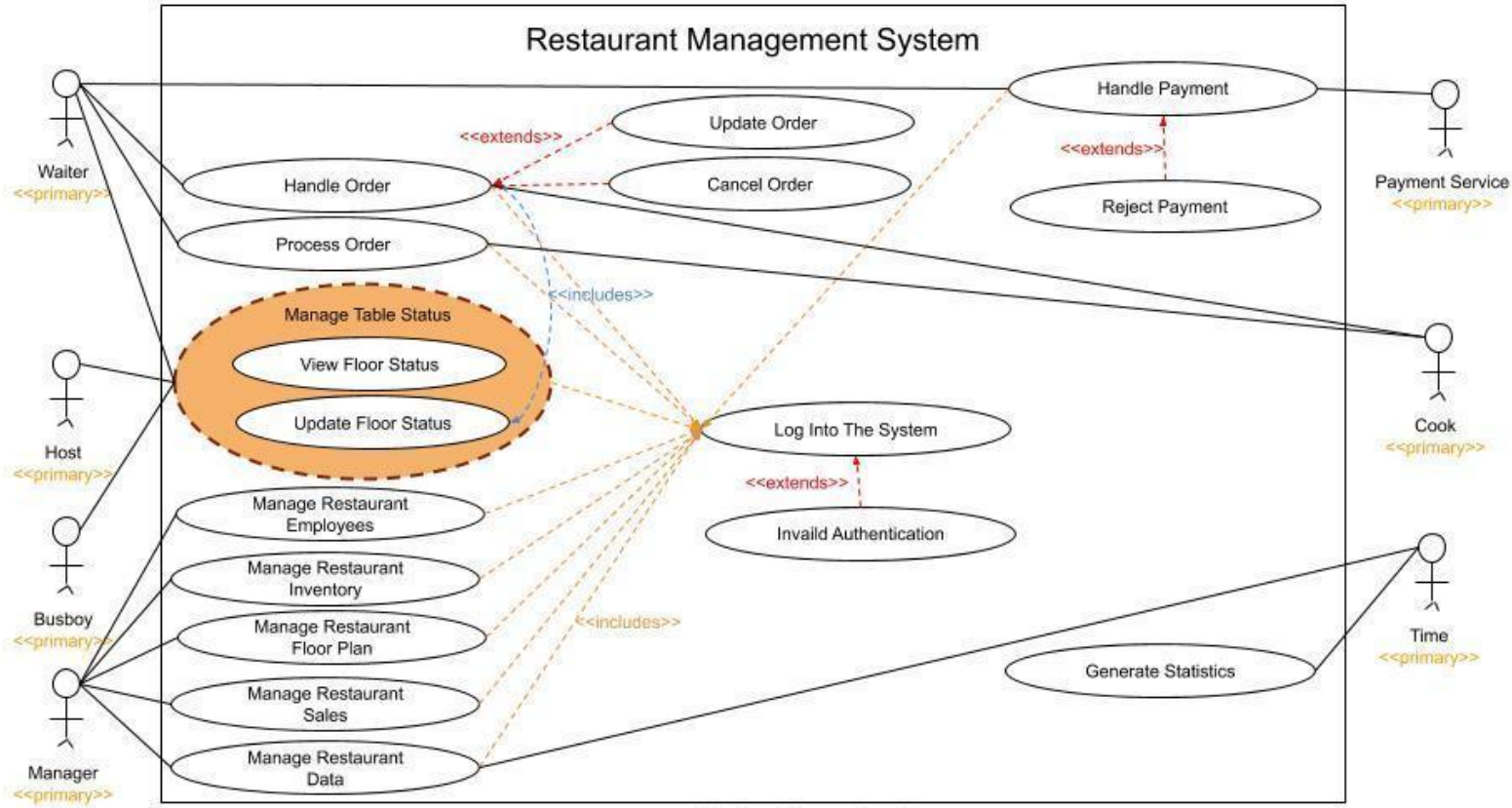
| Version - description | Date | Description | Author |
|---|---|---|---|
| Inception Draft | 10/26/2020 | First draft. To be refined primarily later | Dream Team |
| Elaboration 1 Draft 1 | 10/27/2020 | Refined draft. | Dream Team |
| Elaboration 1 Draft 2 | 10/29/2020 | Refined draft. | Dream Team |
| Elaboration 1 Draft 3 | 11/6/2020 | Revised Use Case Models To Fit New Requirements. | Dream Team |
| Elaboration 1 Draft 2 | 11/26/2020 | Refined Include Use Case. | Dream Team |

## 2.1 UCD Level 0




Use Case Diagram: Level 0

## 2.2 UCD Level 1




Use Case Diagram: Level 1

## *2.3. UCD Prioritization*

### 2.3.1 UCD Prioritization (at level 0)

| Fully Dressed | Casual | Brief |
|---|---|---|
| **1.** Handle Order | 2. Process Order | 4. Manage Table Status |
| | 3. Handle Payment | 5. Manage Restaurant Employees |
| | | 6. Manage Restaurant Data |
| | | 7. Log Into the system |

### 2.3.2 UCD Prioritization (at level 1)

| Fully Dressed | Casual | Brief |
|---|---|---|
| **1.** Handle Order | 2. Process Order | 4. View Floor Status |
| | 3. Handle Payment | 5. Update Floor Status |
| | | 6. Manage Restaurant Employees |
| | | 7. Manage Restaurant Data |
| | | 8. Log Into the system |
| | | 9. Manage Restaurant Inventory |
| | | 10. Manage Restaurant Floor Plan |
| | | 11. Manage Restaurant Sales |
| | | 12. Generate Statistics |
| | | 13. Invalid Authentication |
| | | 14. Reject Payment |
| | | 15. Update Order |
| | | 16. Cancel Order |

# 3. Use Case Descriptions

## 3.1 UC 1

### 3.1.1 UC 1 Description (In Fully Dressed Format)

| Use Case: Handle Order |
|---|
| ID: 1 |
| Brief Description: The waiter selects the table from the floor status screen and adds the item ordered by the user to the tables tab which gets added to the order queue which is viewed by the kitchen staff. The waiter can cancel/modify the order. |
| Primary Actors: Waiter. |
| Secondary Actors: Host. |
| Preconditions:<br>1. The user is identified and authenticated.<br>2. At least one table is empty to seat patrons.<br>3. At least one waiter is available to serve food. |
| Main Flow:<br>1. When patrons enter the restaurant, a host starts the process to log into the system to view the floor status.<br>2. Include (Log into the system).<br>3. The host finds an empty table for the patron.<br>4. The host escorts the patron to the table.<br>5. The host starts the process to assign a waiter for that table.<br>6. Include (Update floor status).<br>7. The waiter goes to the assigned table and takes the patron's order.<br>8. The waiter starts the process to log into the system from their handheld device to handle the patron's order.<br>9. Include (Log into the system).<br>10. The waiter is presented with a page displaying the floor plan of the restaurant and the status of his/her assigned tables.<br>11. Include (Update floor status).<br>12. Then the waiter selects the table from the floor status screen.<br>13. The system provides the waiter with several options.<br>14. The waiter selects "Add Item" to add an item.<br>15. The system provides the waiter with a list of food items on the restaurant's menu.<br>16. For each item ordered by the patron<br>16.1 The waiter selects the item from the list of restaurant menu items.<br>16.2 The waiter adds the desired food item to the order. |

| |
|---|
| 17. After all ordered items have been added, the waiter places the order.<br>18. The order gets added to the order queue.<br>19. The system displays that an order is placed for the waiter.<br>Extension point: Modify order, Extension: Update Order<br>Extension point: Canceling order, Extension: Cancel Order |
| Postconditions: The order was added to the order queue successfully. |
| Open Issues: None. |

### 3.1.2 UC 1 <<Includes>>

| Include Use Case: Update Floor status |
|---|
| ID: 5 |
| Brief Description: The user updates the table status from the floor status screen. |
| Primary Actors: Waiter, Host, Busboy. |
| Secondary Actors: None. |
| Preconditions: The user has viewed the floor status screen. |
| Main Flow:<br>1. If for assigning a wait staff by host<br>1.1. Users select a particular table.<br>1.2. User selects an available wait staff for that table.<br>1.3. The system notifies the waiter and displays the changes.<br>2. If for updating table status by wait staff or busboy<br>2.1. User selects the table<br>2.2. User updates the table status.<br>2.3. System displays the changes. |
| Postconditions: Update has been made successfully. |
| Open Issues: None. |

| Include Use Case: Log Into The System |
|---|
| ID: 8 |
| Brief Description: The user enters their login credentials, after which the system authenticates their user ID and password. Then, the system presents the user with their respective home screen based on the privileges assigned to their role. |
| Primary Actors: Host, Waiter, Cook, Manager, Busboy. |
| Secondary Actors: None. |
| Preconditions: Access to a terminal that has access to the system network. |
| Main Flow:<br>1. The user selects the login button on the screen.<br>1. The system prompts for username and password.<br>2. For each attempt =<3 and (invalid username or password)<br>3.1. The user enters the username and password.<br>3.2. User presses the OK button.<br>Extension point: Invalid ID or password, Extension: Invalid Authentication<br>3. If the staff is a waiter or busboy or host then the system displays the floor screen.<br>4.1. OR If staff is a Manager then the system displays the management screen.<br>4.2. OR If staff is a cook then the system displays the order queue screen.<br>4.3. Else system displays failed. |
| Postconditions: After logging in, the user successfully sees their respective screen. |
| Open Issues: None. |

### 3.1.3 UC 1 <<Extends>>

| Extend Use Case: Update Order |
|---|
| ID: 15 |
| Brief Description: The waiter can modify the menu items according to the patron's request. |
| Primary Actors: Waiter. |
| Secondary Actors: None. |
| Preconditions:<br>1. The waiter logged into the system and viewed the status screen.<br>2. The cooking process was not finished. |
| Main Flow:<br>1.The waiter selects the particular order from the list.<br>2.The system provides the waiter with options like 'Modify Order' or 'Cancel Order'. |

| |
|---|
| 3.The waiter selects “Modify order” and then the system provides the waiter with a list of items of that order and with options like ‘Add Item’ or ‘Delete Item’.<br>4. If the waiter selects ‘Add Item’ then the system provides the waiter with a list of food items on the restaurant’s menu.<br>4.1 For each item ordered by the patron<br>4.1.1 The waiter selects the item from the list of restaurant menu items.<br>4.1.2 The waiter adds the desired food item to the order.<br>5.If the waiter selects “Delete Item” then the system deletes that item from that order.<br>6.After the ordered items have been modified, the waiter places the order.<br>7.The order gets added to the order queue.<br>8.The system displays that an order is placed. |
| Postconditions: Update has been made successfully. |
| Open Issues: None. |

| Extend Use Case: Cancel Order |
|---|
| ID: 16 |
| Brief Description: The waiter can cancel the order according to the patron’s request. |
| Primary Actors: Waiter. |
| Secondary Actors: None. |
| Preconditions:<br>1. The waiter logged into the system and viewed the status screen.<br>2. The cooking process was not finished. |
| Main Flow:<br>1. The waiter selects the order from the list.<br>2. The system provides the waiter with options like ‘Modify Order’ or ‘Cancel Order’.<br>3. The waiter selects “Cancel Order” and then the system deletes that particular order from the list.<br>4. The order gets deleted from the order queue.<br>5. The system displays the changes. |
| Postconditions: Cancel has been made successfully. |
| Open Issues: None. |

### 3.1.4 UC 1 Non-Functional-Requirements

1. The performance should be extremely good. The system should feel snappy to the waiter. Operations (such as placing orders in the order queue) should complete within 3 seconds.
2. The underlying functions required to carry out this use case must be reliable. It must work at all times during business hours to avoid delays in patrons' orders
3. The interface the waiter uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.2 UC 2*

### 3.2.1 UC 2 Description (In Casual Format)

**Process Order**

A cook **logs into the system** and can view incoming orders from the order queue on a first-in-first-out basis. When a cook receives an order from a waiter, he/she immediately begins preparing the items listed on the order. At this stage, the cook updates the order's status to 'cooking'. Any subsequent orders received will be prepared in parallel with the current one if there are kitchen resources available. If not, they will have to wait until the preparation of the current order is complete.

Once the order is ready, the cook updates the status to 'ready' and notifies the waiter associated with the order to serve it. The cook will keep a queue of ready orders to avoid cluttering up the kitchen.

### 3.2.2 UC 2 <<Includes>> Log into the system.

### 3.2.3 UC 2 <<Extends>> None.

### 3.2.4 UC 2 Non-Functional-Requirements

1. The performance should be extremely good. The system should feel snappy to the cook. Operations (such as notifying the waiter) should complete within 3 seconds.

2. The interface the cook uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.3 UC 3*

### 3.3.1 UC 3 Description (In Casual Format)

### Handle Payment

A waiter initiates the payment for a particular table's tab by generating a bill. The patron then selects the mode of payment. If the patron chooses to pay by card, the waiter swipes the card on the POS machine and the transaction is processed. If the patron chooses to pay by cash, the waiter manually processes the payment.

#### 3.3.2 UC 3 <<Includes>> Log into the system.

#### 3.3.3 UC 3 <<Extends>> Reject Payment.

#### 3.4.4 UC 3 Non-Functional-Requirements

1. The Handle Payment use case should be secure. Nobody other than the waiter serving the table should be able to initiate a transaction to pay for the table's tab.
2. The connection to the payment service must be reliable. There should be no point in time during business hours where a waiter is unable to process a transaction due to a reason other than the patron's card being declined.
3. The interface the waiter uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.4 UC 4*

#### 3.4.1 UC 4 Description (In Brief Format)

### View Floor Status

The host or a busboy **logs into the system** and is presented with the floor plan of the restaurant, as well as the current status of all the tables.

A waiter **logs into the system** and is presented with the floor plan of the restaurant, as well as the current status of the tables they are assigned to.

#### 3.4.2 UC 4 <<Includes>> Log into the system.

#### 3.4.3 UC 4 <<Extends>> None.

#### 3.4.4 UC 4 Non-Functional-Requirements

1. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

2. The performance of this use case should be very good. The status of tables should be updated in real-time so anyone viewing a table's status will be up to date with its actual current status.

## *3.5 UC 5*

### 3.5.1 UC 5 Description (In Brief Format)

### Update Floor Status

The host **logs into the system** and is presented with the floor plan of the restaurant, as well as the current status of all the tables. The host checks for available tables (green) and assigns patrons to an available table. Once the patrons are assigned a table, the host changes the status of the table to occupied (yellow) and assigns a waiter to attend to the occupied table.

After the order tab is closed for a table, the waiter assigned to that table **logs into the system** to update the status of the table to dirty (red).

A busboy will then clear the table, after which he or she will **log into the system** and change the status of the table to available (green).

### 3.5.2 UC 5 <<Includes>> Log into the system.

### 3.5.3 UC 5 <<Extends>> None.

### 3.5.4 UC 5 Non-Functional-Requirements

1. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.
2. The performance of this use case should be very good. The status of tables should be updated in real-time so anyone viewing a table's status will be up to date with its actual current status.

## 3.6 UC 6

### 3.6.1 UC 6 Description (In Brief Format)

### Manage Restaurant Employees

A manager **logs into the system** and selects the "Manage Restaurant Employees" from their home screen. After selecting this option, the manager is presented with an interface listing the profiles of all the employees currently working at the restaurant. The manager will be able to create or modify employee profiles and authorize restricted activities for a waiter using his/her user ID.

**3.6.2 UC 6 <<Includes>>** Log into the system.

**3.6.3 UC 6 <<Extends>>** None.

### 3.6.4 UC 6 Non-Functional-Requirements

1. The Manage Restaurant Employees use case should be secure. Nobody other than the manager should be able to manage the restaurant employees because these records contain sensitive information.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## 3.7 UC 7

### 3.7.1 UC 7 Description (In Brief Format)

### Manage Restaurant Data

A manager **logs into the system** and is presented with their respective home screen. From here, the manager can create or modify an employee profile and authorize restricted activities to a waiter with his/her user ID. The manager will also be able to track restaurant inventory and view sales data. Lastly, the manager will be able to view and modify the restaurant's floor plan. Managers can analyze the statistics that allow seeing what portion of the revenue comes from what item, what are the most popular items.

**3.7.2 UC 7 <<Includes>>** Log into the system.

**3.7.3 UC 7 <<Extends>>** None.

### 3.7.4 UC 7 Non-Functional-Requirements

1. The Manage Restaurant Data use case should be secure. Nobody other than the manager should be able to manage the restaurant data since this is sensitive information.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.8 UC 8*

### 3.8.1 UC 8 Description (In Brief Format)

### Log into the system

The user enters their login credentials, after which the system authenticates their user ID and password. Then, the system presents the user with their respective home screen based on the privileges assigned to their role.

### 3.8.2 UC 8 <<Includes>> None.

### 3.8.3 UC 8 <<Extends>> Invalid Authentication.

### 3.8.4 UC 8 Non-Functional-Requirements

1. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.
2. The performance of this use case should be good. Authentication should complete within 5 seconds.

## *3.9 UC 9*

### 3.9.1 UC 9 Description (In Brief Format)

### Manage Restaurant Inventory

A manager **logs into the system** and selects the “Manage Restaurant Inventory” option from their home screen. After selecting this option, the manager is presented with an interface listing all food items in the restaurant’s possession. Additionally, the manager can view the menu items sold each day, along with the number sold.

### 3.9.2 UC 9 <<Includes>> Log into the system.

### 3.9.3 UC 9 <<Extends>> None.

### 3.9.4 UC 9 Non-Functional-Requirements

1. The Manage Restaurant Inventory use case should be secure. Nobody other than the manager should be able to manage the restaurant inventory and view the statistics on the menu items.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.10 UC 10*

### 3.10.1 UC 10 Description (In Brief Format)

### Manage Restaurant Floor Plan

A manager **logs into the system** and selects the "Manage Restaurant Floor Plan" option from their home screen. After selecting this option, the manager is presented with an interface showing them the current floor plan of the restaurant, along with tools that allow for altering the floor plan. The manager will be able to add or remove tables from the floor plan, as well as move them around.

### 3.10.2 UC 10 <<Includes>> Log into the system.

### 3.10.3 UC 10 <<Extends>> None.

### 3.10.4 UC 10 Non-Functional-Requirements

1. The Manage Restaurant Floor Plan use case should be secure. Nobody other than the manager should be able to manage the restaurant's floor plan.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.11 UC 11*

### 3.11.1 UC 11 Description (In Brief Format)

### Manage Restaurant Sales

A manager **logs into the system** and from this profile, the manager should have the ability to manage other aspects of restaurant operations, such as sales analysis. The manager can analyze the statistics that allow him or her to see what portion of the revenue comes from what item, and what the most popular items are.

### 3.11.2 UC 11 <<Includes>> Log into the system.

### 3.11.3 UC 11 <<Extends>> None.

### 3.11.4 UC 11 Non-Functional-Requirements

1. The Manage Restaurant Sales use case should be secure. Nobody other than the manager should be able to view the restaurant's sales data.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.12 UC 12*

### 3.12.1 UC 12 Description (In Brief Format)

### Generate Statistics

The system tracks everything electronically and then organizes it. All the information is collected, saved, and then displayed in table format for easy viewing by the managers. The automatically generated statistics allow the managers to see what portion of the revenue comes from what item and what the most popular items are. All this is done automatically and stays up to date with the latest data inputted into the system.

### 3.12.2 UC 12 <<Includes>> None.

### 3.12.3 UC 12 <<Extends>> None.

### 3.12.4 UC 12 Non-Functional-Requirements

1. The Generate Statistics use case should be secure. Nobody other than the manager should be able to view the statistics generated by the system.
2. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## 3.13 UC 13

### 3.13.1 UC 13 Description (In Brief Format)

**Invalid Authentication**

The user enters their login credentials, after which the system fails to authenticate their user ID and/or password. The system will tell the user to try again or enter in alternate credentials.

**3.13.2 UC 13 <<Includes>>** None.

**3.13.3 UC 13 <<Extends>>** None.

### 3.13.4 UC 13 Non-Functional-Requirements

1. The interface used to carry out this use case must be usable. It should be intuitive so it is easy to understand.
2. The performance of this use case should be good. Authentication should complete within 5 seconds.

## 3.14 UC 14

### 3.14.1 UC 14 Description (In Brief Format)

**Reject Payment**

Should a patron choose to pay by card, the waiter swipes the card on the POS machine and the transaction will not go through no matter how many times the waiter tries. The waiter will return to the patron's table, notifying them the card was declined, then request an alternative means of payment.

**3.14.2 UC 14 <<Includes>>** None.

**3.14.3 UC 14 <<Extends>>** None.

### 3.14.4 UC 14 Non-Functional-Requirements

1. The Handle Payment use case should be secure. Nobody other than the waiter serving the table should be able to initiate a transaction to pay for the table's tab.
2. The connection to the payment service must be reliable. There should be no point in time during business hours where a waiter is unable to process a transaction due to a reason other than the patron's card being declined.

3. The interface the waiter uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.15 UC 15*

### 3.15.1 UC 15 Description (In Brief Format)

**Update Order**

A waiter can modify the items on an order upon a patron's request. This includes adding or removing items on an order. Before updating the order, the waiter must check the cooking status of the order. If the order is not ready yet, then the waiter can update the order.

**3.15.2 UC 15 <<Includes>>** None.

**3.15.3 UC 15 <<Extends>>** None.

### 3.15.4 UC 15 Non-Functional-Requirements

1. The performance should be extremely good. The system should feel snappy to the waiter. Updating the order should complete within 3 seconds.
2. The underlying functions required to carry out this use case must be reliable. It must work at all times during business hours to avoid delays in updating patrons' orders
3. The interface the waiter uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

## *3.16 UC 16*

### 3.16.1 UC 16 Description (In Brief Format)

**Cancel Order**

A waiter can cancel an order upon a patron's request. Before canceling the order, the waiter must check the cooking status of the order. If the order is not ready yet, then the waiter can cancel the order.

**3.16.2 UC 16 <<Includes>>** None.

**3.16.3 UC 16 <<Extends>>** None.

### 3.16.4 UC 16 Non-Functional-Requirements

1. The performance should be extremely good. The system should feel snappy to the waiter. Canceling the order should complete within 3 seconds.
2. The underlying functions required to carry out this use case must be reliable. It must work at all times during business hours to avoid delays in canceling patrons' orders
3. The interface the waiter uses to carry out this use case must be usable. It should be intuitive so it is easy to understand.

# 4. Supplementary Specification

Version 2

Date: 11/11/2020

***Revision History***

| Version - description | Date | Description | Author |
|---|---|---|---|
| Inception draft | 10/28/2020 | First draft. To be refined primarily later | Dream Team |
| Elaboration 1 draft 1 | 10/29/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 2 | 10/30/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 3 | 11/8/2020 | Refined draft. | Dream Team |

## 4.1 Introduction

This document is the repository of all restaurant requirements not captured in the use cases.

## 4.2 Product Non-Functional Requirements

### a. Usability

**Human Factors**

The user must be able to find what the patron wants from the site without any difficulty. The waiter is often looking at the patrons. The text should be easily visible to restaurant staff.

**Program Factors**

The program must be user-friendly, easily navigable, and efficient. The program should be constantly updated. The program must adhere to the branding of the restaurant and the layout of the web pages must be user-friendly, easy to operate. Signals and warnings should be conveyed with sound rather than only via graphics.

### b. Security

All usage requires user authentication. The program must provide the correct privileges for all user roles. The orders, transactions, and information shared between all tablets and computers running the application must be secured (because it can process credit/debit card information). Unauthorized transmission of sensitive information to other sources must be avoided, so this application will need to run on a closed system. All information on registered users must be securely stored in the restaurant's central database.

## c. Reliability

The program must be available during business hours. It should only be shut down an hour after the restaurant is closed. Maintenance should only take place after business hours to avoid complications. All of the restaurant information is organized and saved in the system database for managers to view and archive.

**Recoverability**

If their touch screen device fails, waiters should log into a stationary terminal to complete the order process or payment process. The system should be capable of saving a user's current state from one device and should restore it when they log in from another device.

## d. Performance

The system shall support up to 100 simultaneous users and support up to 100 orders in the order queue at any given time. Any system updates must be completed within 4 seconds. Card payment authorization should take less than 1 minute. The server shall be capable of supporting an arbitrary number of stationary terminals and handheld devices, and be capable of keeping track of all operations conducted on them in real-time.

## e. Functionality

The user should be able to view the status of any of a patron's orders as well as the status of the restaurant (the number of tables available, the number of tables that still need to be cleaned, etc.) and the current activities of the staff (amount of orders that need to be completed, amount of orders that need to be delivered, incoming orders, personnel efficiency, average preparation and turnaround times, etc.).

## f. Supportability

New users can be added and old users can be deleted. All operations will be logged in the database. This will allow for tracking orders and payments. Changes to the system's state will be reflected immediately for all restaurant staff to have access to updated information.

## 4.3 Process Non-Functional Requirements

### a. Design Constraints

The program shall integrate with the existing inventory, including the amount of tables, amount of chairs, ingredients in stock or out of stock, etc.

The program shall keep track of all current employees and their current positions. However, should someone leave, if a new employee is hired, someone comes back, or someone is promoted or demoted, the program should be able to easily make this change upon proper authorization.

The system shall work with current restaurant maintenance protocols to avoid having to rebuild the whole system.

### b. Implementation Constraints

The RMS leadership insists on a Java technologies solution, predicting this will improve long-term porting and supportability, in addition to ease of development.

The system must be capable of parallel operation, and the system design should not introduce scalability issues regarding the number of stationary terminals or tablets connected at any given time. The system should also allow for seamless recovery, without data loss, from individual device failure. There must be a strong audit chain with all system actions logged. If the system is down, then customers must not notice or notice that the system recovers quickly (seconds). The system must be reliable enough to run crash and glitch-free for long periods or facilitate error recovery strong enough such that glitches are never revealed to its end-users.

### c. Free Open Source Components

In general, we recommend it should be written in an object-oriented language with support for all major operating systems, strong GUI links, and a simple, accessible network API. The primary candidate toolchains are Java/Swing, C++/Qt, and Python/Qt.

## 4.4 External Non-Functional Requirements

### a. Hardware and Interfaces

Devices are the stationary terminals, the wireless tablets, the touch displays, Receipt printer, credit/debit card reader, signature reader. All devices must be physically robust and immune to liquid damage and stains. The devices must also have good industrial design aesthetics. The devices behave as 'terminals' in the sense that they never have a full system image, do not store data, and are not used for the core logic of the system. However, they should be fully capable computers that can use textual data from the server along with local UI/interpretation code to display UI elements and take input. All order and transaction records should be stored on the server, not these computers. In all cases, the hardware device takes information from the RMS and processes the information to display. It also provides user input information to the RMS.

### b. Software Interfaces

The RMS will interface with a Database Management System (DBMS) that stores the information necessary for the RMS to operate. The DBMS must be able to provide, on request and with low latency, data concerning the restaurant's menu, employees (and their passwords), and available dietary facts. Additionally, it should take, and archive data provided to it by the RMS. This data will include records of all orders and transactions (system states and state changes) executed by the RMS. The DBMS must store all data such that it can be used for accounting, as well as accountability.

### c. Legal Issues

All tax rules must be applied during the payment process. These can change frequently.

# 5. Glossary

Version 1

Date: 11/9/2020

### *Revision History*

| Version - description | Date | Description | Author |
|---|---|---|---|
| Inception draft | 11/6/2020 | First draft. To be refined primarily later | Dream Team |
| Elaboration 1 Draft 1 | 11/8/2020 | Refinement of first draft. | Dream Team |
| Elaboration 1 Draft 2 | 11/9/2020 | Second refinement of first draft. | Dream Team |

## 5.1 Glossary

| Term | Definition and Information | Format | Validation Rules | Aliases |
|---|---|---|---|---|
| **Restaurant Staff** | Any individual who is involved in the day to day activities of the restaurant, including Cooks, Busboys, Waiters, Managers, and Hosts. | | | |
| **Manager** | Restaurant Staff with the highest administrative authority, with the power to change the status of all other employee types, have access to restaurant statistics, and can modify the floor plan. | | | |
| **Patron** | A customer who is served by the restaurant staff. | | | |
| **Cook** | Restaurant Staff who are responsible for processing the orders in the order queue sent by Waiters. | | | |
| **Waiter** | Restaurant Staff who are responsible for interacting with patrons by creating, modifying, and cancelling orders. These individuals send orders to the order queue for processing by Cooks, and update the orders as needed. | | | |
| **Host** | Restaurant Staff who welcomes patrons and directs them to an available table. The Host role can also be performed by a Waiter. | | | |
| **Busboy** | Restaurant Staff who are responsible for cleaning Tables. These individuals are also responsible for updating the status of Tables from Dirty to Clean after a table is cleaned. | | | |
| **Table** | A physical surface located within the restaurant on which prepared food is presented to Patrons. Each Table has a Waiter assigned to it. | | | |
| **Table Status** | Shows the status of a Table (Ready, Dirty, Busy). | | | |
| **Dirty** | A Table status indicating a Table is not currently occupied by any Patrons, but needs to be cleaned to seat new Patrons. | | | |
| **Busy** | A Table status indicating a Table is currently occupied by Patrons. | | | |
| **Ready** | A Table status indicating a Table is ready to seat new Patrons. | | | |
| **Order** | A list of items submitted by a Waiter, which are to be prepared by a Cook and eventually delivered to the Table from which it was ordered from. | | | |
| **Order Status** | The status of an Order (Cooking, Ready). This tells a Waiter what should be done with an Order (whether to serve it or wait). | | | |
| **Processing** | An Order status indicating an Order is waiting to be prepared. | | | |
| **Cooking** | An Order status indicating an Order is currently being prepared. Orders cannot be modified or cancelled once they reach this stage. | | | |
| **Ready** | An Order status indicating an Order has been prepared and is ready to be served to the Patron. | | | |
| **Staff Payroll** | This is the payroll for Restaurant Staff. Their wages are calculated from the number of hours worked and their hourly rate. The hourly rate is based on the role the individual serves (Cook, Busboy, Cook, Waiter, Manager, Host) . | | | |
| **Menu** | A list of food items served by the restaurant. A Patron will choose items from this Menu and the Waiter will place an Order for them. | | | |
| **Floor Plan** | This is the arrangement of all the tables in the restaurant. The Floor Plan can be modified on demand only by the Manager. | | | |
| **Payment Authorization** | This is an action performed by the Payment Service. A debit or credit card will have a transaction performed on it. Validation of the transaction guarantees payment to the restaurant for an Order. | | | |
| **Payment Authorization Request** | This is an action performed by a Waiter. A waiter initiates a transaction with a Patron's debit or credit card, and this information is sent to the Payment Service for authorization. Data elements in the transaction include restaurant's account no, the Patron's account no, and timestamp. | | | |
| **Payment Service** | An external Payment Service that handles payments to the restaurant by a debit or credit card. | | | |

# 6. System Sequence Diagrams

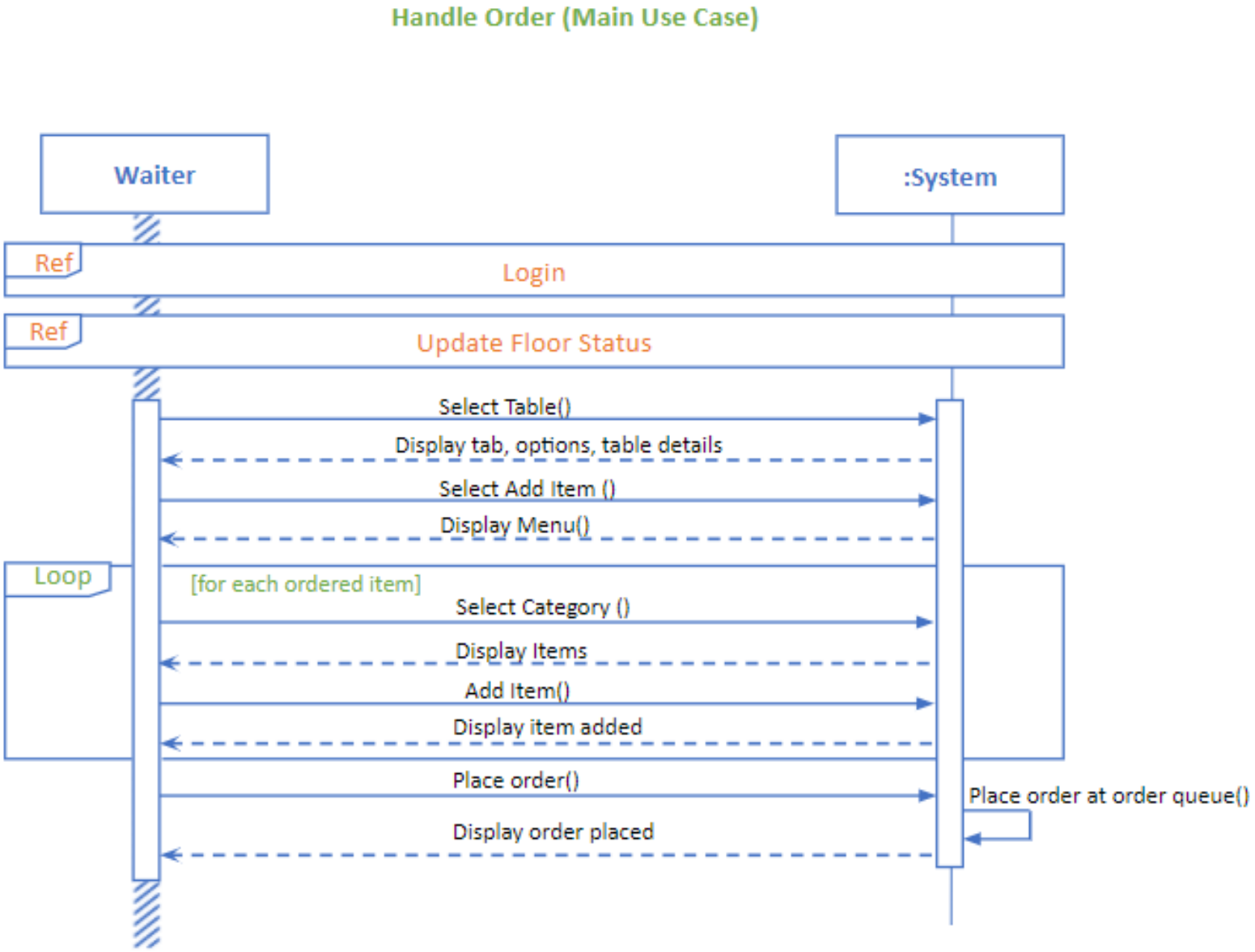
Version 2

Date:11/26/2020

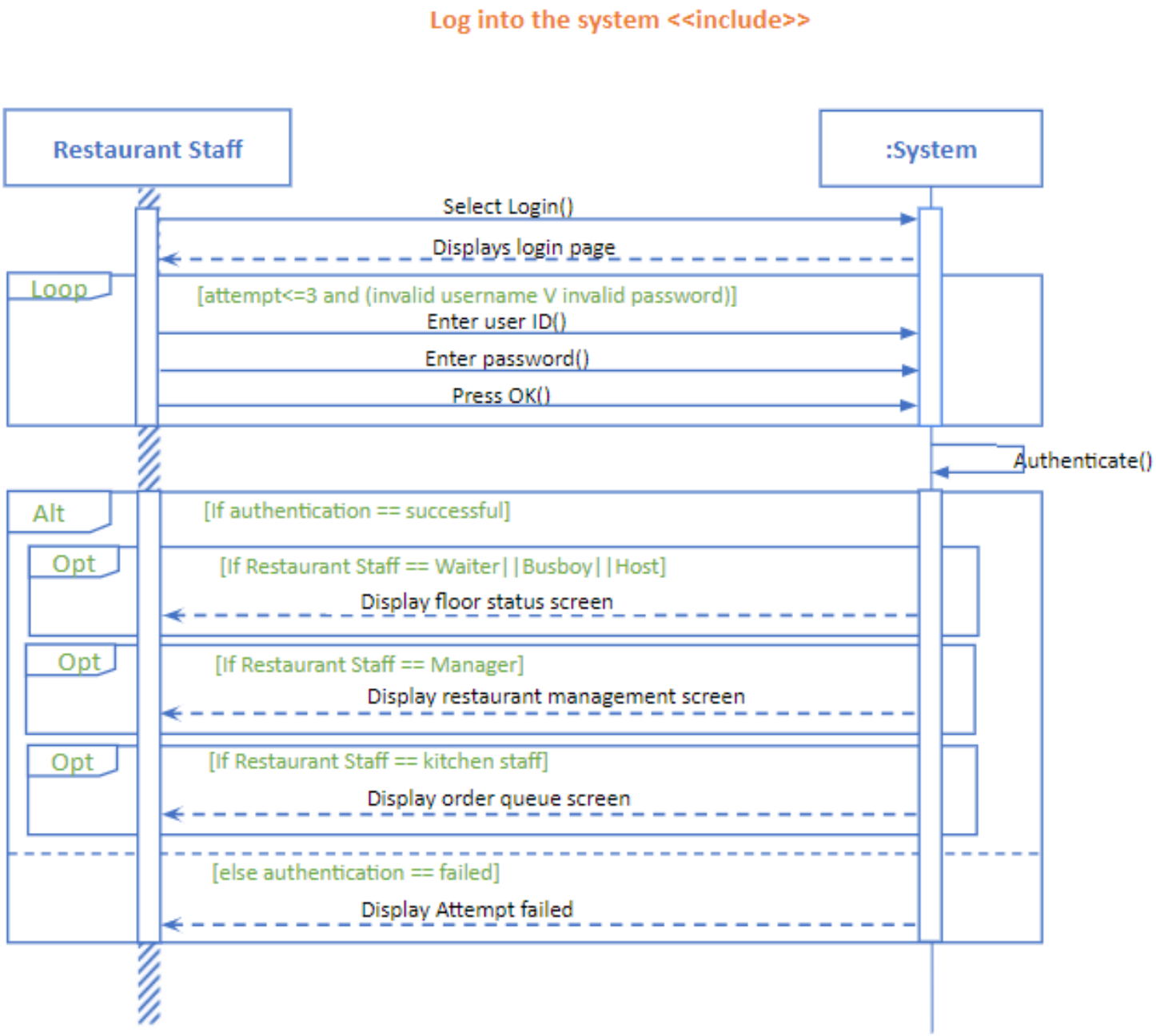
***Revision History***

| Version - description | Date | Description | Author |
|---|---|---|---|
| Inception draft | 11/04/2020 | First draft. To be refined primarily later | Dream Team |
| Elaboration 1 draft 1 | 11/05/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 2 | 11/06/2020 | Refined draft. | Dream Team |
| Elaboration 1 draft 3 | 11/26/2020 | Refined draft. | Dream Team |

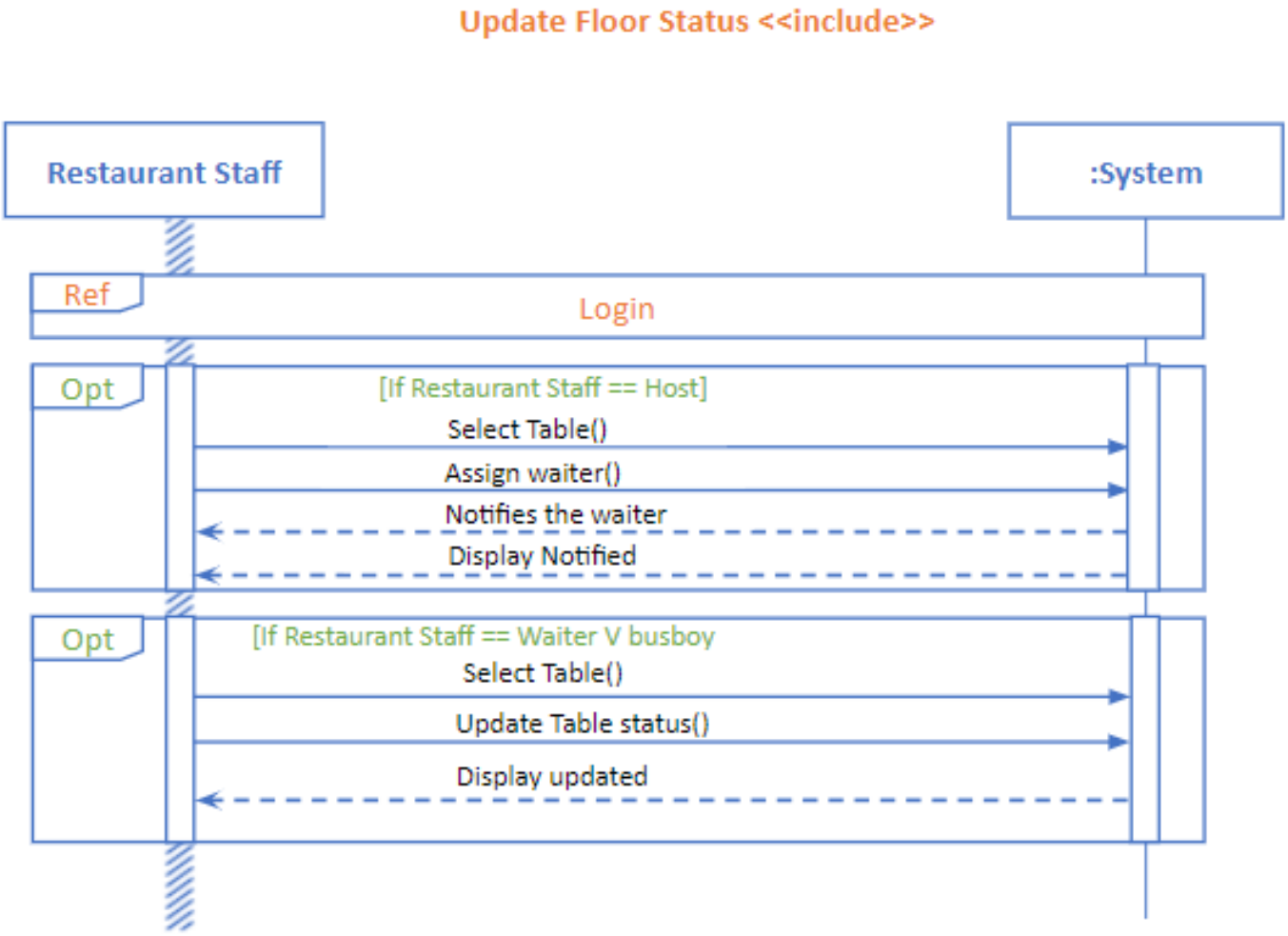
## 6.1 UC 1 System Sequence Diagram



## 6.2 UC 8 System Sequence Diagram

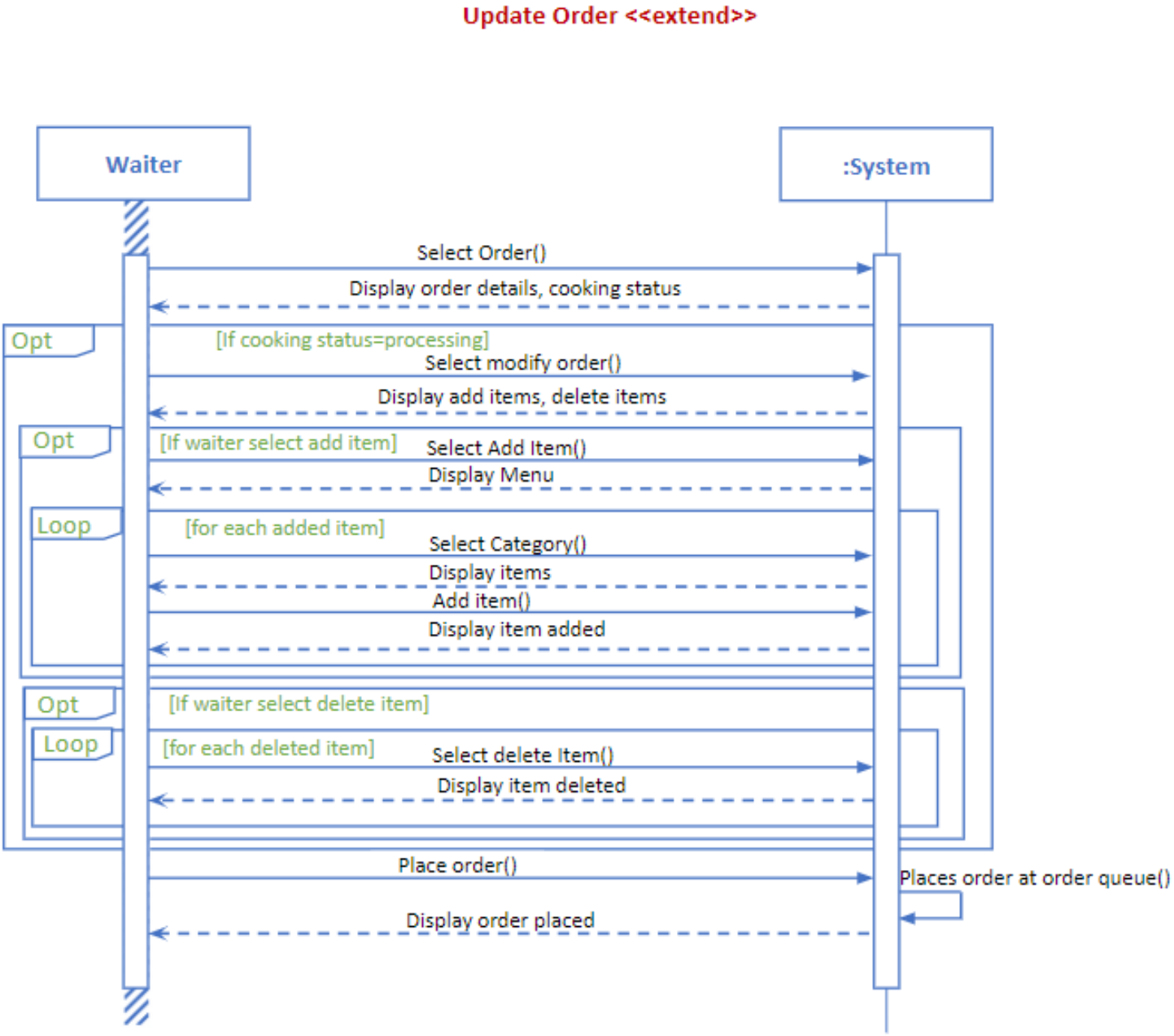


## 6.3 UC 5 System Sequence Diagram




**a.**

## 6.4 UC 15 System Sequence Diagram



## 6.5 UC 16 System Sequence Diagram