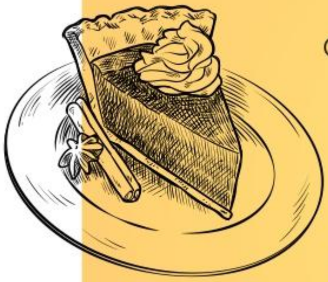
FINAL PROJECT REPORT

Canal Lounge

PRESENTED BY:

TEAM #4



CASSANDRA MACRI (40157528)
DANIEL HAMZE (40116438)
ELLIOT CETRAN (40043326)
DAVID HAZAN (40018105)
YI EN LIU (40128896)
SHIRLEY ARNSTEIN (40177567)
JULIAN SINTIM (40125943)

PRESENTED TO:

HOSSEIN AZARPANAH
BTM 495: INFORMATION SYSTEMS DESIGN AND IMPLEMENTATION
SECTION A
WINTER 2023

Table of Contents

Executive Summary

The Canal Lounge, a seasonal Amsterdam-inspired cocktail lounge floating on the Lachine Canal, has identified poor inventory tracking and resupply as issues affecting its operations. Currently, inventory tracking is done manually, leading to data inaccuracy and miscalculations of stock. Similarly, resupply decisions are based on manual inspections, leading to overstocking, stockouts, inaccurate costing, and reduced efficiency. As a solution, an automated inventory tracking and resupply system is proposed. The system will automatically update inventory levels and notify the owner of low-stock products, eliminating the need for manual tracking and inspections. The proposed system will increase efficiency, minimize waste, and maximize profitability. The project's special constraint is the short time frame due to the seasonal operation of the Canal Lounge.

The automated inventory tracking system was developed using various modeling diagrams. These include functional modeling which aims to describe the various functions of the system through use cases. The next diagram is structural modeling whereby the system is broken down into its constituent parts, which are represented as nodes or entities. As a result, a class diagram was developed to show the static structure of the inventory tracking system in terms of its classes, attributes, operations, and relationships. Next, is the behavioral modeling which involves the use of models, such as state machines, activity diagrams, and sequence diagrams, to describe the behavior of our system.

The class and methods design was used to create a well-organized and maintainable software system that meets its functional requirements. The classes are designed to represent the different entities in our system, such as orders, ingredients, and menu item. Each class has a set of attributes that describe its state and a set of methods that define its behavior. The methods then represent the specific tasks or operations on the data stored in a class.

The human-computer interaction design was utilized to create the user interface of the inventory tracking system. The HCI design focuses on how users would interact with our solution, with the goal of creating an interface that is intuitive, efficient, and satisfying.

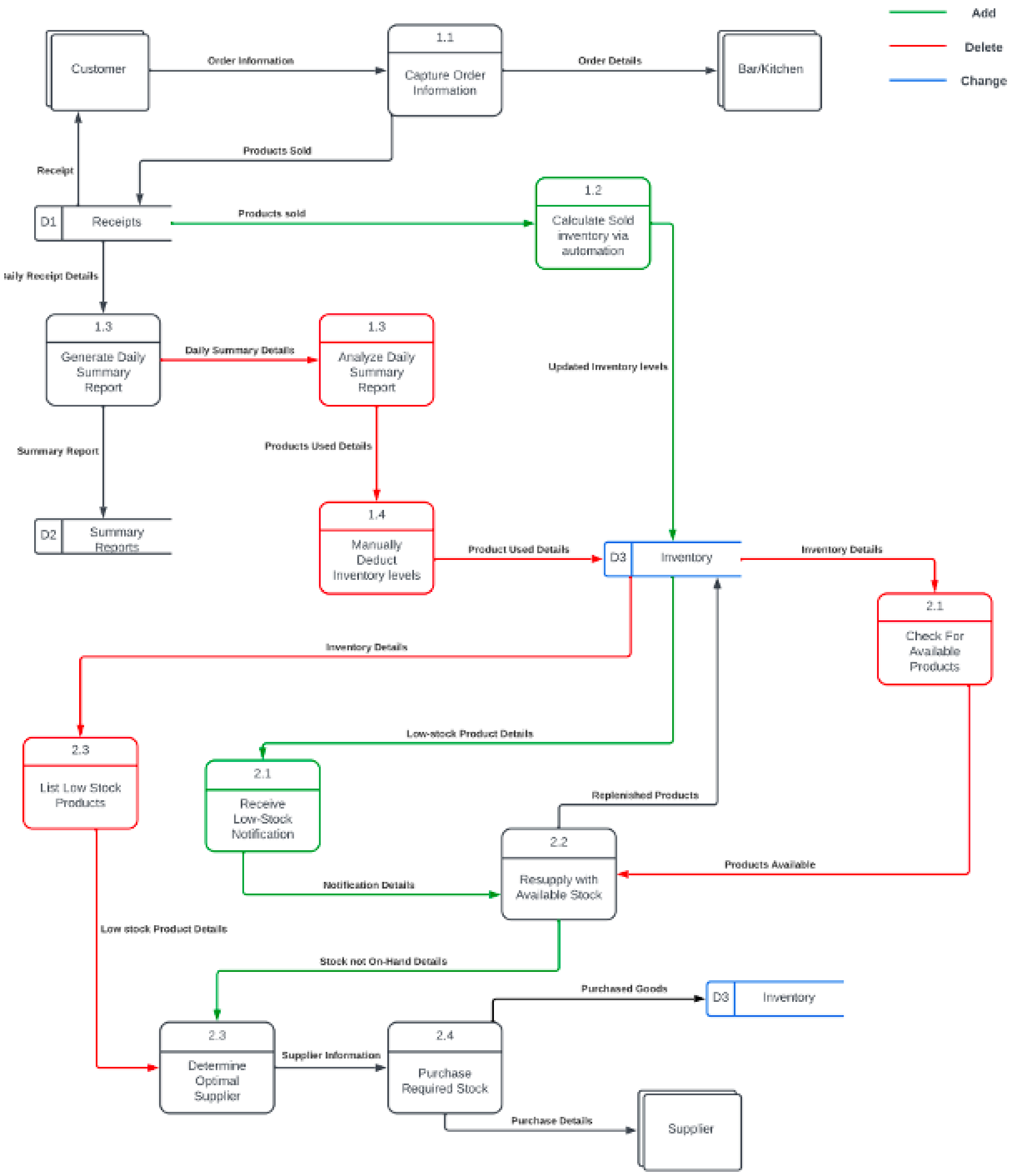
The deployment diagram shows the physical deployment of our system into the production environment. It also lists the different hardware components, software specifications, and cloud specifications, and shows the relationship between these. Lastly, the Python code for our prototype is included.

Team Project Proposal

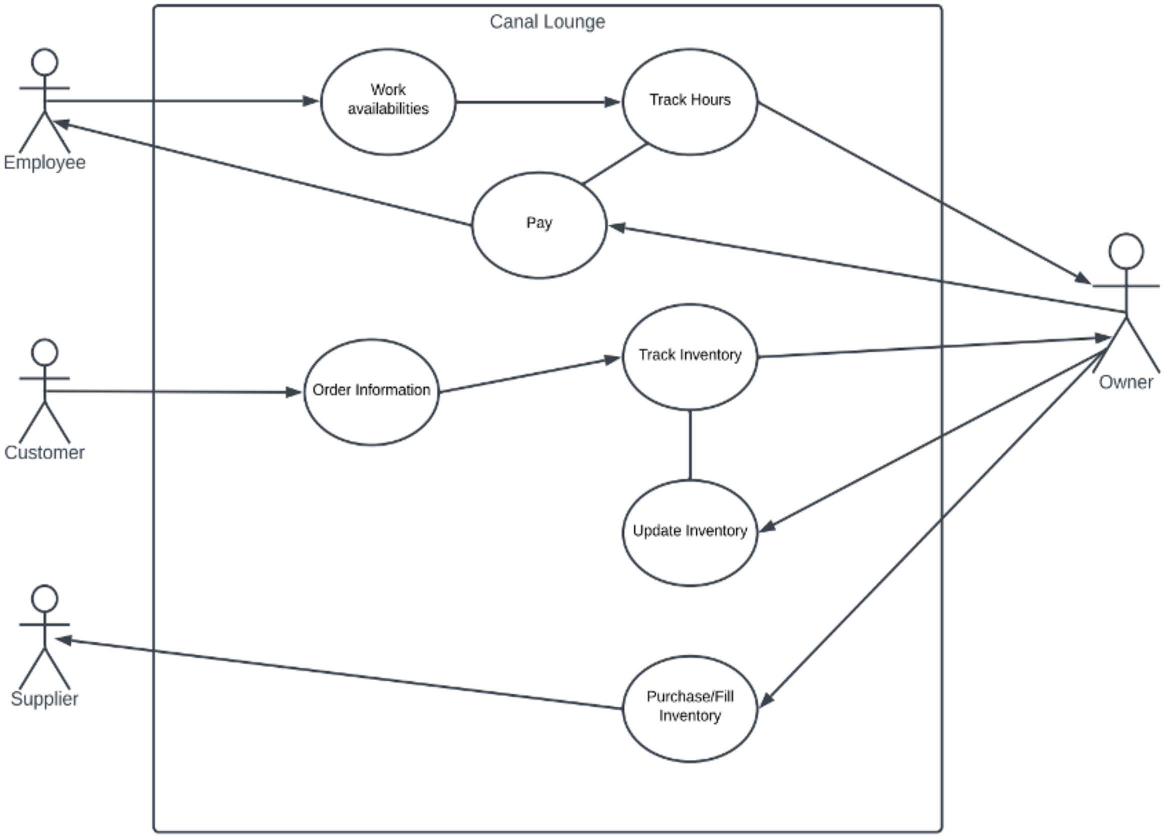
BTM 481 Project Information

Project Sub-System Proposed for this Project:

<u>Inventory Tracking & Resupply</u>



Use Case:



Narrative:

<u>Inventory Tracking & Resupply</u>

Proposed Solution: Automated Tracking Software

The inventory tracking is currently done manually by the owner, having to keep track of the inventory in his mind and some loose pieces of paper here and there. He then proceeds to update his excel sheet at home with the information he remembered and the summary receipt for the day. This method is less than optimal and often leads to miscalculations of stock and a data inaccuracy. While the current methods have allowed the business to run, it is a very inefficient process that not only causes data issues but as well as a lot of stress and time consumption for the owner. This allows for an IT opportunity to be exploited as these issues could be solved with an automated software that would track inventory after it is inputted in a new “Inventory” data store and would update automatically as products and ingredients are being used. According to user requirements, this data store should be online and available to the owner on location as well as his laptop at home, allowing him to track and verify in real time his stock levels.

As illustrated in our proposed system data-flow diagram, this proposed system would eliminate the previous processes 1.3 and 1.4, as the owner no longer would have to manually inspect the daily summary receipt and make note of what was sold and hence what stock was used. Instead, we now create a new process 1.2 where the software automatically updates the newly improved inventory data store D3 by subtracting the used products from the data store as soon as they are sold, allowing for real-time tracking of his inventory. This newly improved data store is now completely digitized, available across multiple devices and will now update itself after each product sold.

Similarly, the inventory resupply subsystem is closely tied to inventory tracking, as both systems go hand-in-hand; this is shown in the current system DFD as they both share the same data store D3 labeled “Inventory”. As in the previous system and explained in the current system, the problem encountered in the inventory resupply is very similar, lack of automation. The owner is already currently tracking his stock levels manually, and hence decides to resupply when he believes his stocks are low. He does not apply any known strategies for resupplying such as periodic replenishment or top-off replenishment, he just does so when he believes it is necessary. Since there is a lack of correct inventory tracking and demand forecasting this directly affects inventory resupply as the data may be inaccurate and lead to incorrect resupply of stock levels.

In our proposed system, the automation software used previously in the first subsystem inventory tracking, would also be applied to this subsystem. This would allow an all-in-one simple and efficient software system to track and be informed of his stock levels to be able to resupply accordingly. As illustrated in our proposed system data flow diagram, we removed process 2.1 and 2.3 which required the owner to manually verify his own stock both on location and at his home and make a list on a piece of paper of what product is low and would need to be re-ordered. As we have seen in our user requirements analysis, the new application of the software would send him a notification when he is consulting his data store telling him what product stock is low and needs to be replenished. This system would allow him to set a reorder point strategy that would ensure he is never out of stock entirely of a certain product. As a result, this proposed system allows for an elimination of data inaccuracy, human error, as well as saving the owner a lot of time and stress for the future.

Part 1 of Project Proposal

Project Sponsor:

The sponsor is our client **Bernard Roach.**

The Canal Lounge, a seasonal business is an Amsterdam inspired cocktail lounge alluring to many as the bar is on a remodeled, 50-year-old Canadian Fly boat, can be found floating on the Lachine Canal.

Business Need:

The three primary company subsystems that have been dissected are: employee payroll and scheduling, inventory tracking, and inventory resupply issues. The subsystems that we will be focusing on are the inventory tracking and resupply issues. Poor inventory tracking and resupply in a restaurant can lead to a variety of issues such as stockouts, overstocking, inaccurate costing, poor cash flow, and reduced efficiency. These subsystems are essential to ensure the establishment has the necessary ingredients and supplies to run smoothly, minimize waste, and maximize profitability.

Business Requirements:

Changes to the inventory tracking subsystem focus on facilitating the owner's control and monitoring over his inventory. Additionally, the inventory should be automatically updated once orders are inputted into the system. Here are requirements for this sub-system. Moreover, the inventory replenishment sub-system focuses on sending notifications to the owner regarding products that are low in stock, eliminating the need to constantly check current stock levels and determine the best times to replenish products.

Functional Requirements:

- Viewing current inventory from anywhere at any time, in order to gain an accurate picture of the products currently available.
- Inventory levels should be automatically edited/reduced once fulfilled orders are inputted into the system.

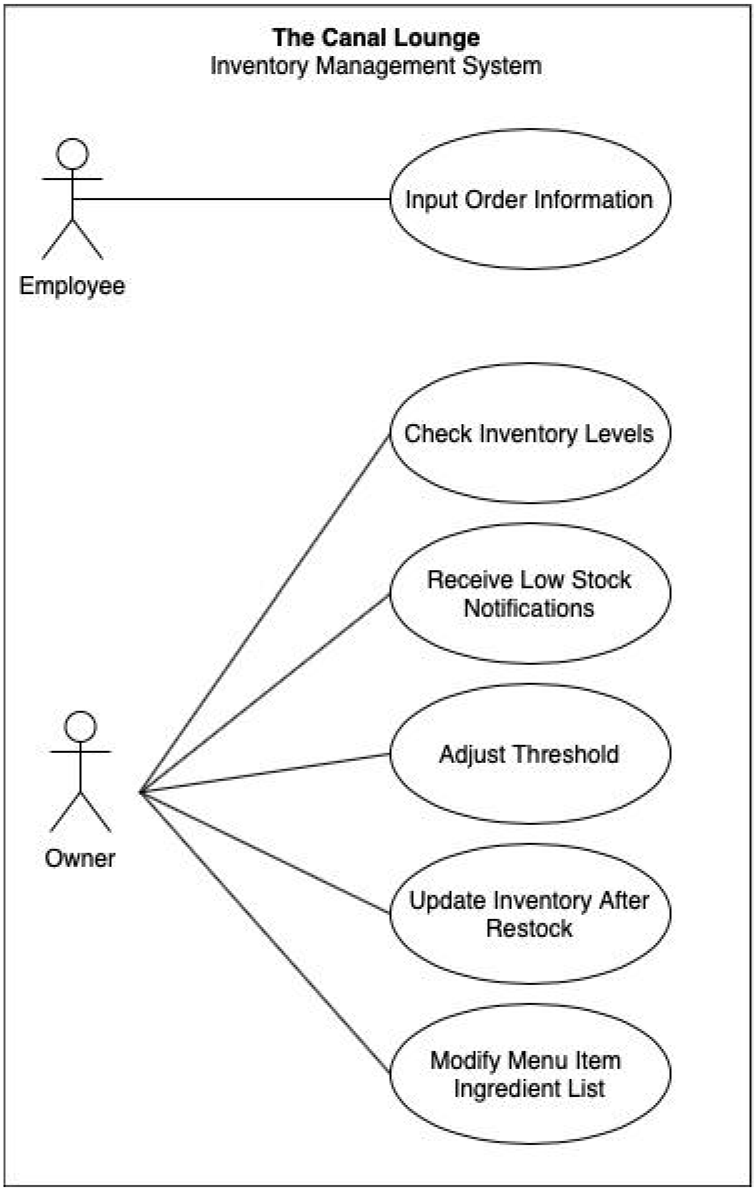
Non-Functional Requirements:

- Inventory updates should reflect shortly after the order is inputted; process should take less than a minute.
- Receiving alerts about low-stock products in order to eliminate the need to regularly check inventory.

Business Value: The Canal Lounge will benefit from a more efficient and timely system thanks to our project. We will create an easier system for the business operations.

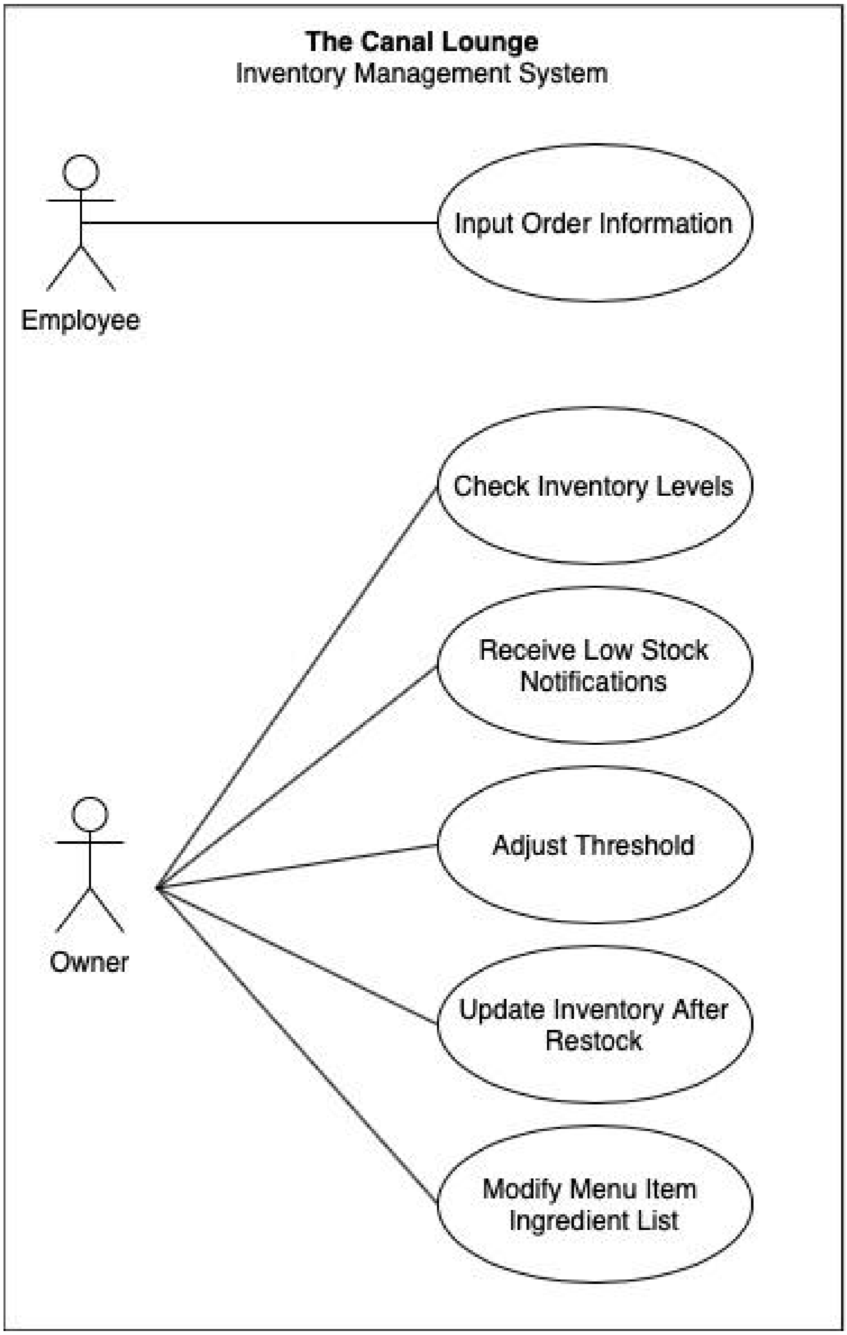
Special Issues or Constraints: A notable constraint is the seasonal operation time of the Canal Lounge as the business closes over the winter period. For this reason, all implementations have to be completed in a short time frame.

Part 2 of Project Proposal


The Canal Lounge
Inventory Management System
Employee
Input Order Information
Owner
Check Inventory Levels
Receive Low Stock Notifications
Adjust Threshold
Update Inventory After Restock
Modify Menu Item Ingredient List

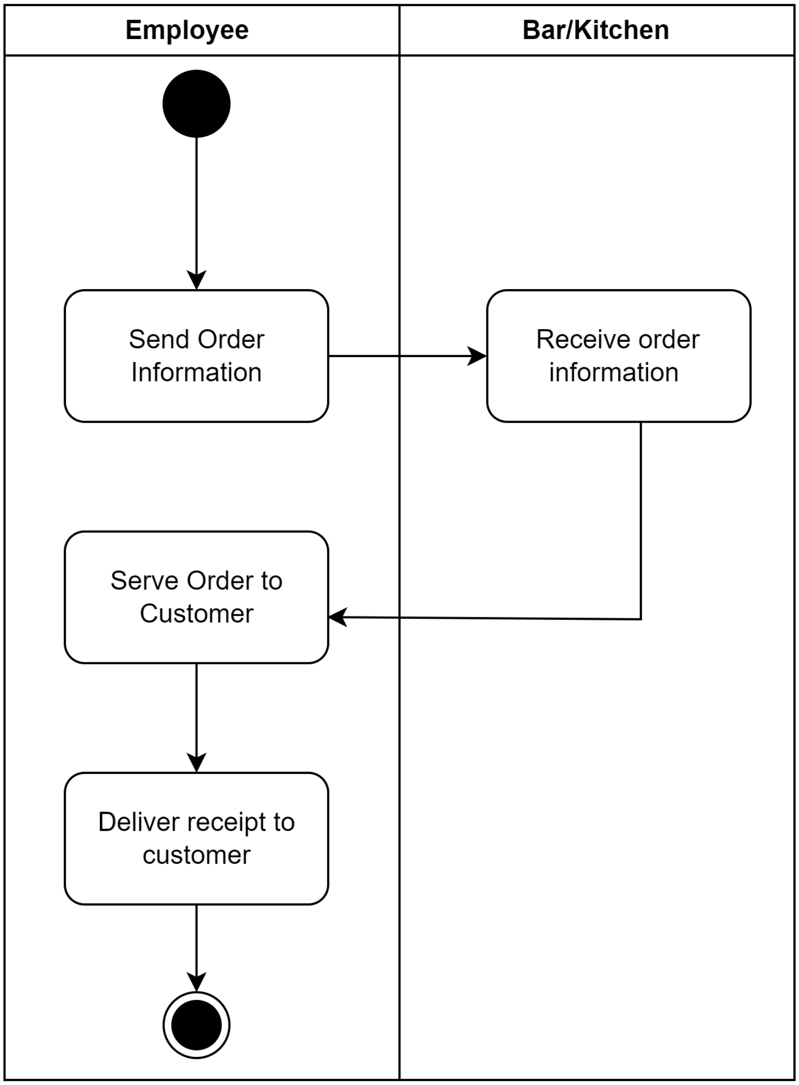
Functional Modeling

Use-Case Diagram

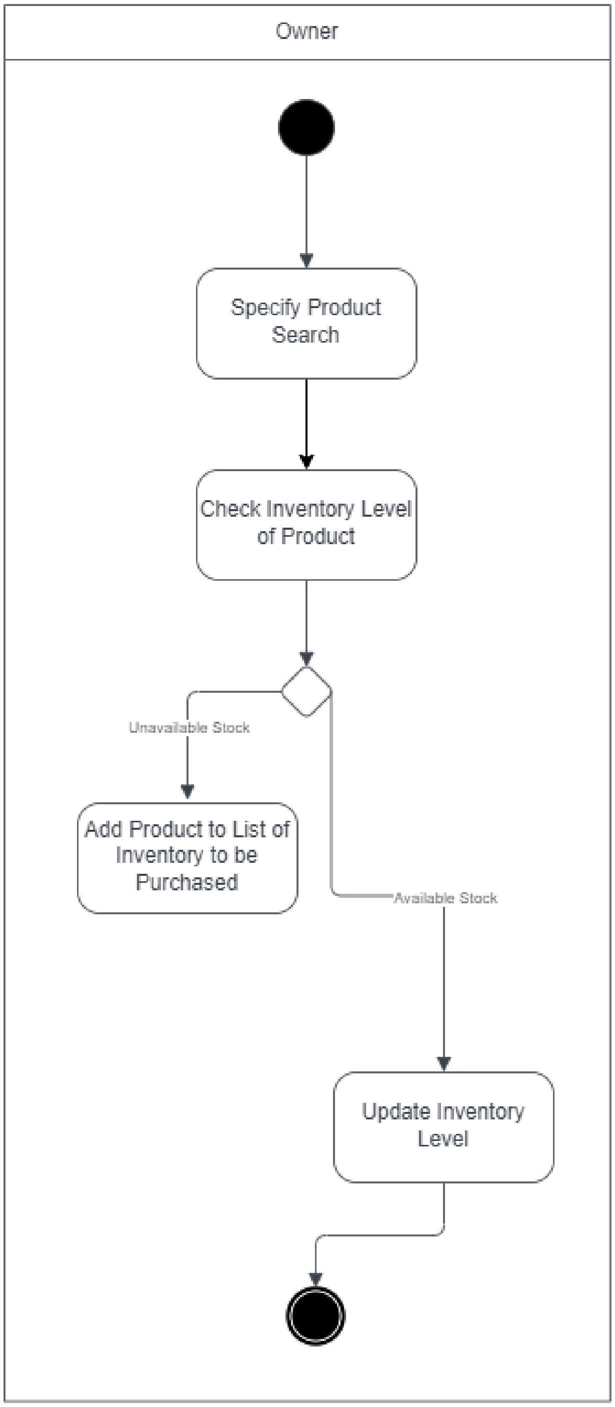

The Canal Lounge
Inventory Management System
Employee
Input Order Information
Check Inventory Levels
Receive Low Stock Notifications
Adjust Threshold
Owner
Update Inventory After Restock
Modify Menu Item Ingredient List

Activity Diagrams

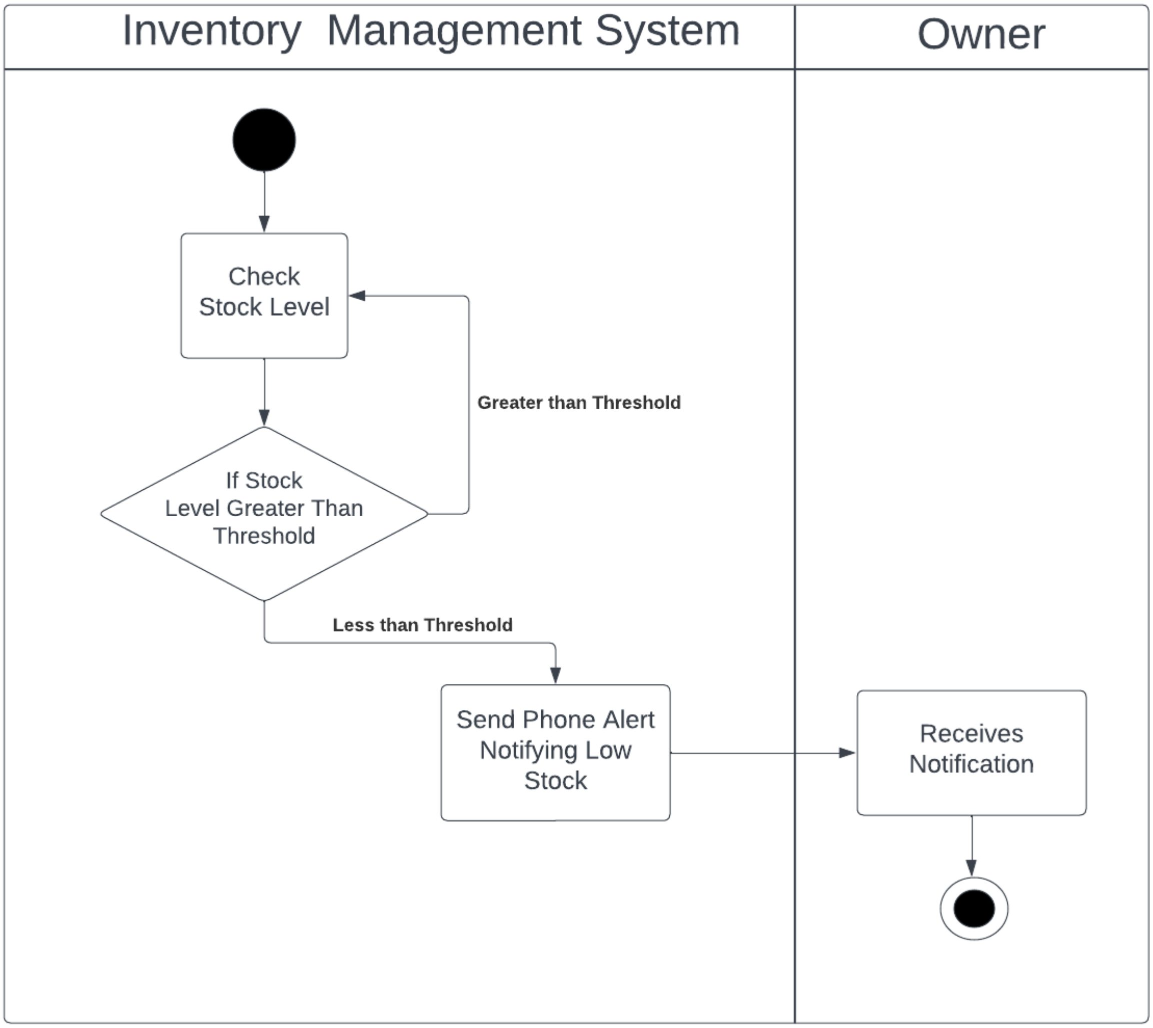
Activity Diagram for **“Input Order Information”** Use-Case:



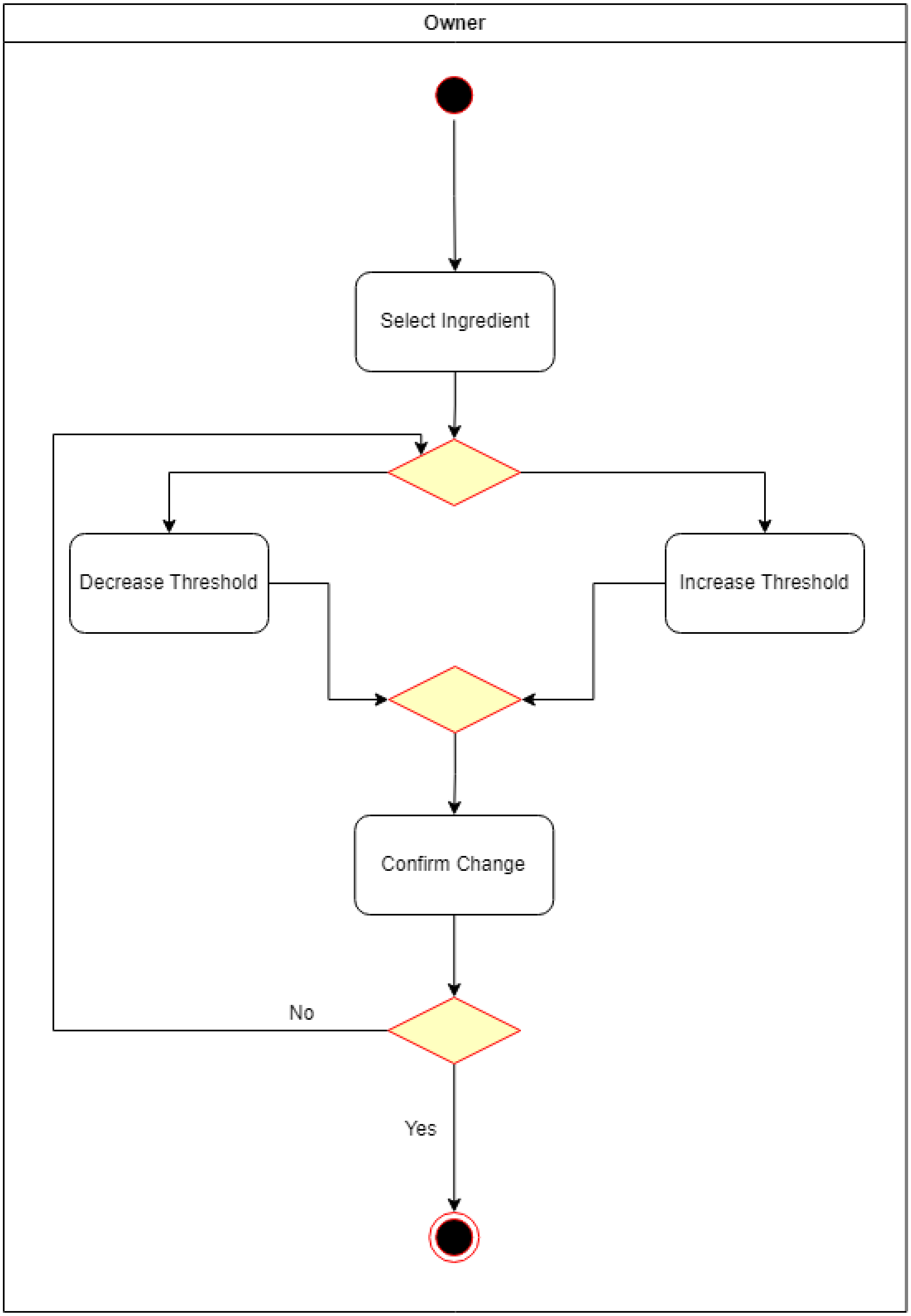
Activity Diagram for **“Check Inventory Levels”** Use-Case:



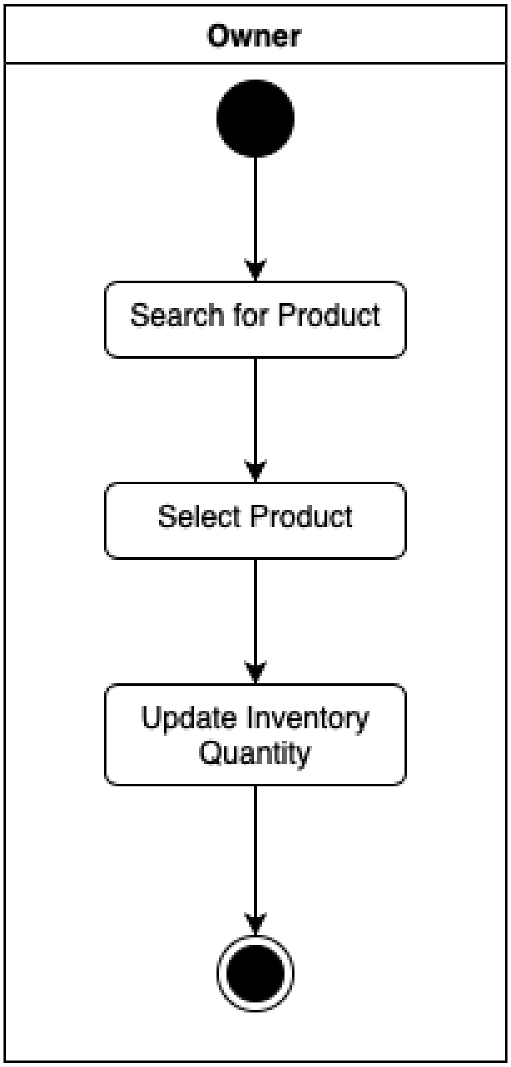
Activity Diagram for **"Receive Low Stock Notifications"** Use-Case:


Inventory Management System
Owner
Check Stock Level
Greater than Threshold
If Stock Level Greater Than Threshold
Less than Threshold
Send Phone Alert Notifying Low Stock
Receives Notification

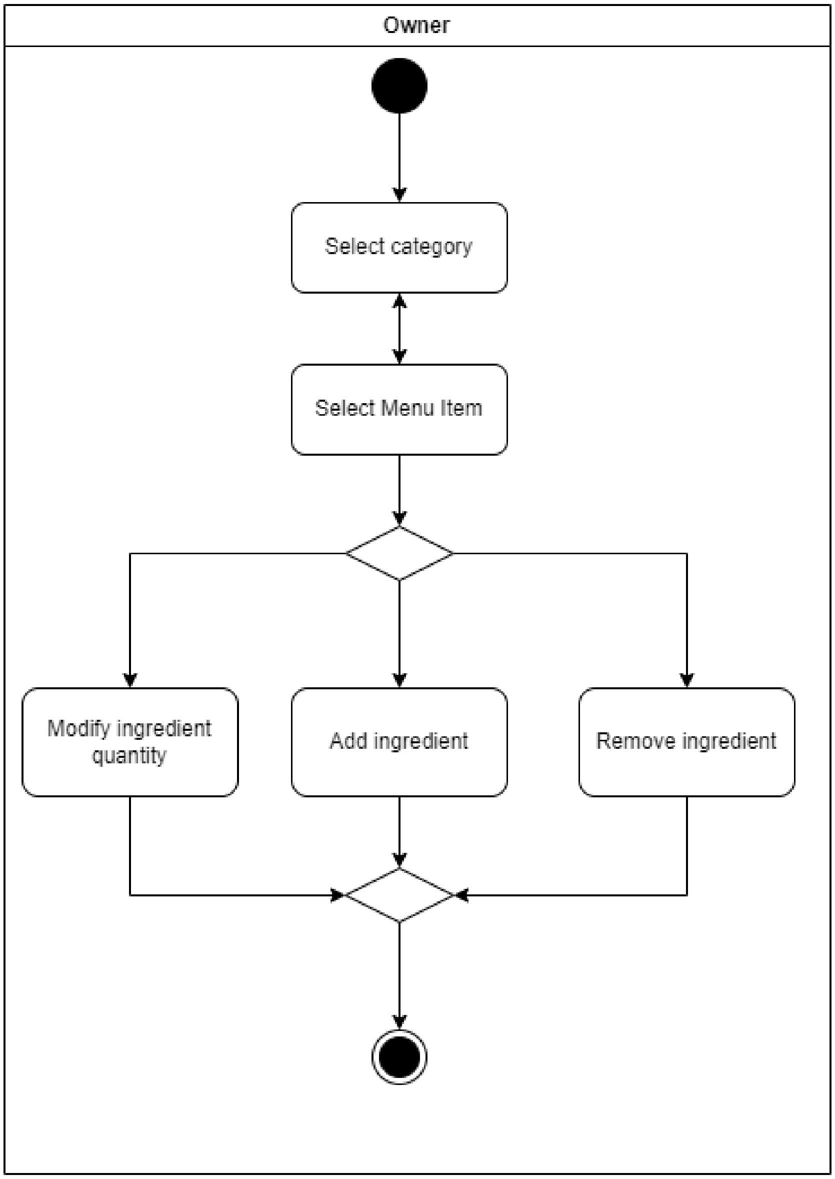
Activity Diagram for **“Adjust Threshold”** Use-Case:



Activity Diagram for **“Update Inventory After Restock”** Use-Case:



Activity Diagram for **“Modify Menu Item Ingredients/Quantities”** Use-Case:



Use Case Descriptions

Use-Case Description for **"Input Order Information"** Use-Case:

Use Case Name: **Input order information**	ID: **01**	Importance Level: **High**
Primary Actor: Employee	Use Case Type: Detail, Essential	
Stakeholders and Interests: ● Employee captures customers order information ● Bar/Kitchen receives order		
Brief Description: **This use case shows an employee input order information**		
Trigger: Employee open POS system Type: External		
Relationships: Association: Employee Include: Extend: Generalization:		
Normal Flow of Events: 1. Employee sends the order information from the customer. 2. Kitchen staff receives the order and begins preparation. 3. Employee serves the order to the customer. 4. Employee delivers receipt to customer.		
SubFlows: N/A		
Alternate/Exceptional Flows: N/A		

Use-Case Description for **“Check Inventory Levels”** Use-Case:

Use Case Name: Update Inventory Levels	ID: 02	Importance Level: High
Primary Actor: Owner	Use Case Type: Essential, Details	
Stakeholders and Interests: ● Owner: Would like to accurately repurchase stock. ● Customers: Expect the bar to be stocked with all their items on the menu ● Employees: Want the bar to be stocked with all the items so they can provide a better experience thus receiving a better tip.		
Brief Description: The owner checks inventory availability and restocks if inventory unavailable		
Trigger: Stock Needs Manual Daily Update by Owner Type:		
Relationships: Association: Owner Include: Extend: Generalization:		
Normal Flow of Events: 1. Owner Checks Inventory for Low Stock Products 2. Owner Updates Inventory Information		
SubFlows: 1. Items Available in Stock 2. Items Unavailable in Stock a. Create List of Products to be Purchased b. Purchase Stock Outside the System		
Alternate/Exceptional Flows: N/A		

Use-Case Description for **“Receive Low Stock Notifications”** Use-Case:

Use Case Name: Receive Low Stock Notifications	ID: 03	Importance Level: Medium
Primary Actor: Inventory Management System	Use Case Type: Essential	
Stakeholders and Interests: ● Owner: Would like to accurately repurchase stock. ● Suppliers: Would like to have clear communication and no problems with too little or too much purchased. ● Customers: Expect the bar to be stocked with all their items on the menu ● Employees: Want the bar to be stocked with all the items so they can provide a better experience thus receiving a better tip.		
Brief Description: The system keeps checking inventory levels until a threshold is triggered notifying the owner.		
Trigger: Stock goes below threshold Type: Internal		
Relationships: N/A Association: Owner Include: Extend: Generalization:		
Normal Flow of Events: 1. System checks the stock levels 2. If the stock levels are greater than the threshold the system will check the stock levels again 3. If the stock levels are lower than the threshold a. Send low stock notification to the owner 4. Owner receives notification		
SubFlows: N/A		
Alternate/Exceptional Flows: N/A		

Use-Case Description for **“Adjust Threshold”** Use-Case:

Use Case Name: Adjust Threshold	ID: 04	Importance Level: Medium
Primary Actor: Owner	Use Case Type: Essential, Detail	
Stakeholders and Interests: ● Owner: Decides when the right time to purchase new quantities is; ● Supplier: Supplies product for owner based on the purchase order		
Brief Description: The owner can set the threshold quantity for ingredients. Owner must confirm the threshold quantity for an ingredient.		
Trigger: Owner decides to purchase new quantities based on low inventory levels Type: External		
Relationships: Association: Owner Include: N/A Extend: N/A Generalization: N/A		
Normal Flow of Events: 1. Owner Selects Ingredient 2. Owner Increases Threshold / Decreases Threshold 3. Owner Confirms Change		
SubFlows: N/A		
Alternate/Exceptional Flows: N/A		

Use-Case Description for **“Update Inventory After Restock”** Use-Case:

Use Case Name: Update Inventory After Restock

ID: 05

Importance Level: High

Primary Actor: Owner

Use Case Type: Essential (purpose)
Detail (amount of information)

Stakeholders and Interests:

- The owner wants to update inventory quantities after restocking.

Brief Description: The owner is updating inventory quantities after restocking inventory.

Trigger: Inventory is restocked.
Type: External

Relationships: N/A

Association: Owner
Include:
Extend:
Generalization:

Normal Flow of Events:

1. Owner searches for the product that he wants to update.
2. Owner selected product.
3. Owner updates inventory following restock.

SubFlows: N/A

Alternate/Exceptional Flows: N/A

Use-Case Description for **“Create Purchase Order”** Use-Case:

Use Case Name: Create Purchase Order

ID: 06

Importance Level: High

Primary Actor: Owner

Use Case Type: Essential, Detail

Stakeholders and Interests:

- Owner: Decides when the right time to purchase new quantities is;
- Supplier: Supplies product for owner based on the purchase order

Brief Description: When inventory levels are low, the restaurant needs to order more products from their different suppliers.

Trigger: Owner decides to purchase new quantities based on low inventory levels

Type: External

Relationships:

Association: Supplier
Include: N/A
Extend: N/A
Generalization: N/A

Normal Flow of Events:

1. Create purchase order
2. Review Purchase order
3. Check inventory levels
4. Confirm delivery date
5. Prepare and package the products for shipment
6. Arrange for the product to be shipped
7. Send an invoice to the customer
8. Pay invoice

SubFlows: N/A

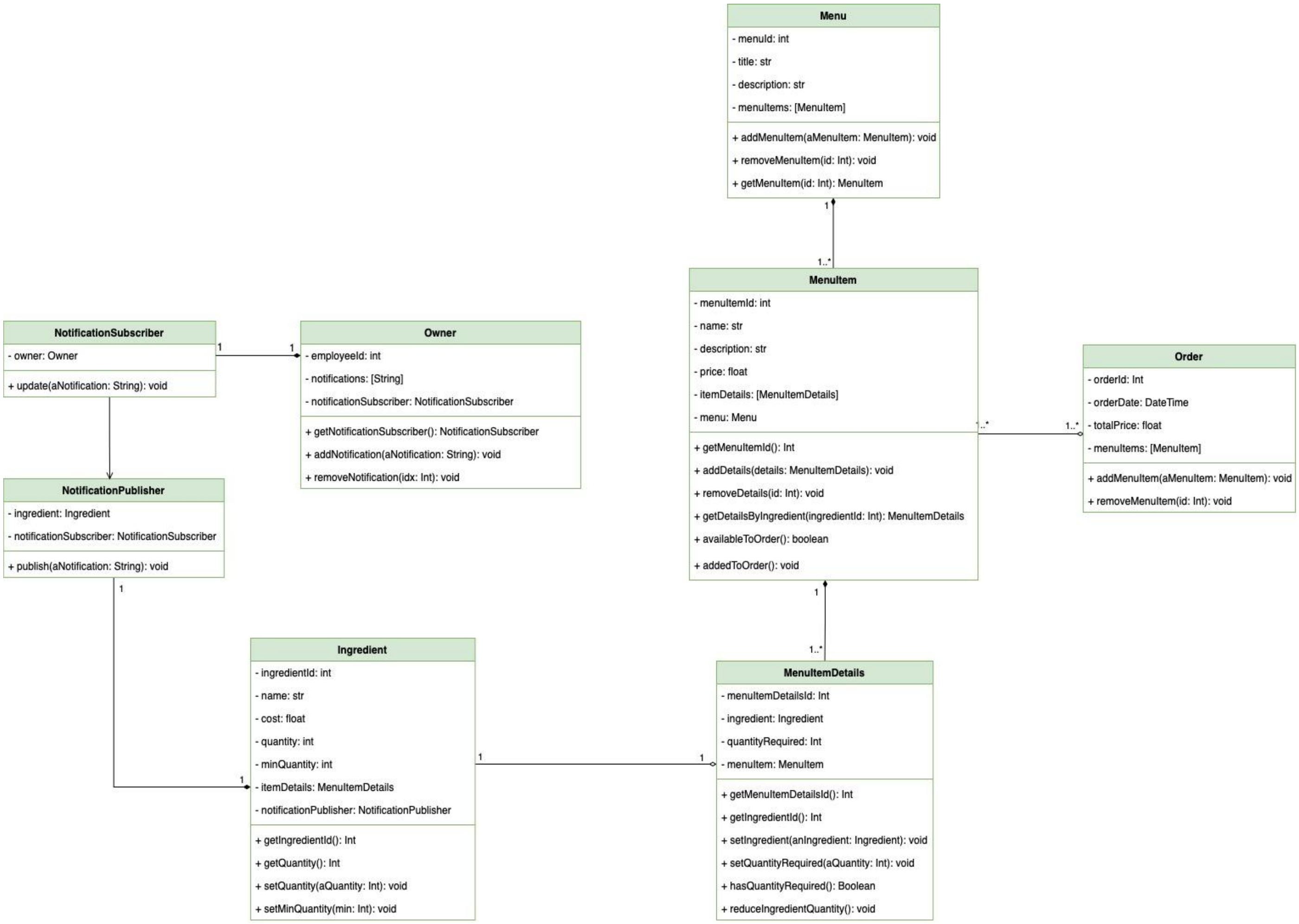
Alternate/Exceptional Flows: N/A

Use-Case Description for **“Modify Menu Item Ingredients/Quantities” Use-Case:**

Use Case Name: Modify Menu Item Ingredients and Quantities	ID: 07	Importance Level: Medium
Primary Actor: Owner	Use Case Type: Essential	
Stakeholders and Interests: ● Owner: controls how much inventory should be reduced every time a menu item is inputted by a waiter		
Brief Description: Each menu item will have ingredients and quantities associated with it. When an order comprised of different menu items is inputted, the inventory levels of the ingredients included in the menu items are reduced accordingly. Thus, the owner may want to modify the ingredients and their quantities of a menu item.		
Trigger: Click on button in system to access area to modify menu items Type: Internal		
Relationships: N/A Association: Owner Include: Extend: Generalization:		
Normal Flow of Events: 1. Owner Select Category (Drinks, Entrees, Mains) 2. Owner Select menu item 3. Owner Modify ingredient quantity/ add ingredient/ remove ingredient 4. The menu item’s ingredients and their quantities are updated		
SubFlows: N/A		
Alternate/Exceptional Flows: N/A		

Structural Modeling

Class Diagram


Menu
- menuId: int
- title: str
- description: str
- menuItems: [MenuItem]
+ addMenuItem(aMenuItem: MenuItem): void
+ removeMenuItem(id: Int): void
+ getMenuItem(id: Int): MenuItem
1
1..*
NotificationSubscriber
- owner: Owner
+ update(aNotification: String): void
1
1
Owner
- employeeId: int
- notifications: [String]
- notificationSubscriber: NotificationSubscriber
+ getNotificationSubscriber(): NotificationSubscriber
+ addNotification(aNotification: String): void
+ removeNotification(idx: Int): void
MenuItem
- menuItemId: int
- name: str
- description: str
- price: float
- itemDetails: [MenuItemDetails]
- menu: Menu
+ getMenuItemId(): Int
+ addDetails(details: MenuItemDetails): void
+ removeDetails(id: Int): void
+ getDetailsByIngredient(ingredientId: Int): MenuItemDetails
+ availableToOrder(): boolean
+ addedToOrder(): void
1..*
1..*
Order
- orderId: Int
- orderDate: DateTime
- totalPrice: float
- menuItems: [MenuItem]
+ addMenuItem(aMenuItem: MenuItem): void
+ removeMenuItem(id: Int): void
NotificationPublisher
- ingredient: Ingredient
- notificationSubscriber: NotificationSubscriber
+ publish(aNotification: String): void
1
1
1..*
Ingredient
- ingredientId: int
- name: str
- cost: float
- quantity: int
- minQuantity: int
- itemDetails: MenuItemDetails
- notificationPublisher: NotificationPublisher
+ getIngredientId(): Int
+ getQuantity(): Int
+ setQuantity(aQuantity: Int): void
+ setMinQuantity(min: Int): void
1
1
1
MenuItemDetails
- menuItemDetailsId: Int
- ingredient: Ingredient
- quantityRequired: Int
- menuItem: MenuItem
+ getMenuItemDetailsId(): Int
+ getIngredientId(): Int
+ setIngredient(anIngredient: Ingredient): void
+ setQuantityRequired(aQuantity: Int): void
+ hasQuantityRequired(): Boolean
+ reduceIngredientQuantity(): void

CRC Cards

Front:

Class Name: Menu	**ID:** 01	**Type:** Concrete, Domain
Description: List of the items currently available on the restaurant/owner’s menu.		**Associated Use Cases:** 01, 07

Responsibilities	**Collaborators**
- Add menu item	Menu item, Menu item details, Ingredient
- Remove menu item	Menu item, Menu item details, Ingredient

Back:

Attributes:

- Menu ID
- Title
- Description
- Menu items

Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: MenuItem (**composition**)

Front:

Class Name: Order	**ID:** 02	**Type:** Concrete, Domain
Description: List of the menu items ordered, the price, and the date of the order.		**Associated Use Cases:** 01

Responsibilities	**Collaborators**
- Add menu item	Menu item
- Remove menu item	Menu item

Back:

Attributes: - Order ID - Order date - Total price - Menu items
Relationships: **Generalization (a-kind-of):** **Aggregation (has-parts):** MenuItem **Other Associations:**

Front:

Class Name: MenuItem	**ID:** 03	**Type:** Concrete, Domain
Description: Item in the menu and list of its menu item details, the price, the name, and the description.		**Associated Use Cases:** 01, 07

Responsibilities	**Collaborators**
- Add details	Menu item details, Ingredient
- Remove details	Menu item details, Ingredient
- Available to order	Menu item details, Ingredient
- Added to order	Menu item details, Ingredient

Back:

Attributes: - Menu item ID - Name - Description - Price - Item Details - Menu

Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: MenuItemDetails (**composition**)

Front:

Class Name: Ingredient	**ID:** 04	**Type:** Concrete, Domain
Description: Each necessary ingredient for a menu item.		**Associated Use Cases:** 02, 03, 04, 05, 06, 07

<u>**Responsibilities**</u>	<u>**Collaborators**</u>
N/A	N/A

Back:

Attributes: - Ingredient ID - Name - Cost - Quantity - Minimum quantity - Item details - Notification publisher

Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: NotificationPublisher (**composition)**

Front:

Class Name: NotificationPublisher	**ID:** 05	**Type:** Abstract, Domain
Description: Mechanism for owner to subscribe to events related to low stock inventory threshold.		**Associated Use Cases:** 03

<u>**Responsibilities**</u>	<u>**Collaborators**</u>
- Subscribe	Notification subscriber
- Unsubscribe	Notification subscriber
- Publish	N/A

Back:

Attributes: - Ingredient - Notification subscriber

Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: NotificationSubcriber (**association**)

Front:

Class Name: NotificationSubscriber	**ID:** 06	**Type:** Abstract, Domain
Description: Owner (subscriber) is notified of low stock inventory threshold.		**Associated Use Cases:** 03

<u>**Responsibilities**</u>	<u>**Collaborators**</u>
- Update	N/A

Back:

Attributes: - Owner

Relationships: N/A
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations:

Front:

Class Name: MenuItemDetails	**ID:** 07	**Type:** Concrete, Domain
Description: Details of the item in the menu containing an ingredient and the quantity required of that ingredient.		**Associated Use Cases:** 01, 07

Responsibilities	**Collaborators**
- Has quantity required	- Ingredient
- Reduce ingredient quantity	- Ingredient

Back:

Attributes: - Menu item details ID - Ingredient - Quantity required - Menu item

Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: MenuItem

Front:

Class Name: Owner	**ID:** 08	**Type:** Concrete, Domain
Description: Maintains a list of low stock notifications.		**Associated Use Cases:** 03

Responsibilities	**Collaborators**
- Add notification	- N/A
- Remove notification	- N/A

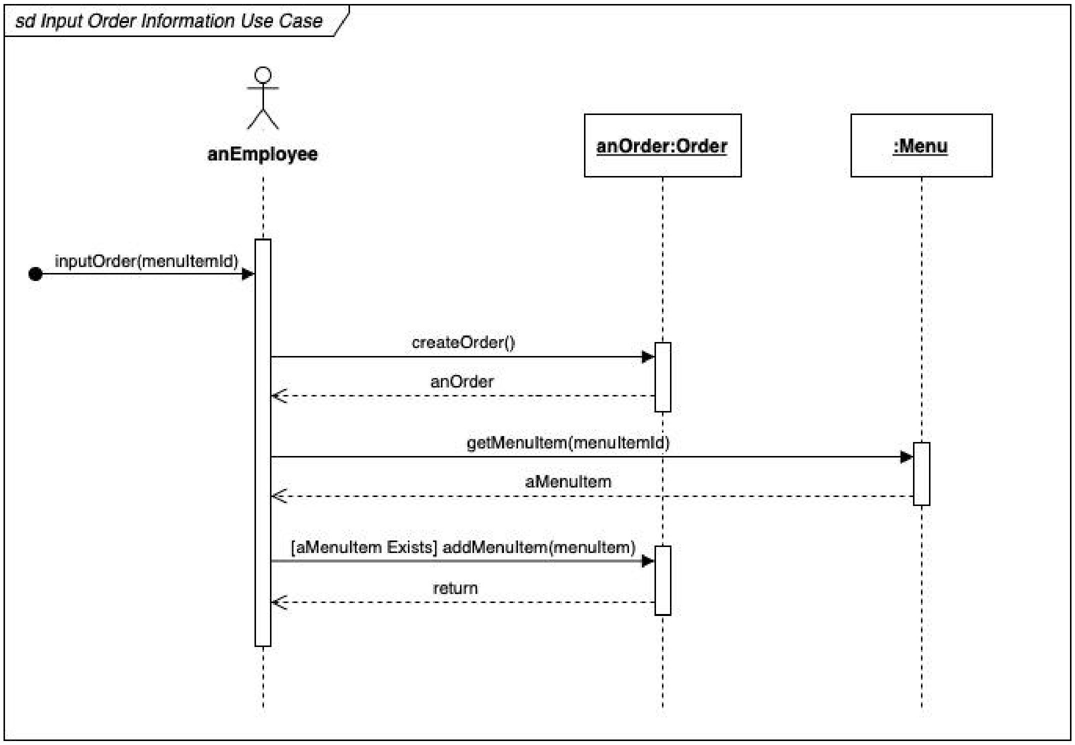
Back:

Attributes: - Employee ID - Notifications - Notification subscriber

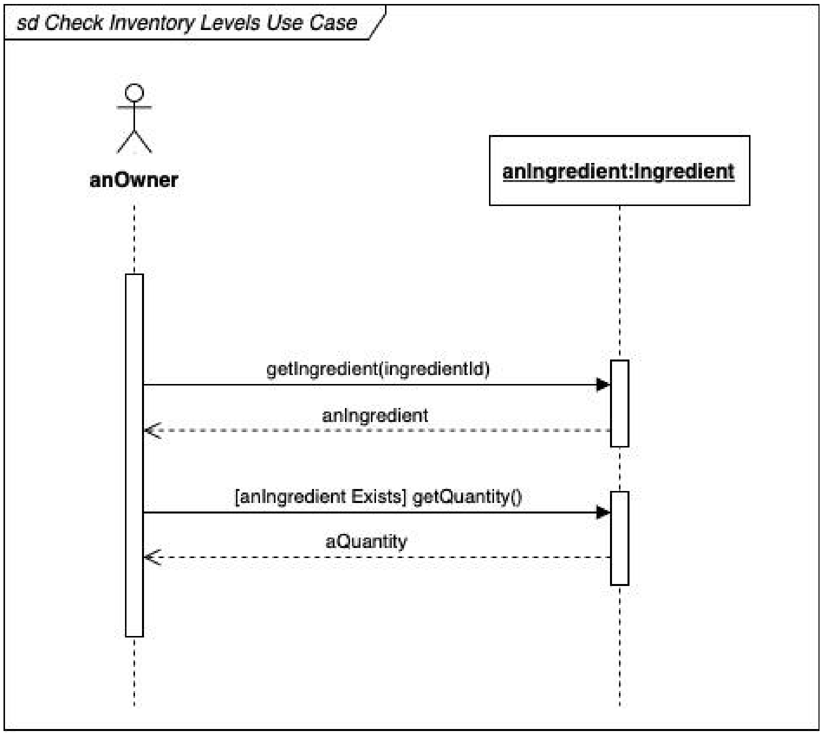
Relationships:
Generalization (a-kind-of):
Aggregation (has-parts):
Other Associations: NotificationSubscriber (**composition**)

Behavioral Modeling

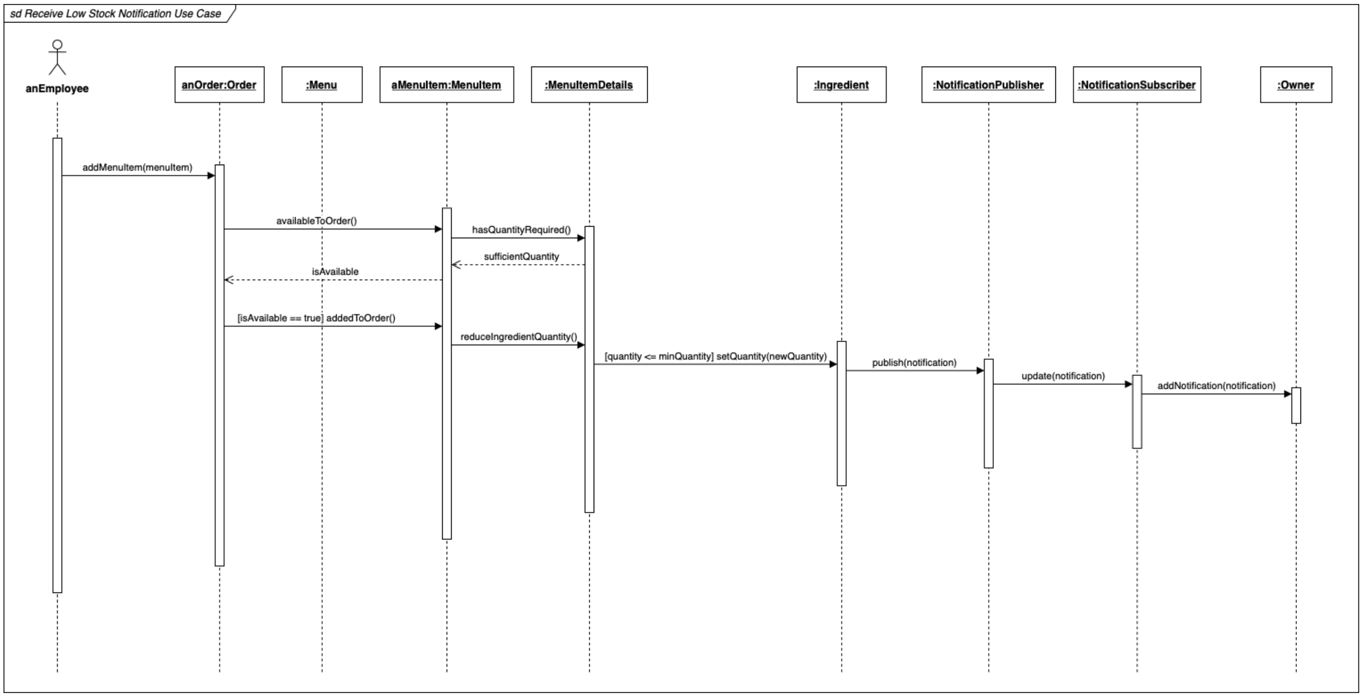
Use Case 1: **Input Order Information**



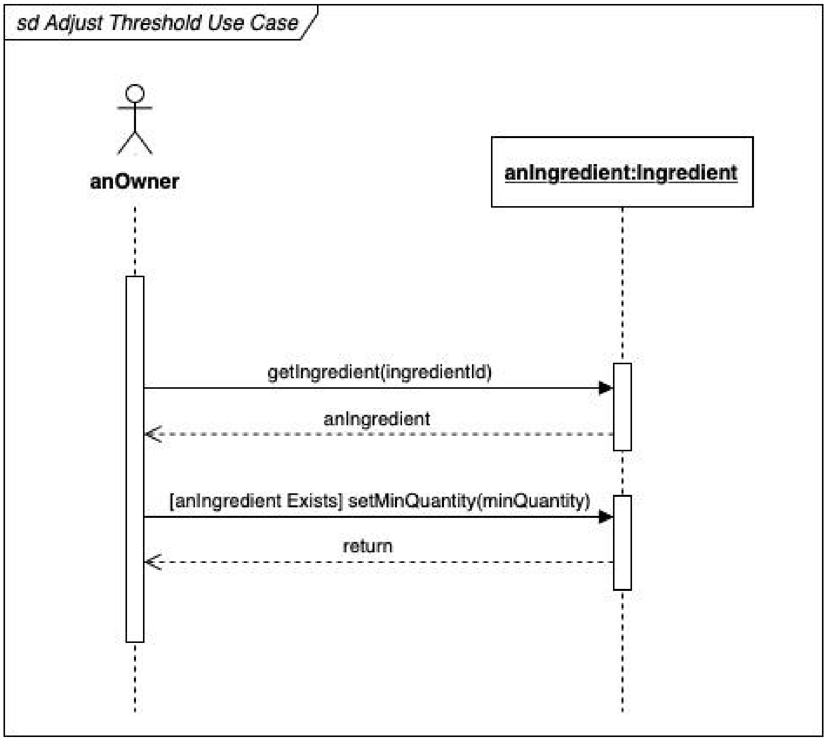
Use Case 2: **Check Inventory Levels**


sd Check Inventory Levels Use Case
anOwner
anIngredient:Ingredient
getIngredient(ingredientId)
anIngredient
[anIngredient Exists] getQuantity()
aQuantity

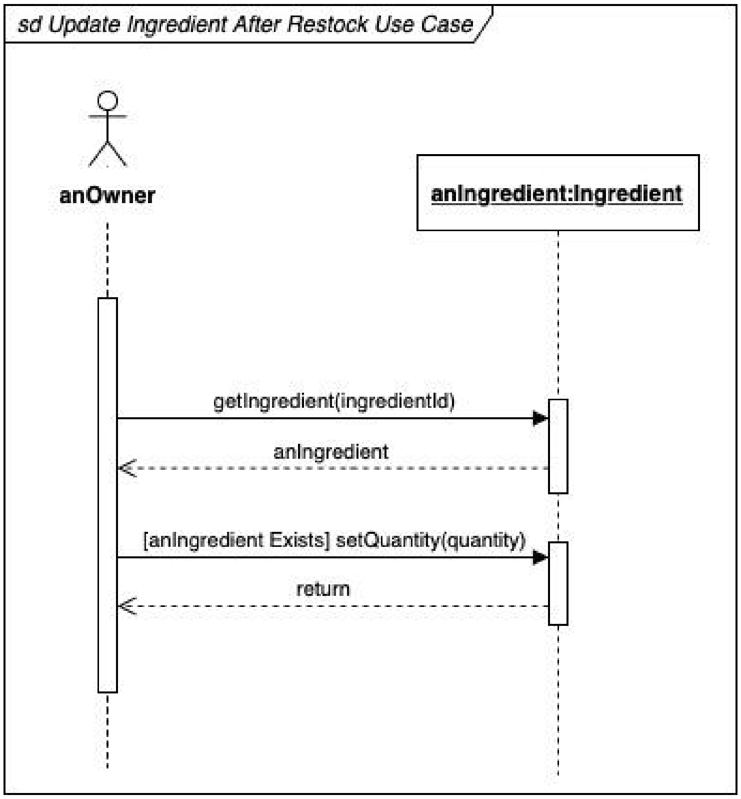
Use Case 3: **Receive Low Stock Notifications**



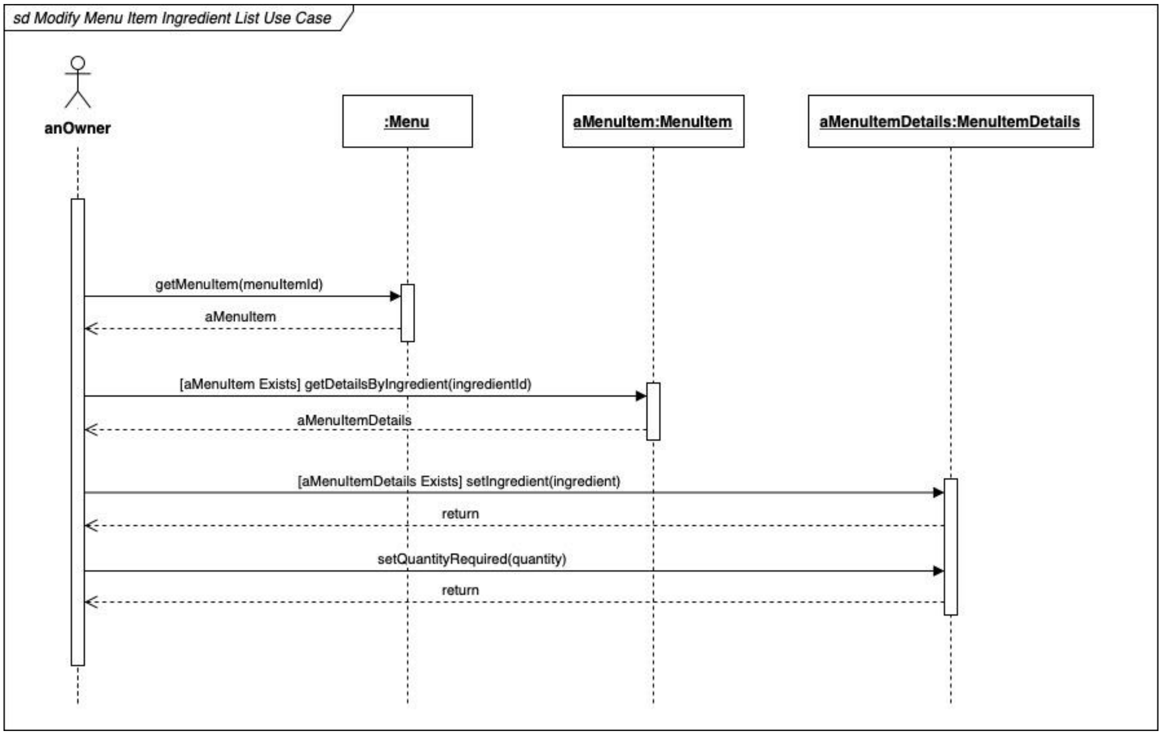
Use Case 4: **Adjust Threshold**


sd Adjust Threshold Use Case
anOwner
anIngredient:Ingredient
getIngredient(ingredientId)
anIngredient
[anIngredient Exists] setMinQuantity(minQuantity)
return

Use Case 5: **Update Inventory After Restock**

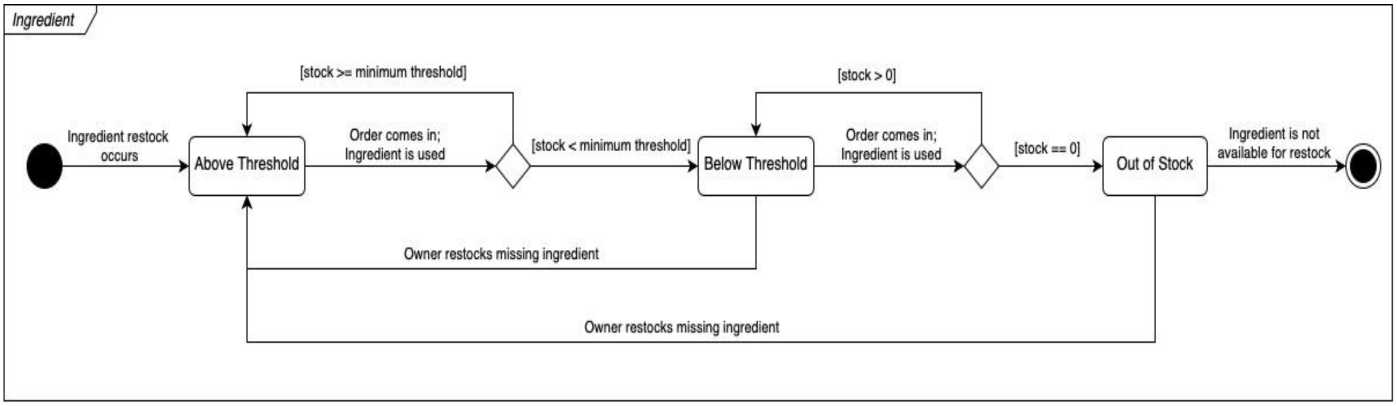

sd Update Ingredient After Restock Use Case
anOwner
anIngredient:Ingredient
getIngredient(ingredientId)
anIngredient
[anIngredient Exists] setQuantity(quantity)
return

Use Case 6: **Modify Menu Item Ingredient**

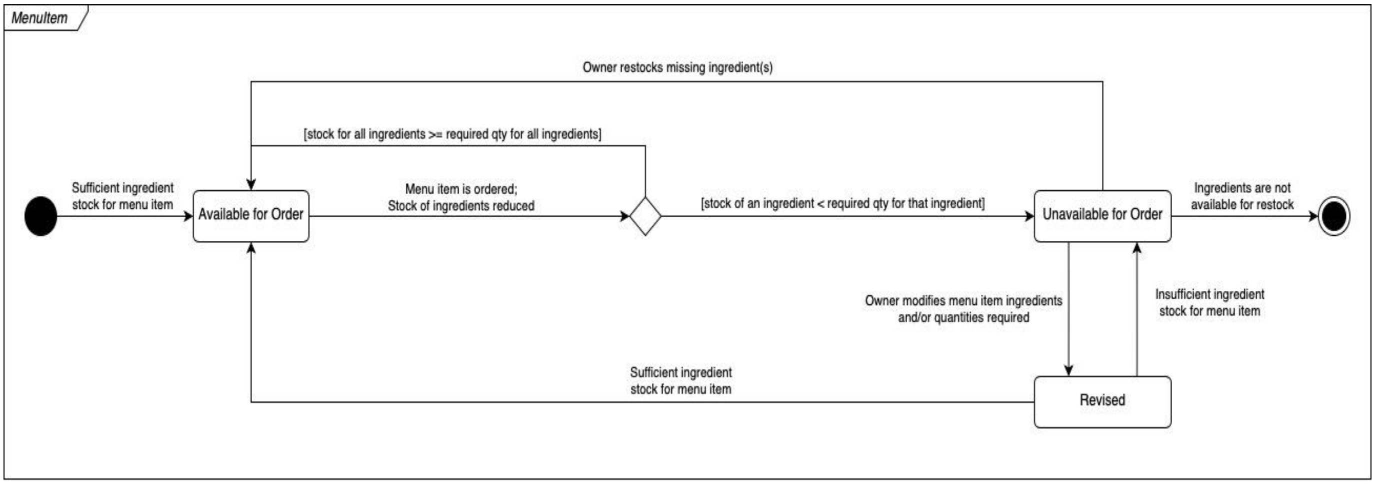

sd Modify Menu Item Ingredient List Use Case
anOwner
:Menu
aMenuItem:MenuItem
aMenuItemDetails:MenuItemDetails
getMenuItem(menuItemId)
aMenuItem
[aMenuItem Exists] getDetailsByIngredient(ingredientId)
aMenuItemDetails
[aMenuItemDetails Exists] setIngredient(ingredient)
return
setQuantityRequired(quantity)
return

Behavioral State Machine:

Object #1: **Ingredient**

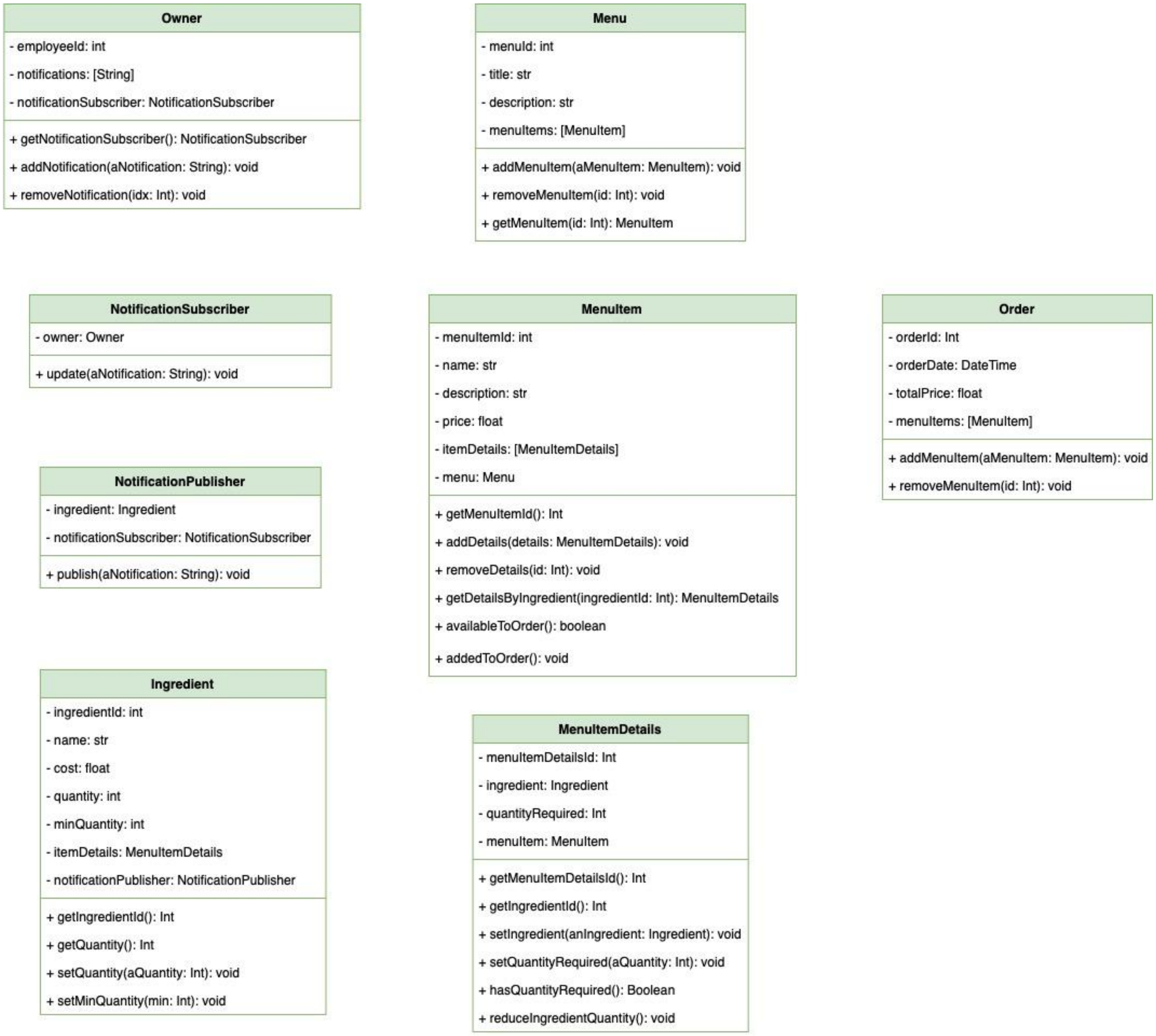

Ingredient
[stock >= minimum threshold]
[stock > 0]
Ingredient restock occurs
Above Threshold
Order comes in; Ingredient is used
[stock < minimum threshold]
Below Threshold
Order comes in; Ingredient is used
[stock == 0]
Out of Stock
Ingredient is not available for restock
Owner restocks missing ingredient
Owner restocks missing ingredient

Object #2: **MenuItem**


MenuItem
Owner restocks missing ingredient(s)
[stock for all ingredients >= required qty for all ingredients]
Sufficient ingredient stock for menu item
Available for Order
Menu item is ordered; Stock of ingredients reduced
[stock of an ingredient < required qty for that ingredient]
Unavailable for Order
Ingredients are not available for restock
Owner modifies menu item ingredients and/or quantities required
Insufficient ingredient stock for menu item
Sufficient ingredient stock for menu item
Revised

Class and Methods Design

Implementation Class Diagram


Owner
- employeeId: int
- notifications: [String]
- notificationSubscriber: NotificationSubscriber
+ getNotificationSubscriber(): NotificationSubscriber
+ addNotification(aNotification: String): void
+ removeNotification(idx: Int): void
Menu
- menuId: int
- title: str
- description: str
- menuItems: [MenuItem]
+ addMenuItem(aMenuItem: MenuItem): void
+ removeMenuItem(id: Int): void
+ getMenuItem(id: Int): MenuItem
NotificationSubscriber
- owner: Owner
+ update(aNotification: String): void
MenuItem
- menuItemId: int
- name: str
- description: str
- price: float
- itemDetails: [MenuItemDetails]
- menu: Menu
+ getMenuItemId(): Int
+ addDetails(details: MenuItemDetails): void
+ removeDetails(id: Int): void
+ getDetailsByIngredient(ingredientId: Int): MenuItemDetails
+ availableToOrder(): boolean
+ addedToOrder(): void
Order
- orderId: Int
- orderDate: DateTime
- totalPrice: float
- menuItems: [MenuItem]
+ addMenuItem(aMenuItem: MenuItem): void
+ removeMenuItem(id: Int): void
NotificationPublisher
- ingredient: Ingredient
- notificationSubscriber: NotificationSubscriber
+ publish(aNotification: String): void
Ingredient
- ingredientId: int
- name: str
- cost: float
- quantity: int
- minQuantity: int
- itemDetails: MenuItemDetails
- notificationPublisher: NotificationPublisher
+ getIngredientId(): Int
+ getQuantity(): Int
+ setQuantity(aQuantity: Int): void
+ setMinQuantity(min: Int): void
MenuItemDetails
- menuItemDetailsId: Int
- ingredient: Ingredient
- quantityRequired: Int
- menuItem: MenuItem
+ getMenuItemDetailsId(): Int
+ getIngredientId(): Int
+ setIngredient(anIngredient: Ingredient): void
+ setQuantityRequired(aQuantity: Int): void
+ hasQuantityRequired(): Boolean
+ reduceIngredientQuantity(): void

Method Contracts

Method Name: availableToOrder	**Class Name**: MenuItem	**ID:** 01
Clients (Consumers): Order		
Associated Use Cases: Receive Low Stock Notification, Input Order Information		
Description of Responsibilities: Implement the necessary behavior to determine whether a menu item is available to order based on ingredient quantities required.		
Arguments Received: Menu item identification		
Type of Value Returned: boolean		
Pre-Conditions: None		
Post-Conditions: None		

Method Name: reduceIngredientStock	**Class Name**: MenuItemDetails	**ID:** 02
Clients (Consumers): MenuItem		
Associated Use Cases: Receive Low Stock Notification, Input Order Information		
Description of Responsibilities: Implement the necessary behavior to reduce the stock of the ingredients used in the menu item after the menu item has been added to an Order.		
Arguments Received: none		
Type of Value Returned: void		
Pre-Conditions: menuItem.itemDetails.notEmpty() menuItem.availableToOrder() = true		
Post-Conditions: ingredient.quantity >= 0		

Method Name: setQuantity	**Class Name**: Ingredient	**ID:** 03
Clients (Consumers): MenuItemDetails		
Associated Use Cases: Receive Low Stock Notification		
Description of Responsibilities: Implement the necessary behavior to update the stock quantity of an ingredient and trigger a low stock event when applicable.		
Arguments Received: aQuantity:Int		
Type of Value Returned: void		
Pre-Conditions: aQuantity >= 0		
Post-Conditions: None		

Method Name: update	**Class Name**: NotificationSubscriber	**ID:** 04
Clients (Consumers): NotificationPublisher, NotificationPublisher.publish		
Associated Use Cases: Receive Low Stock Notification		
Description of Responsibilities: Implement the necessary behavior to update the existing owner with a new low stock notification.		
Arguments Received: void		
Type of Value Returned: aNotification:String		
Pre-Conditions: None		
Post-Conditions: owner.notifications = owner.notifications@pre.including(aNotification)		

Method Specifications

Method Name: availableToOrder	**Class Name:** MenuItem	**ID:** 01
Contract ID: N/A	**Programmer:** Cassandra Macri	**Date Due:** 19/03/2023

Programming Language: Python
Triggers/Events: An employee tries to add a menu item to the order.

Arguments Received: **Data Type:**	**Notes:**
-	-

Messages Sent & Arguments Passed: **ClassName.MethodName:**	**Argument Data Type:**	**Notes:**
MenuItemDetails.hasQuantityRequired()	-	-

Argument Returned: **Data Type:**	**Notes:**
-	-

Algorithm Specification:

```
FOR details in itemDetails:
    IF not details.hasQuantityRequired():
        return false
return true
```

Misc.Notes: There is a threshold within the system (i.e. “adjust threshold” use case)to identify if the quantity of an item is below or above, in order to determine when restock should occur.

Method Name: reduceIngredientStock	**Class Name:** MenuItemDetails	**ID:** 02
Contract ID: N/A	**Programmer:** Cassandra Macri	**Date Due:** 19/03/2023

Programming Language: Python
Triggers/Events: An employee adds a menu item to the order.

Arguments Received: **Data Type:**	**Notes:**
-	-

Messages Sent & Arguments Passed: **ClassName.MethodName:**	**Argument Data Type:**	**Notes:**
Ingredient.getQuantity()		
Ingredient.setQuantity(newQuantity)	Int	

Argument Returned: **Data Type:**	**Notes:**

-	-

Algorithm Specification:

```
prevQuantity = ingredient.getQuantity()
newQuantity = prevQuantity - quantityRequired
IF newQuantity > 0
    ingredient.setQuantity(newQuantity)
ELSE
    ingredient.setQuantity(0)
```

Misc.Notes:

Method Name: setQuantity	**Class Name:** Ingredient	**ID:** 03
Contract ID: N/A	**Programmer:** Cassandra Macri	**Date Due:** 19/03/2023

Programming Language: Python

Triggers/Events:

An employee adds a menu item to the order.
The owner modifies the low quantity threshold of an ingredient.
The owner restocks an ingredient.

Arguments Received: **Data Type:**	**Notes:**
Int	The new quantity (i.e. stock) for the ingredient.

Messages Sent & Arguments Passed: **ClassName.MethodName:**	**Argument Data Type:**	**Notes:**
NotificationPublisher.publish(notification)	String	

Argument Returned: **Data Type:**	**Notes:**
-	-

Algorithm Specification:

```
quantity = aQuantity
IF quantity < minQuantity:
    notification = “...”
    notificationPublisher.publish(message)
```

Misc.Notes: The “notification” variable above would be assigned something meaningful.
For example, it would specify the id, name, and quantity of the ingredient who is now below the minimum quantity threshold.

Method Name: update	**Class Name:** NotificationSubscriber	**ID:** 04
Contract ID: N/A	**Programmer:** Cassandra Macri	**Date Due:** 19/03/2023

Programming Language: Python

Triggers/Events:
An employee adds a menu item to the order.
The owner modifies the low quantity threshold of an ingredient.
The owner restocks an ingredient.

Arguments Received: **Data Type:**	**Notes:**
String	The notification to send to the owner.

Messages Sent & Arguments Passed: **ClassName.MethodName:**	**Argument** **Data Type:**	**Notes:**
Owner.addNotification(aNotification)	String	

Argument Returned: **Data Type:**	**Notes:**
-	-

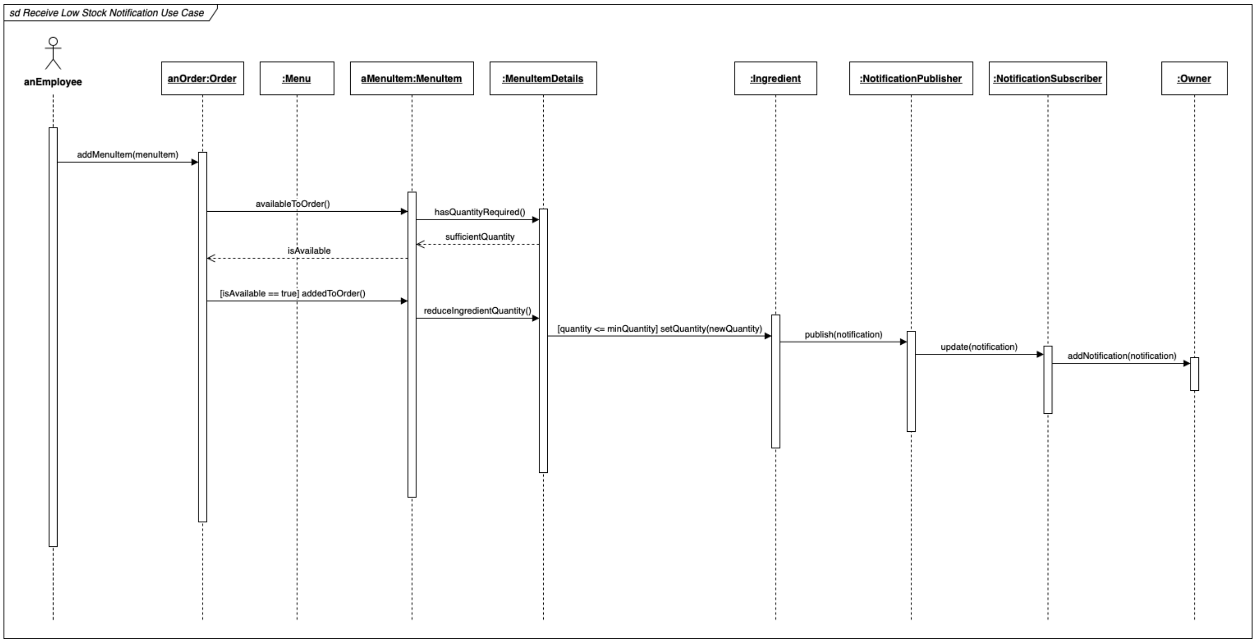
Algorithm Specification:

```
owner.addNotification(aNotification)
```

Misc.Notes:

Human-Computer Interaction Design

Use Case Scenario #1: Receive Low Stock Notification



=== User Story ===

As an *Owner*
I want to *be notified when an ingredient is low in stock*
So I can *prepare a purchase order and replenish the ingredient*

=== Interaction Points ===

:: Main screen ::

Input: Employee clicks on “Input Order” button from main screen
Output: New screen for inputting the order is displayed

:: Input Order screen ::

Input: Employee clicks on the “Add” button to add the menu item to the order
Outputs:

1. The “Add” button gets disabled when menu item is out of stock
2. A low stock notification is added to the notification list when an item’s stock has decreased past the low stock threshold.

Input: Employee clicks on the “Input Order” button
Output: Order is placed and system returns to the main screen

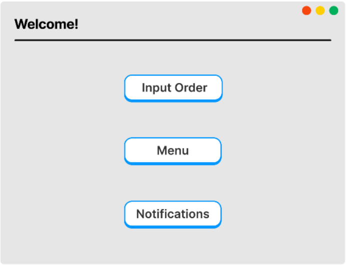
Input: Employee clicks on the “Cancel” button
Output: System returns to the main screen

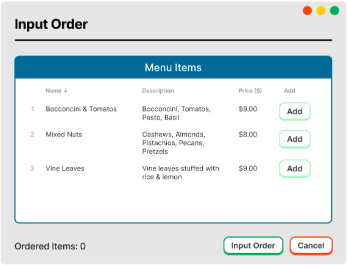
:: View Notifications screen ::

Input: Owner clicks on the “Dismiss” button
Output: The notification is removed from the table

Input: Owner clicks on the “Home” button
Output: System returns to the main screen

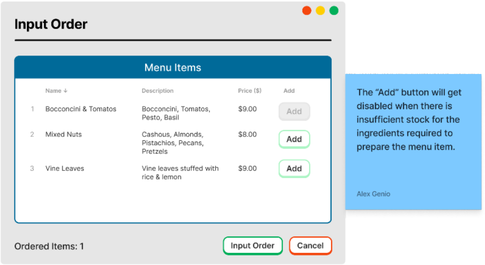
=== Wireframes ===





	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

Ordered Items: 0

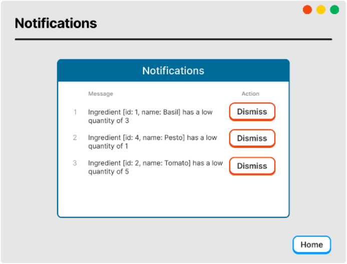


	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashous, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

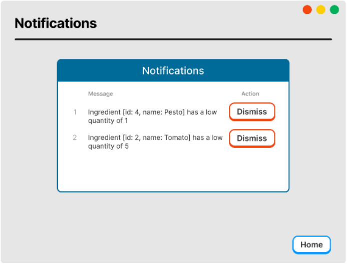
Ordered Items: 1

The “Add” button will get disabled when there is insufficient stock for the ingredients required to prepare the menu item.

Alex Genio

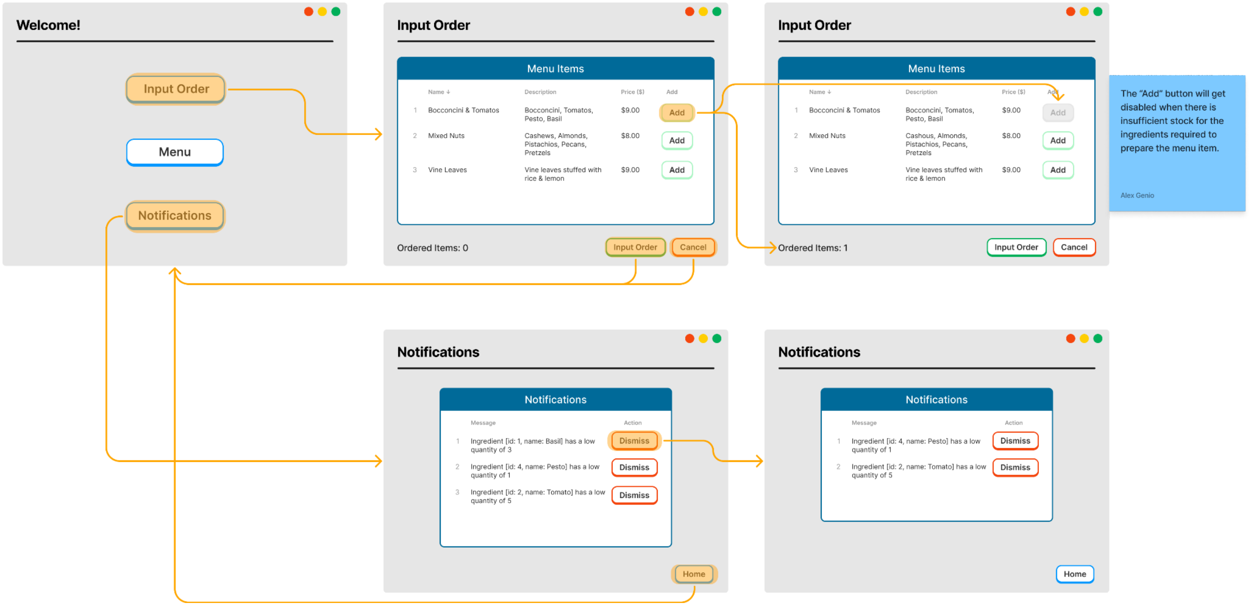


	Message	Action
1	Ingredient [id: 1, name: Basil] has a low quantity of 3	Dismiss
2	Ingredient [id: 4, name: Pesto] has a low quantity of 1	Dismiss
3	Ingredient [id: 2, name: Tomato] has a low quantity of 5	Dismiss



	Message	Action
1	Ingredient [id: 4, name: Pesto] has a low quantity of 1	Dismiss
2	Ingredient [id: 2, name: Tomato] has a low quantity of 5	Dismiss

=== Wireflow ===



	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

Ordered Items: 0

	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashous, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

Ordered Items: 1

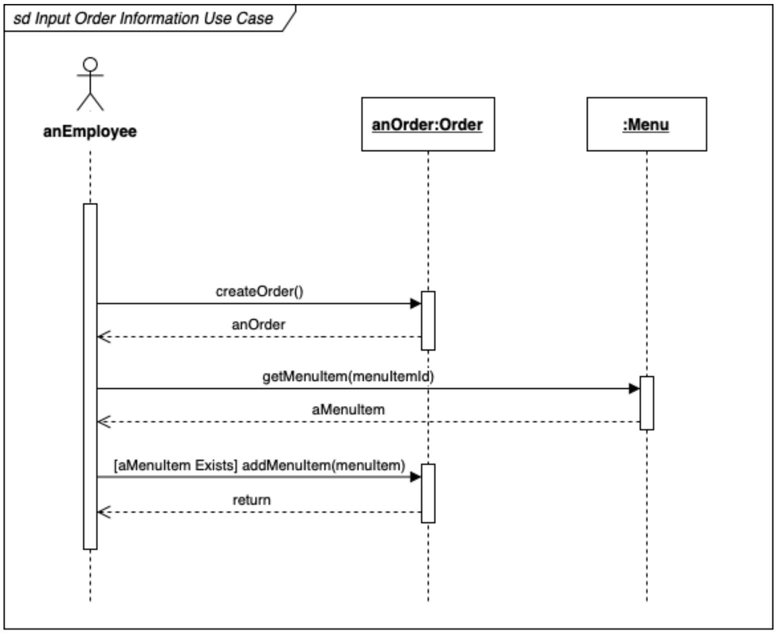
The “Add” button will get disabled when there is insufficient stock for the ingredients required to prepare the menu item.

Alex Genio

	Message	Action
1	Ingredient [id: 1, name: Basil] has a low quantity of 3	Dismiss
2	Ingredient [id: 4, name: Pesto] has a low quantity of 1	Dismiss
3	Ingredient [id: 2, name: Tomato] has a low quantity of 5	Dismiss

	Message	Action
1	Ingredient [id: 4, name: Pesto] has a low quantity of 1	Dismiss
2	Ingredient [id: 2, name: Tomato] has a low quantity of 5	Dismiss

Use Case Scenario #2: Input Order Information



=== User Story ===

As an *Employee*
I want to *create orders*
So I can *serve the customer the menu item(s) they desire*

=== Interaction Points ===

:: Main screen ::

Input: Employee clicks on “Input Order” button from main screen
Output: New screen for inputting the order is displayed

:: Input Order screen ::

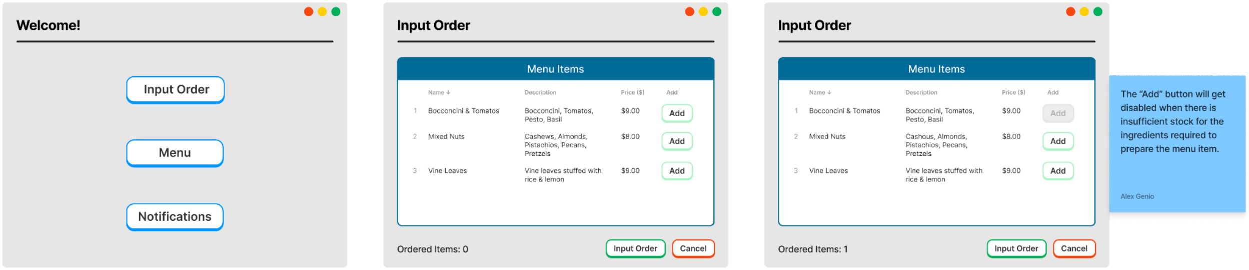
Input: Employee clicks on the “Add” button to add the menu item to the order
Output:

1. The “Add” button gets disabled when menu item is out of stock
2. Total items in order counter is incremented

Input: Employee clicks on the “Input Order” button
Output: Order is placed and system returns to the main screen

Input: Employee clicks on the “Cancel” button
Output: System returns to the main screen

=== Wireframes ===



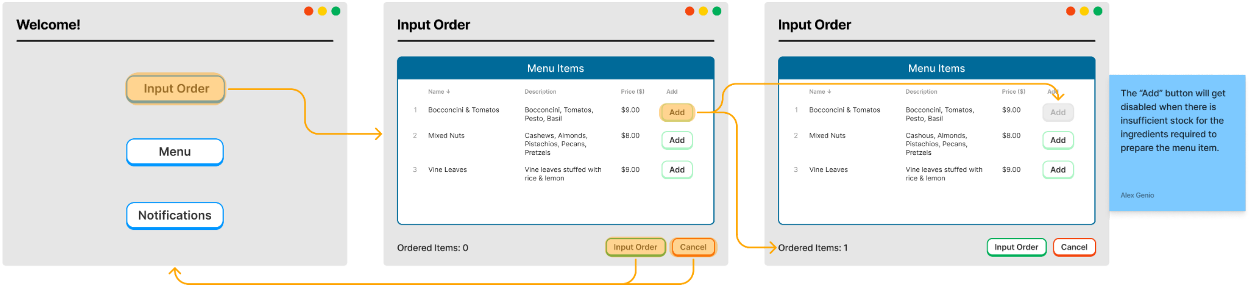
	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashous, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

The “Add” button will get disabled when there is insufficient stock for the ingredients required to prepare the menu item.

Alex Genio

=== Wireflow ===



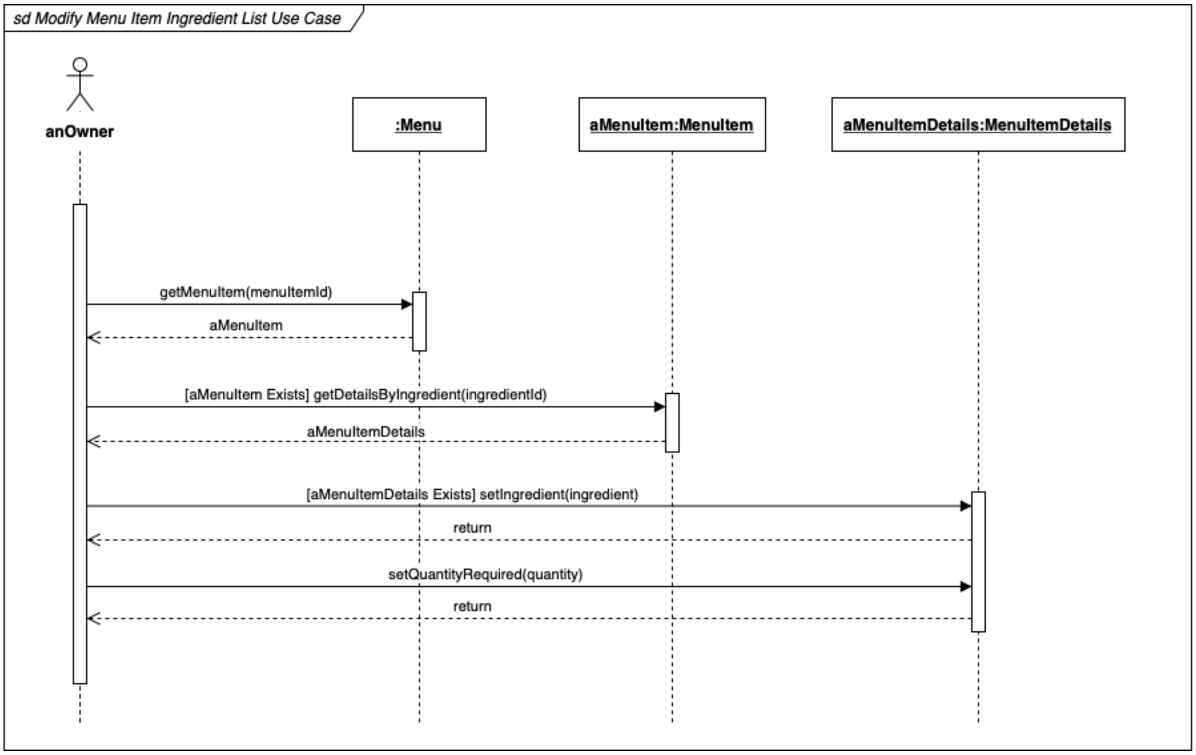
	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

	Name ↓	Description	Price ($)	Add
1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	$9.00	Add
2	Mixed Nuts	Cashous, Almonds, Pistachios, Pecans, Pretzels	$8.00	Add
3	Vine Leaves	Vine leaves stuffed with rice & lemon	$9.00	Add

The “Add” button will get disabled when there is insufficient stock for the ingredients required to prepare the menu item.

Alex Genio

Use Case Scenario #3: Modify Menu Item Ingredient List


sd Modify Menu Item Ingredient List Use Case
anOwner
:Menu
aMenuItem:MenuItem
aMenuItemDetails:MenuItemDetails
getMenuItem(menuItemId)
aMenuItem
[aMenuItem Exists] getDetailsByIngredient(ingredientId)
aMenuItemDetails
[aMenuItemDetails Exists] setIngredient(ingredient)
return
setQuantityRequired(quantity)
return

=== User Story ===

As an *Owner*
I want to *modify the items offered on the menu*
So I can *ensure all menu items can be served*

=== Interaction Points ===

:: Main screen ::

Input: Owner clicks on “Menu” button from the main screen
Output: New screen for viewing the menu’s items is displayed

:: Menu screen ::

Input: Owner clicks on “Details” button
Output: New screen for editing the selected menu item’s details is displayed

Input: Owner clicks on the “Home” button
Output: System returns to the main screen

:: Edit Menu Item Details screen ::

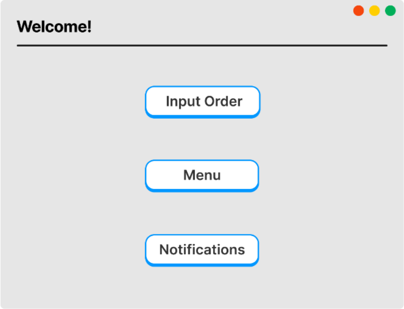
Input: Owner clicks on the “Ingredient” dropdown to select a new ingredient
Output: Selected “ingredient” value is updated on the screen

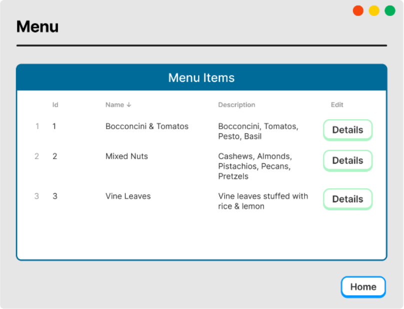
Input: Owner inputs a new quantity required for an ingredient
Output: Quantity required is updated on the screen

Input: Owner clicks on the “Home” button
Output: System returns to the main screen

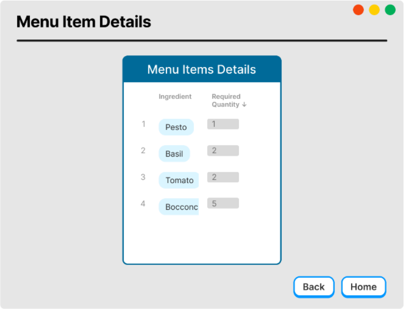
Input: Owner clicks on the “Back” button
Output: System returns to the “Menu” screen

=== Wireframes ===

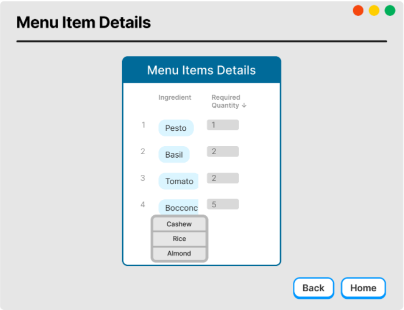




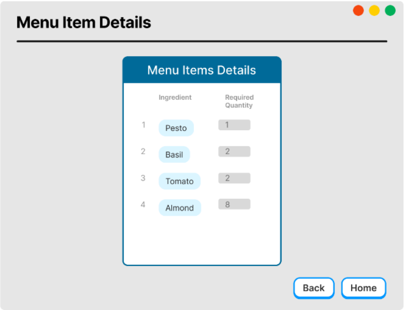
	Id	Name ↓	Description	Edit
1	1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	Details
2	2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	Details
3	3	Vine Leaves	Vine leaves stuffed with rice & lemon	Details



	Ingredient	Required Quantity ↓
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Bocconc	5

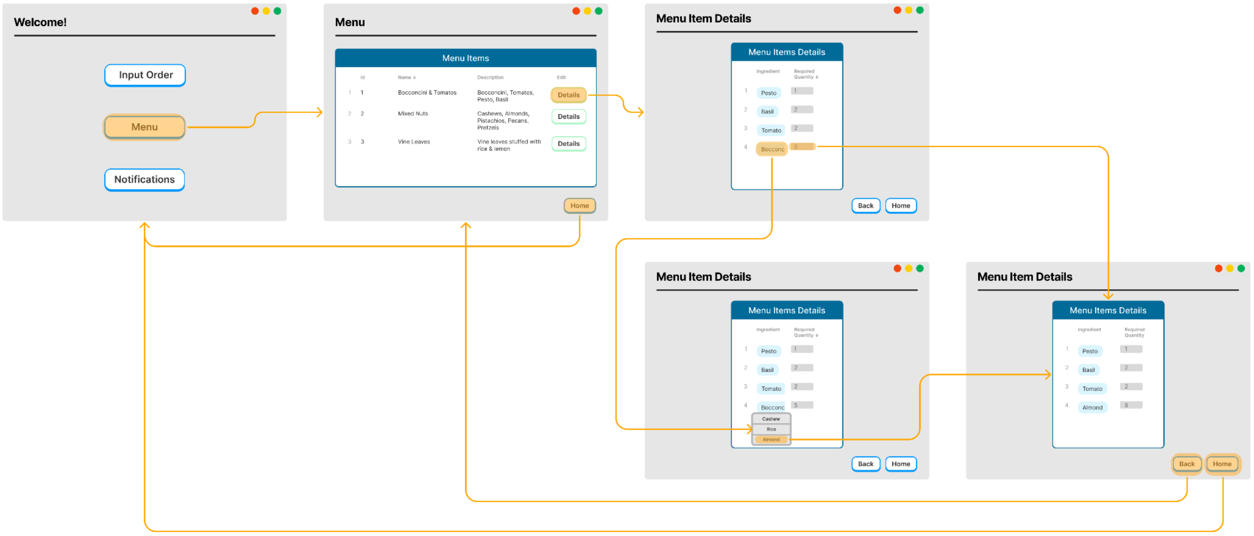


	Ingredient	Required Quantity ↓
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Bocconc	5



	Ingredient	Required Quantity
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Almond	8

=== Wireflow ===



Menu Items

	Id	Name ↓	Description	Edit
1	1	Bocconcini & Tomatos	Bocconcini, Tomatos, Pesto, Basil	Details
2	2	Mixed Nuts	Cashews, Almonds, Pistachios, Pecans, Pretzels	Details
3	3	Vine Leaves	Vine leaves stuffed with rice & lemon	Details

Menu Items Details

	Ingredient	Required Quantity ↓
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Bocconc	5

Menu Items Details

	Ingredient	Required Quantity ↓
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Bocconc	5

Menu Items Details

	Ingredient	Required Quantity
1	Pesto	1
2	Basil	2
3	Tomato	2
4	Almond	8

Deployment Diagram

Installation & Operation

- **Choices and justification regarding the technology stack selected:**

- **MySQL** is a popular open-source relational database management system (RDBMS) that is widely used in web development. The system is fast, scalable, and can handle large amounts of data. In the context of an inventory management application for a restaurant, MySQL can be used as the database to store and manage the business' inventory data. MySQL can also be easily integrated with other technologies such as ReactJS and NodeJS, which makes it an ideal choice for an inventory management application built on this technology stack. This integration can enable seamless communication between the application and the database, allowing for efficient management of data.

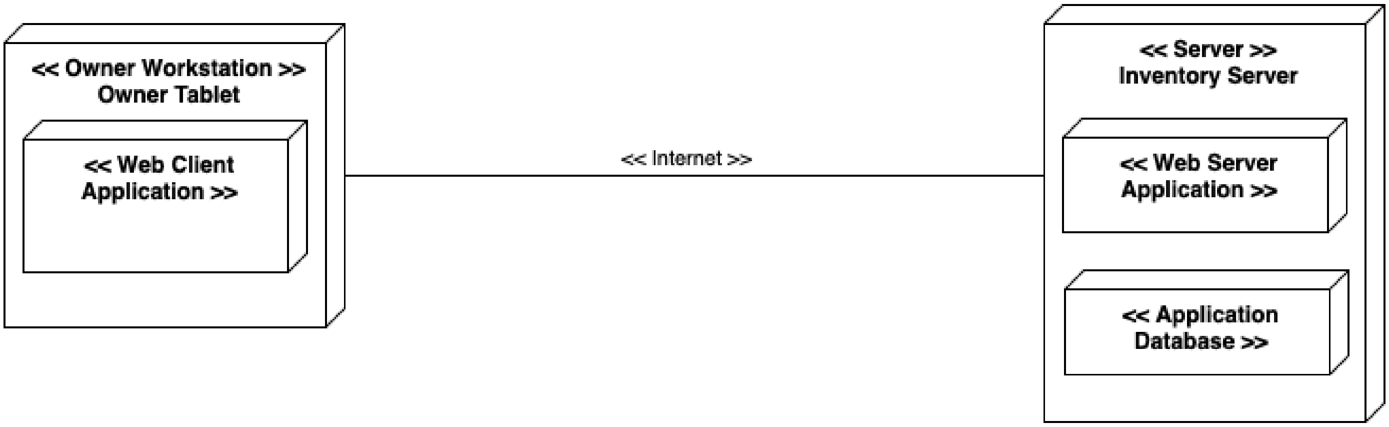
- **ReactJS** is a JavaScript library for building user interfaces. In the context of an inventory management system for a restaurant, ReactJS can be used to build the user interface for managing inventory items, inputting orders, and tracking inventory levels. This application can be used to create a component for displaying the list of inventory items, which can be reused throughout the application wherever the list of inventory items needs to be displayed. ReactJS can also be used to create a component for the order form, which can be used for placing orders for inventory items. It can include input fields for selecting the item, quantity, and other details. In addition, the software can display real-time information about the inventory levels of various items, including visualizations to help users understand the current inventory status. Moreover, ReactJS can be used to create a component for displaying notifications, in order to notify users about low inventory levels.

- **NodeJS** is an open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code outside of a web browser. In the context of an inventory management system for a restaurant, NodeJS can be used to build the server-side back-end of the application. It can handle requests from the front-end and interact with the database to fetch or update inventory data. NodeJS can also provide an API for other systems or applications to access inventory data. In addition, NodeJS can be used to handle real-time data processing, such as updating inventory levels in real-time, generating reports, or sending notifications. Furthermore, NodeJS can be used to interact with the MySQL database and retrieve or store inventory data. NodeJS can use the MySQL module to connect to the database and perform queries.

- **Visual Studio Code** is a source code editor that can be used in the development of an inventory management system. It provides debugging capabilities, allows for collaboration, and supports version control systems. Developers can use Visual Studio to write and debug code for the front-end (ReactJS), back-end (Node.js), and database (MySQL) components of the application.

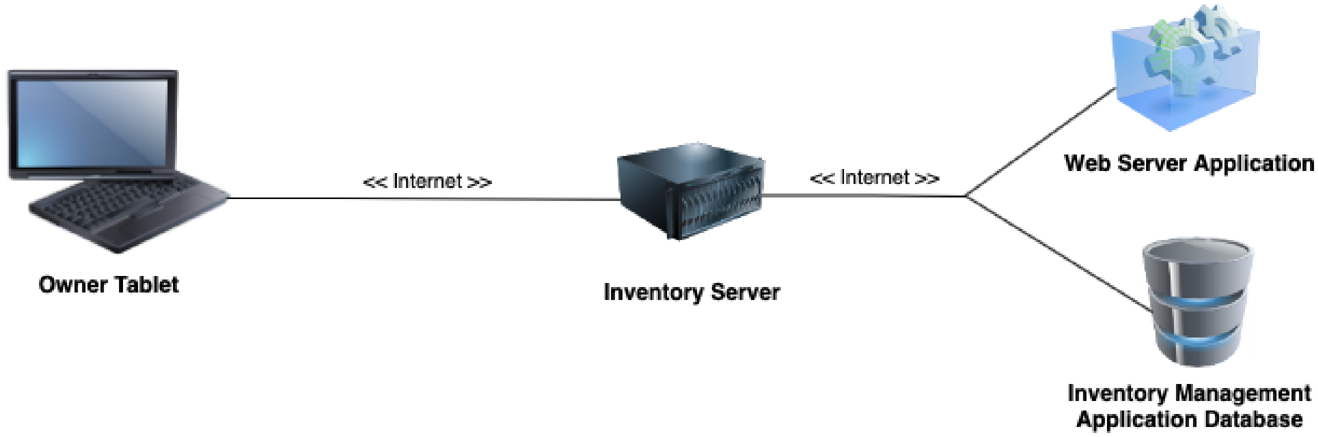
- **Deployment Diagram (e.g., Figure 11-9, particularly b & c):**

a. Deployment of the artifact to a node (e.g., Figure 11-9, b):



<< Owner Workstation >>
Owner Tablet
<< Web Client Application >>
<< Internet >>
<< Server >>
Inventory Server
<< Web Server Application >>
<< Application Database >>

b. Deployment of the artifact to a node with extended notation (e.g., Figure 11-9):



c. **Hardware and software specifications (e.g., Figure 11-12); if using serverless / cloud, then discuss the recommended platform:**

- ***Hardware Specification*: Owner's Tablet** (E.g. Lenovo Smart Tab M8 *)

  * *Specs:* MediaTek® Helio A22 Processor (4 Cores, 4x A53 @2.0 GHz);

  Android™ 9 Pie™ Operating System

  2 GB LPDDR3 (Soldered) Memory

  32 GB eMMC Storage

  8" HD (1280 x 800) IPS Display, touchscreen, 350 nits

- ***Software Specification:* Windows** & **Google Chrome**
- Windows provides a stable and reliable platform for running the application and its components, while Google Chrome provides a user-friendly interface and advanced features for accessing the front-end of the application.
- ***Cloud Specification:* Microsoft Azure** is a cloud computing platform that can be used to build and deploy an inventory management system. It offers virtual machines, database services, cloud storage, security features, and monitoring and analytics services. Azure can be used to manage the application's infrastructure, store and manage data, ensure security, and monitor performance.

● **The reasoning behind your deployment decisions:**

The deployment decisions were made based on the requirements and limitations of the hardware, software, and cloud infrastructure, as well as the needs of the business. The deployment decisions were made due to the following:

- Hardware: The choice of hardware was based on the owner's tablet (assumed), which has a MediaTek Helio A22 processor, 2GB RAM, and 32GB eMMC storage. The specifications of this tablet are sufficient for running a basic inventory management system. If the business's inventory management needs grow, it may be necessary to upgrade the hardware to a more powerful device or use a dedicated server.
- Operating System: The choice of operating system was based on the need for a reliable platform for running the inventory management system. Android 9 Pie is a mobile operating system that may not be optimized for business applications, so Windows was chosen as it provides a stable platform for running the application. Additionally, Windows provides compatibility with most business software, which may be required for the inventory management system to interface with other applications.
- Software: The software specification was chosen based on the need for a user-friendly interface and features for accessing the front-end of the application. Google Chrome was

chosen because it is a widely used web browser that is compatible with most devices and platforms, making it easier to access the application from anywhere.

- Cloud: The choice of cloud infrastructure was based on the need for reliability and security. Microsoft Azure was chosen because it provides virtual machines, database services, cloud storage, security features, and monitoring and analytics services that can be used to build and deploy an inventory management system.

- **Briefly discuss the following requirements:**

    - ***Operational requirements (Figure 11-14):***
        - **Technical Environment** (special hardware, software, and network requirements imposed by business requirements):
            - The system will work over the Web environment with Google Chrome.
            - The restaurant location will have an always-on network connection to enable real-time database updates.
            - The system should automatically back up at the end of each day on the Microsoft Azure cloud.

    - ***Performance requirements (Figure 11-15):***
        - **Speed** (time within which the system must perform its functions):
            - The inventory database must be updated in real time.
        - **Availability & Reliability** (extent to which the system will be available to the users and the permissible failure rate due to errors):
            - The system shall have 99% uptime performance.

    - ***Security requirements (Figure 11-16):***
        - **Access Control** (limitations on who can access the data):
            - The Owner is the only user able to manage inventory, modify menu items, and view notifications.
                - Access can be granted by the owner to other employees
            - The Employees can only input orders.

    - ***Cultural and Political requirements (Figure 11-17)***
        - As a local business, no cultural or political requirements are expected, and the business is not anticipated to expand to other markets.

User Testing & Training

A training and testing plan are developed to ensure the smooth implementation and adoption of the new system amongst the users of the Canal Lounge.

User Agreement Testing (UAT) is the final step before moving a software application to the production environment:

- Create test environment with production-like data setup
- Develop test scenarios for the users
- Ensure user testing is documented to track bugs and recommendations
- Each user should be involved in the UAT process, and complete their respective testing

A mandatory training must be completed by all the employees of the Canal Lounge, the owner included.

- Covers the new application's features and processes
- In the form of a video to provide visual aid, accompanied with UAT testing for practical training
- Video takes 30 minutes to complete
- The training period is included in employee worktime

System support/maintenance training for the owner of the Canal Lounge:

- Provide application specifications training to the owner
- Includes best practices for contingency planning to ensure business continuity

Prototype

Source Code

Main

```
from menu import Menu, MenuItem, MenuItemDetails
from ingredient import Ingredient
from owner import Owner
from order import Order


def useCaseReceiveLowStockNotification(menuItem):
    # simulate adding a menu item that has already been selected
    order = Order("2023-03-19")
    print("Adding {} to order...".format(menuItem))
    order.addMenuItem(menuItem)
    print(order)

    print("Adding {} to order...".format(menuItem))
    order.addMenuItem(menuItem)
    print(order)

    # receive low stock notification
    print("Adding {} to order...".format(menuItem))
    order.addMenuItem(menuItem)
    print(order)

def useCaseInputOrderInformation(menu, menuItemId):
    # simulate selecting and adding a menu item to the order
    order = Order("2023-04-10")
    menuItem = menu.getMenuItem(menuItemId)
    if menuItem is not None:
        print("Before adding {} to order --> {}".format(menuItem, order))
        order.addMenuItem(menuItem)
        print("After adding {} to order --> {}".format(menuItem, order))

def useCaseModifyMenuItemIngredientList(menu, menuItemId, ingredientId,
newIngredient, newQuantityRequired):
    menuItem = menu.getMenuItem(menuItemId)
    if menuItem is not None:
        details = menuItem.getDetailsByIngredient(ingredientId)
        if details is not None:
            print("Before MenuItem details modification --->
{}".format(details))
            print("Changing ingredient to {}".format(newIngredient))
            details.setIngredient(newIngredient)
            print("Changing quantity required to
{}".format(newQuantityRequired))
            details.setQuantityRequired(newQuantityRequired)
            print("After MenuItem details modification --->
{}".format(details))

def main():

    # create an owner
    owner1 = Owner()
    subscriber = owner1.getNotificationSubscriber()

    # build the inventory of ingredients
    boconccini = Ingredient("Boconccini", 5.99, 7, 3, subscriber)
    pesto = Ingredient("Pesto", 7.99, 6, 4, subscriber)
    tomato = Ingredient("Tomato", 4.99, 11, 5, subscriber)
    basil = Ingredient("Basil", 2.99, 8, 6, subscriber)

    cashew = Ingredient("Cashew", 1.99, 100, 20, subscriber)
    almond = Ingredient("Almond", 2.99, 100, 30, subscriber)
    pistachio = Ingredient("Pistachio", 4.99, 100, 25, subscriber)
    pecan = Ingredient("Pecan", 1.99, 100, 20, subscriber)
    pretzel = Ingredient("Pretzel", 0.99, 100, 15, subscriber)

    vineLeaf = Ingredient("Vine Leaf", 0.20, 75, 10, subscriber)
    rice = Ingredient("White Rice", 10.99, 200, 20, subscriber)
    lemon = Ingredient("Lemon", 3.99, 30, 5, subscriber)

    # build the menu
    menu = Menu("Canal Lounge Menu", "Lunch Menu")

    bocAndTomato = MenuItem("Boconccini & Tomatos", "Boconccini, Tomato, Pesto,
Basil", 8.99, menu)
    bocAndTomato.addDetails(MenuItemDetails(boconccini, 2, bocAndTomato))
    bocAndTomato.addDetails(MenuItemDetails(pesto, 1, bocAndTomato))
    bocAndTomato.addDetails(MenuItemDetails(tomato, 2, bocAndTomato))
    bocAndTomato.addDetails(MenuItemDetails(basil, 1, bocAndTomato))

    mixedNuts = MenuItem("Mixed Nuts", "Cashews, Almonds, Pistachios, Pecans,
Pretzels", 7.99, menu)
    mixedNuts.addDetails(MenuItemDetails(cashew, 10, mixedNuts))
    mixedNuts.addDetails(MenuItemDetails(almond, 10, mixedNuts))
    mixedNuts.addDetails(MenuItemDetails(pistachio, 10, mixedNuts))
    mixedNuts.addDetails(MenuItemDetails(pecan, 10, mixedNuts))
    mixedNuts.addDetails(MenuItemDetails(pretzel, 10, mixedNuts))

    vineLeaves = MenuItem("Vine Leaves", "Vine leaves stuffed with rice and
lemon", 8.99, menu)
    vineLeaves.addDetails(MenuItemDetails(vineLeaf, 5, vineLeaves))
    vineLeaves.addDetails(MenuItemDetails(rice, 1, vineLeaves))
    vineLeaves.addDetails(MenuItemDetails(lemon, 1, vineLeaves))

    menu.addMenuItem(bocAndTomato)
    menu.addMenuItem(mixedNuts)
    menu.addMenuItem(vineLeaves)

    # Use Case #1: Receive Low Stock Notification
    print("")
    print("*** Use Case #1: Receive Low Stock Notification ***")
    useCaseReceiveLowStockNotification(bocAndTomato)
    print("")

    # Use Case #2: Input Order Information
    print("*** Use Case #2: Input Order Information ***")
    useCaseInputOrderInformation(menu, mixedNuts.getMenuItemId())
    print("")
    # Use Case #3: Modify Menu Item Ingredient List
    print("*** Use Case #3: Modify Menu Item Ingredient List ***")
    useCaseModifyMenuItemIngredientList(menu, bocAndTomato.getMenuItemId(),
boconccini.getIngredientId(), almond, 10)
    print("")


if __name__ == '__main__':
    main()
```

Ingredient

```
from notification import NotificationPublisher


class Ingredient:

    nextId = 1

    def __init__(self, name, cost, quantity, minQuantity,
notificationSubscriber):
        self.__ingredientId = Ingredient.nextId
        Ingredient.nextId += 1
        self.__name = name
        self.__cost = cost
        self.__quantity = quantity
        self.__minQuantity = minQuantity
        self.__notificationPublisher = NotificationPublisher(self,
notificationSubscriber)
        self.__menuItemDetails = None

    def getIngredientId(self):
        return self.__ingredientId

    def getQuantity(self):
        return self.__quantity

    def setQuantity(self, aQuantity):
        self.__quantity = aQuantity
        self.__maybePublishNotification()

    def setMinQuantity(self, aQuantity):
        self.__minQuantity = aQuantity
        self.__maybePublishNotification()

    def __maybePublishNotification(self):
        if (self.__quantity < self.__minQuantity):
            notification = "Notification --- {} has a low quantity of
{}".format(self.__str__(), self.__quantity)
            self.__notificationPublisher.publish(notification)

    def __str__(self):
        return "Ingredient [id: {}, name: {}]".format(self.__ingredientId,
self.__name, )
```

```
    def __eq__(self, other):
        return self.__ingredientId == other.__ingredientId
```

Menu

```
class Menu:

    nextId = 1

    def __init__(self, title, description):
        self.__menuId = Menu.nextId
        Menu.nextId += 1
        self.__title = title
        self.__description = description

        # set of MenuItem
        self.__menuItems = set()

    def addMenuItem(self, aMenuItem):
        self.__menuItems.add(aMenuItem)

    def removeMenuItem(self, menuItemId):
        menuItem = self.getMenuItem(menuItemId)
        if menuItem is not None:
            self.__menuItems.remove(menuItem)

    def getMenuItem(self, menuItemId):
        for menuItem in self.__menuItems:
            if menuItem.getMenuItemId() == menuItemId:
                return menuItem


class MenuItem:

    nextId = 1

    def __init__(self, name, description, price, menu):
        self.__menuItemId = MenuItem.nextId
        MenuItem.nextId += 1
        self.__name = name
        self.__description = description
        self.__price = price
        self.__itemDetails = set()
        self.__menu = menu

    def getMenuItemId(self):
        return self.__menuItemId

    def addDetails(self, details):
        self.__itemDetails.add(details)

    def removeDetails(self, detailsId):
        for details in self.__itemDetails:
            if details.getMenuItemDetailsId() == detailsId:
                self.__itemDetails.remove(details)

    def getDetailsByIngredient(self, ingredientId):
        for details in self.__itemDetails:
            if details.getIngredientId() == ingredientId:
                return details

    def availableToOrder(self):
        for details in self.__itemDetails:
            if not details.hasQuantityRequired():
                return False
        return True

    def addedToOrder(self):
        for details in self.__itemDetails:
            details.reduceIngredientQuantity()

    def __str__(self):
        return "MenuItem [id:{}, name: {}, description: {}]".format(
            self.__menuItemId, self.__name, self.__description)


class MenuItemDetails:

    nextId = 1

    def __init__(self, anIngredient, aQuantityRequired, aMenuItem):
        self.__menuItemDetailsId = MenuItemDetails.nextId
        MenuItemDetails.nextId += 1
        self.__ingredient = anIngredient
        self.__quantityRequired = aQuantityRequired
        self.__menuItem = aMenuItem

    def getMenuItemDetailsId(self):
        return self.__menuItemDetailsId

    def getIngredientId(self):
        return self.__ingredient.getIngredientId()

    def setIngredient(self, anIngredient):
        self.__ingredient = anIngredient

    def setQuantityRequired(self, aQuantity):
        self.__quantityRequired = aQuantity

    def hasQuantityRequired(self):
        return self.__ingredient.getQuantity() >= self.__quantityRequired

    def reduceIngredientQuantity(self):
        prevQuantity = self.__ingredient.getQuantity()
        newQuantity = prevQuantity - self.__quantityRequired
        if newQuantity > 0:
            self.__ingredient.setQuantity(newQuantity)
        else:
            self.__ingredient.setQuantity(0)
    def __str__(self):
        return "{}, quantityRequired: {}".format(self.__ingredient, 
self.__quantityRequired)
```

Notifications

```
class NotificationSubscriber:

    def __init__(self, owner):
        self.__owner = owner

    def update(self, aNotification):
        self.__owner.addNotification(aNotification)


class NotificationPublisher:

    def __init__(self, ingredient, notificationSubscriber):
        self.__ingredient = ingredient
        self.__notificationSubscriber = notificationSubscriber

    def publish(self, aNotification):
        if self.__notificationSubscriber is not None:
            self.__notificationSubscriber.update(aNotification)
```

Order

```
class Order:

    nextId = 1

    def __init__(self, orderDate):
        self.__orderId = Order.nextId
        Order.nextId += 1
        self.__orderDate = orderDate
        self.__totalPrice = 0.0
        self.__menuItems = []

    def addMenuItem(self, aMenuItem):
        if aMenuItem.availableToOrder():
            self.__menuItems.append(aMenuItem)
            aMenuItem.addedToOrder()

    def removeMenuItem(self, menuItemId):
        for menuItem in self.__menuItems:
            if menuItem.getMenuItemId() == menuItemId:
                self.__menuItems.remove(menuItem)

    def __str__(self):
        return "Ordered Items: {}".format(len(self.__menuItems))
```

Owner

```
from notification import NotificationSubscriber


class Owner:

    nextId = 1

    def __init__(self):
        self.__employeeId = Owner.nextId
        Owner.nextId += 1
        self.__notifications = set()
        self.__notificationSubscriber = NotificationSubscriber(self)

    def getNotificationSubscriber(self):
        return self.__notificationSubscriber

    def addNotification(self, aNotification):
        self.__notifications.add(aNotification)
        print(aNotification)

    def removeNotification(self, idx):
        self.__notifications.pop(idx)
```

Output:

```
*** Use Case #1: Receive Low Stock Notification ***
Adding MenuItem [id:1, name: Boconccini & Tomatos, description: Boconccini,
Tomato, Pesto, Basil] to order...
Ordered Items: 1
Adding MenuItem [id:1, name: Boconccini & Tomatos, description: Boconccini,
Tomato, Pesto, Basil] to order...
Ordered Items: 2
Adding MenuItem [id:1, name: Boconccini & Tomatos, description: Boconccini,
Tomato, Pesto, Basil] to order...
Notification --- Ingredient [id: 4, name: Basil] has a low quantity of 5
Notification --- Ingredient [id: 1, name: Boconccini] has a low quantity of 1
Notification --- Ingredient [id: 2, name: Pesto] has a low quantity of 3
Ordered Items: 3

*** Use Case #2: Input Order Information ***
Before adding MenuItem [id:2, name: Mixed Nuts, description: Cashews, Almonds,
Pistachios, Pecans, Pretzels] to order --> Ordered Items: 0
After adding MenuItem [id:2, name: Mixed Nuts, description: Cashews, Almonds,
Pistachios, Pecans, Pretzels] to order --> Ordered Items: 1

*** Use Case #3: Modify Menu Item Ingredient List ***
Before MenuItem details modification ---> Ingredient [id: 1, name: Boconccini],
quantityRequired: 2
```

```
Changing ingredient to Ingredient [id: 6, name: Almond]
Changing quantity required to 10
After MenuItem details modification ---> Ingredient [id: 6, name: Almond],
quantityRequired: 10
```

Construction

Tools Used: Sublime 3 Text Editor

Programming Environment: Python 3 (programming language) & MacOS Terminal

Frameworks: N/A

Libraries: N/A

Patterns: N/A

Appendices

Meeting agendas and minutes

Milestone: Team Project Proposal

January 21, 2023 at 5:00 pm - 5:30 pm / 30 minute meeting to work and submit milestone

Milestone: Functional Modeling

Wednesday January 24, 2023 at 7:00 pm-10:00 pm / 3 hour meeting discussing and work distribution

Saturday January 28, 2023 at 5:00 pm - 5:30 pm / 30 minute meeting to finalize and submit

Milestone: Structural Modeling

February 7, 2023 at 6:00 pm - 8:30 pm / 2 hours and 30 minute meeting to work on the milestone and work distribution

February 11, 2023 at 7:00 - 7:15 pm / 15 minute meeting to finalize and submit

Milestone: Behavioral Modeling

February 19, 2023 at 4:00 pm - 6:00 pm / 2 hour meeting to work on milestone and distribute the work

February 25, 2023 at 6:00 pm - 7:00 pm / 1 hour meeting to finalize the milestone

Milestone: Class and Methods design

March 13, 2023 at 5:30 pm - 8:30 pm / 3 hour meeting to work on milestone

March 18, 2023, at 3:00 pm - 4:00 pm / 1 hour meeting to work and finalize the milestone

Milestone: HCI Design and Deployment Diagram

Monday March 20, 2023 at 6:00 pm - 8:00 pm /2 hour meeting discussing HCI milestone. Deployment Diagram and work distribution.

Presentation and Final Report

Monday April 12 , 2023 at 5:22 - 5:57: / 35 minute meeting discussing presentation and work distribution.

Saturday April 15, 2023 at 4:00 pm - 6:30 pm // 2 hour 30 minute meeting to work and finalize reports + presentations

Project management documents

Google Drive

Software used to support your work (project management, modeling, programming, implementation)

Project management

Google Docs

Google Slides

Modeling

Google Drawio

Figma - HCI Design/WireFrame and WireFlow

Programming

Sublime 3 text editor