Software Design Description

for

Food Buddy

Version 2.0 approved

Prepared by

Vishwareena Vanoo 194697

Wan Nur Atiqah Binti Wan Ahmad Yusri 193909

Muhammad Harith Bin Zainudin 192171

Mohd Shakir Asyraf Bin Shazali 193908

Muhammad Haris Bin Harun 194701

3/3/2020

1.0 INTRODUCTION

This section provides the introduction to the Software Design Documentation(SDD) of the system. It explain the purposes, scope, definition, acronyms, abbreviations and the references that have been used in this SDD

1.1 Purposes

FoodBuddy is focusing on providing food services specifically for students. This project is related to the UPM students' lifestyle. Nowadays, students are facing problems in finding and selling their food due to higher delivery fee and no platform to sell their foods. That's the reason for us to choose students as our users. Therefore, we think that through this FoodBuddy project, we can help the students to manage their problems. We believe that this project can help them to buy food with low delivery fees and also can sell their foods to the students.

1.2 Scope

The FoodBuddy system focuses on finding the foods with low delivery fees and also sells their foods to the other students. This system will help them more by decreasing their expenses on food.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
SDD	Software Design Documentation
UPM	Universiti Putra Malaysia

1.4 References

The user of this SDD may need the following documents for reference: IEEE Standard 1016-1998, IEEE Recommended Practice for Software Requirements Specifications, IEEE Computer Society, 1998.

The user also needs to refer to Software Requirement Specifications(SRS) documentation of FoodBuddy System for understanding this Software Design Documentation(SDD).

2.0 SYSTEM OVERVIEW

Food Buddy is an application that is created for students in university to buy and sell foods. If the user is a seller, they can add their menu and manage price, description and availability of the menu. Finally, they must wait for the notification from the customer. However, if the user logs in as a buyer, this application can view and choose the menu they want, add to the cart and may proceed to the payment. Buyer can choose either they want to pay online or cash on delivery (COD). Students also can choose either they want to pick up food or they want the seller to send the food to them. Food buddy can be used for mobile applications and it is an android user.

3.0 DESIGN CONSIDERATION

3.1 Assumptions and Dependencies

AS1 - We assume that the user can’t find any retailer that provides low delivery fees.
AS2 - We assume that the user doesn't know where they can sell their foods for the other students.
D1 - The system is depend on the changes made by the user in their software to a latest version.

3.2 General Constraints

C1 - The constraint that we have found is the memory requirement in the hardware. The system needs more than one server as a requirement for its better performance on view the product and also make a transaction.
C2 - Another constraint is network traffic where it requires smooth internet connection in order for the transaction to be done correctly without lack.

3.3 Goals And Guidelines

G1 - Provides full help towards Universiti Putra Malaysia(UPM) students in finding and selling their foods.
G2 - Create a full functioned system

3.4 Development Methods

The system has been created on waterfall model methodology. It is a classical model used in system development life cycle to create a system with a linear and sequential approach. It contains certain phases such as Requirement Gathering , Analysis, System Design, Implementation Development , Testing, Integrating and Maintenance Fixing. The reason of using the methodology is because of its easy management for small project sand clear defined stages.

4.0 DETAILED DESIGN

This section shows the detailed design feature specifications of smaller pieces of the design.

4.1 User Class Diagram

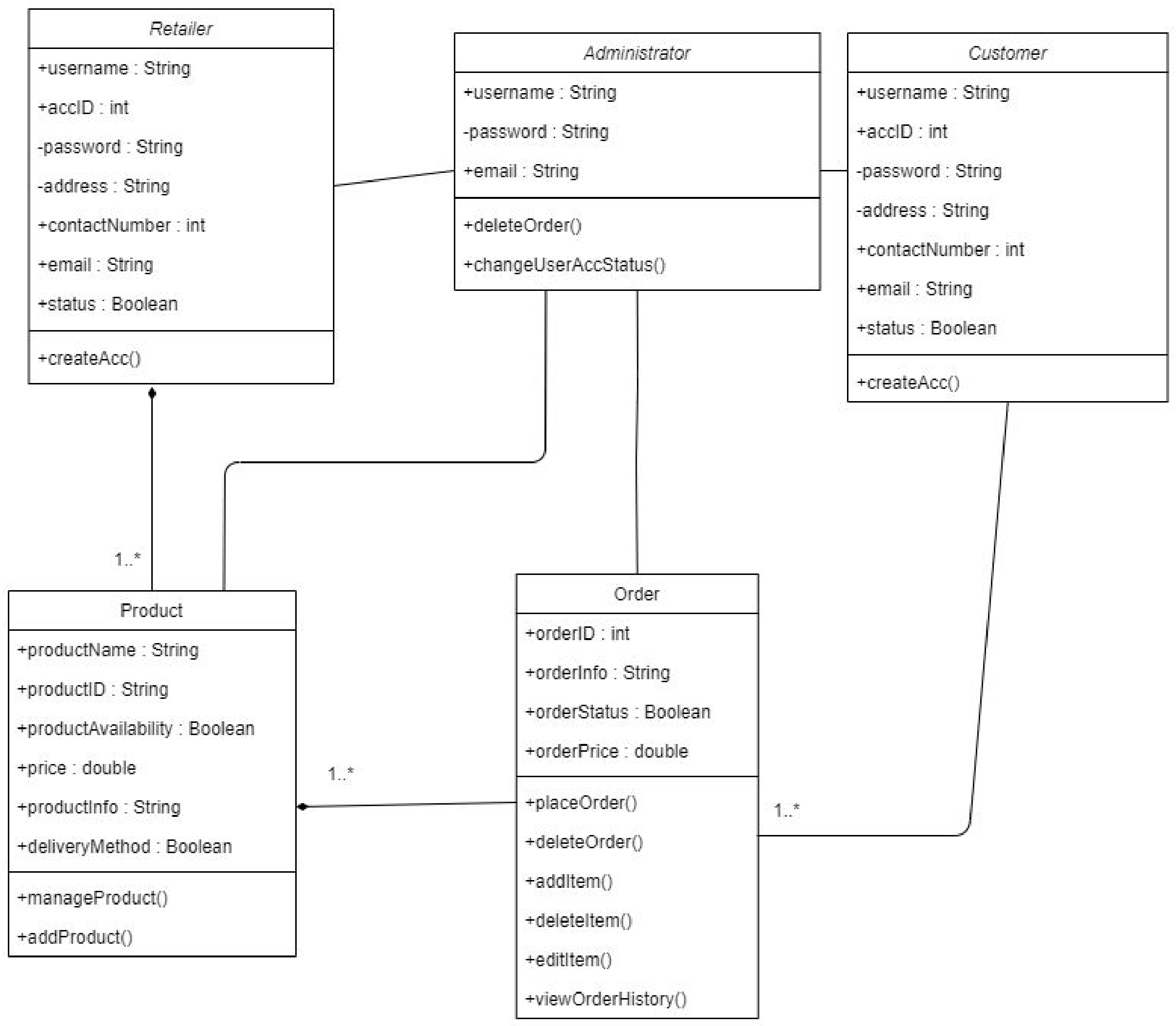


Diagram 1 : The diagram shows the complete User class diagram of the FoodBuddy System

4.1.1 Retailer

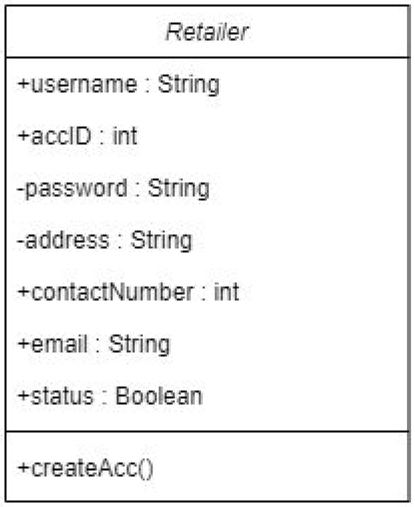


Diagram 2 : The diagram shows the retailer class diagram that consists of attribute and operation

a. Classification

The class component consists of attributes which are username type String, accID type integer, password type String, address type String, contactNumber type integer, email type String and status type boolean.

b. Definition

The retailer class does contain methods or functions which are createAcc() only. It also shows the behavior of each attribute of the class which is declared as public and private accordingly by referring to diagram 2.

c. Responsibilities

There are 7 attributes in retailer class. The first attribute is username which is responsible to get the username from the user. The second is accID which is a unique number that will generate automatically for the user that registers into the system. Third one is password which is to get the password from the user. Next is the address which is responsible to get the address from the user. The contactNumber is responsible to get the user contact number. The email will be responsible for getting the email from the user. Lastly, the status have a responsible to show the status of the retailer either it is active or not

4.1.2 Administrator

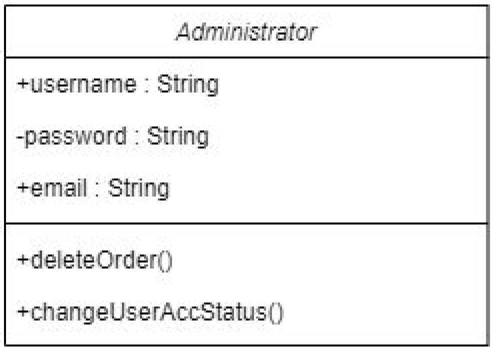


Diagram 3 : The diagram shows the administrator class diagram that consists of attribute and operation

a. Classification

The class component consists of attributes which are username type String, password type String and email type String.

b. Definition

The administrator does contain any methods or functions which are deleteOrder() and changeUserAccStatus. It also shows the behavior of each attribute of the class which is declared as public and private.

c. Responsibilities

There are 3 attributes in administrator class. The first attribute username which will store the username for the administrator. The second one is the password which is responsible to get the password from the administrator. The last one is email that is responsible for getting the email from the administrator.

4.1.3 Customer

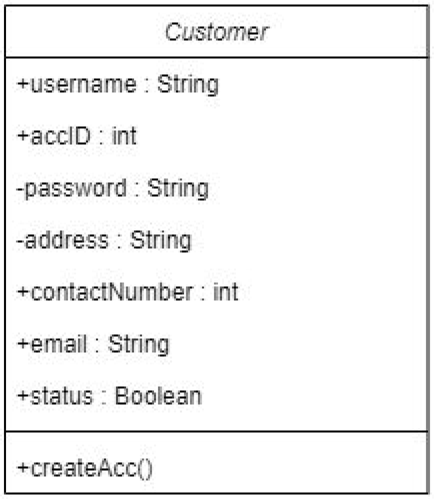


Diagram 4 : The diagram shows the customer class diagram that consists of attribute and operation

a. Classification

The class component consists of attributes which are username type String, accID type integer, password type String, address type String, contactNumber type integer, email type String and status type boolean.

b. Definition

The administrator does contain any methods or functions which is createAcc(). It also shows the behavior of each attribute of the class which is declared as public and private.

c. Responsibilities

This customer class is basically the same as the retailer class. There are 7 attributes in customer class. The first attribute is username which is responsible to get the username from the user. The second is accID which is a unique number that will generate automatically for the user that registers into the system. Third one is password which is to get the password from the user. Next is the address which is responsible to get the address from the user. The contactNumber is responsible to get the user contact number. The email will be responsible for getting the email from the user. Lastly, the status have a responsible to show the status of the customer either it is active or not

4.1.4 Order

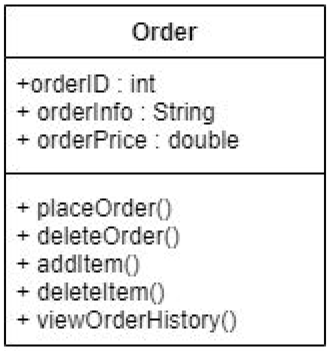


Diagram 5 : The diagram shows the order class diagram that consists of attribute and operation

a. **Classification**

The class component consists of attributes which are orderID type int, orderInfo type String and orderPrice type double.

b. **Definition**

The order class defines the order object which stores all of necessary order information. The 3 attributes do not contain any specific attribute as it will be accessed by the Order class. It also shows that every attribute is labelled as public.

c. **Responsibilities**

It has 5 method which placeOrder(), to insert order, deleteOrder() to delete order, addItem() to add item to the order, deleteItem() to delete an item in the order and viewOrderHistory(), to grab all order that has been made by specific user.

4.1.5 Product

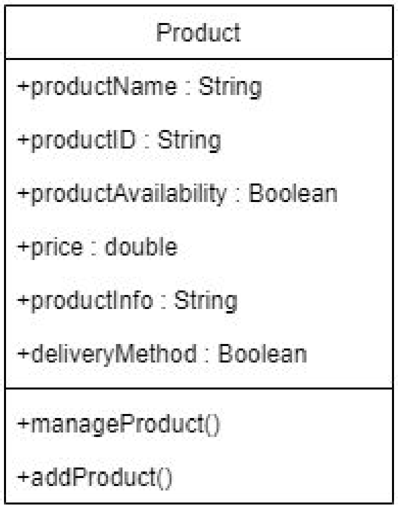


Diagram 6 : The diagram shows the product class diagram that consists of attribute and operation

a. Classification

The class component consists of attributes which are productName type String, productID type String, productAvailability type Boolean, price type double, productInfo type String, deliveryMethod type boolean.

b. Definition

The product class defines the product object which stores all of necessary order information. The 6 attributes do not contain any specific attribute as it will be accessed by the order, customer and retailer class. It also shows that every attribute is labelled as public.

c. Responsibilities

It has 2 methods which manageProduct() that the retailer can manage either to delete or change the necessary information about their own product and addProduct() that the retailer can add their new product.

5.0 DATABASE DESIGN

This section shows all the database design that will be involved and created in the FoodBuddy system.

5.1 Data Model

RETAILER			
username	VARCHAR2	PRIMARY KEY	NOT NULL
accID	INTEGER	UNIQUE	NOT NULL
password	VARCHAR2		NOT NULL
address	VARCHAR2		
contactNumber	INTEGER		NOT NULL
email	VARCHAR2		NOT NULL
status	BOOLEAN		NOT NULL

Table 1.1 : Table above show the data model for the retailer

CUSTOMER			
username	VARCHAR2	PRIMARY KEY	NOT NULL
accID	INTEGER	UNIQUE	NOT NULL
password	VARCHAR2		NOT NULL
address	VARCHAR2		
contactNumber	INTEGER		NOT NULL
email	VARCHAR2		NOT NULL
status	BOOLEAN		NOT NULL

Table 1.2 : Table above show the data model for the customer

ADMINISTRATOR			
<u>username</u>	VARCHAR2	PRIMARY KEY	NOT NULL
password	VARCHAR2		NOT NULL
email	VARCHAR2		NOT NULL

Table 1.3 : Table above show the data model for the administrator

PRODUCT			
productName	VARCHAR2		NOT NULL
productID	VARCHAR2	UNIQUE	NOT NULL
productAvailablity	BOOLEAN		NOT NULL
price	DOUBLE		NOT NULL
productinfo	VARCHAR2		
deliveryMethod	BOOLEAN		NOT NULL

Table 1.4 : Table above show the data model for the product

ORDER			
orderID	INTEGER	UNIQUE	NOT NULL
orderInfo	VARCHAR 2		NOT NULL
OrderStatus	BOOLEAN		NOT NULL
orderPrice	DOUBLE		NOT NULL

Table 1.5 : Table above show the data model for the order

5.2 Data Dictionary

accId	username	password	address	contactNumber	email

Table 2 : The above table shows the relation table for retailer

From table 1.1, we can understand that username, password, address and email have datatype varchar2 in relation table and string in class diagram. The primary key in this table is username. The accID is unique. All the attributes are not null except for address.

accId	username	password	address	contactNumber	email

Table 3 : The above table shows the relation table for retailer

From table 1.2, we can understand that username, password, address and email have datatype varchar2 in relation table and string in class diagram. The primary key in this table is username. The accID is unique. All the attributes are not null except for address.

username	password	email

Table 4 : The above table shows the relation table for administrator

From table 1.3, we can understand that username, password, and email have datatype varchar2 in relation table and string in class diagram. The primary key in this table is username. All the attributes are not null except for address.

productId	productName	productAvailability	Price	ProductInfo	accId

Table 5 : The above table shows the relation table for product

From table 1.4, we can understand productName, productID and productInfo have datatype varchar2 in relation table and string in class diagram. While for productAvailability and deliveryMethod have datatype boolean. For the price it have datatype double. There is no primary key. All the attributes are not null except for productInfo.

orderId	orderInfo	orderPrice	accId

Table 6 : The above table shows the relation table for order

From the diagram above, we can see that int remains as integer in the relation table, string as varchar2, and double changes to number int relation table. In relation table have foreign key which is accID and its datatype is number. accID is the primary key for the customer/retailer table. All data in the order table is not null. Which cannot be empty. orderID is unique in this table.

6.0 INTERFACES DESIGN

This section shows all the interface design for the FoodBuddy system that will be developed later by the designer.

6.1 Sequence Diagram And Boundary Control Entity Approach

6.1.1 Create Account (UC1)

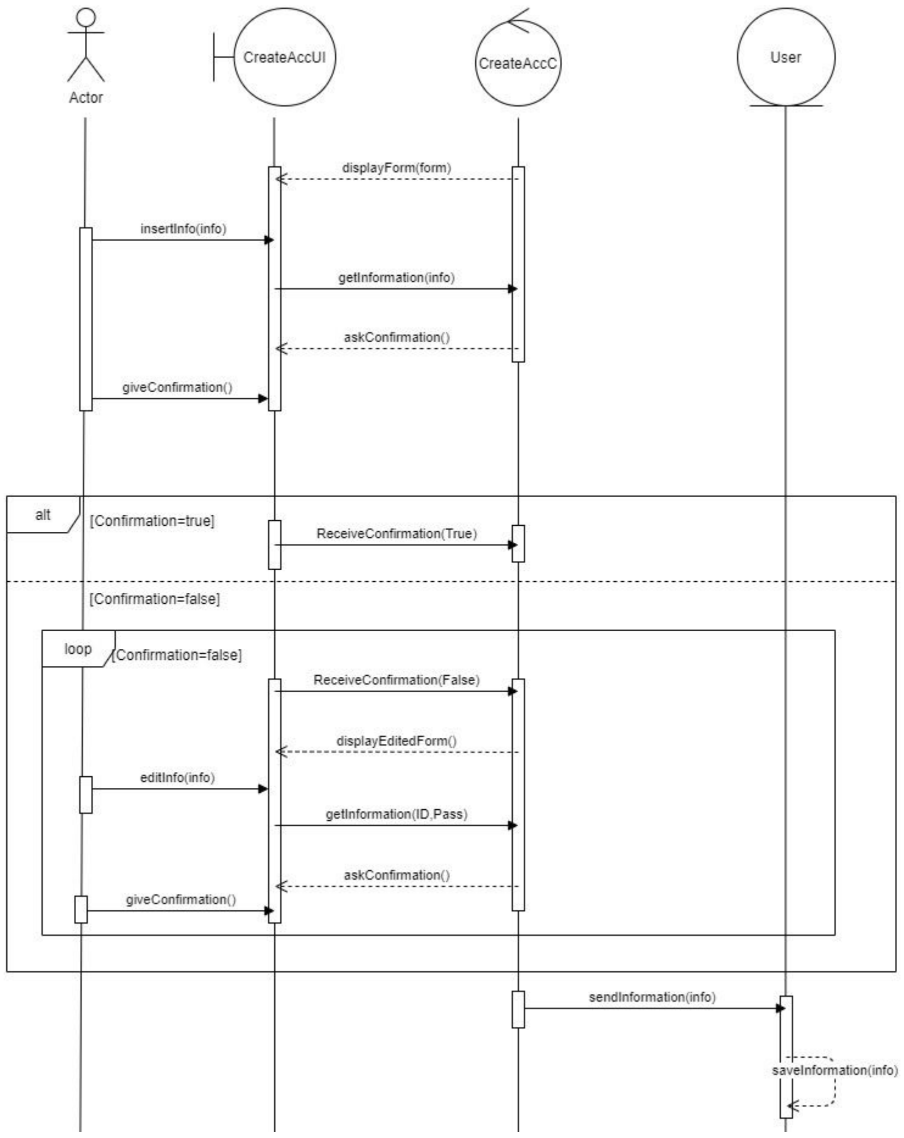


Diagram 7.1 : The above diagram shows the sequence diagram for Creating an Account

Diagram above shows the Sequence Diagram for the Create Account use case. The sequence diagram has 4 entities which are Actor, CreateAccUI (the boundary class), CreateAccC (the control class), User (the entity class). The activities in this use case take place among these entities.

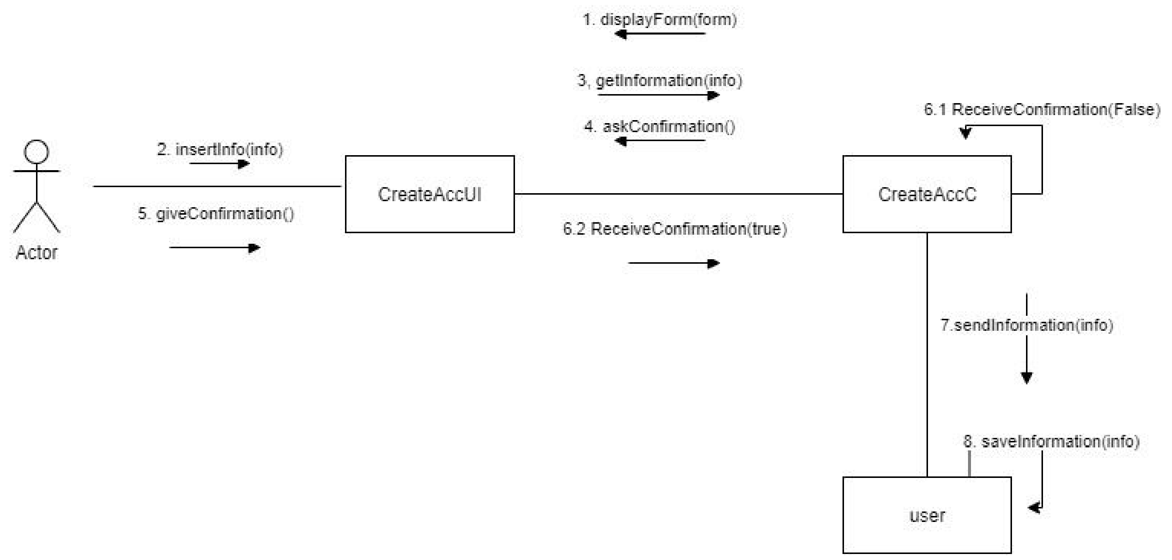
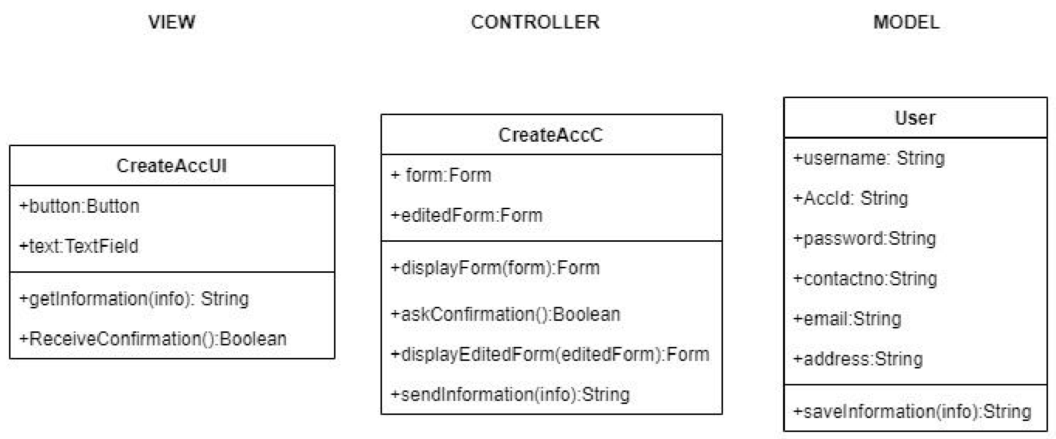


Diagram 7.2 : The above diagram shows the communication Diagram for creating an account

Based on diagrams, an empty form in CreateAccUI is displayed and the user needs to insert the information. Upon user clicking the Submit button, the boundary class acts by getting, asking confirmation and submitting the submit response to User() and saves all the information into User (entity class). On the other hand, the alternatives when the user does not confirm, CreateAccUI will display the edited form. Users will continue editing the form at CreateAccUI. When the user presses the submit button and confirms the information will be saved.


VIEW

CreateAccUI
+button:Button +text:TextField
+getInformation(info): String +ReceiveConfirmation():Boolean

CONTROLLER

CreateAccC
+ form:Form +editedForm:Form
+displayForm(form):Form +askConfirmation():Boolean +displayEditedForm(editedForm):Form +sendInformation(info):String

MODEL

User
+username: String +AccId: String +password:String +contactno:String +email:String +address:String
+saveInformation(info):String

Diagram 7.3 : The diagram above shows the MVC Class Diagram for Creating an account

The diagram shows three classes .The first class that involved is CreateAccUI class which is a boundary class for view. CreateAccUI class contains Buttons to submit the form., reset the form and confirmation for the detail; TextFields to insert information about the user. The methods that are involved in this class are getInformation(info) to fetch the information being inserted and ReceiveConfirmation() to fetch the type of confirmation chosen by the user.

The second class is CreateAccC class which is a control class which contains forms and editedForm to get user information. The methods that are involved in this class are displayForm(Form) which fetch the form and display to boundary class; askConfirmation() to ask for confirmation for the detail before send it to save; displayForm(Form) which fetch the form and display to boundary class; displayEditedForm(Form) which fetch already editedForm and display to boundary class for the user to make changes; sendInformation(info) which send the information to entity class to be stored.

The third class is User class which is a model class. The class contains information such as username, AccId, password, contactNo, email and address. This class contains one method which is saveInformation() to store the information.

6.1.2 Login (UC2)

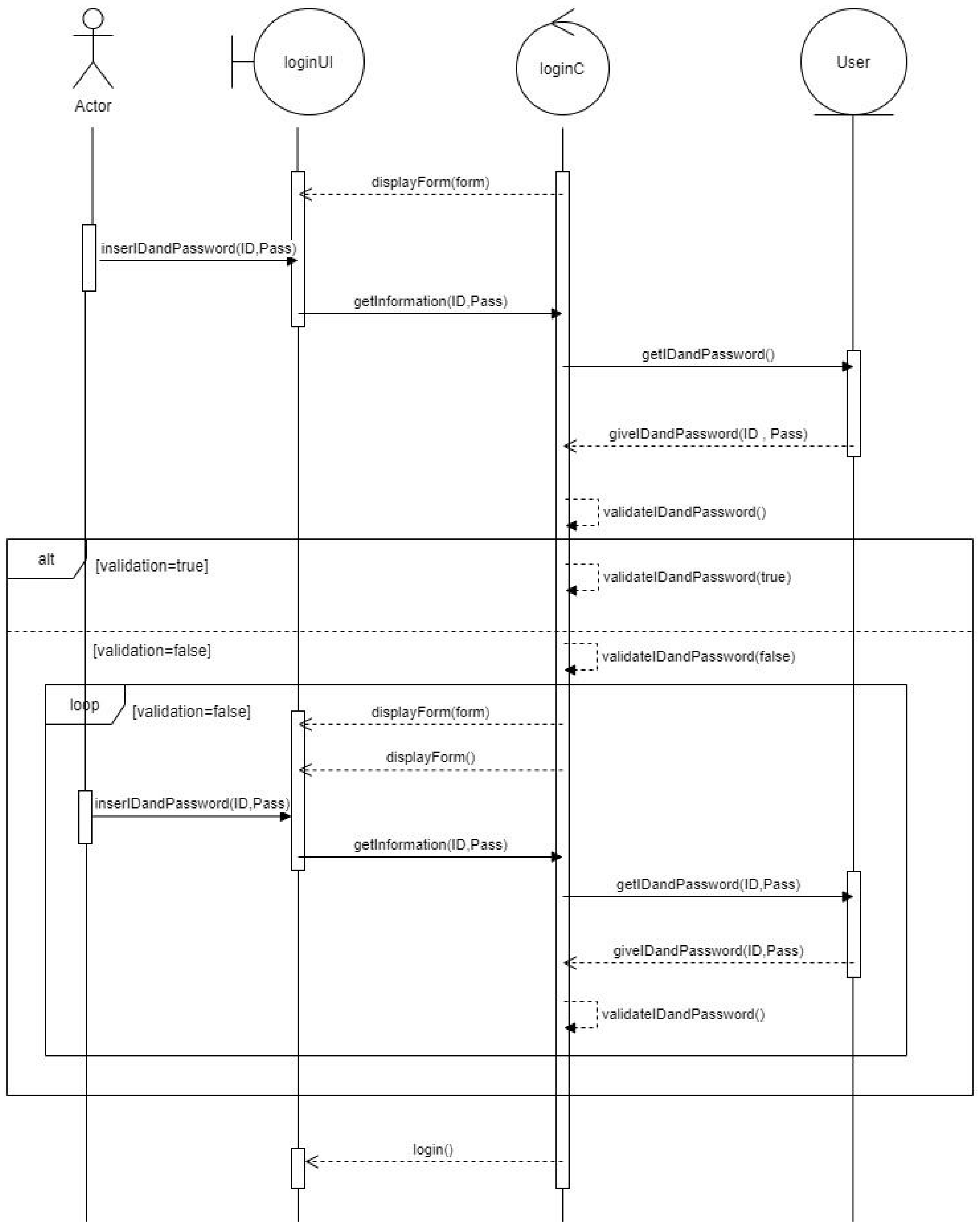


Diagram 8.1 : The above diagram shows the sequence diagram for login

Diagram above shows the Sequence Diagram for Login use case. The sequence diagram has 4 entities which are Actor, LoginUI (the interface class), LoginC (the control class), User (the model class). The activities in this use case take place among these entities.

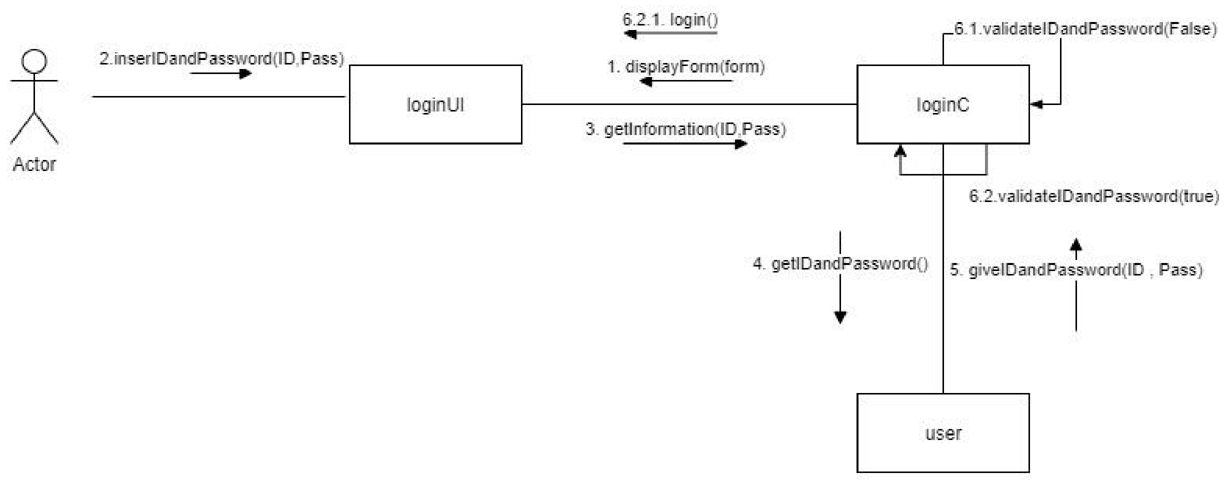

Actor
2.inserIDandPassword(ID,Pass)
loginUI
6.2.1. login()
1. displayForm(form)
3. getInformation(ID,Pass)
loginC
6.1.validateIDandPassword(False)
6.2.validateIDandPassword(true)
4. getIDandPassword()
5. giveIDandPassword(ID , Pass)
user

Diagram 8.2 : The above diagram shows the Communication Diagram for Login

Based on diagrams, empty text fields in LoginUI are displayed and the user needs to enter the following attributes which are ID and Password. Upon user clicking the Login button, the boundary class acts by getting the inserted information and fetching the ID and Password from User(entity class) for validation. LoginUI will display a message on LoginUI informing the user that he or she successfully logged in. On the other hand, the alternatives when the validation result is invalid, LoginUI will pop out informing that the user has entered the wrong ID or Password while asking the user to re-enter ID and Password. The system will validate again after the user enters the Login button.

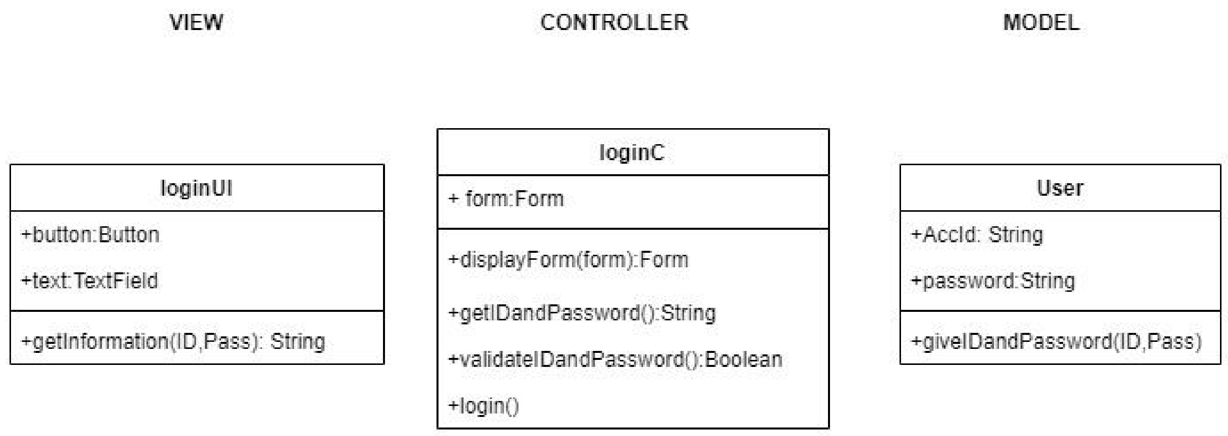


Diagram 8.3 : The above diagram shows the MVC Class Diagram for Login

The diagram shows three classes. The first class that is involved is LoginUI class which is a boundary class for view. LoginUI class contains Buttons to submit the form; TextFields to insert ID and password. The method that is involved in this class is getInformation(info) to fetch the information being inserted.

The second class is LoginC class which is a control class which contains forms and to get ID and password. The methods that are involved in this class are displayForm(Form) which fetch the form and display to boundary class; getIDandPassword() which get the ID and Password from User (the entity class) to further validation; validateIDandPassword() to compare the inserted ID and Password with the database; login() to show failure and success of validation and login.

The third class is User class which is a model class. The class contains information such as AccId, password. This class contains one method which is giveID and Password(ID,Pass) to send information for validation.

6.1.3 Manage Product (UC3)

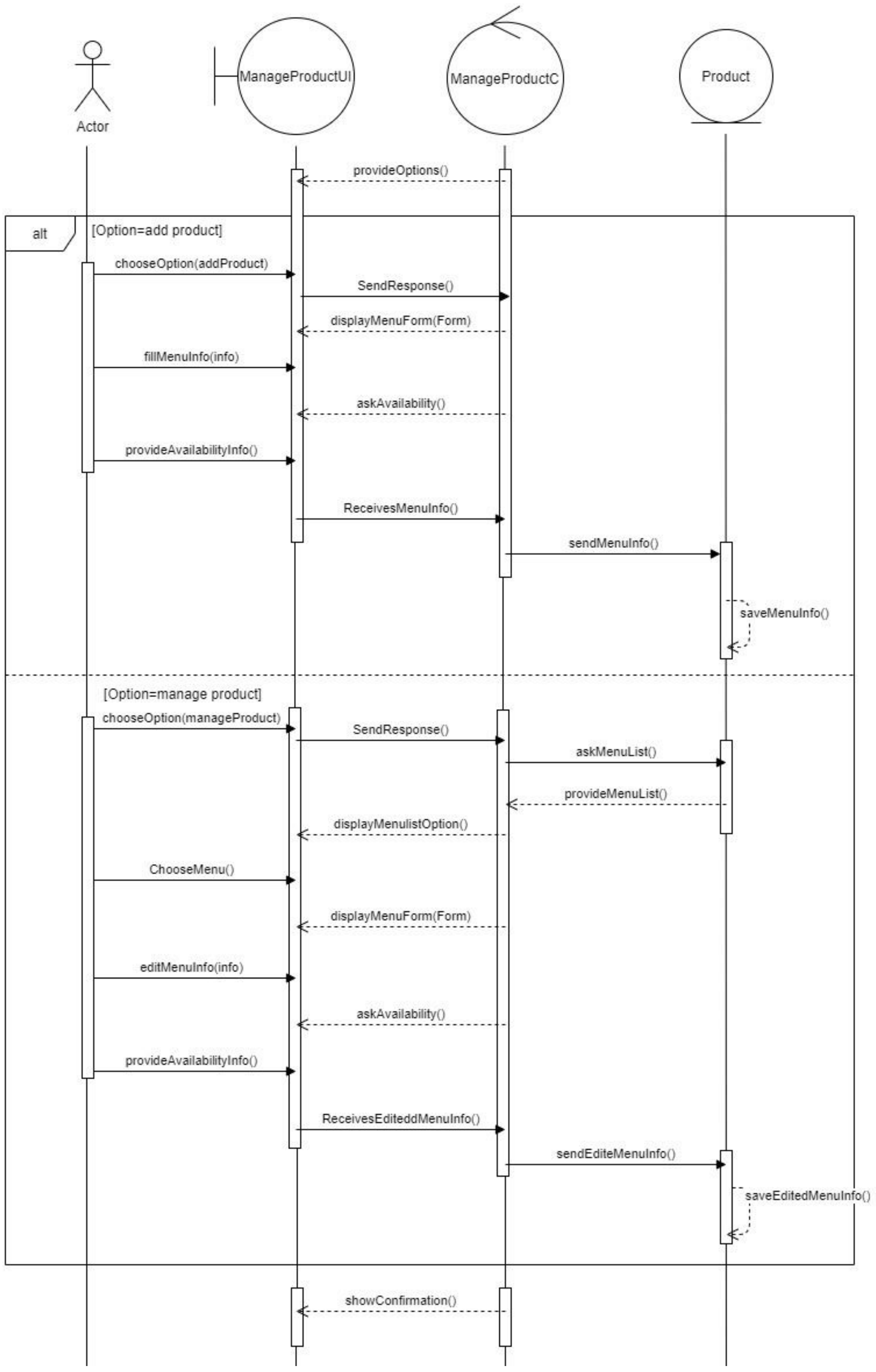


Diagram 9.1 : The above diagram shows the Sequence Diagram for Manage Product

Diagram above shows the Sequence Diagram for Manage Product use case. The sequence diagram has 4 entities which are Actor, ManageProductUI (the interface class), ManageProductC (the control class), Product (the model class). The activities in this use case take place among these entities.

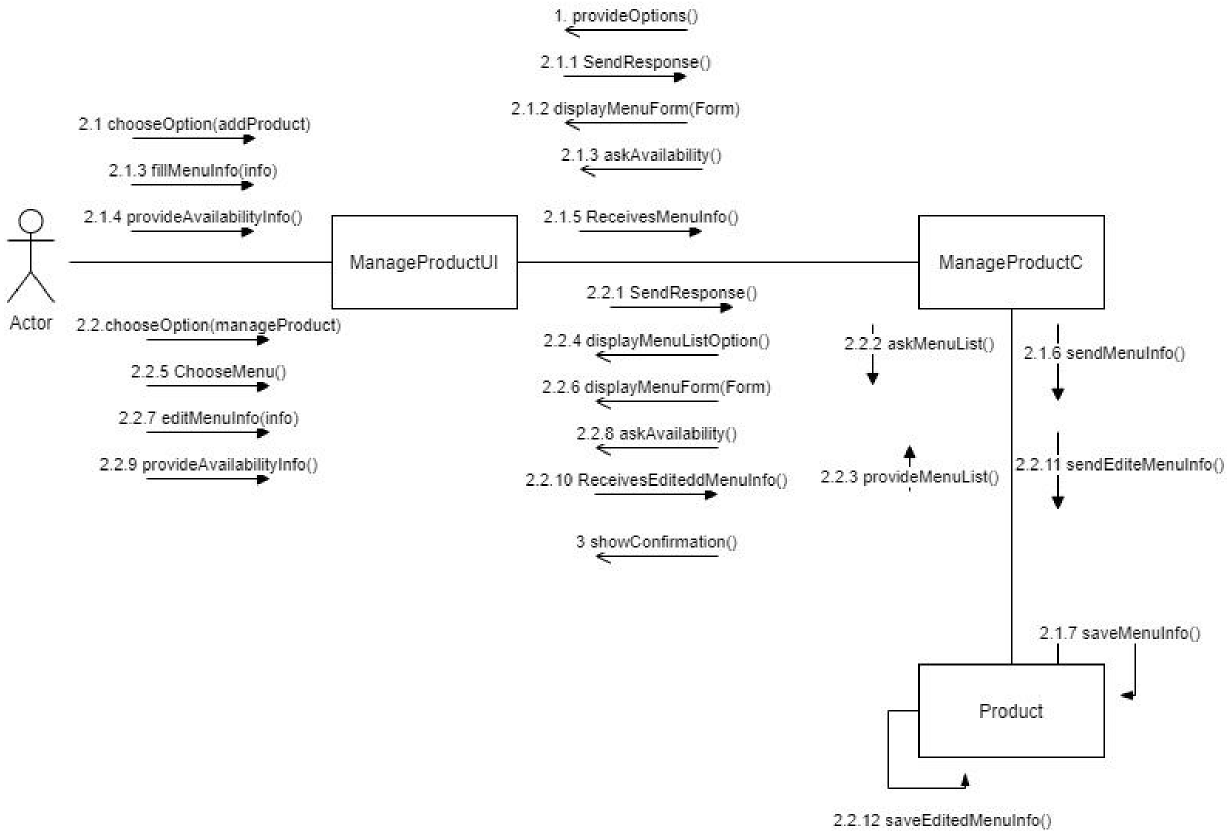


Diagram 9.2 : The above diagram shows the Communication Diagram for Manage Product

Based on diagrams, options will be provided in ManageAccountUI and the user needs to choose one of the options. Upon user clicking the add product option the boundary class acts by submitting the response and proceeding to get the form to fill in information about the new menu from ManageProduct entity. Then, the user has to fill in the information which will be sent to a ManageProduct class through boundary class. The menu info will be saved. On the other hand, the alternatives when the user chooses to manage product options the boundary class will submit the response and to get the existing product list from ManageProduct entity. Then, the user has to manage product information which will be sent to a ManageProduct class through boundary class to be saved.

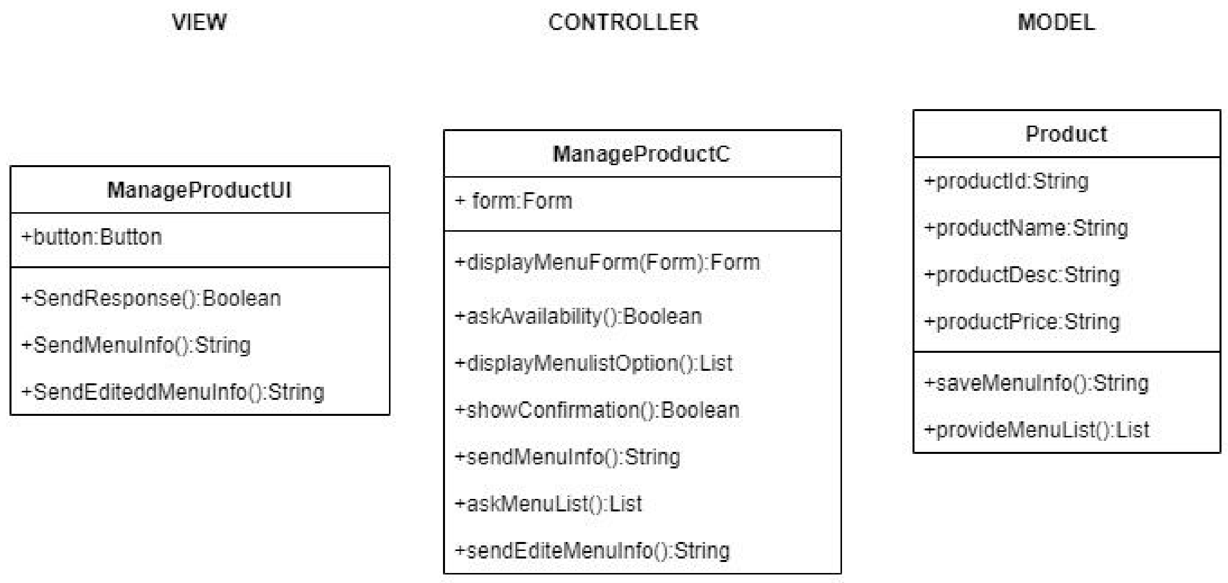


Diagram 9.3 : The above diagram shows the MVC Class Diagram for Manage Product

The diagram shows three classes. The first class that is involved is ManageProductUI class which is a boundary class for view. ManageProductUI class contains Buttons to submit the form; TextFields to insert information about the product . The methods that are involved in this class are SendResponse() which sends a response on the choice given; sendMenuInfo(info) to fetch the information being inserted; sendEditedMenuInfo(info) to fetch the edited information being inserted in alternative choice.

The second class is ManageProductC class which is a control class which contains forms and to get information about the product. The methods that are involved in this class are displayMenuForm(Form) which fetch the form and display to boundary class; askAvailability() to ask for availability of the menu; displayMenuListOption() which show options of menu to be edited; showConfirmation() to show whether the information is saved; sendMenuInfo() to send the information to be stored in entity class; askMenuList() which fetch menu list from entity class to display; sendEditedMenuInfo() to send the edited information to be stored in entity class

The third class is Product class which is a model class. The class contains information such as ProductId, ProductName, ProductDesc, ProductPrice. This class contains two methods which are sendMenuInfo() to store information about products and provideMenuInfo() to display.

6.1.4 View Product (UC4)

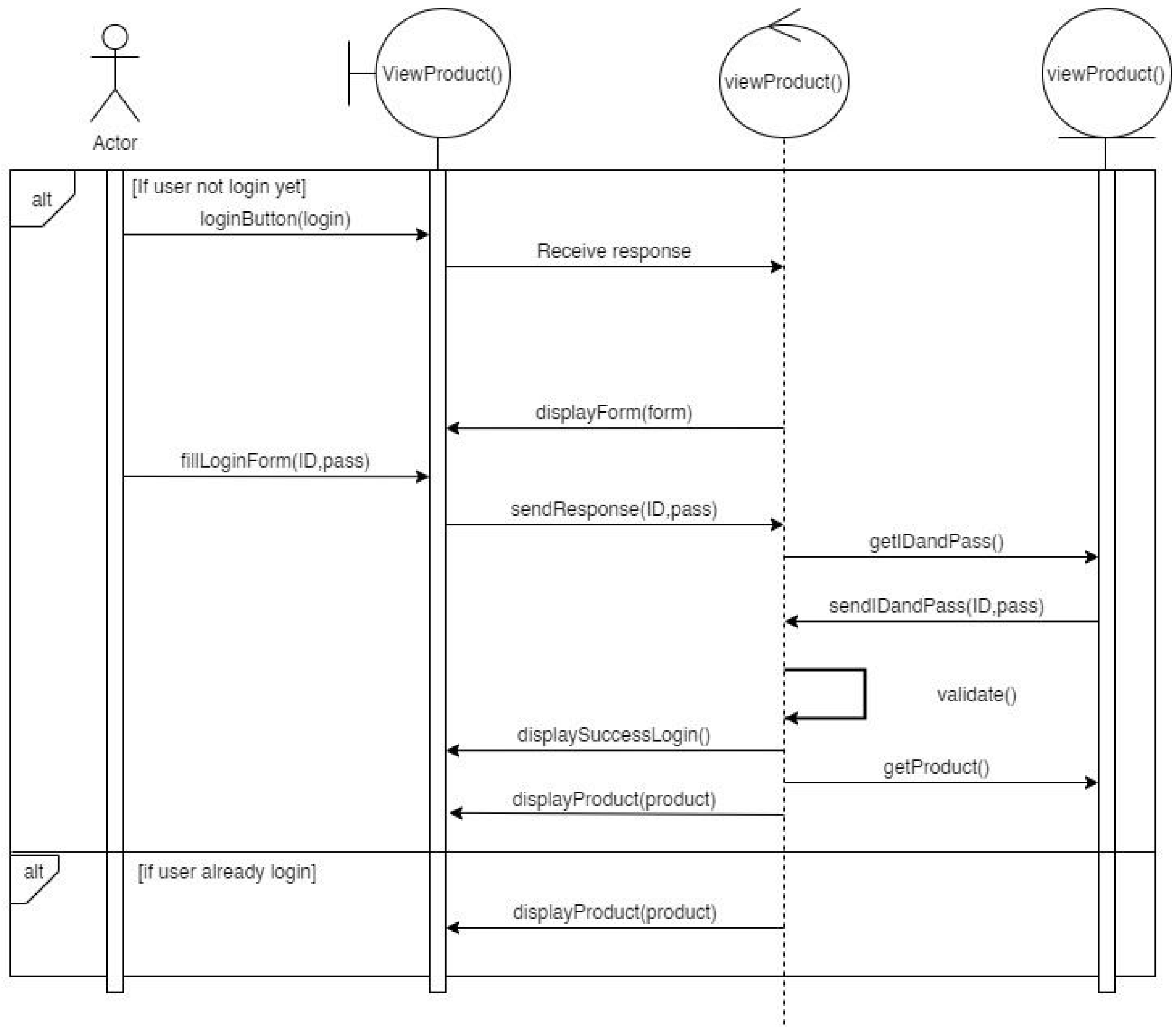


Diagram 10.1 : The above diagram shows the Sequence Diagram for View Product

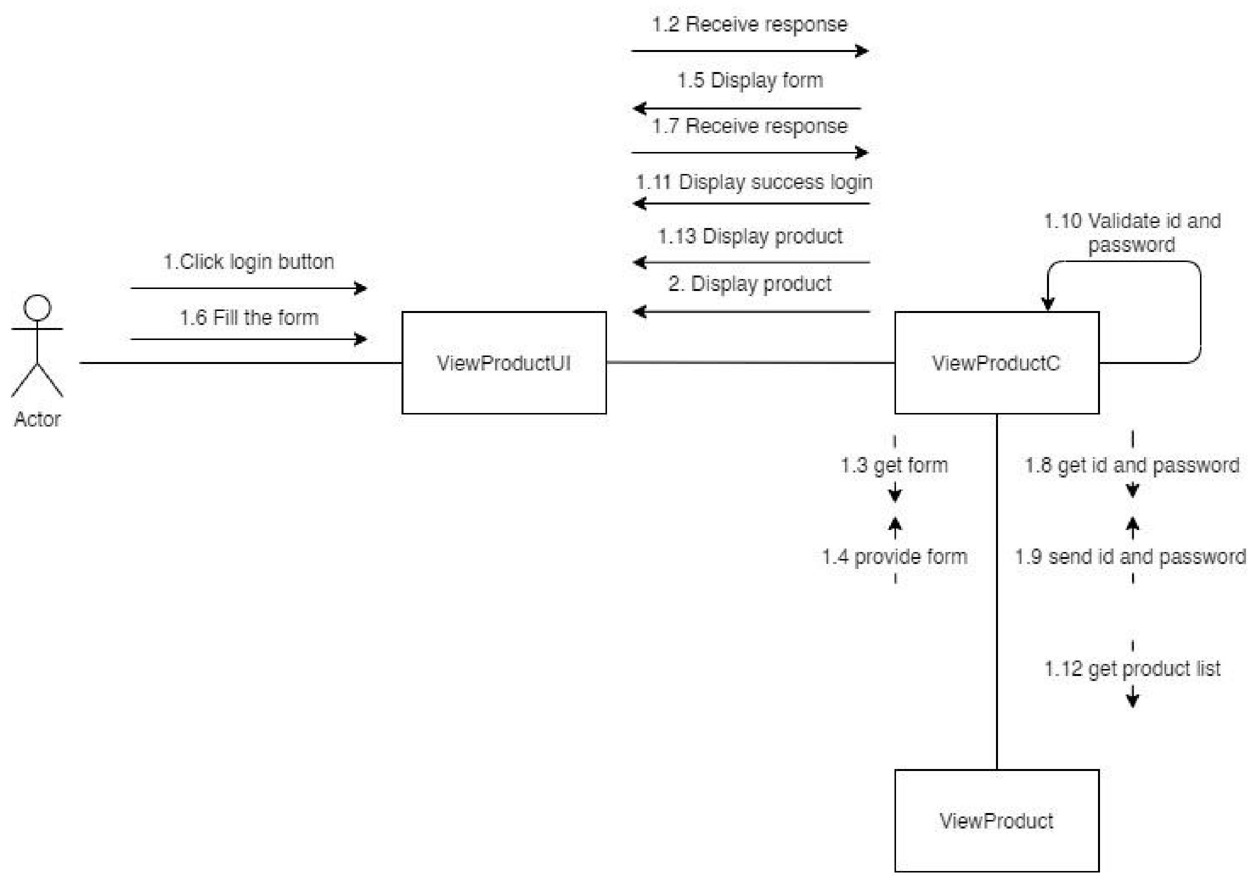


Diagram 10.2 : The above diagram shows the Communication Diagram for View Product

Based on diagrams, empty text fields in LoginUI are displayed and the user needs to enter the following attributes which are ID and Password. Upon user clicking the Login button, the boundary class acts by getting the inserted information and fetching the ID and Password from User(entity class) for validation. LoginUI will display a message on LoginUI informing the user that he or she successfully logged in. After login, ViewProductUI will display the product on the page. On the other hand, the alternatives when the user already login, the system will display the product in ViewProductUI without login.

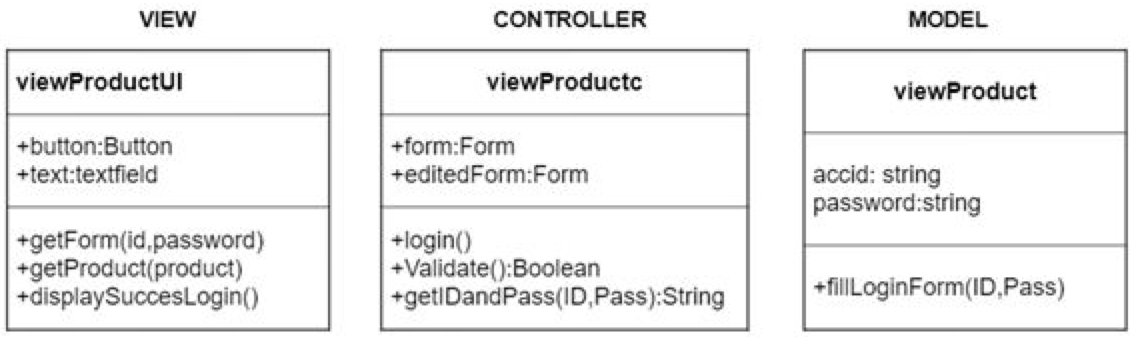


Diagram 10.3 : The above diagram shows the MVC Class Diagram for View Product

The diagram shows three classes .The first class that involved is viewProductUI class which is a boundary class for view. ViewProductUI class contains Buttons to view products; TextFields to insert user information and search keywords about the product. The methods that are involved in this class are getForm(info) to fetch the information being inserted, getProduct to fetch the product and displaySuccessLogin() to fetch the message to the user.

The second class is CreateAccC class which is a control class which contains forms and editedForm to get user information. The methods that are involved in this class are login() which fetch the form and display to the boundary class; validate() to ask for confirmation for the detail before sending it to save; getIDandPass(ID,Pass) which fetch the ID and password to the boundary class.

The third class is User class which is a model class. The class contains information such as username, AccId and password. This class contains one method which isfillLoginForm(ID,pass) to login into the system.

6.1.5 Order (UC5)

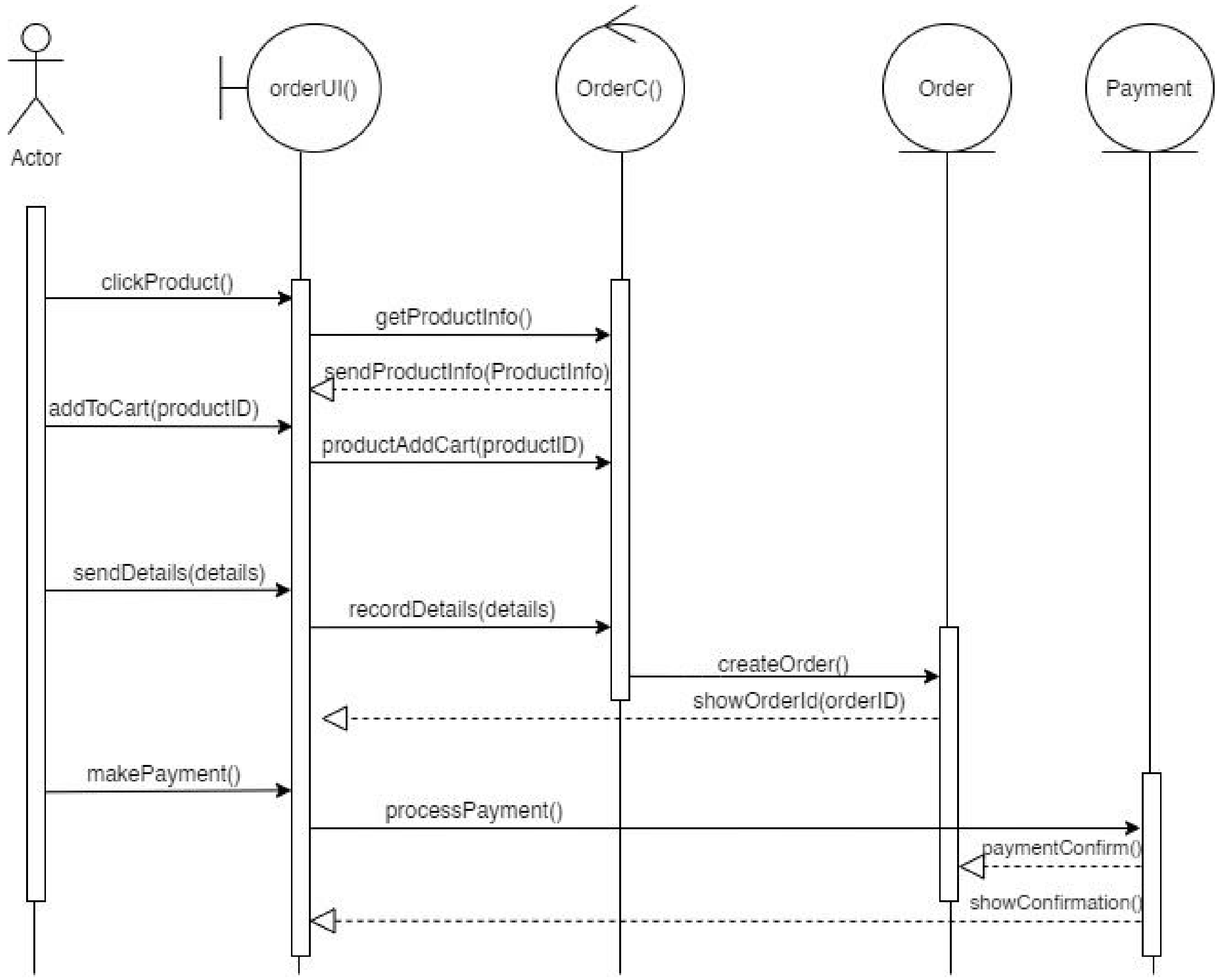


Diagram 11.1 : The above diagram shows the Sequence Diagram for Order

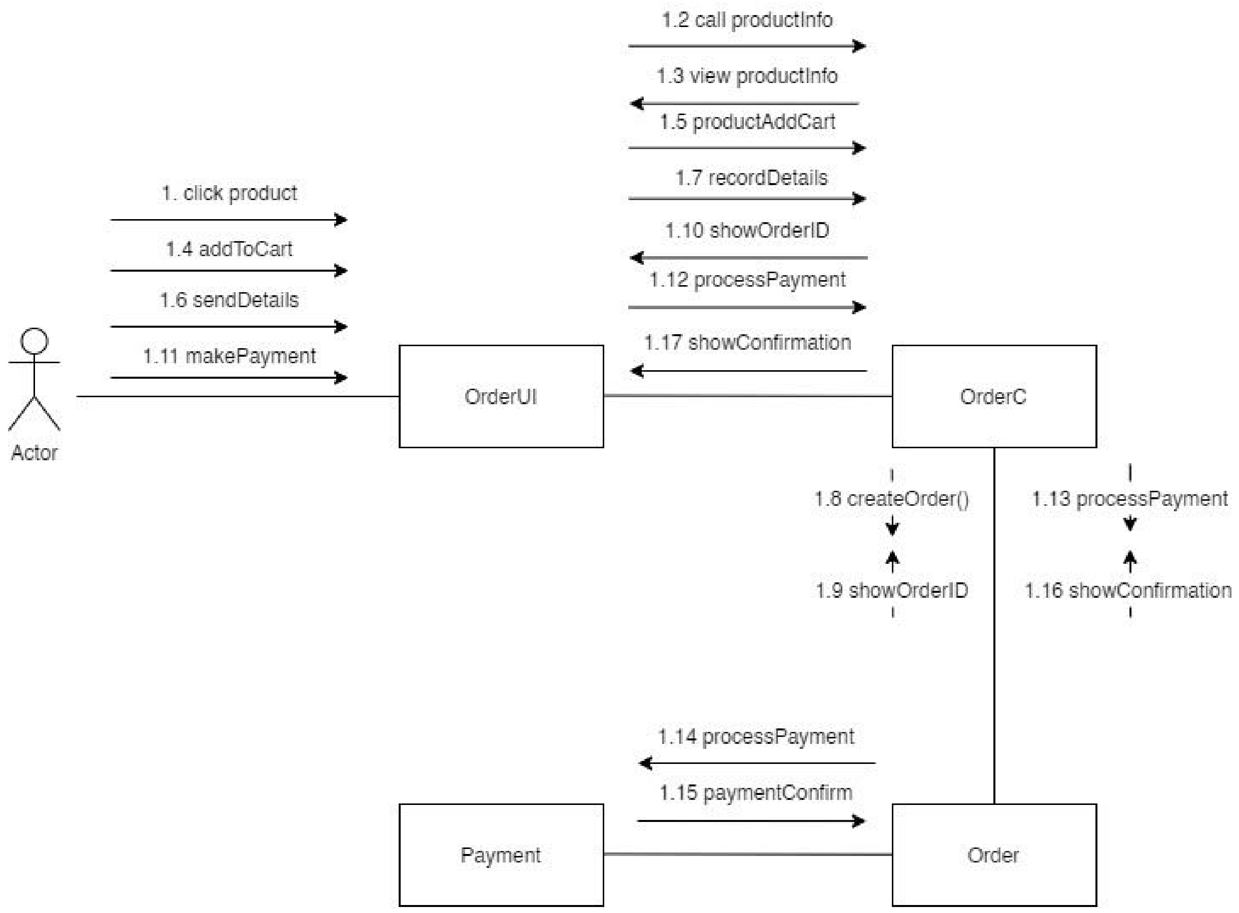


Diagram 11.2 : The above diagram shows the Communication Diagram for Order

Based on diagrams, the user needs to click one of the products in OrderUI and the products info will be shown. Upon user clicking the add to cart button, products will be added into cart and users need to fill details. OrderUI will display a message on LoginUI informing the orderID. OrderUI will direct the user to Payment to complete the payment. OrderUI will display a confirmation message after the payment is successfully made.

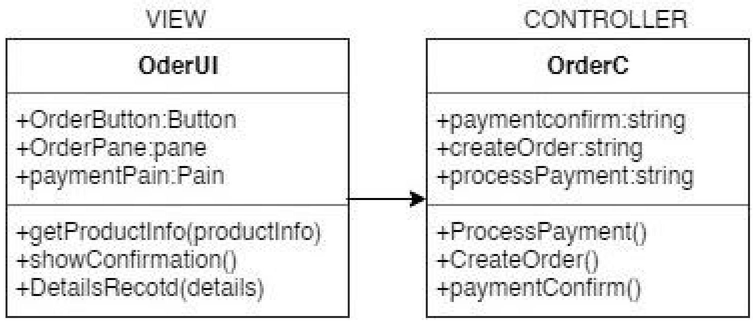


Diagram 11.3 : The above diagram shows the MVC Class Diagram for Payment Gateway

The diagram shows two classes. The first class that is involved is OrderUI class which is a boundary class for view. OrderUI class contains OrderButton to submit the order; OrderPane to create pane and paymentPain. The method that is involved in this class is getProductInfo(productInfo) to fetch the product information, showConfirmation() to show confirmation to the user and detailsRecord(details) to fetch details..

The second class is OrderC class which is a control class which contains paymentconfirm, createOrder and processPayment. The methods that are involved in this class are processPayment(), createOrder() and paymentConfirm().

6.1.6 Payment Gateway (UC6)

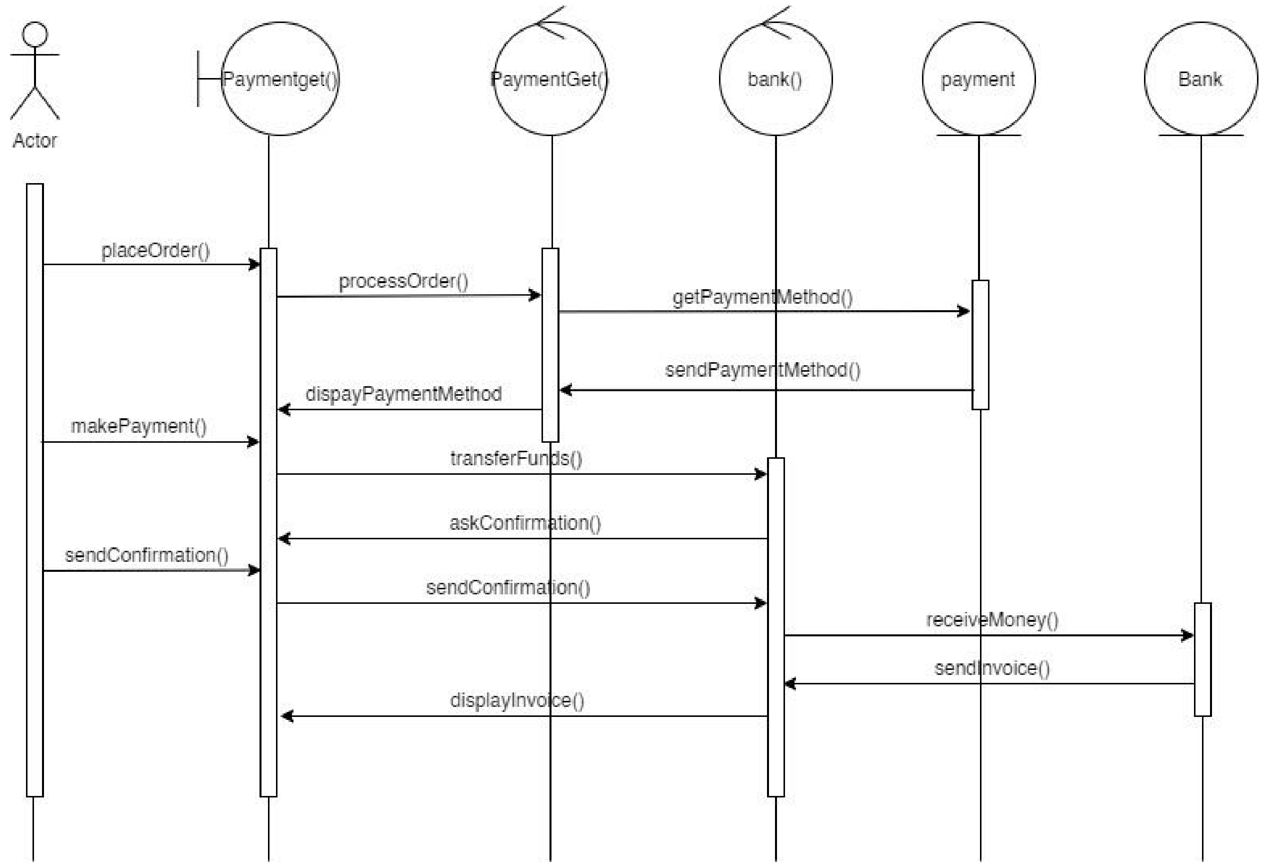


Diagram 12.1 : The above diagram shows the Sequence Diagram for Payment Gateway

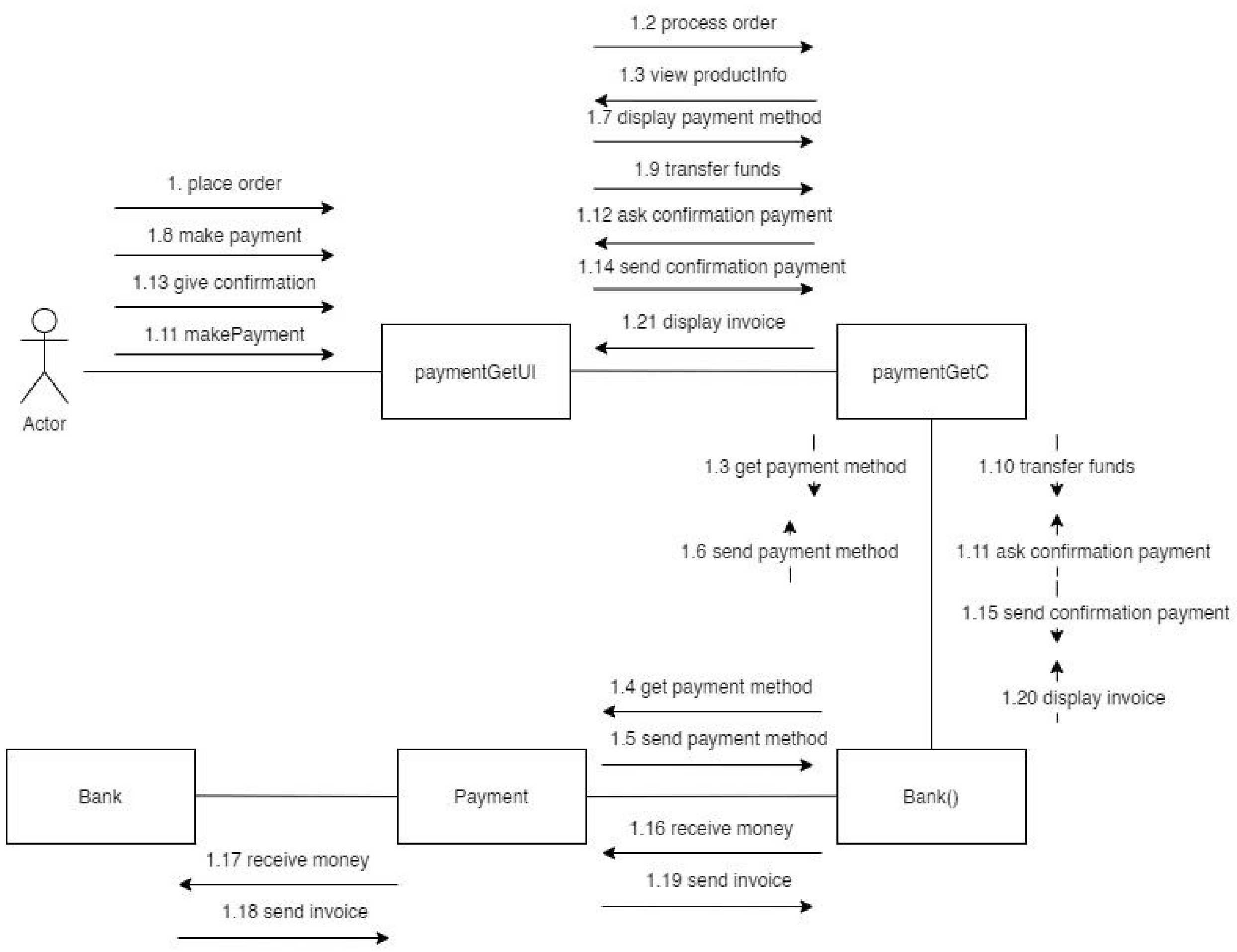


Diagram 12.2 : The above diagram shows the Communication Diagram for Payment Gateway

Based on diagrams, payment methods in PaymentGetUI are displayed and the user needs to choose either one. Upon user clicking the selected method, the user will make the payment to the bank. To confirm the payment, Payment will ask for confirmation from the user in PaymentGetUI. Once the user gives confirmation, the Bank will receive the money and will send an invoice to the user through paymentGetUI.

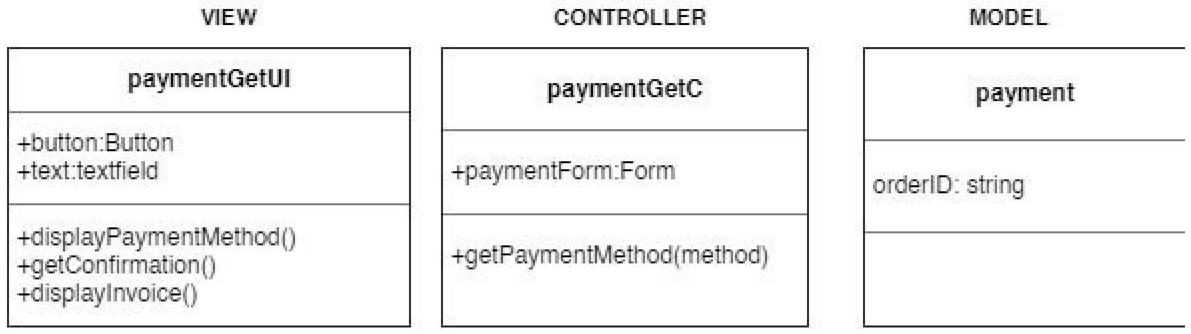


Diagram 12.3 : The above diagram shows the MVC Class Diagram for Payment Gateway

The diagram shows three classes .The first class that is involved is paymentGetUI class which is a boundary class for view. CreateAccUI class contains Buttons to submit the form,and TextFields to insert payment information about the order. The methods that are involved in this class are displayPaymentMethod() to display method of payment, getConfirmation() to fetch the type of confirmation chosen by the user and displayInvoice() to send invoice to the user.

The second class is paymentGetC class which is a control class which contains paymentForm to get payment information. The methods that are involved in this class are getPaymentMethod(method) which fetch the payment method and display it to boundary class.

The third class is User class which is a model class. The class contains information such as orderID.

6.1.7 Receive Notification for Order (UC7)

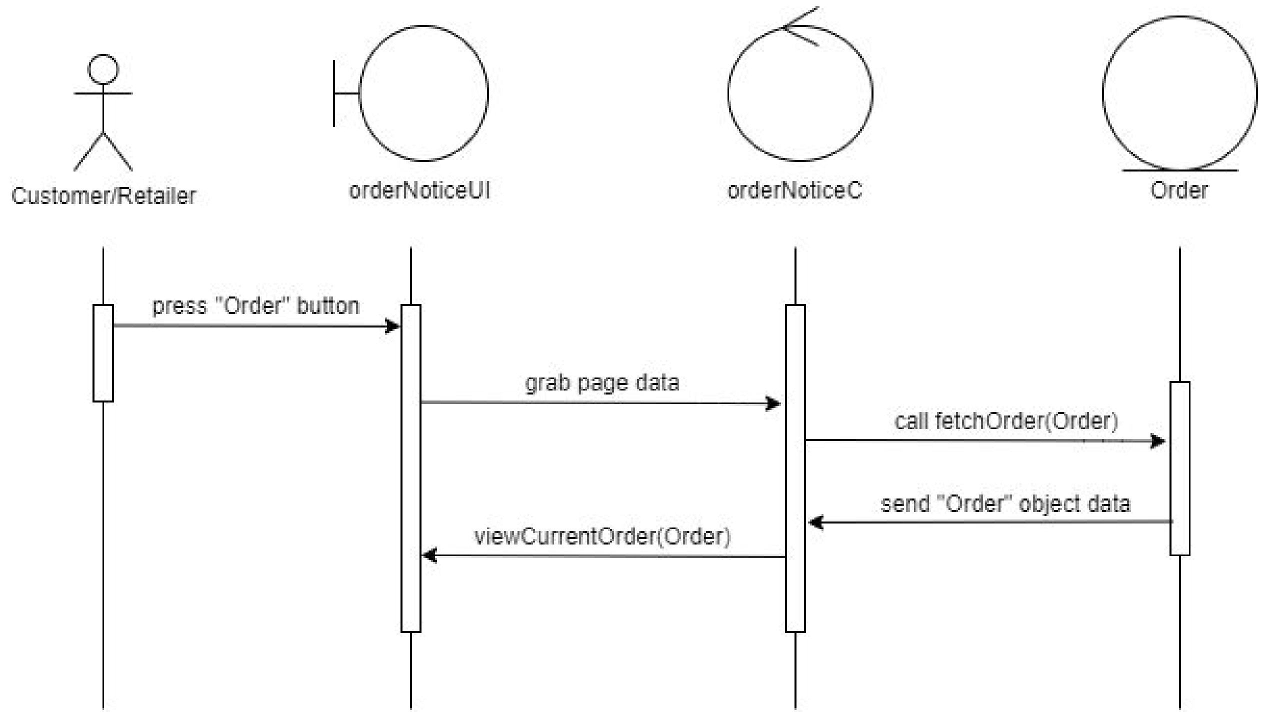


Diagram 13.1 : The above diagram shows the Sequence Diagram for Receive Notification for order

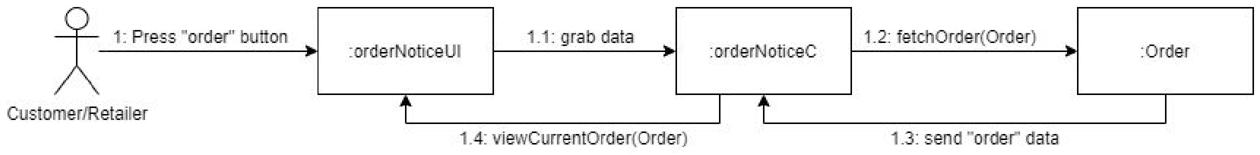


Diagram 13.2 : The above diagram shows the Communication Diagram for Receive Notification for order

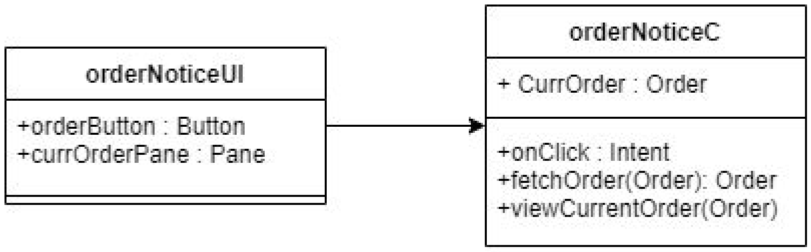


Diagram 13.3 : The above diagram shows the MVC Diagram for Receive Notification for order

Based on the diagram, the user will need to press the “Order” button in order to start this use case. The orderNoticeC will grab the current order from the database and then display it using the viewCurrentOrder method to the user.

6.1.8 Receive Notification for Confirmation (UC8)

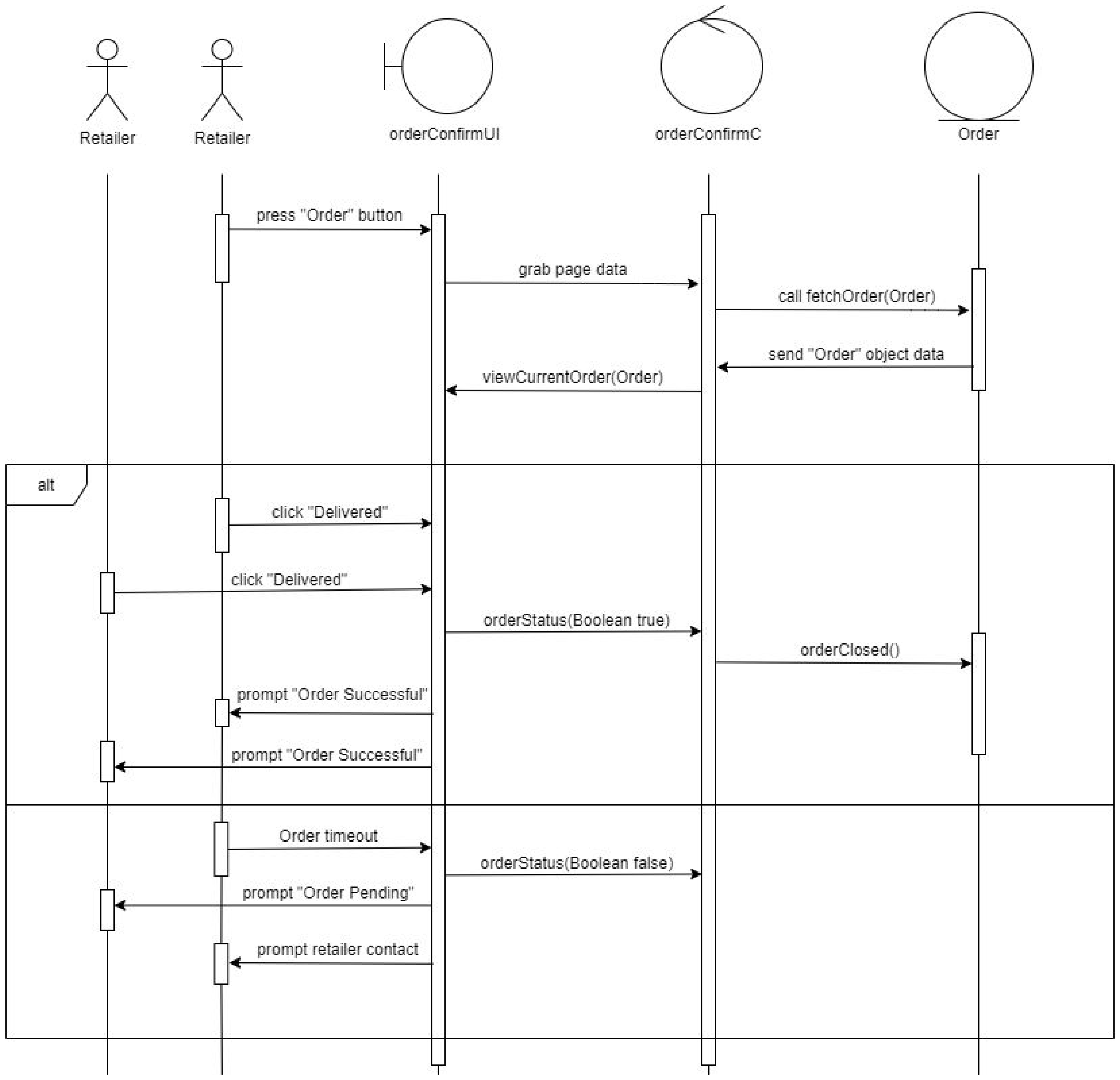


Diagram 14.1 : The above diagram shows the Sequence Diagram for Receive Notification for Confirmation

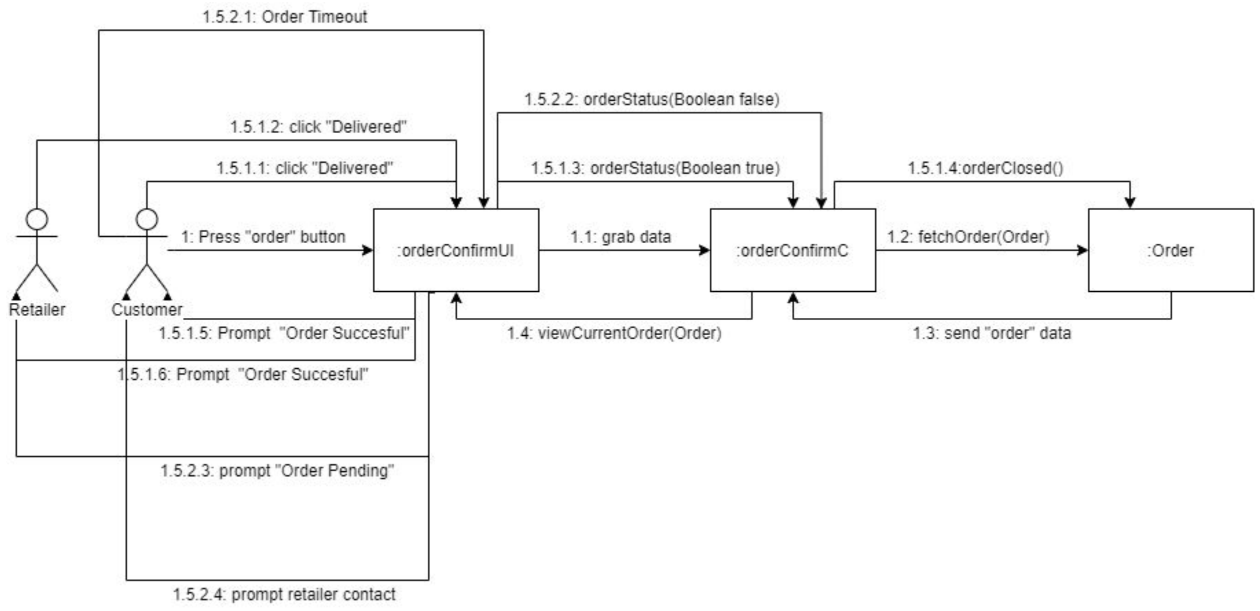

1.5.2.1: Order Timeout
1.5.2.2: orderStatus(Boolean false)
1.5.1.2: click "Delivered"
1.5.1.1: click "Delivered"
1.5.1.3: orderStatus(Boolean true)
1.5.1.4:orderClosed()
1: Press "order" button
:orderConfirmUI
1.1: grab data
:orderConfirmC
1.2: fetchOrder(Order)
:Order
Retailer
Customer
1.5.1.5: Prompt "Order Succesful"
1.4: viewCurrentOrder(Order)
1.3: send "order" data
1.5.1.6: Prompt "Order Succesful"
1.5.2.3: prompt "Order Pending"
1.5.2.4: prompt retailer contact

Diagram 14.2 : The above diagram shows the Communication Diagram for Receive Notification for confirmation

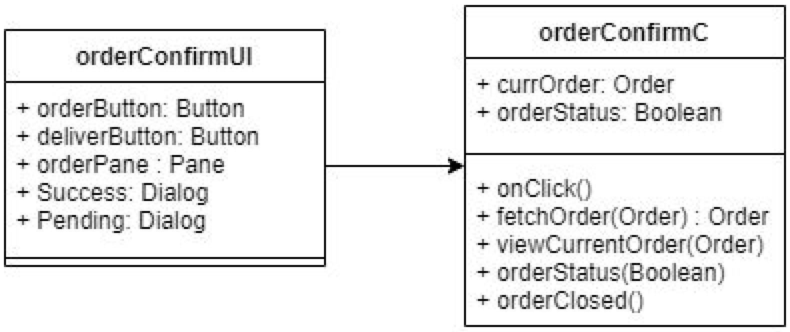
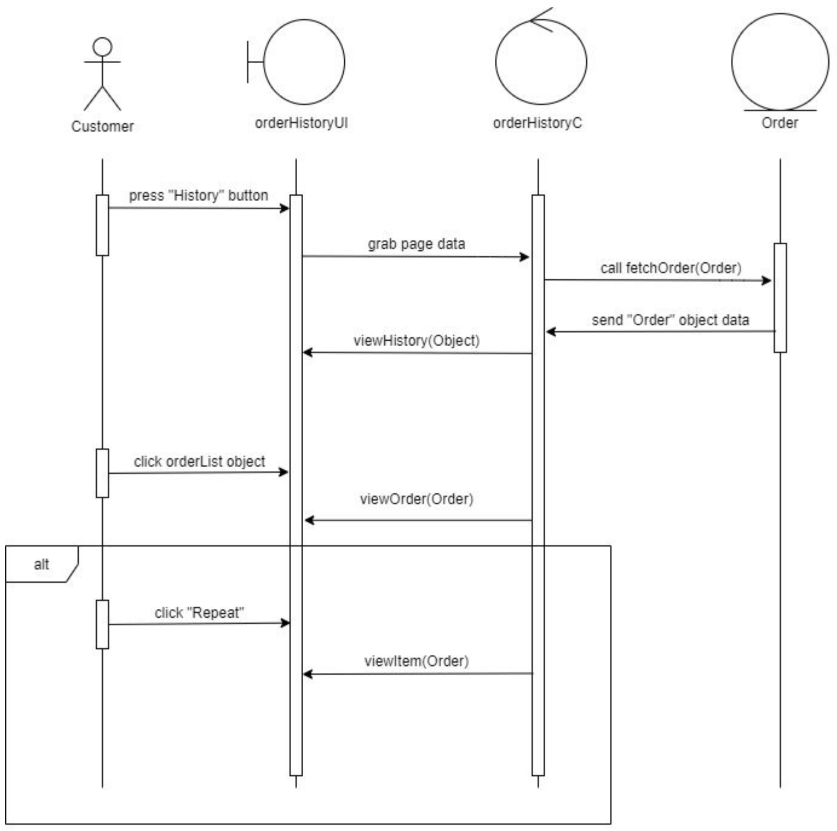


Diagram 14.3 : The above diagram shows the MVC Diagram for Receive Notification for confirmation

Based on the diagram, the use case starts when the user has filed an order. If the customer and the retailer pressed the “Delivered” button on their order page, the system would run the orderClosed() method to notify the database that the order was successful and needed to be closed. Else, if the order reaches a certain time, the UI will prompt “Order Pending” to the retailer and provide the customer with the retailer contact details. This use case ends when both users have pressed the “Delivered” button, until then the order will be in pending state.

6.1.9 Order History (UC9)



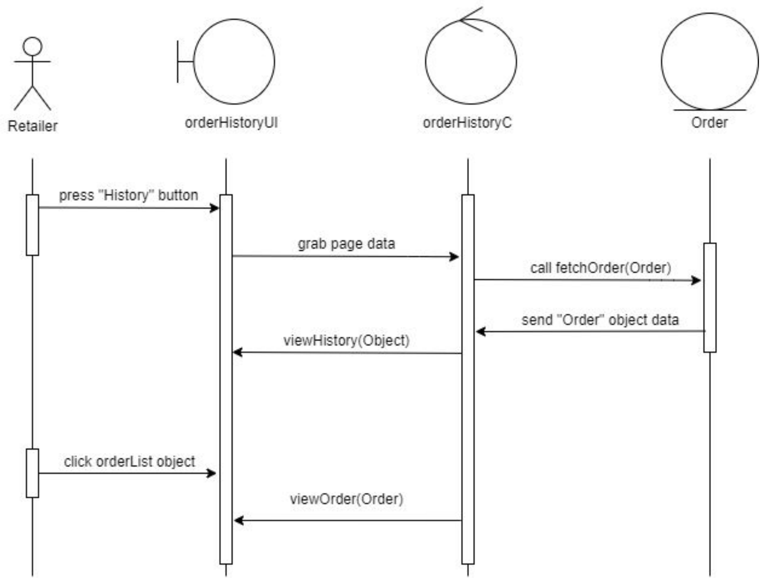
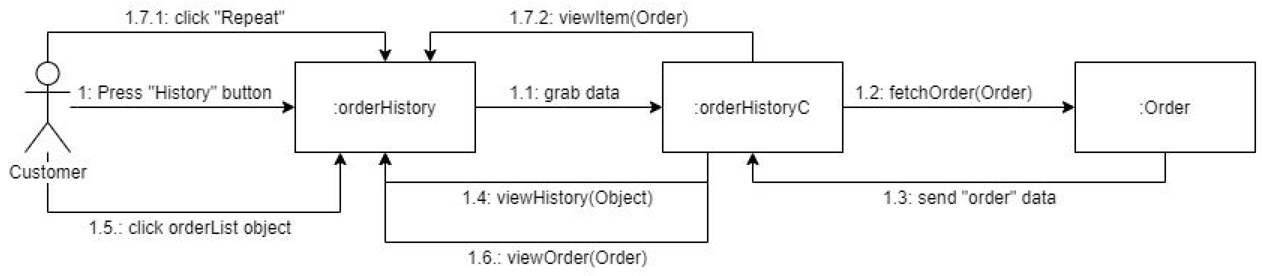


Diagram 15.1 : The above diagram shows the Sequence Diagram for Order History



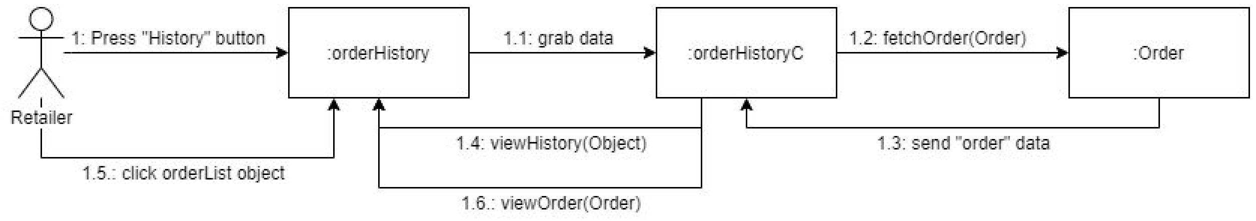


Diagram 15.2 : The above diagram shows the Communication Diagram for Order History

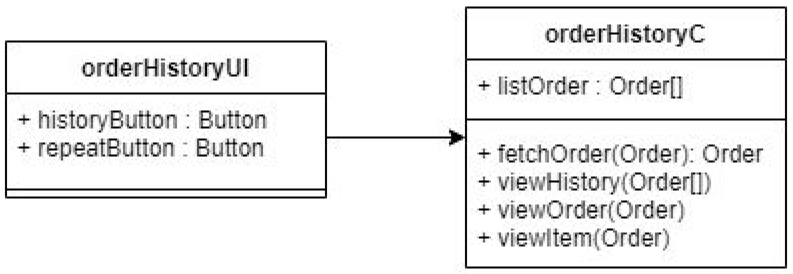


Diagram 15.3 : The above diagram shows the MVC Diagram for Order History

Based on the diagram, the use case will start when the customer/retailer presses the “History” button. For the customer side, it will prompt all of the customer's previous orders and when the customer clicks one of the order lists, it will prompt the order detail. If the customer presses the “Repeat” button, the user will be redirected to the item page to place an order of that item. For the retailer side, it will prompt all of the orders made by customers for their product. If the retailer clicks one of the order lists, it will prompt the order detail.

7.0 PROTOTYPE (iqah)

In this section, we are going to show interface design for this project. Figure 7.1 shows sign in interface. Once the user clicks on our apps, the first interface will appear in 5 seconds, then the second interface will come.

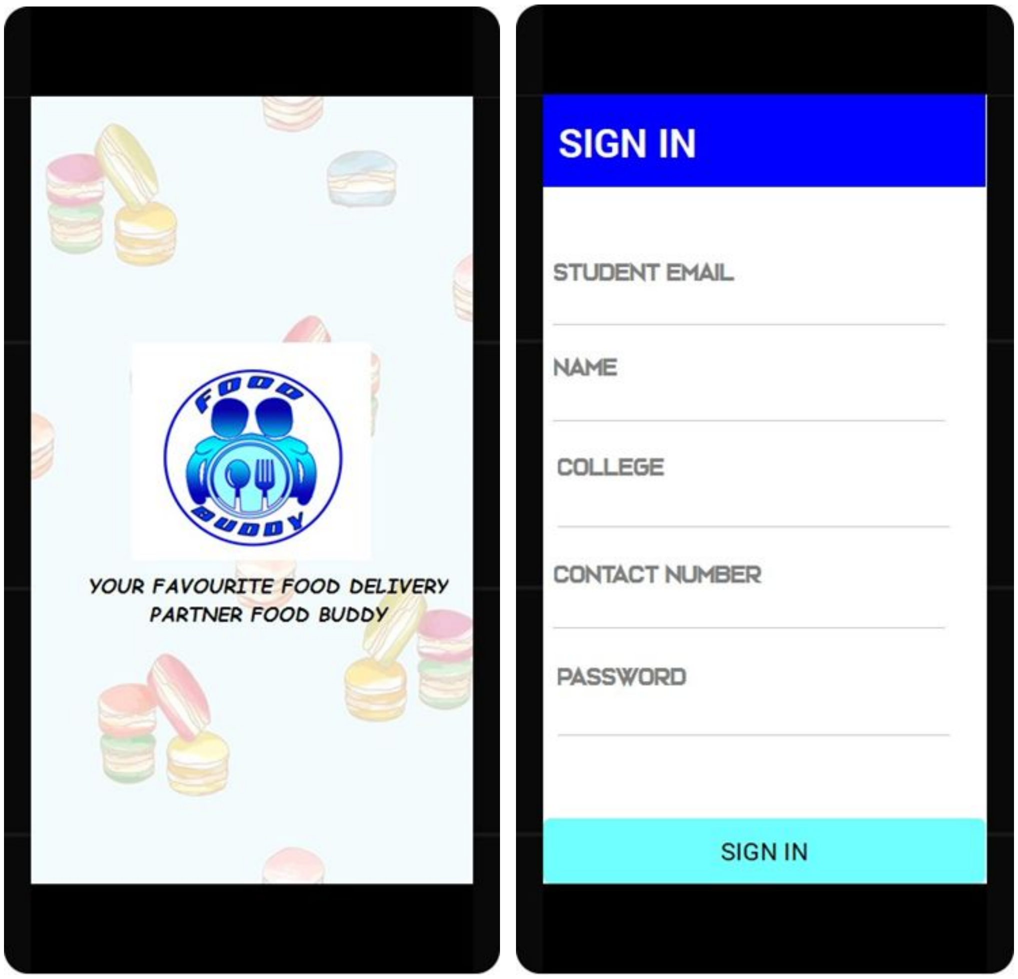


Figure 7.1 The picture above show the sign in interface

Figure 7.2 below shows logging interface. If user already have account, they can straight away login to our apps and can start activity there.

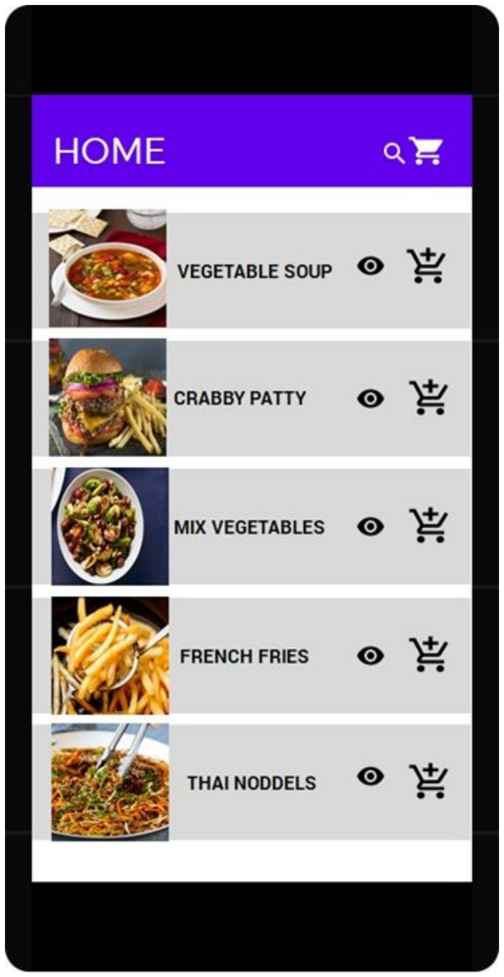


Figure 7.2 The picture above shows the logging interface

Figure 7.3 shows the main menu for these apps. Once users already log in to our apps, this interface will appear. Users can choose user profile to update their profile, manage menu for user as retailers update menu, and check order to user as customer to check their order status.

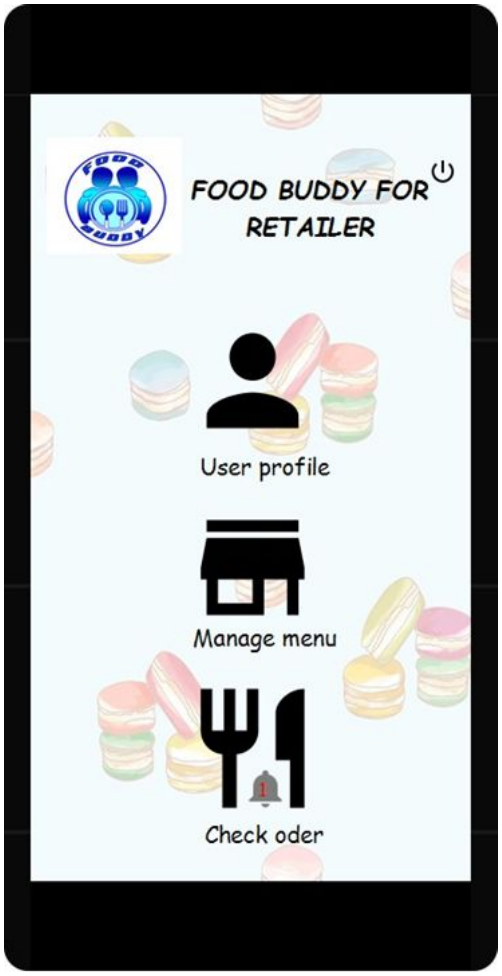
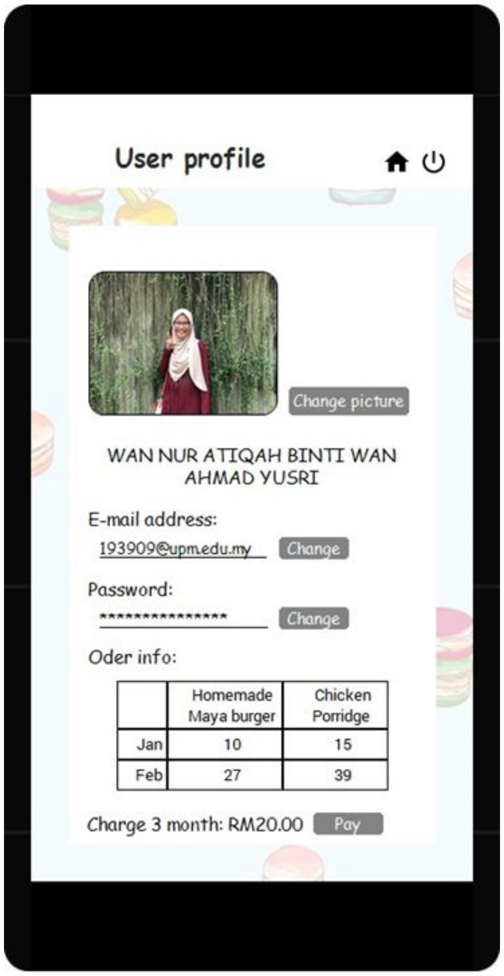


Figure 7.3 The above picture show the main menu interface

Figure 7.4 shows user profile. User profile will appear when the user clicks “user profile” in the main menu. Here users can update their picture, email address, and password. Here during implementation, we will add one information which is the user address.



WAN NUR ATIQAH BINTI WAN AHMAD YUSRI

E-mail address:
193909@upm.edu.my

Password:
****

Oder info:

	Homemade Maya burger	Chicken Porridge
Jan	10	15
Feb	27	39

Charge 3 month: RM20.00

Figure 7.4 The above picture show the user profile interface

Figure 7.5 shows a list of menu interfaces. From here, customers can choose any menu they want and add to cart by clicking at the “add trolley icon”. Customers also can view to get more information about the menu by clicking at “eye icon”.

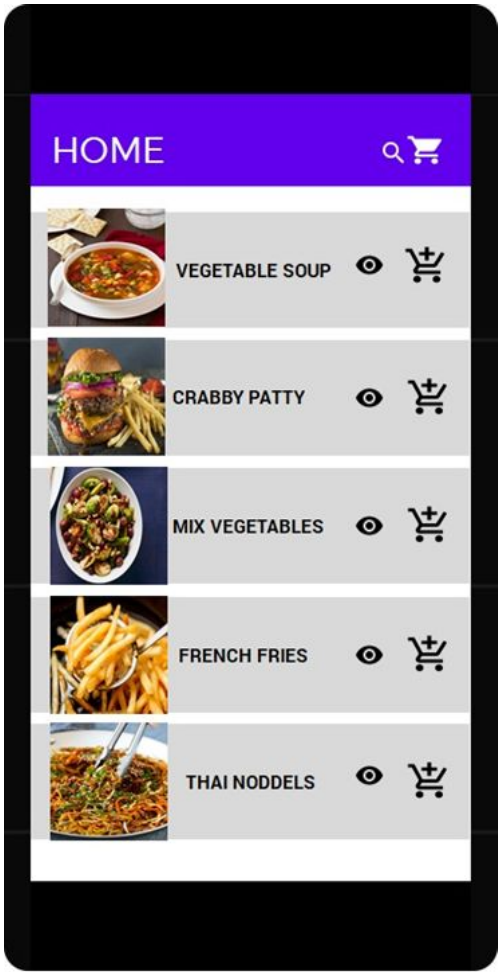
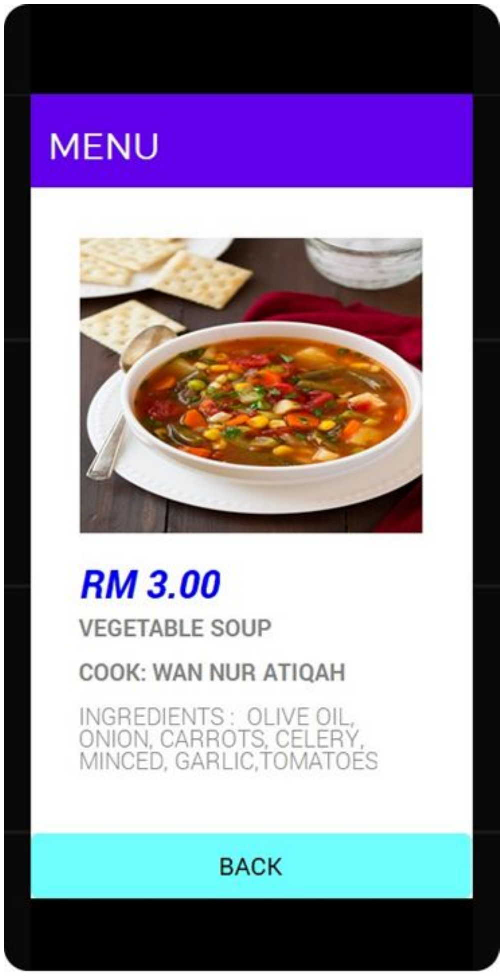


Figure 7.5 The picture above show the list of menu interface

Figure 7.6 shows menu description. This interface will appear when the user clicks “eye icon” in the menu list interface. They will show all description from retailer.



RM 3.00

VEGETABLE SOUP

COOK: WAN NUR ATIQAH

INGREDIENTS : OLIVE OIL, ONION, CARROTS, CELERY, MINCED, GARLIC,TOMATOES

Figure 7.6 The picture above show the description of the food interface

Figure 7.7 shows cart interface. When user click ad trolly icon in menu list interface, it will bring user to this interface, This interface user can see the menu they want check out. And users also can remove menus.

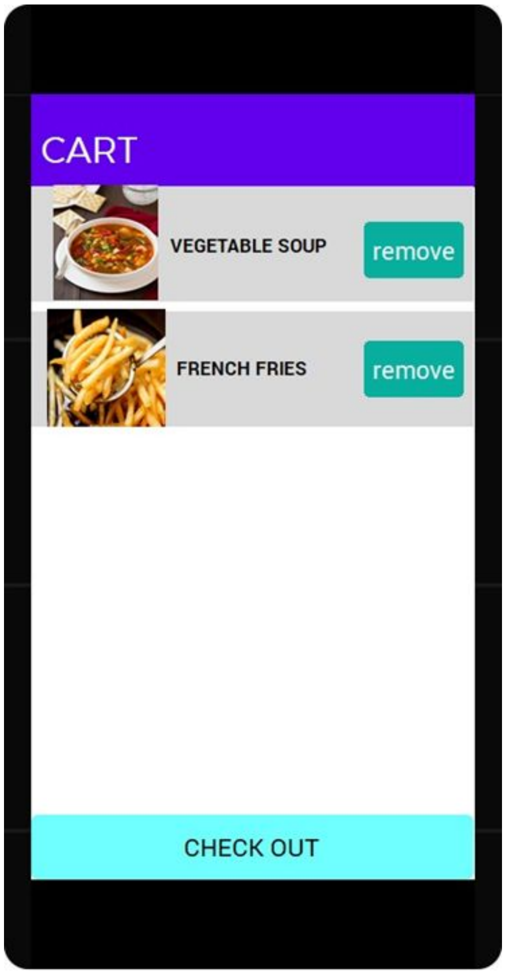


Figure 7.7 The picture above show the interface of the customer shopping cart

Figure 7.8 shows the check out interface. This will appear when users click on the checkout button in cart interface. In this interface, it will show all the menu that customers want, price, delivery fee and also total payment.

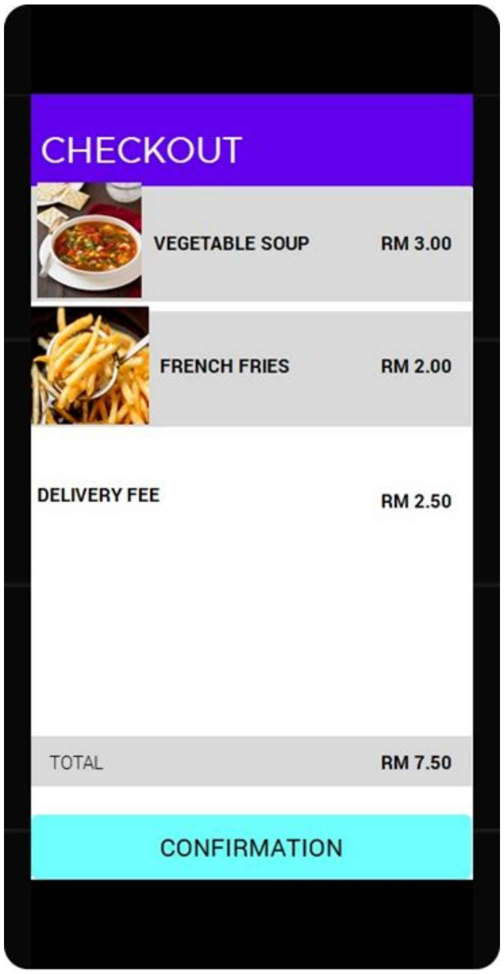


Figure 7.8 The picture above show the interface for the customer when they have select check out

Figure 7.9 shows the managed menu interface. This interface will appear when the user clicks on the manage menu in the main interface. In this interface users (retailers) can decide whether the menu is available or not at the moment. They also can delete the menu and add a new menu.

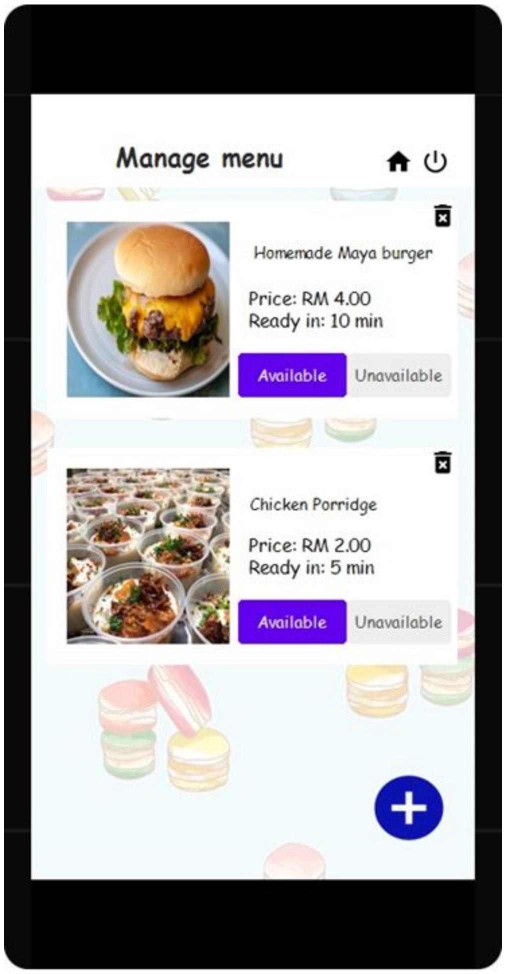


Figure 7.9 The picture above show the interface for the retailer about the status of their own menu

Figure 7.10 shows the add menu interface. This interface will appear when the user clicks on the manage menu in the main interface and clicks the add icon. Users can add food photos, name, price and decide when it is ready.

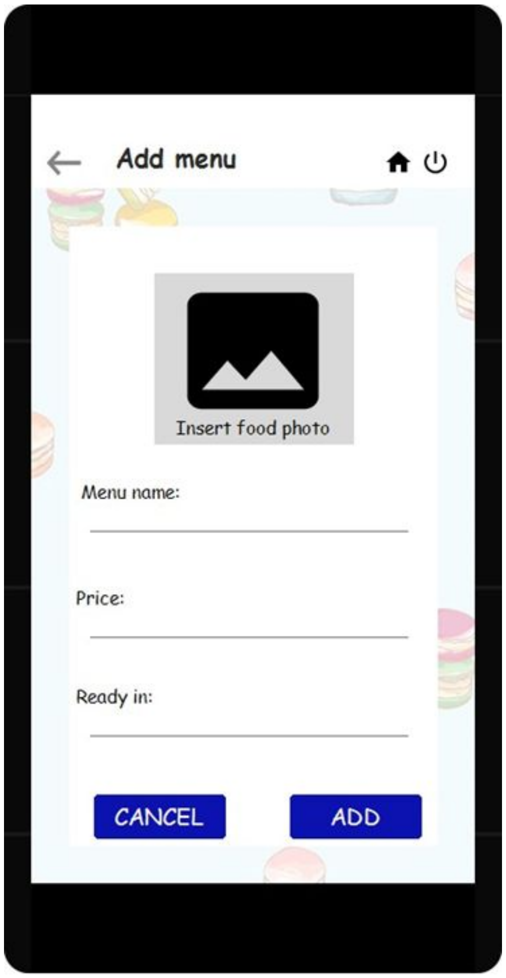


Figure 7.10 The picture above show the interface for the retailer to add new menu

Figure 7.11 shows the order interface.This interface will appear when the user (retailer) clicks the check order in the main interface. Once retailers get an order, they can decide to accept or reject the order. And they also need to give order status to customers.

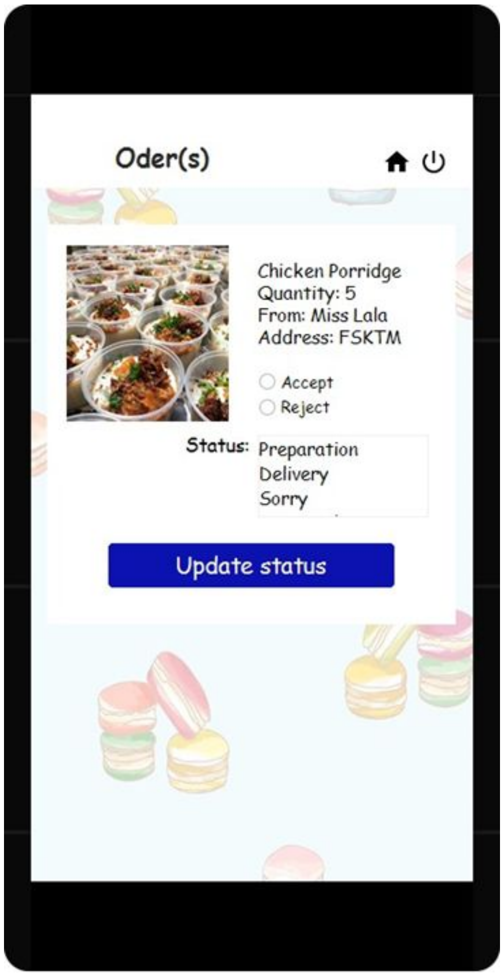


Figure 7.11 The picture above show the order interface when customer have order the food

9.0 BIBLIOGRAPHY