VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Software Engineering

Report

Point of Sale

Lecturer: Assoc. Prof. Quan Thanh Tho
TA: Bùi Công Tuấn
Student:

Vũ Khánh Hưng – 1910232
Ngô Minh Hồng Thái – 1912046
Nguyễn Văn Vinh Quang – 1911907
Vũ Nguyễn Minh Huy – 1952733
Tô Thanh Phong – 1914637

Ho Chi Minh, September 2021

Contents

1 Changelog

No.	Date	Changes	Actor
5	24 / 10 / 2021	"Section 3.5.8 Update menu" updated. "Section 3.5.9 Process order" updated. "Section 3.6.3 Update menu" updated. "Section 3.6.4 Process order" updated. "Section 3.7.5 Update menu" updated. "Section 3.7.6 Process order" updated. "Section 4.1.2 Advantage" added. "Section 4.1.3 Disadvantage" added	Tô Thanh Phong
		"Section 3.5.7 View information" updated. "Section 3.7.2 View information" updated. "Section 4.1.1 SPA introduction" added.	Nguyễn Văn Vinh Quang
		"Section 3.5.1 Login account" updated. "Section 3.5.2 Register account" updated. "Section 3.5.3 Retrieve password account" updated. "Section 3.7.1 Login" updated.	Vũ Khánh Hưng
		"Section 3.5.6 Manage acocunt" updated. "Section 3.7.3 Account management" updated. "Section 4.2 Component diagram" added. "Section 4.3 Deployment diagram" added.	Vũ Nguyễn Minh Huy
		"Section 3.5.4 Place an order" updated. "Section 3.5.5 Pay for order" updated. "Section 3.6.1 Login" updated. "Section 3.6.2 Order and pay" updated. "Section 3.6.5 View transaction history. "Section 3.6.6 View order history" updated. "Section 3.6.7 Feedback" updated. "Section 3.6.8 Update profile" updated. "Section 3.6.9 View and management profile" updated. "Section 3.7.4 Order and pay" updated.	Ngô Minh Hồng Thái

No.	Date	Changes	Actor
4	22 / 10 / 2021	"Section 3.4 Activity diagram" updated.	Nguyễn Văn Vinh Quang
3	10 / 10 / 2021	"Section 3.4 Activitity diagram" added. "Section 3.5.7 View information" added. "Section 3.6.6 View information" added. "Section 2.2 Scope of project" updated. "Section 3.7 Class detail" edited.	Nguyễn Văn Vinh Quang
		"Section 3.5.1 Login account" added. "Section 3.5.2 Register account" added. "Section 3.5.3 Retrieve password account" added. "Section 3.6.1 Login" added. "Section 3.7 Class detail" edited.	Vũ Khánh Hưng
		"Section 3.5.4 Place an order" added. "Section 3.5.5 Pay for order" added. "Section 3.6.2 Place an order" added. "Section 3.6.3 Pay for order" added. "Section 3.7 Class detail" edited.	Ngô Minh Hồng Thái
		"Section 3.5.6 Manage account" added. "Section 3.6.4 Manage account" added. "Section 3.6.8 User generalization" added. "Section 3.7 Class detail" edited.	Vũ Nguyễn Minh Huy
		"Section 3.5.8 Update menu" added. "Section 3.5.9 Process order" added. "Section 3.6.4 Update menu" edited. "Section 3.6.5 Process order" edited. "Section 3.7 Class detail" edited.	Tô Thanh Phong
2	25 / 09 / 2021	"Section 3.3.2.h Account management" added.	Vũ Nguyễn Minh Huy
		"Section 3.3.1 General use case" added. "Section 3.3.2.f Manage menu" added. "Section 3.3.2.g Process order" added.	Tô Thanh Phong
1	24 / 09 / 2021	"Section 3.1 Function" added. "Section 3.2 Non-functional" added. "Section 3.3.2.g Profile management" added.	Vũ Nguyễn Minh Huy
		"Section 3.2.2.b Place an order" added. "Section 3.2.2.c Feed back" added. "Section 3.2.2.d Pay for order" added.	Ngô Minh Hồng Thái
		"Section 2.3 Stakeholder" added. "Section 3.2.2.e Manage order history" added.	Nguyễn Văn Vinh Quang
		"Section 2.1 Intro" added. "Section 2.2 Scope of project" added. "Section 3.2.2.a Login" added	Vũ Khánh Hưng

2 Introduction

2.1 Intro

A point-of-sale (POS) system is a popular tool for brick-and-mortar businesses. They've replaced the old-school cash register with a more sophisticated, tech-forward approach to the checkout process. Especially during the COVID-19 pandemic when all transactions need to reduce human interactions as much as possible to avoid the spread, technological devices play an important role in saving time in making purchase and sale transactions, reducing errors in the payment process, and updating the restaurant's latest information quickly without face-to-face meetings.

More specifically for restaurants, POS is even more important as it can serve a large number of customers at the same time which can reduce the workload for the staff. Typically, restaurant POS systems include table reservation, ordering food, alerts, billing, credit card processing and customer management.

2.2 Scope of project

	FastFood
Example	Circle K, Ministop, cafe
Customer	All, mostly young people
Payment	Pay first with credit card, visa. Option cash.
Food	Use-now food, option: instance food.
Role	Registered customer, guest, clerk and administrator
Table	None
Business process	1. Customer comes to the restaurant, chooses a table to sit or go directly to the counter. 2. The customer scans QR code to access the restaurant's website. 3. The customer selects and orders food. 4. The customer pays for the order. 5. The kitchen manager prepares food for customer's order. The clerk may wrap the food when the food is ready. 6. The customers takes food at the counter.

2.3 Stakeholder

Stakeholder	**Description**
Customer	Customers manage their account, select food, pay for the order and follow the order, after using service customer can send feedback, view their transaction history.
Clerk	Clerks manage his account, refund money for customer if their order denied by kitchen, update menu on system, view statistics of their restaurant.
Administrator	Administrators manage clerk account, customer account and restaurant information.
Kitchen manager	Kitchen manager manage his account, proccess order from customer, can view food they have to cook and confirm after cooking.

3 Functional requirement

3.1 Function

1. Login:
   - Customers may login or create a new account if they want.
   - Clerks, kitchen managers and registered customers can regain their password via email if they forget.
2. Place order:
   - Customers can select food from the menu to order.
3. Pay for order:
   - Customer pays for the order they have placed.
4. Feed back:
   - Registered customer can post feedback, rate the restaurant.
   - Guest customerand clerk can view the feedback.
5. Manage order history:
   - Registered customers can view their transaction history including order, price, ...
   - Clerks can view the order history of their restaurant in statistics.

6. Manage account:
   - Administrators can manage customers, clerks, and restaurant information.
   - Registered customers, kitchen managers and clerks can manage their personal profile.
   - Clerks can also manage their associated restaurant.
7. View order status:
   - Customer can track personal order status
8. Process order:
   - Clerk and kitchen manager can involve in processing order in order to make it complete.
9. Manage menu:
   - Clerk can adds, deletes, updates items in the menu.

3.2 Non-Functional

Product requirement	• All functionalities of the system must behave with no crashes up to 300 orders. • Response delay must not exceed 1 second.
Organization requirement	• Users can access the system by any browsers. • System serves one specific restaurant with many branches. • System must support all functionality of the Food take away service. • System is accessed through Web technology and QR code. • Users using mobile, tablet or PCs can access the system and be supported with all the same functionalities. • Users using the service interact only with the system and can be served end-to-end at the restaurants.

3.3 Use case diagram

3.3.1 General use case

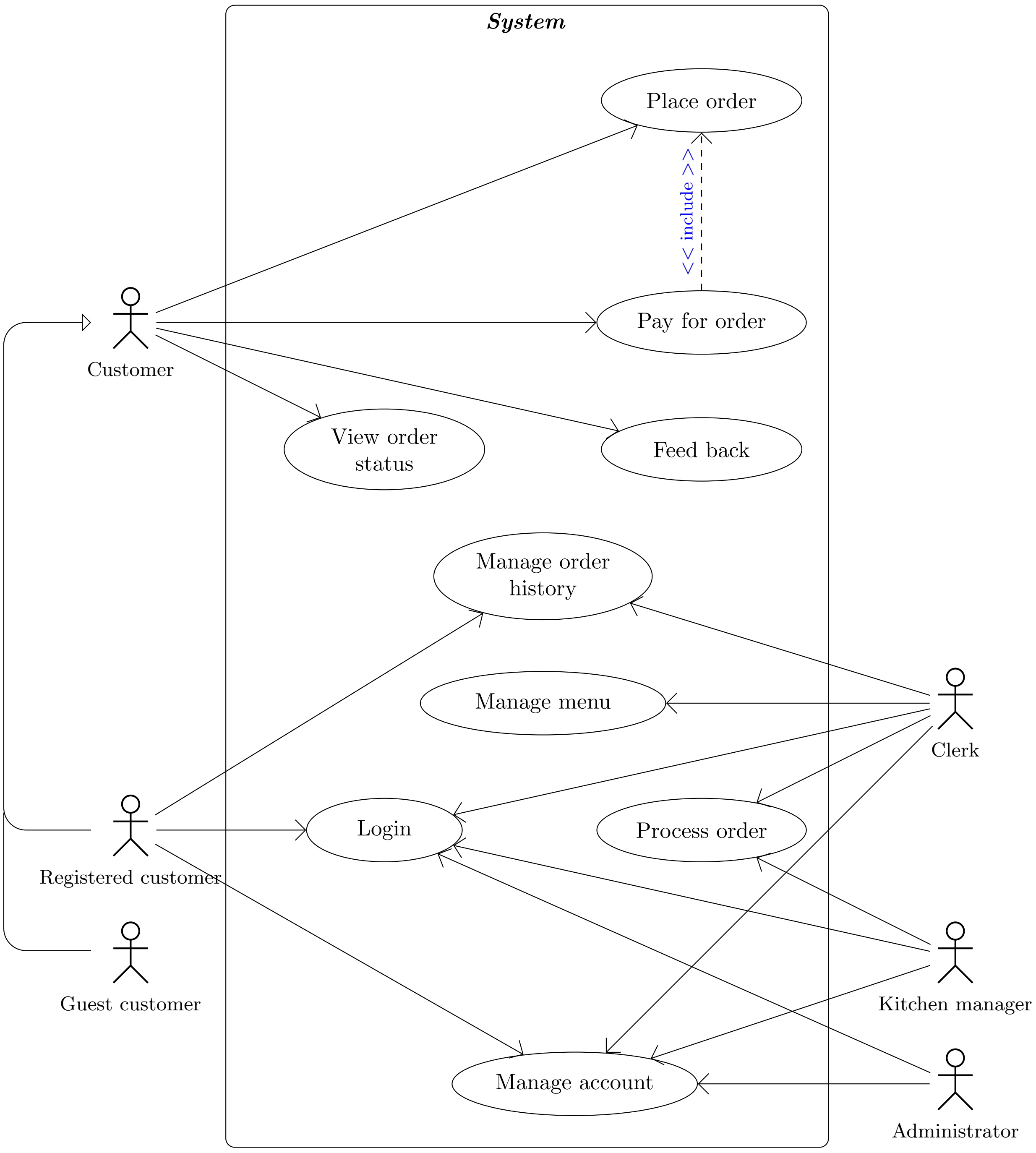


Figure 1: General use case

3.3.2 Use case description

3.3.2.1 Login

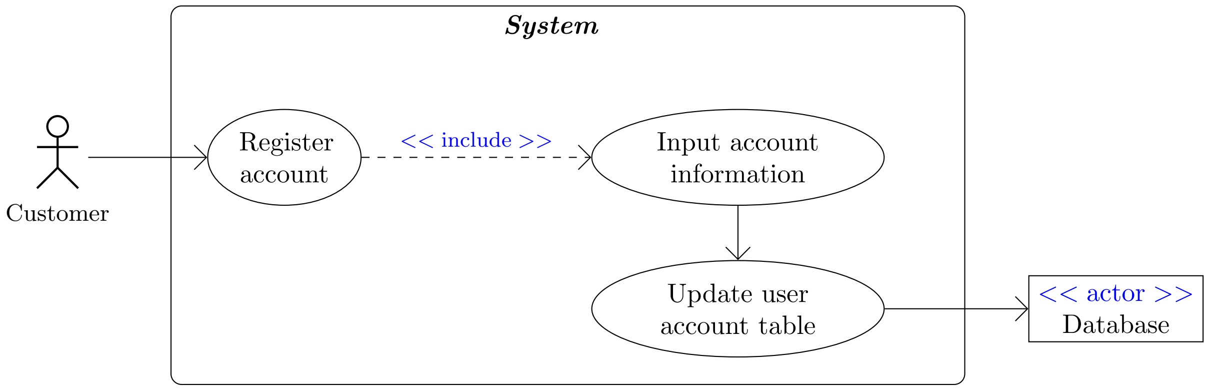


Figure 2: Register account use case

Name	Register account
Actor	Customer
Description	With register, Customer must input some relevant personal information, including username, password, repeat password, email, phone number.
Precondition	Users need to access to home page by scanning QR code.
Action	1. Users go to home page of website by scanning QR code. 2. Users click ***Register*** 3. Users input username, password, repeat password, email, phone number 4. Users click the button ***Register***
Exception	Exception at step 3: If password not matches with repeat password, alern by text ***Your password is not same as repeat password.*** If any field is empty, alern by ***You must be fill in all field***, which make the register unsuccessfully
Alternative flow	At step 2, user can click ***Login*** then click to ***Don't have an account*** to access the register link

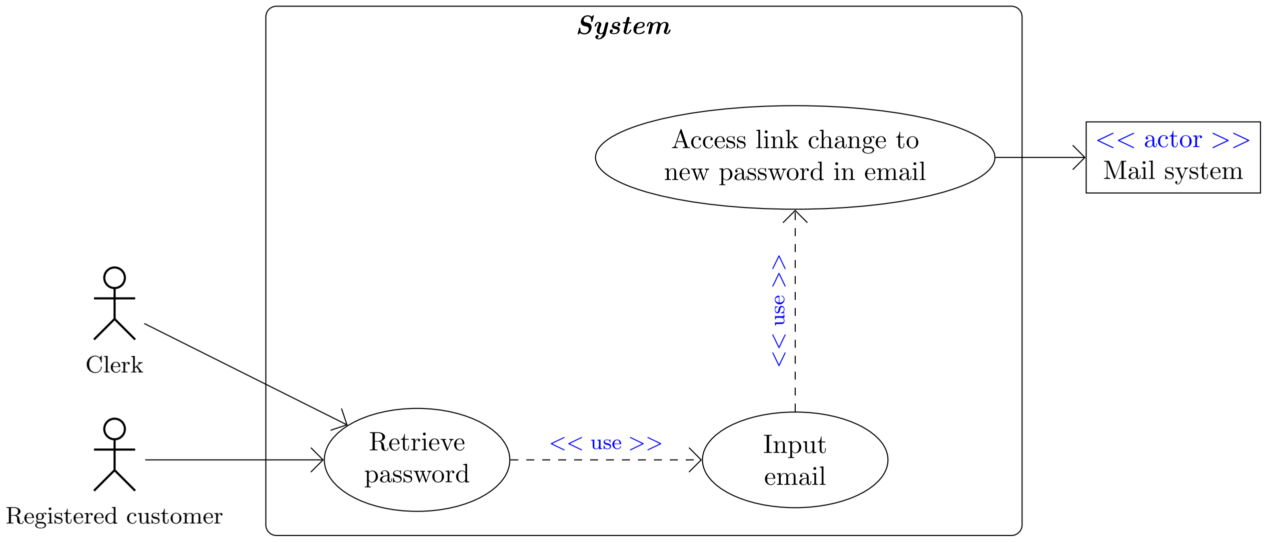


Figure 3: Retrieve password use case

Name	Retrieve password
Actor	Registered customer, Clerk
Description	With doing retrieve password, Registered customer, Clerk will change to new password by link showed in email.
Precondition	Users need to access to login page.
Action	1. Users go to login page. 2. Users click ***Forget password*** 3. Users input user's email. 4. Users click the button ***Give password*** 5. Users change to new password by the link showed in email.
Exception	Exception at step 5: User's email is not valid.
Alternative flow	None

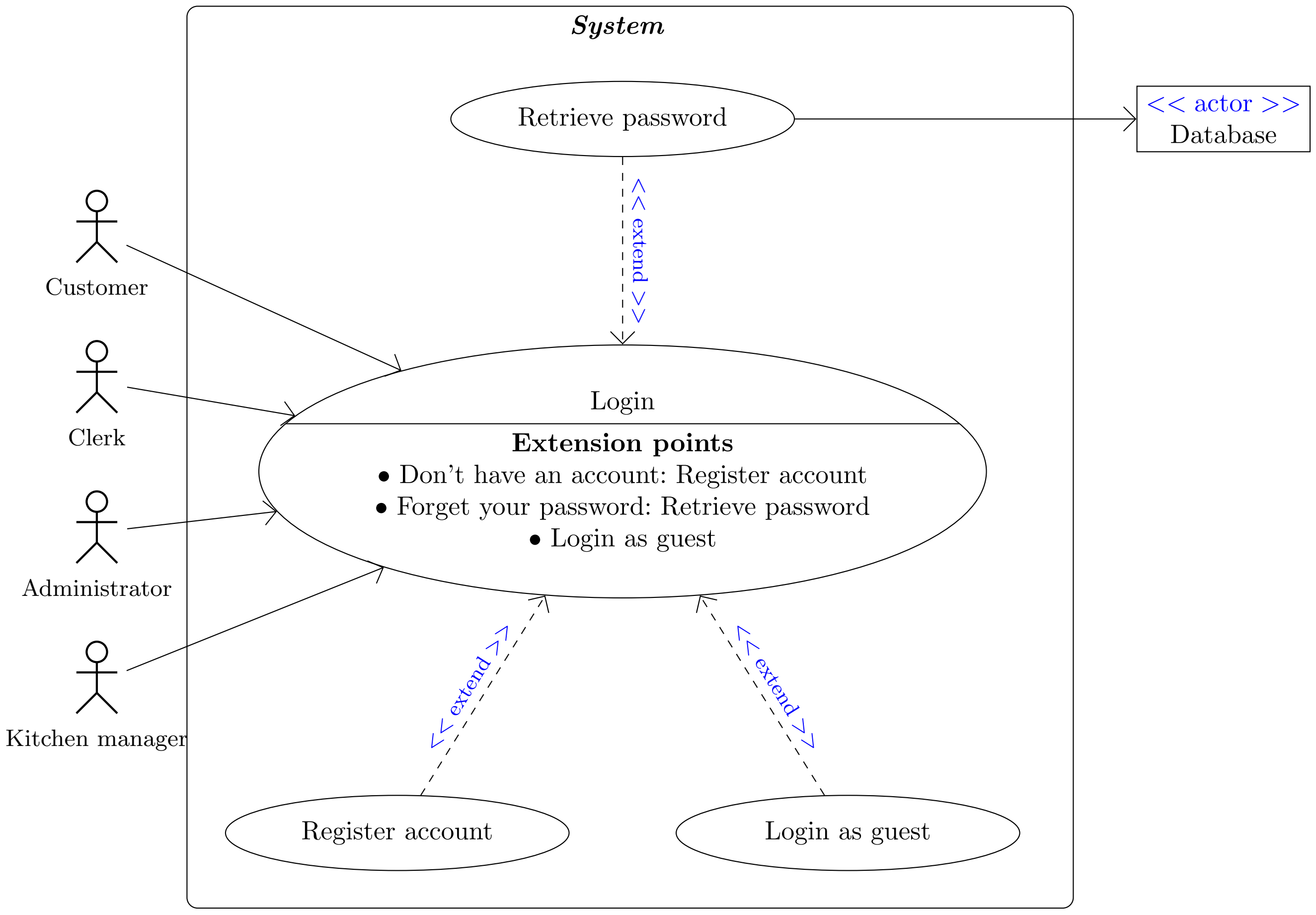


Figure 4: Login use case

Name	Login
Actor	Customer, Clerk, Kitchen manager and Administrator
Description	With login, Customers, Clerks, Kitchen managers and Administrators login by enter the email and password. In addition, a user who does not have an account can click ***Don't have an account*** to register an account or a user who forgets the password can click ***Forget password*** to retrieve password from user's email. If users don't want to login, users can click ***Login as guest***.
Precondition	Users need to access to home page by scanning QR code.
Action	1. Users click ***Login***. 2. Users input the email and password. 3. Users click the button ***Login***
Exception	Exception at step 3: If users input wrong password or an email, alert ***Your password or email is wrong***
Alternative flow	[**New users**] At step 3, users can click ***Don't have an account*** to register [**User forget password**] At step 3, users can click ***Forget password*** to retrieve password [**User don't want to login**] At step 3, users can click ***Login as guest*** to login

3.3.2.2 Place an order

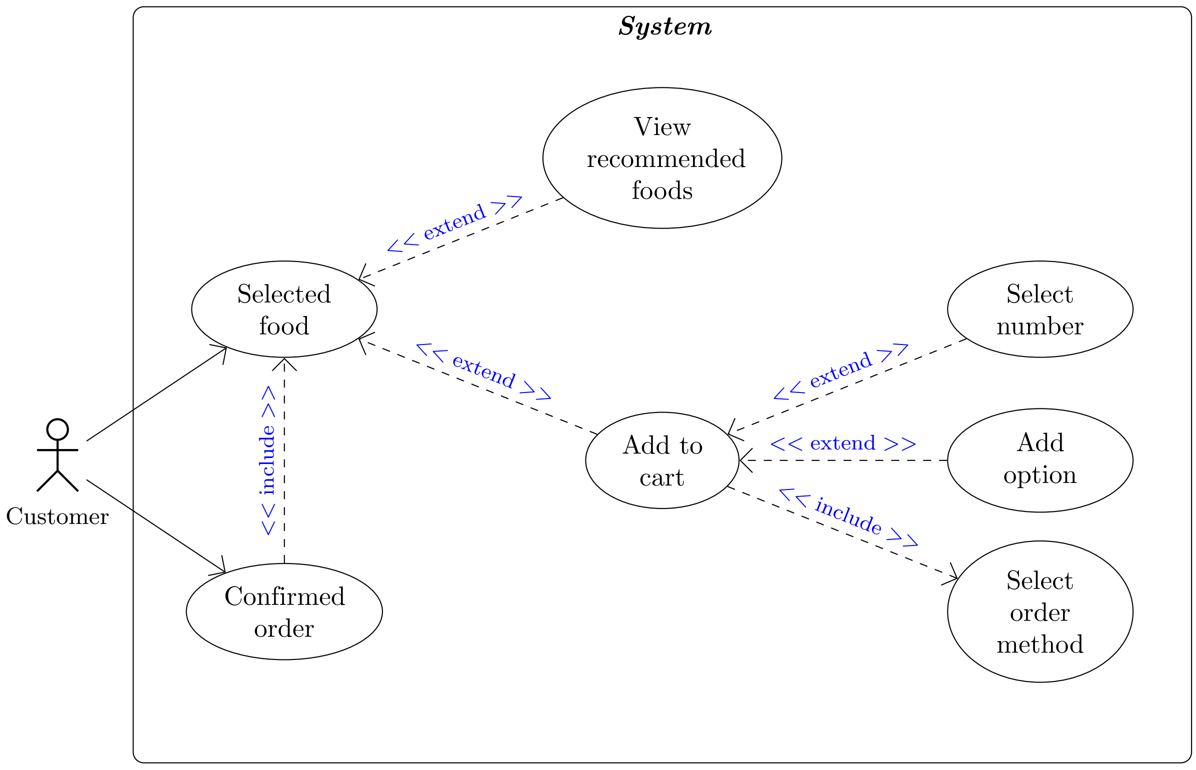


Figure 5: Place an order use case

Name	Place an order
Actor	Customer
Description	Customers can select food from the menu to order.
Precondition	Customer need to access the menu page.
Action	1. Customers skim the menu and can see recommended foods on the menu. 2. The customer selects the foods that he/she wants to add to the cart.When choosing foods, customers must select order method (take away or eat-in). In addition, when choosing foods, customers can choose the quantity and the options that go with the dish. 3. The customer confirms the order. 4. System confirms the order.
Exception	Exception 1: at step 3: If the customer does not add any foods to the cart, the order will be cancelled. Exception 2: at step 3: If there are not enough ingredients to make the food, the order will be cancelled.
Alternative flow	Alternative 1: at exception 1: The system notify that the cart is empty and redirect the user to the menu page. Alternative 2: at exception 2: The system notify that there are not enough ingredients to make certain foods in the order and remove those foods from the cart. The customer is then redirected to the menu page. Alternative 3: at step 3: The customer can return to the menu page to choose more foods for their order.

3.3.2.3 Feed back

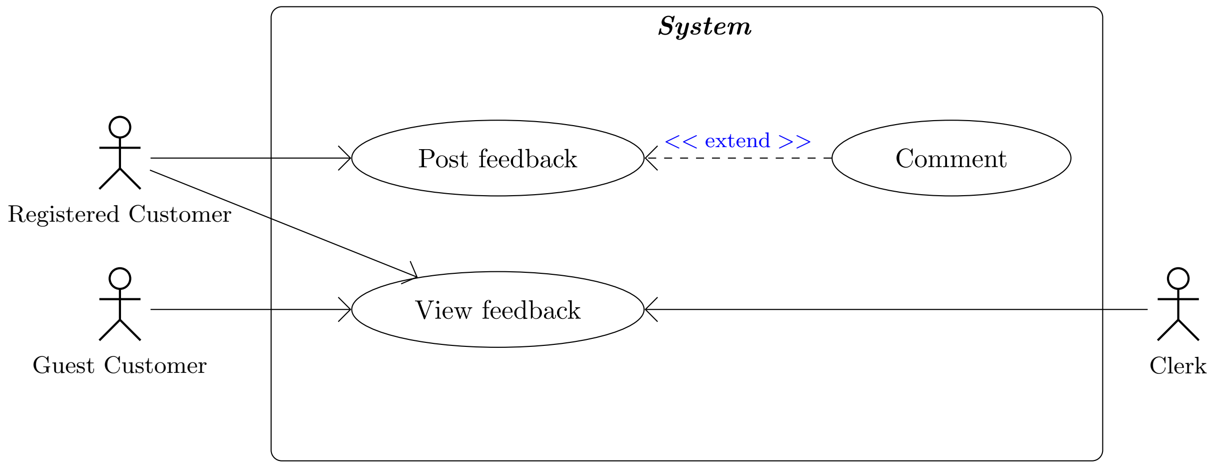


Figure 6: Feed back use case

Name	Feed back
Actor	Registered customer, guest customer, and clerk.
Description	Registered customer can post feedback, rate the restaurant. Guest customer and clerk can view the feedback.
Precondition	Customer need to have an account and login first. Also, the customer must be on the feedback page.
Action	[**Registered customer**] 1. Registered customer can view other feedback, rate with stars (up to 5 stars), and add comment about the restaurant. 2. Registered customer select the ***Post*** button to post the feedback. [**Guest customer and clerk**] 1. Guest customer and clerk can browse the feedback page to see the rates and comments.
Exception	[**Registered customer**] Exception 1: at step 2: If the customer does not press the ***Post*** button, the feedback cannot be posted.
Alternative flow	[**Registered customer**] Alternative 1: exception 1: If the customer leaves the feedback page, the system will notify the customer that the feedback was not posted.

3.3.2.4 Pay for order

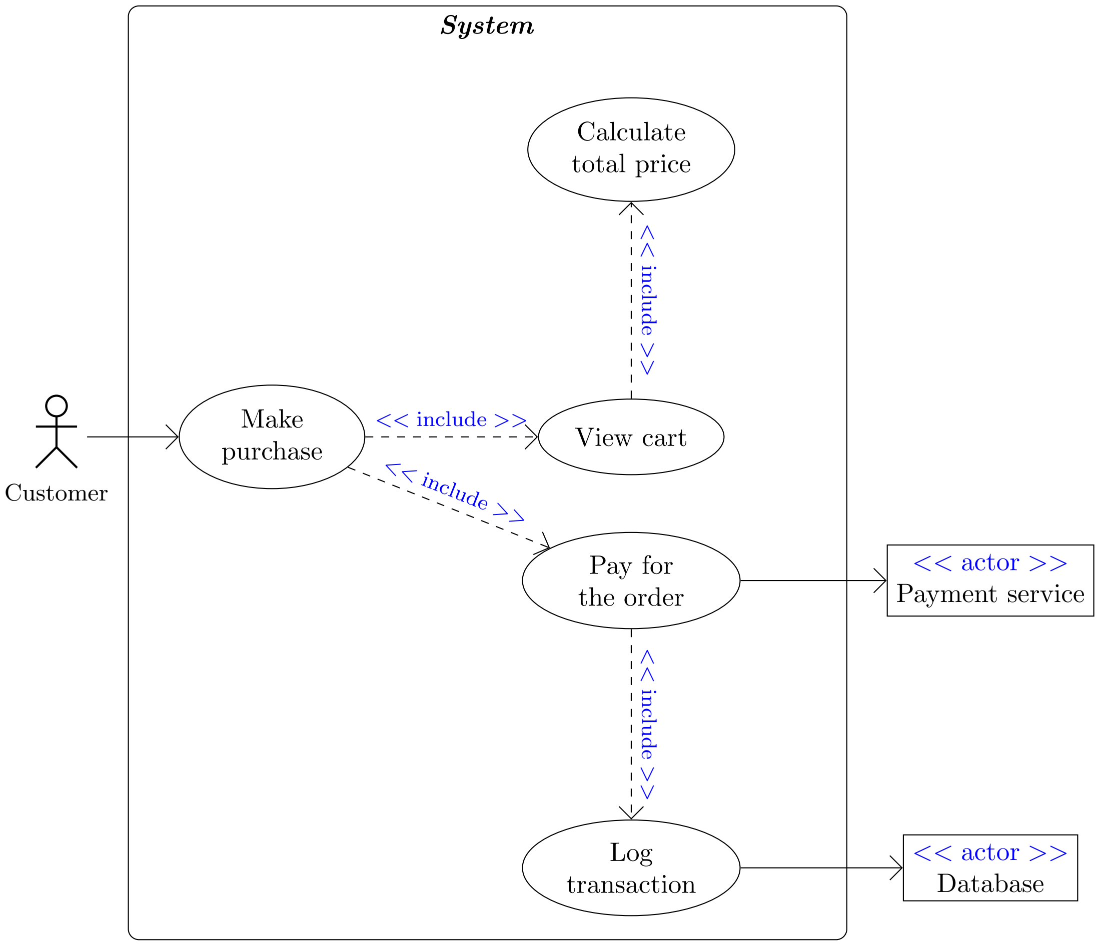


Figure 7: Pay for order use case

Name	Pay for order
Actor	Customer, payment service, and database.
Description	Customer pays for the order they have placed.
Precondition	Customer must place an order before pay for it.
Action	1. Customer can see the list of foods in the cart, see the total amount to pay for the order and select ***Payment*** button. 2. Customer selects the appropriate payment method and makes the payment with the Payment service. 3. The system records transaction information into the database.
Exception	Exception 1: at step 2: Customer enters wrong information, causing errors in the payment process, then the system will notify the customer that the transaction is canceled due to wrong information. Exception 2: at step 2: Customer does not have enough money for that payment method, the system will notify that the transaction is canceled due to insufficient payment.
Alternative flow	Alternative 1: at step 1: Customer can cancel order payment and return to menu page. Alternative 2: at exception 1 and 2: 2a. Customer can cancel order payment and return to menu page. 2b. Customer can select another payment method and continue step 2.

3.3.2.5 View history

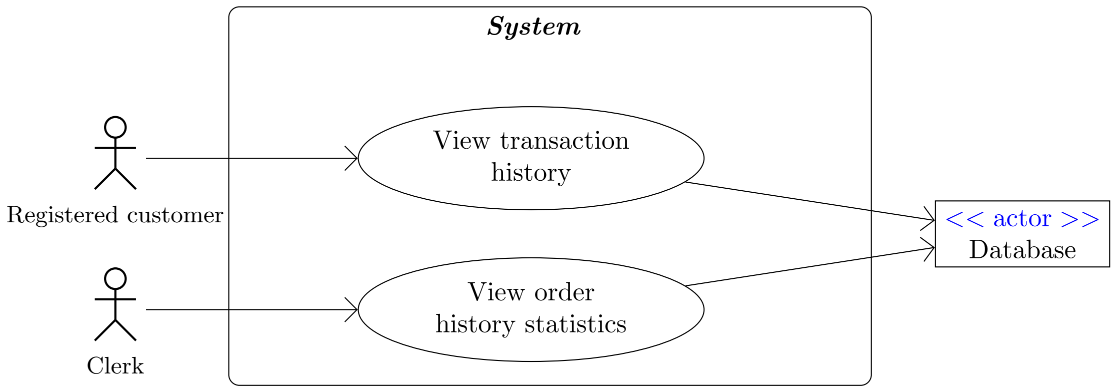


Figure 8: View history use case

Name	View transaction history
Actor	Registered customer
Description	Registered customers can use the history interface to view their transaction history
Precondition	Customers have to login
Action	1. Customers move to history interface 2. Database retrieve transaction history to customer 3. Customer check their transaction history
Exception	None
Alternative flow	None

Name	View order history statistics
Actor	Clerk
Description	Clerk can view his restaurant order history statistics in an interval.
Precondition	Clerk have to login
Action	1. Clerk move to statistics interface 2. Clerk choose button begin date and end date to view statistics
Exception	Exception at step 2: If clerk choose the begin date before the date they begin or the end date exceed current date, alert by text "The date you choose is not validate"
Alternative flow	None

3.3.2.6 Manage menu

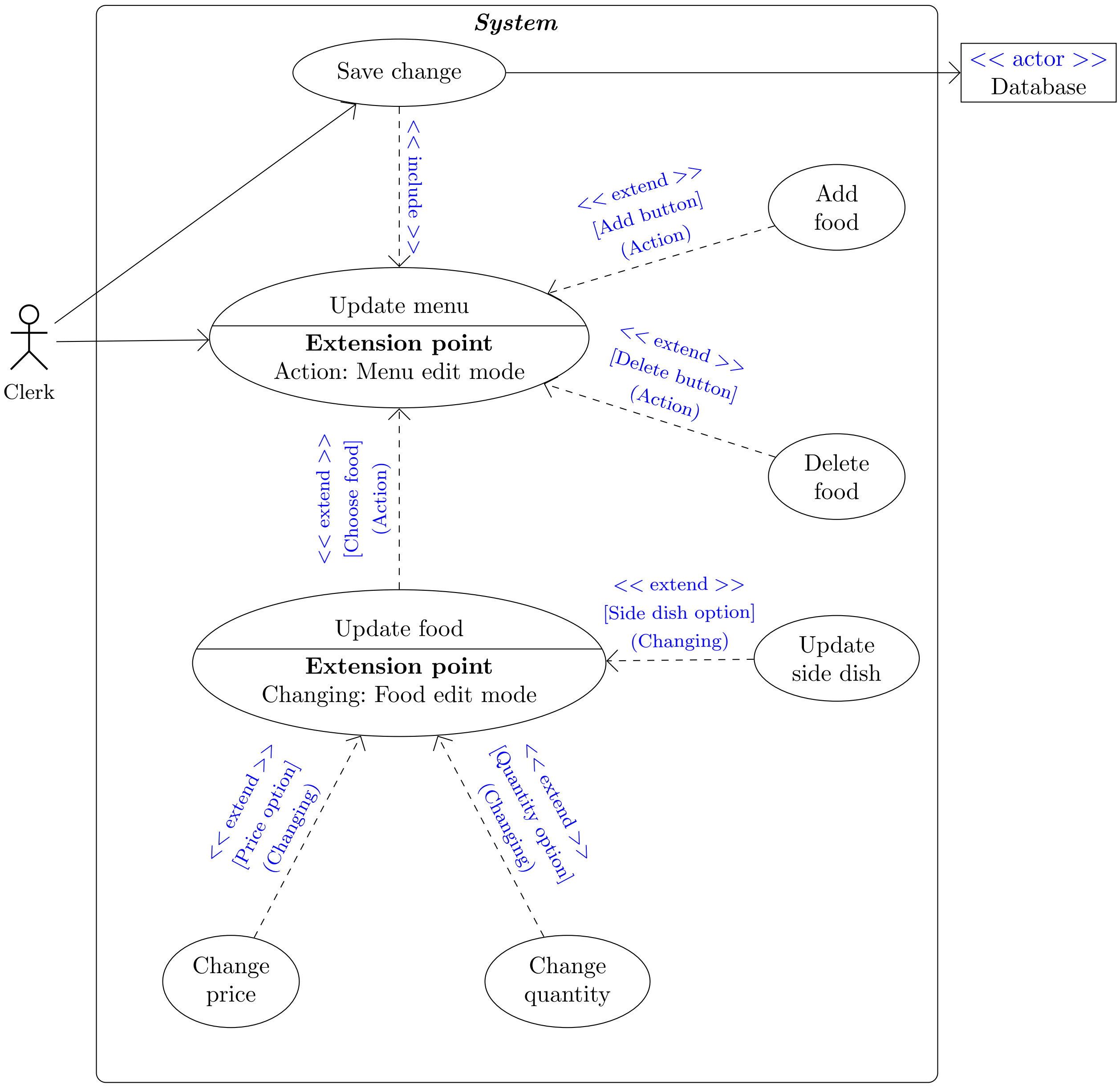


Figure 9: Manage menu use case

Name	Manage menu
Actor	Clerk and database
Description	Clerk can add new foods, delete old foods and update attributes of current foods such as price, quantity, side dish. This changes are occured to the menu of restaurant, which is viewed by customers
Precondition	Clerk must login to their account first, then access to the edit mode of menu
Action	1. Clerk can perform tasks such as add new foods, delete old foods, update attribute of current foods (price, quantity, side dish). 2. After changing, clerk must save the changed infomation of menu. 3. System saves information to database.
Exception	Exception 1: at step 1: When changing the menu, if clerk hasn't specified all compulsory information or used invalid information, clerk can't make the change. Exception 2: at step 2: If the clerk did not save, new information would not be saved to the database.
Alternative flow	Alternative 1: at step 2: Clerk can click "cancel" button if clerk doesn't need to change the menu anymore. Alternative 2: at exception 1: System notify the clerk. Alternative 3: at exception 2: System notify the clerk then clerk can save or leave without saving.

3.3.2.7 Process order

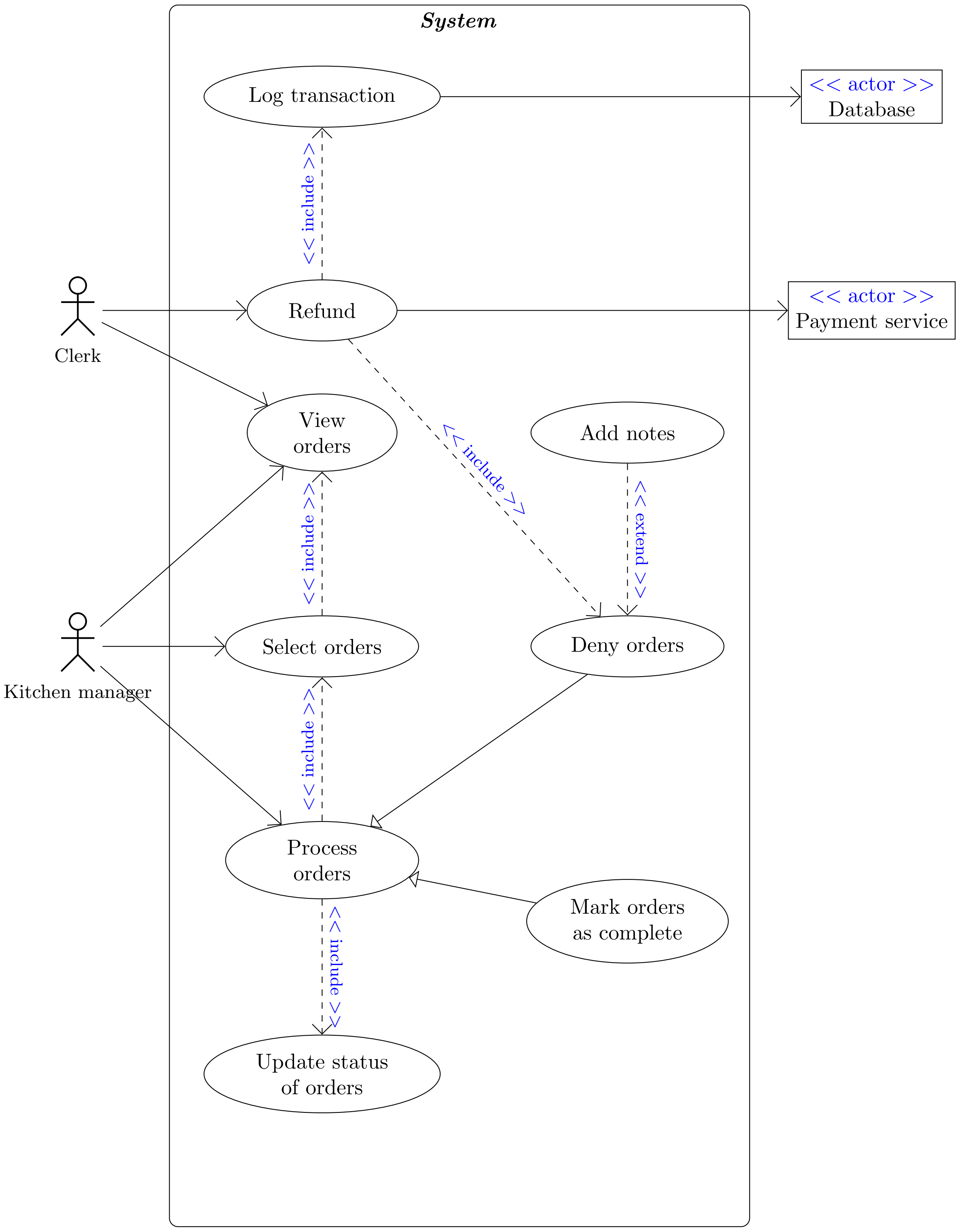


Figure 10: Process order use case

Name	Process order
Actor	Kitchen manager, clerk
Description	Kitchen manager and clerk can view all orders of customers. Kitchen manager can choose to accept or deny orders, then completes accepted ones. If kitchen manager denies orders for a few reasons, clerk will refund the customer.
Precondition	Kitchen manager and clerk must login, the orders have been paid
Action	[**Kitchen manager**] 1. Kitchen manager views all customer's orders. 2. Kitchen manager selects orders to process. 3. Kitchen manager accepts or denies those orders and can add reason for denying them. 4. After completing accpeted orders, kitchen manager must mark them as complete. 5. System updates status of orders. [**Clerk**] 1. Clerk can view all customner's orders. 2. If a order is denied, clerk must refund customer the price of that order. 3. The system logs transaction information into the database.
Exception	None
Alternative flow	None

3.3.2.8 Manage account

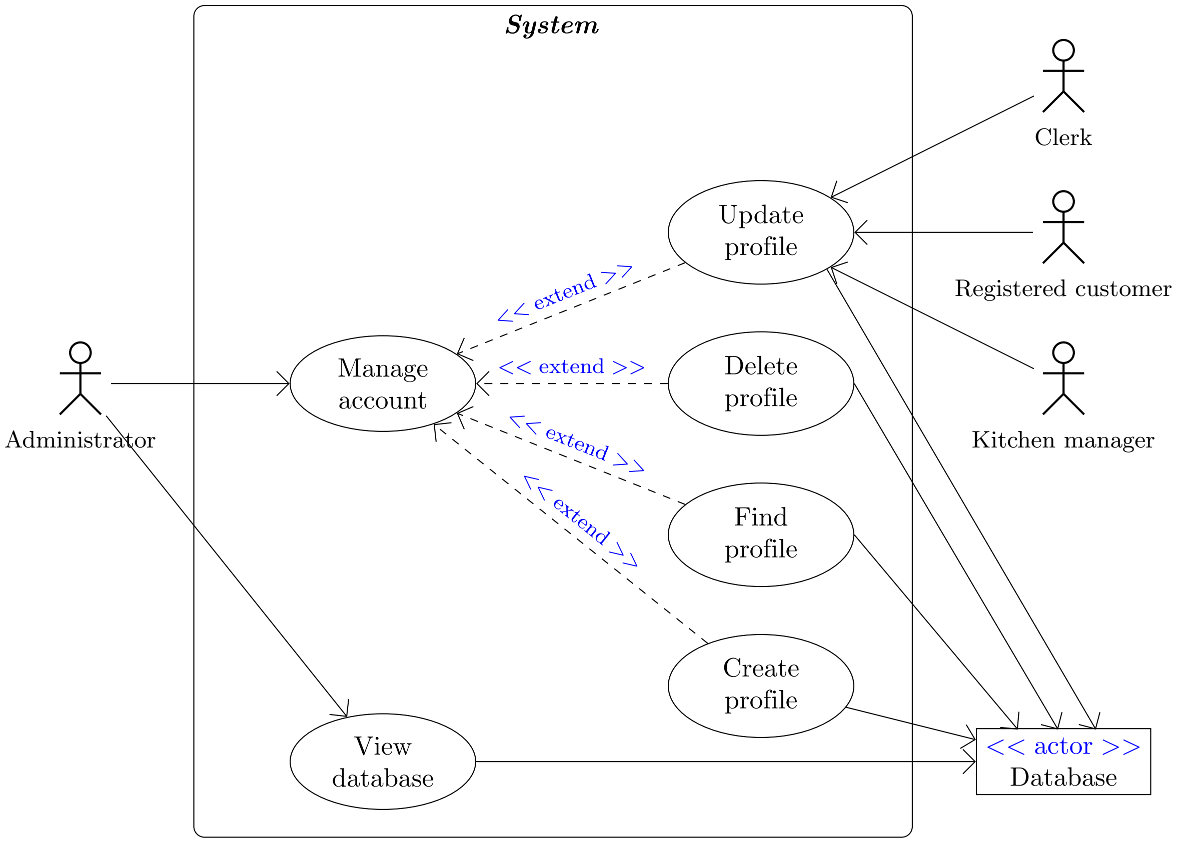


Figure 11: Manage account use case

Name	Manage account
Actor	Administrator, clerk, registered customers, kitchen manager and database
Description	Clerk, registered customers and kitchen manager can manage their personal profile. Administrators can manage accounts of all users and databases.
Precondition	Clerk, kitchen manager, customer and administrator have logined
Action	[**Clerk, kitchen manager and customer**] 1. Clerk, kitchen manager and customer update their personal profile. Clerk updates their restaurants information. Editing information is updated to the database. [**Administrator**] 1. Administrator can search for a user, update, create or delete a user. In the case of clerks, restaurants and kitchen managers, if an administrator creates, deletes one of them, the actor must do the same operation with others. Editing information is updated to the database. 2. Administrator can view database's information. Information is retrieving from the database.
Exception	None
Alternative flow	None

3.4 Activity diagram

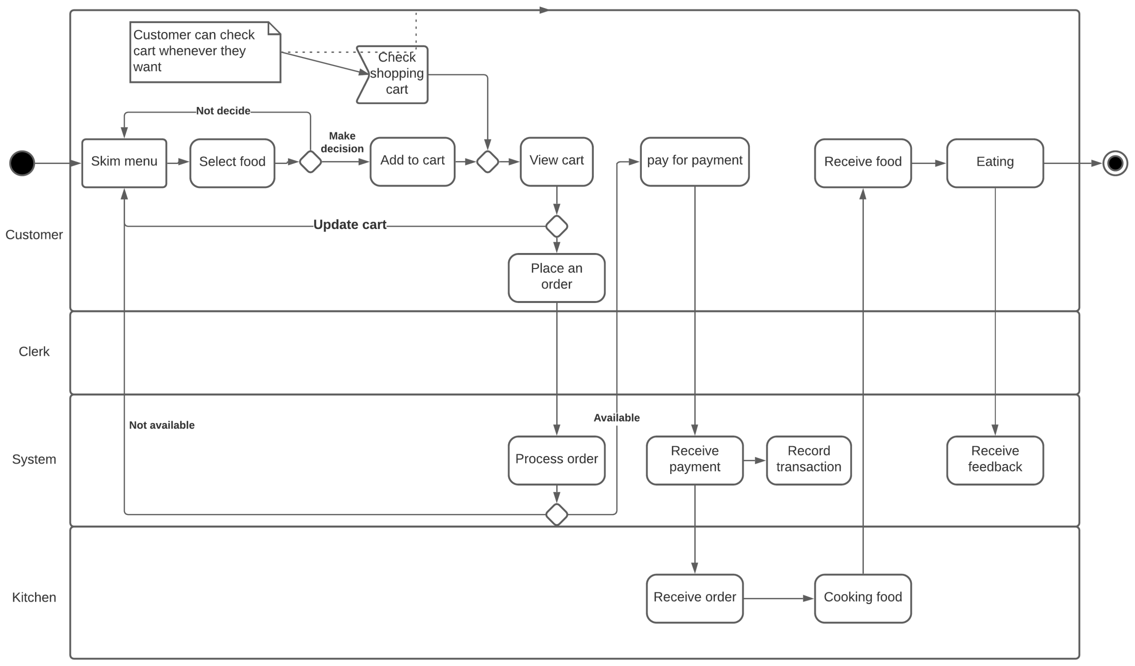


Figure 12: Activity diagram

3.5 Sequence diagram

3.5.1 Login account

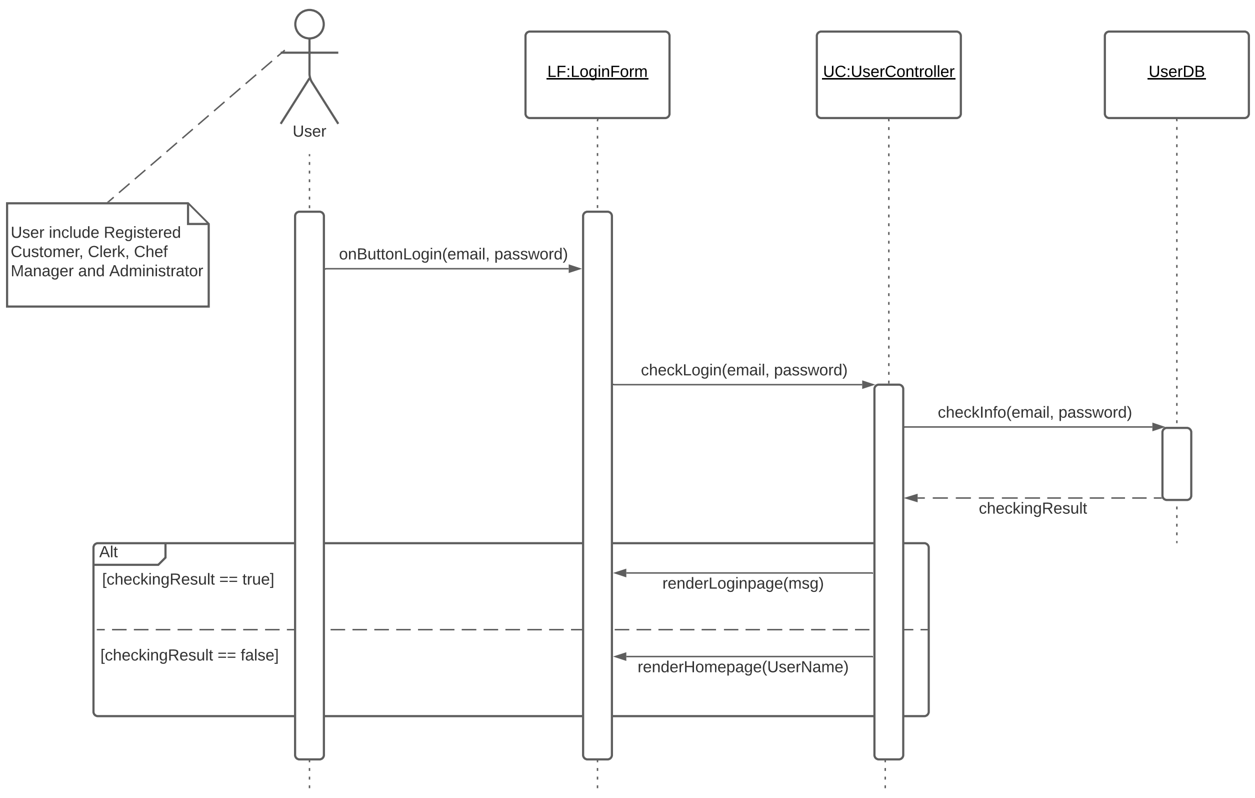

User
LF:LoginForm
UC:UserController
UserDB
User include Registered Customer, Clerk, Chef Manager and Administrator
onButtonLogin(email, password)
checkLogin(email, password)
checkInfo(email, password)
checkingResult
Alt
[checkingResult == true]
renderLoginpage(msg)
[checkingResult == false]
renderHomepage(UserName)

Figure 13: Login account's sequence diagram

Description:

1. `Registered Customer, Clerk, Chef Manager and Administrator` click ***SUBMIT*** button, which called `onButtonLogin` method of instance `LP`, providing with email, password.

2. `LoginPage`'s instance then calls `checkLogin` method of instance `UC`, providing with user email and password, which is checked the valid account or not.

3. `User Controller` process the checking task by calling `checkInfo` method of instance `UserInfo`, providing with user email and password. After, `UserInfo` calling `checkingResult` method of instance `UC` to announce the checking result.

4. If the checking result is "False", it mean invalid account, `User Controller` calling `renderLoginpage` method of instance `LP`, providing with the message about error during login. Other case, `UserController` calling `renderHomepage` method of instance `LP` providing with the name logined show in home page.

3.5.2 Register account

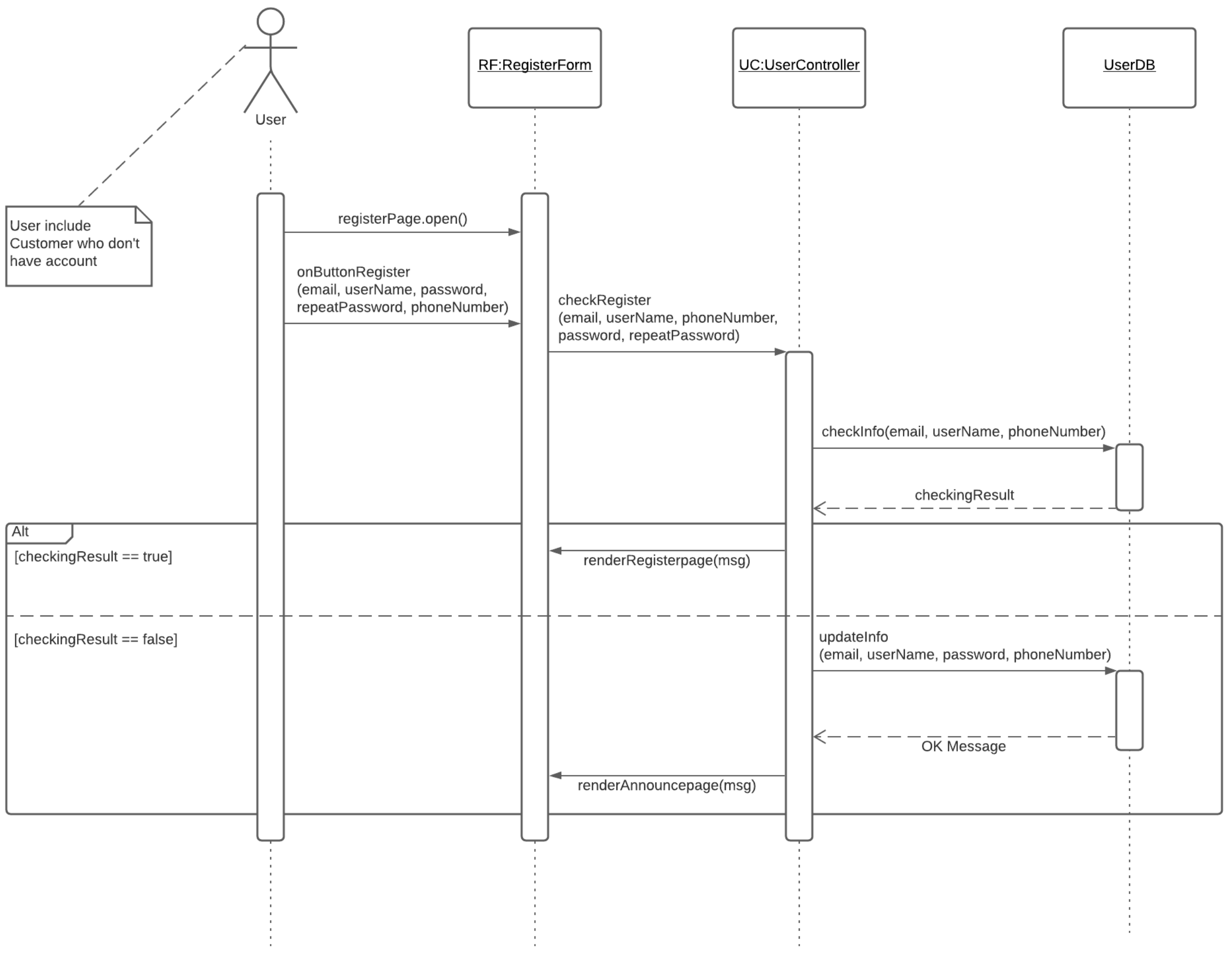

User
RF:RegisterForm
UC:UserController
UserDB
User include Customer who don't have account
registerPage.open()
onButtonRegister (email, userName, password, repeatPassword, phoneNumber)
checkRegister (email, userName, phoneNumber, password, repeatPassword)
checkInfo(email, userName, phoneNumber)
checkingResult
Alt
[checkingResult == true]
renderRegisterpage(msg)
[checkingResult == false]
updateInfo (email, userName, password, phoneNumber)
OK Message
renderAnnouncepage(msg)

Figure 14: Register account's sequence diagram

Description:

1. When `Registered Customer, Clerk, Chef Manager and Administrator` click to **REGISTER** in home page, `RegisterForm` will be called the method `registerPage.open` to show the UI register page to user.
2. `Registered Customer, Clerk, Chef Manager and Administrator` click ***SUBMIT*** button, which called `onButtonRegister` method of instance `RegisterForm`, providing with email, userName, password, RepeatPassword, phoneNumber information of user register.
3. `UserController`'s instance then calls `checkRegister` method of instance `UC`, providing with email, userName, password, RepeatPassword, phoneNumber information of user register, to check both the syntax of all field and check the existance of information user registered.
4. `UserController` process the checking existance task by calling `checkInfo` method of instance `UserInfo`, providing email, Username and phone Number to checking if one of all field is exist. After, `UserInfo` calling `checkingResult` method of instance `UC` to announce the checking result.
5. If the checking result is "False", it mean it have something wrong during register, `UserController` calling `renderRegisterpage` method of instance `RF`, providing with the message about error during register. Other case, `UserController` update the account as new account by calling `updateInfo` method of instance `UserInfo`, then calling `renderAnnouncePage` method of instance `RF` providing with the message announce to user register successfully.

3.5.3 Retrieve password account

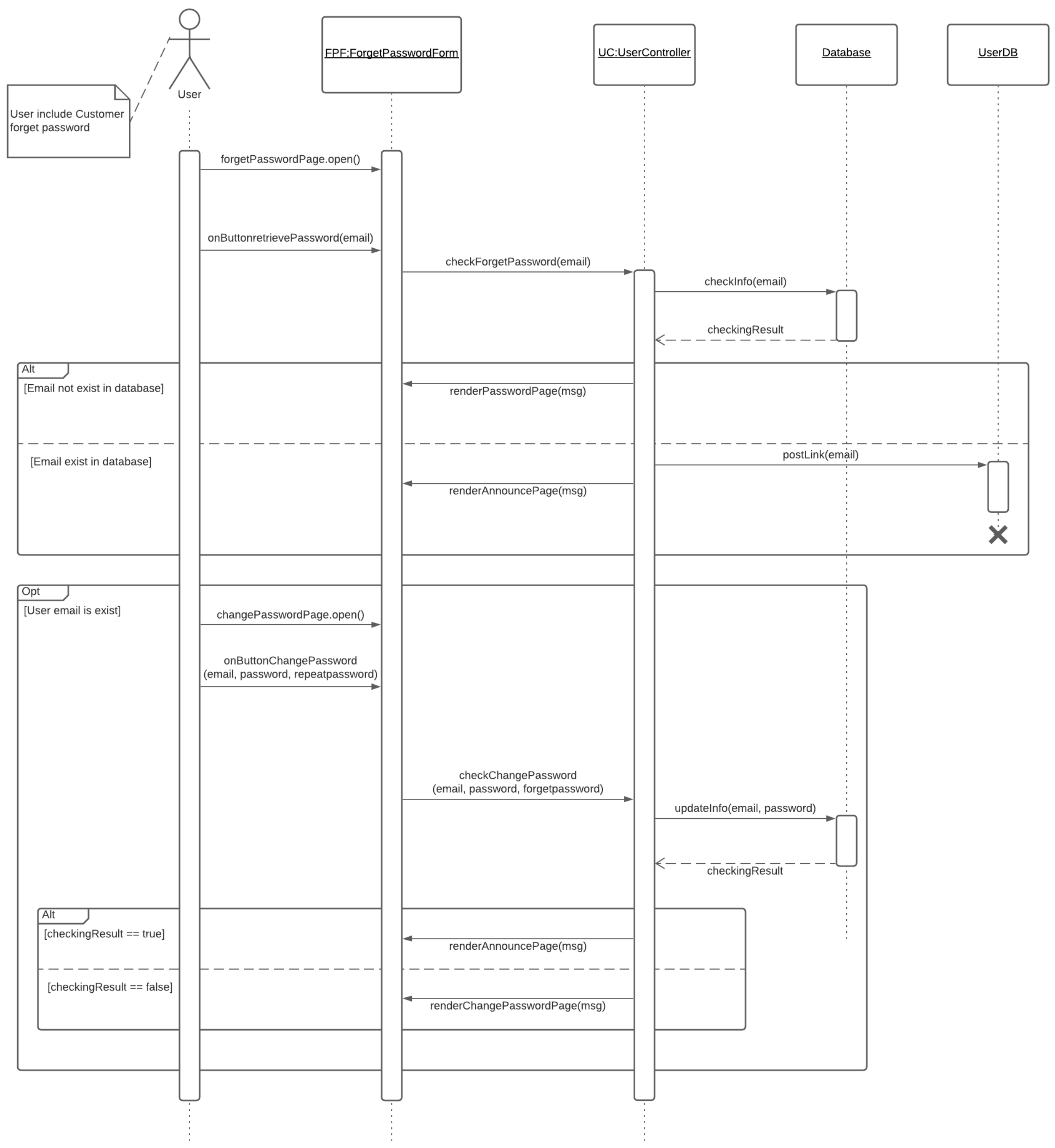

User include Customer forget password
User
FPF:ForgetPasswordForm
UC:UserController
Database
UserDB
forgetPasswordPage.open()
onButtonretrievePassword(email)
checkForgetPassword(email)
checkInfo(email)
checkingResult
Alt
[Email not exist in database]
renderPasswordPage(msg)
[Email exist in database]
postLink(email)
renderAnnouncePage(msg)
Opt
[User email is exist]
changePasswordPage.open()
onButtonChangePassword (email, password, repeatpassword)
checkChangePassword (email, password, forgetpassword)
updateInfo(email, password)
checkingResult
Alt
[checkingResult == true]
renderAnnouncePage(msg)
[checkingResult == false]
renderChangePasswordPage(msg)

Figure 15: Retrieve password account's sequence diagram

Description:

1. When `User` click to ***FORGET YOUR PASSWORD*** in Login page, `ForgetPasswordPage` will be called by User the method `forgetPasswordPage.open` to show the UI forget password page to user.

2. `User` click ***SUBMIT*** button, which called `onButtonretrievePassword` method of instance `FP`, providing with user email.

3. `UserController`'s instance then calls `checkForgetPassword` method of instance `UC`, providing with user email.

4. `UserController` process the checking task by calling `checkInfo` method of instance `UserInfo`, providing email to check the email is exist in restaurant's system or not. After, `Database` calling `checkingResult` method of instance `UC` to announce the checking result.

5. If the checking result is "Wrong", it mean email input not exist in database, `User Controller` calling `renderPasswordpage` method of instance `FPF`, providing with the message about error. Other case, `User Controller` calling `postLink` method of `Mail system` to send the link to user email and then calling `renderAnnouncePage` method of instance `FPF` providing with the message announce to user to access user's email to change password.

6. If user email is exist, user can be access to the change password link. When user clicked it, `ForgetPasswordForm` will be called by User the method `changePasswordPage.open` to show the UI change password page to user.

7. `User` click `SUBMIT` button, which called `onButtonretrievePassword` method of instance `FPF`, providing with user email, user password and repeatPassword.

8. `UserController`'s instance then calls `checkChangePassword` method of instance `UC`, providing with user email, user password and repeatPassword to check the valid syntax of password and the existance of user email.

9. `UserController` process the checking task by calling `updateInfo` method of instance `UserInfo`, providing email to check the email is exist in restaurant's system or not and password to update to user if email user is exist. After, `Database` calling `checkingResult` method of instance `UC` to announce the checking result.

10. If all field is valid input, `UserController` calling `renderAnnouncePage` method of instance `FPF`, providing with the message to announce that user account have been changed password successfully. If not it, `UserController` calling `renderChangePasswordPage` method of instance `FPF`, providing with the message to announce the error during the change password action.

3.5.4 Place an order

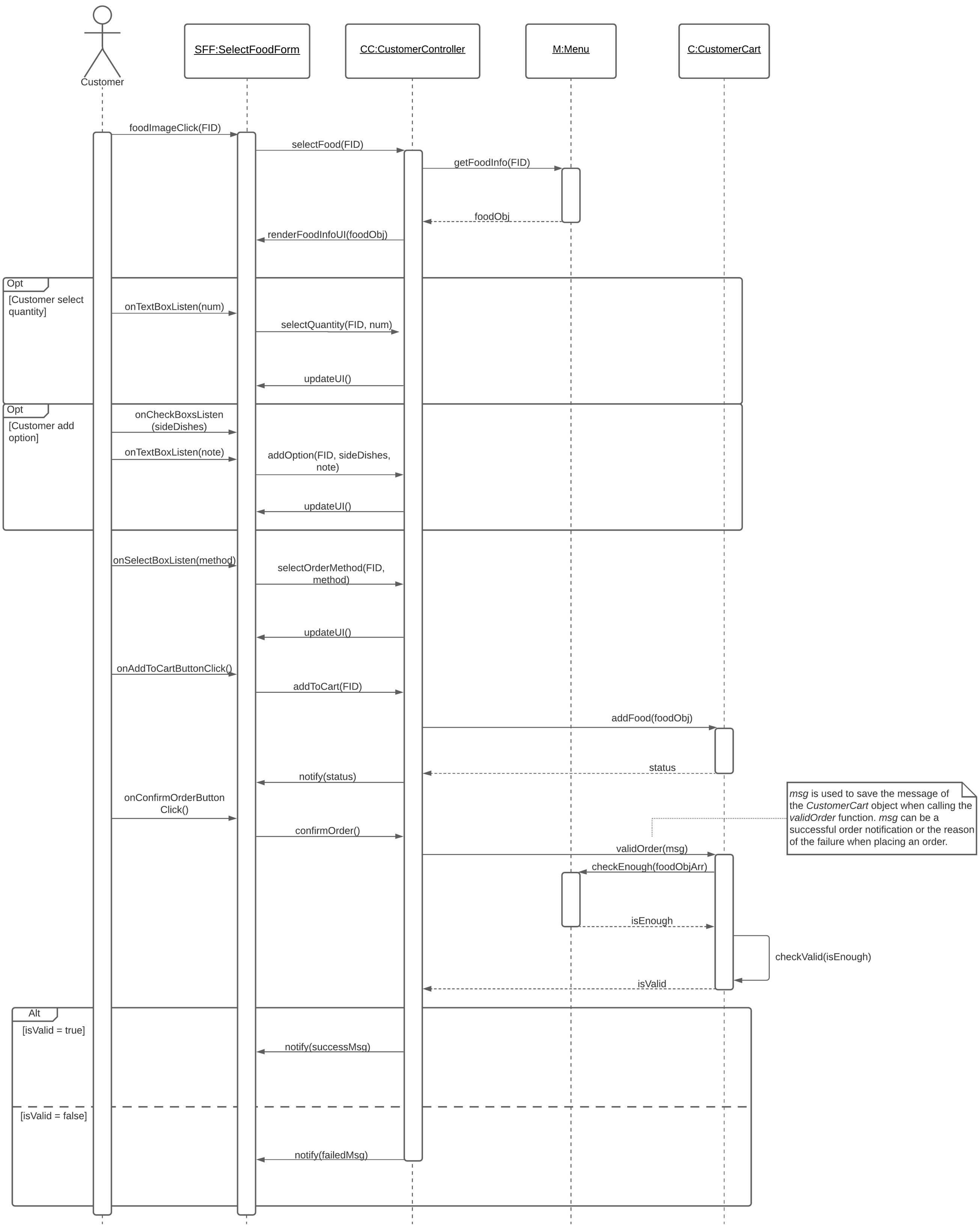


Figure 16: Place an order's sequence diagram

Description:

1. Customer selects the food that they want by clicking on the image of that food on the menu. Then, the Interface triggers the selectFood method of UserController, supplying the food's identifier – FID to identify the required food's information.

2. UserController calls the getFoodInfo method of the instance M of the Menu object class, providing the food's identifier – FID. Then, the instance M returns the required instance foodObj of the MainFood class to the UserController.

3. UserController calls approriate function of the Interface to render the corresponding UI.

4. If customer wants to select the quantity of food, he/she can click on the button "+", "–" or directly type the approriate number in the box. It will trigger the Interface to call the selectQuantity method of UserController, supplying the food's identifier – FID and the number of food – num to change the quantiy property of the instance foodObj with the coresponding FID property.

5. If customer wants to add option to the food, he/she can tick on the box to select the side dishes that he/she wants. Also, there is a text box for customer to write a note for their selected dish. After that, the Interface calls the addOption method, providing the food's identifier – FID, the list of side dishes – sideDishes and the note of the food – note.

6. Customer selects the order method by clicking on the approriate checkbox. The is two type of order method: take-away or eat-in. Then, the Interface triggers the selectOrderMethod method of the UserController, supplying the food's identifier – FID, and the order method – orderMethod.

7. Customer clicks the ***ADD TO CART*** button to add the food to cart. The Interface then triggers the addToCart method of UserController, supplying the food's identifier – FID.

8. UserController calls the addFood method of the instance C of the CustomerCart class, providing the instance foodObj of the class MainFood. The instance C adds the foodObj to the its list and returns the status to the UserController.

9. UserController notify the status to the Interface after adding the food to the cart.

10. Customer clicks the ***CONFIRM ORDER*** button to confirm the order. The Interface calls the confirmOrder method of UserController without providing parameter.

11. UserController calls the validOrder method of the instance C, providing the msg to store the message of the instance C to the UserController.

12. Instance C calls the checkEnough method of the instance M to check if there is enough food for the order. After that, M returns the isEnough to indicate that there is enough food or not. Then, C returns the isValid, which is calculated from isEnough and some other factors, to the UserController.

13. If isValid is true, then UserController notifys that the order is success. If isValid is false, UserController notifys the order is fail and announce the reason why it failed.

3.5.5 Pay for order

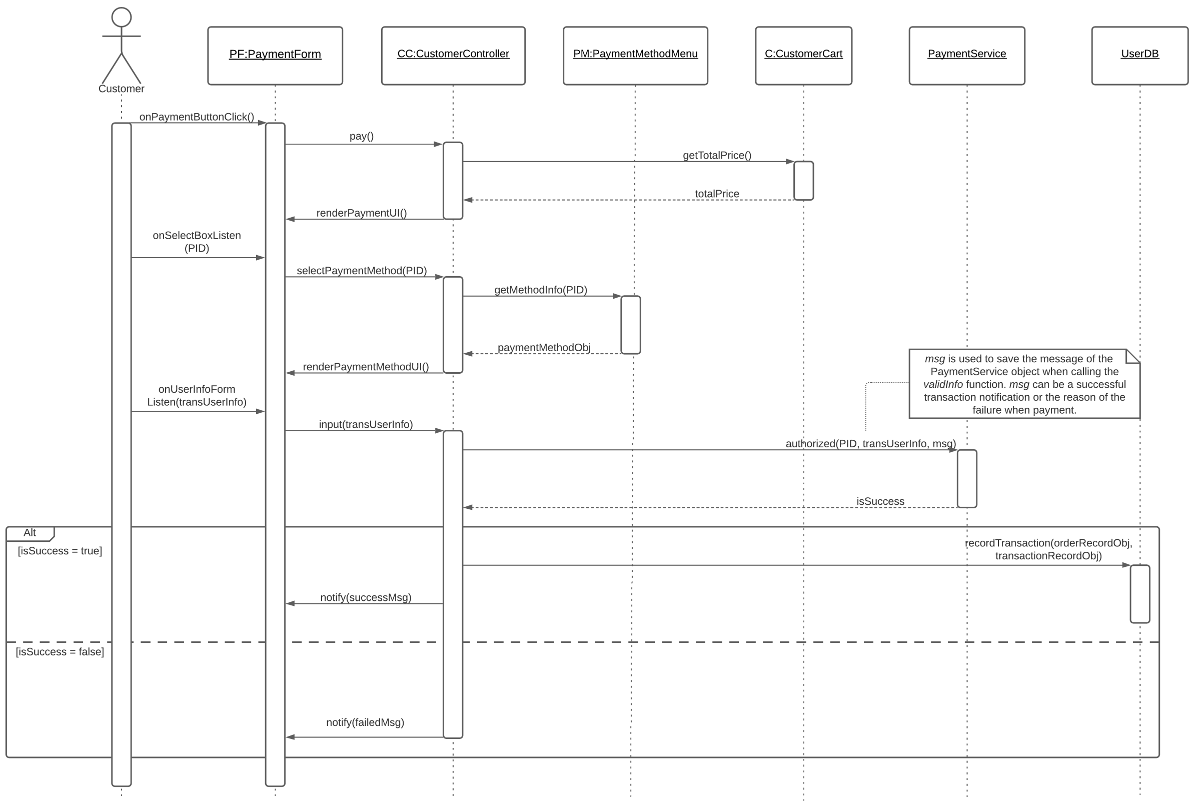


Figure 17: Pay for order's sequence diagram

Description:

1. `Customer` clicks the ***PAYMENT*** button. Then, the Interface triggers the pay method of `UserController`.

2. `UserController` calls the getTotalPrice method of the instance C of the CustomerCart object class. Then, the instance C returns total price – totalPrice of all the main foods in the cart.

3. `UserController` calls approriate function of the Interface to render the corresponding UI.

4. `Customer` selects the payment method by clicking on the approriate checkbox. After that, the Interface calls the selectPaymentMethod, providing the payment method's identifier – PID.

5. `UserController` triggers the getMethodInfo method of the instance PM of the `PaymentMethodMenu` object class, providing the payment method's identifier – PID. Then, the instance PM returns the required instance paymentMethodObj of the PaymentMethod class to the `UserController`.

6. `UserController` calls approriate function of the Interface to render the corresponding UI.

7. `Customer` inputs the user information to the textbox and clicks the ***PAY*** button. For each method, they are required other kinds of information. Then, the Interface calls input method of the `UserController`, providing all the needed information – transUserInfo.

8. `UserController` then triggers the authorized function of the `PaymentService` to check the user information and make a payment, providing payment method's identifier – PID, required user informations – transUserInfo, and message of the `PaymentService` to the `UserController` – msg. After that, `PaymentService` return the isSuccess to the `UserController`.

9. If isSuccess is true, `UserController` writes the transaction to the `Database` via the recordTransaction method and notifys that the transaction is success. If isSuccess is false, `UserController` notifys the transaction is fail and announce the reason why it failed.

3.5.6 Manage account

Update profile

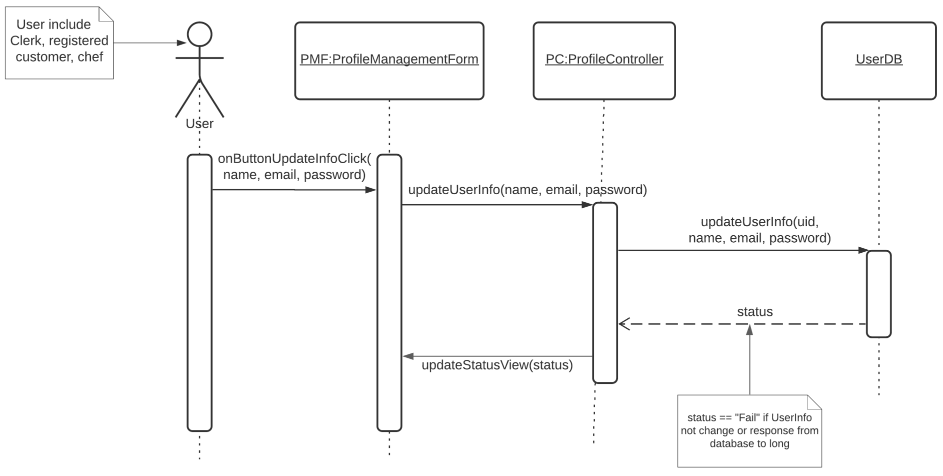

User include Clerk, registered customer, chef
User
PMF:ProfileManagementForm
PC:ProfileController
UserDB
onButtonUpdateInfoClick(name, email, password)
updateUserInfo(name, email, password)
updateUserInfo(uid, name, email, password)
status
updateStatusView(status)
status == "Fail" if UserInfo not change or response from database to long

Figure 18: Update profile's sequence diagram

Description:

1. `Registered customer, clerk or chef` click update button, which called onButtonUpdateInfoClick method of instance PMF, providing with new name, email, password (Unedited field will be remained).
2. Interface's instance then calls updateUserInfo method of instance PC, providing with new user information.
3. ProfileController's instance update user information to Object's UserDB via updateUserInfo method, supplying with userid.
4. Database process's status is returned to instance PC before status is rendered to interface via method updateStatusView of instance PMF, call by controller.

Create profile

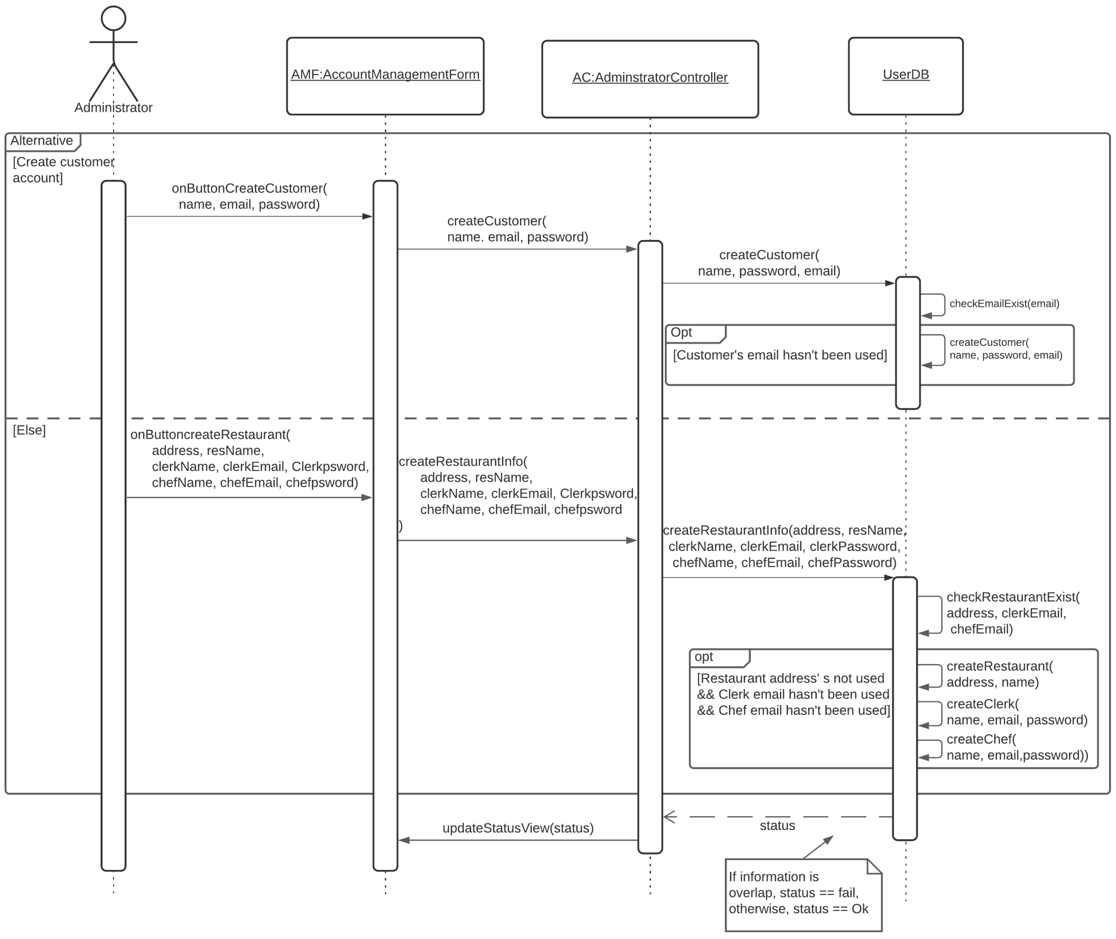

Administrator
AMF:AccountManagementForm
AC:AdminstratorController
UserDB
Alternative
[Create customer account]
onButtonCreateCustomer(name, email, password)
createCustomer(name. email, password)
createCustomer(name, password, email)
checkEmailExist(email)
Opt
[Customer's email hasn't been used]
createCustomer(name, password, email)
[Else]
onButtoncreateRestaurant(address, resName, clerkName, clerkEmail, Clerkpsword, chefName, chefEmail, chefpsword)
createRestaurantInfo(address, resName, clerkName, clerkEmail, Clerkpsword, chefName, chefEmail, chefpsword)
createRestaurantInfo(address, resName, clerkName, clerkEmail, clerkPassword, chefName, chefEmail, chefPassword)
checkRestaurantExist(address, clerkEmail, chefEmail)
opt
[Restaurant address' s not used && Clerk email hasn't been used && Chef email hasn't been used]
createRestaurant(address, name)
createClerk(name, email, password)
createChef(name, email,password))
updateStatusView(status)
status
If information is overlap, status == fail, otherwise, status == Ok

Figure 19: Create profile's sequence diagram

Description:

Case 1: Administrator creates customer

1. `Administrator` click create button, which called onButtonCreateCustomer method of instance AMF, providing with customer's name, email, password.
2. Interface's instance then calls createCustomer method of instance PC, providing with customer information.
3. AdminstartorController's instance push user information to Object's UserDB via createCustomer method.
4. Database first check whether email has been used by checkEmailExist method. If this email hasn't been used, UserDB update customer profile via createCustomer method, providing with name, email and password.

Case 2: Administrator creates restaurant's object (Clerk, restaurant, chef)

1. `Administrator` click create button, which called onButtonCreateRestaurant method of instance AMF, providing with Restaurant's address, name, Clerk's name, email, password, Kitchen manager's name, email and password.
2. Interface's instance then calls createRestaurantInfo method of instance AC, providing with restaurant information.
3. AdminstratorController's instance push restaurant information to Object's UserDB via createRestaurantInfo method.

4. Database first check whether restaurant's address, Clerk, Kitchen manager have been used by checkRestaurantExist method. If these information haven't been used, UserDB update restaurant, clerk and Kitchen manager profile via createRestaurant, createClerk, createChef method respectively.

Find profile

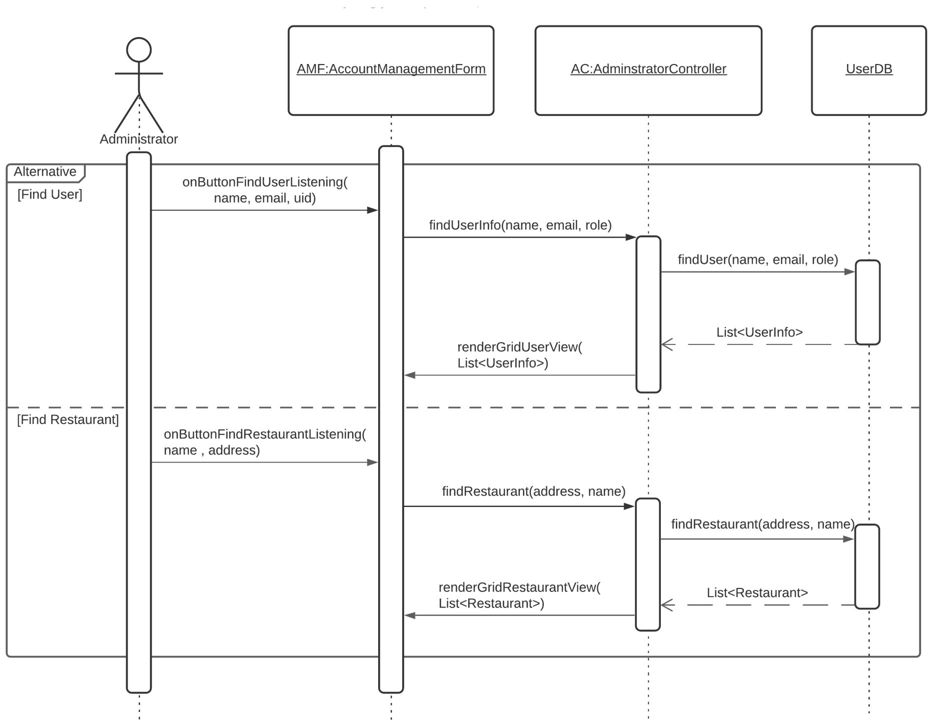

AMF:AccountManagementForm
AC:AdminstratorController
UserDB
Administrator
Alternative
[Find User]
onButtonFindUserListening(name, email, uid)
findUserInfo(name, email, role)
findUser(name, email, role)
List<UserInfo>
renderGridUserView(List<UserInfo>)
[Find Restaurant]
onButtonFindRestaurantListening(name , address)
findRestaurant(address, name)
findRestaurant(address, name)
List<Restaurant>
renderGridRestaurantView(List<Restaurant>)

Figure 20: Find profile's sequence diagram

Description: Adminstrator find user/restaurant need to be deleted via Find profile usecase. *Case 1: Administrator delete customer's account:*

1. `Administrator` click delete button, which called onButtonCreateCustomer method of instance AMF, providing with customer's uid.
2. Interface's instance then calls deleteCustomer method of instance AC, providing with customer uid.
3. AministratorController's instance delete customer by Object's UserDB via deleteCustomer method, providing with uid.
4. Database check if customer's account is activating (login), object's UserDB then deactivate customer by call instance UMF's logout's method.

Case 2: Administrator delete restaurant, clerk or kitchen manager

1. `Administrator` click delete button, which called onButtonCreateRestaurant method of instance AMF, providing with Clerk or Kitchen manager uid if user is deleted; or restaurant's id if restaurant is deleted.
2. Interface's instance then calls deleteRestaurant method of instance AC, providing with target's id.
3. AdministratorController's instance delete target by Object's UserDB via deleteRestaurant method, providing with id.
4. Database first find restaurant via this id as id is unique to each restaurant and each clerk/kitchen manager account is associated with unique restaurant.
5. This instance restaurant also have clerk, kitchen manager information (including uid), UserDB can log out Clerk/Kitchen manager via log out method if they has logined, before each targets via deleteClerk, deleteChef and deleteRestaurant providing with id.

Delete profile

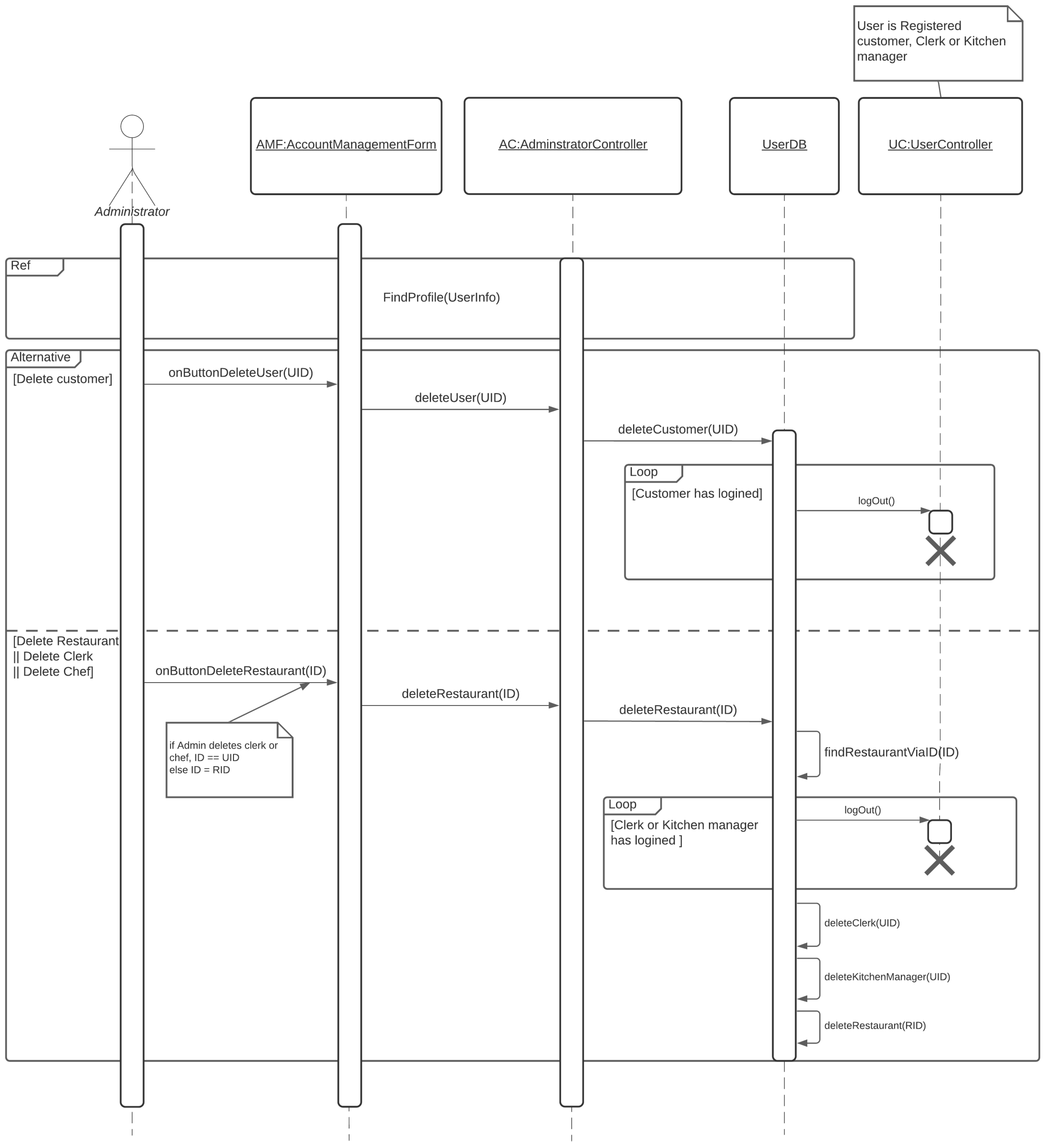

User is Registered customer, Clerk or Kitchen manager
AMF:AccountManagementForm
AC:AdminstratorController
UserDB
UC:UserController
Administrator
Ref
FindProfile(UserInfo)
Alternative
[Delete customer]
onButtonDeleteUser(UID)
deleteUser(UID)
deleteCustomer(UID)
Loop
[Customer has logined]
logOut()
[Delete Restaurant || Delete Clerk || Delete Chef]
onButtonDeleteRestaurant(ID)
deleteRestaurant(ID)
deleteRestaurant(ID)
if Admin deletes clerk or chef, ID == UID else ID = RID
findRestaurantViaID(ID)
Loop
[Clerk or Kitchen manager has logined]
logOut()
deleteClerk(UID)
deleteKitchenManager(UID)
deleteRestaurant(RID)

Figure 21: Delete profile's sequence diagram

Description:

1. `Administrator` find profile of user needed to be delete as same as ***FIND PROFILE*** sequence
2. `Administrators` click delete button associating with the user that needs to be deleted, supplying `UserController` the UID of that user.
3. `UserController` sends this user's UID to the `Database`. `Database` then delete this user and return `Administrator` a status message.

View Database

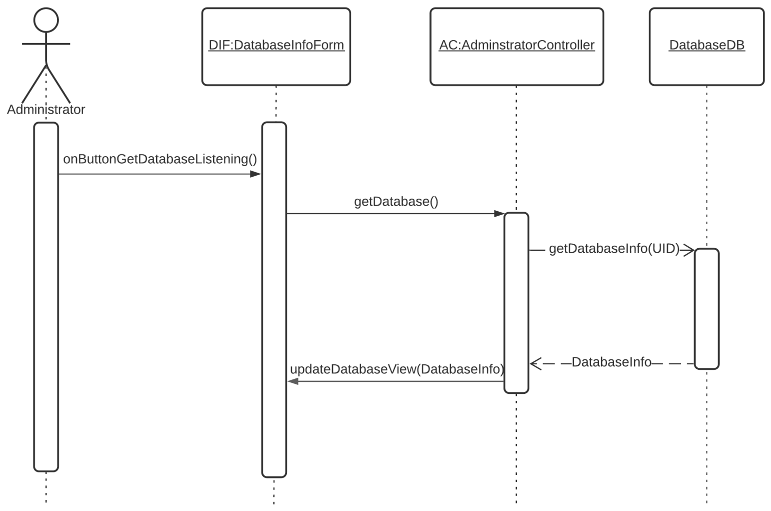


Figure 22: View database's sequence diagram

Description:

1. `Adminstrator` click ***VIEW*** button, which called onButtonGetDatabaseListening method of instance AMF.
2. Interface's instance then calls getDatabase method of instance AC.
3. AccountManagementController's instance retrieve database information through Object's UserDB via getDatabaseInfo method, supplying with admin id.
4. Database returns to instance AC Database information before data is rendered to interface via method updateDatabaseView of instance PMF, call by controller.

3.5.7 View information

View transaction history

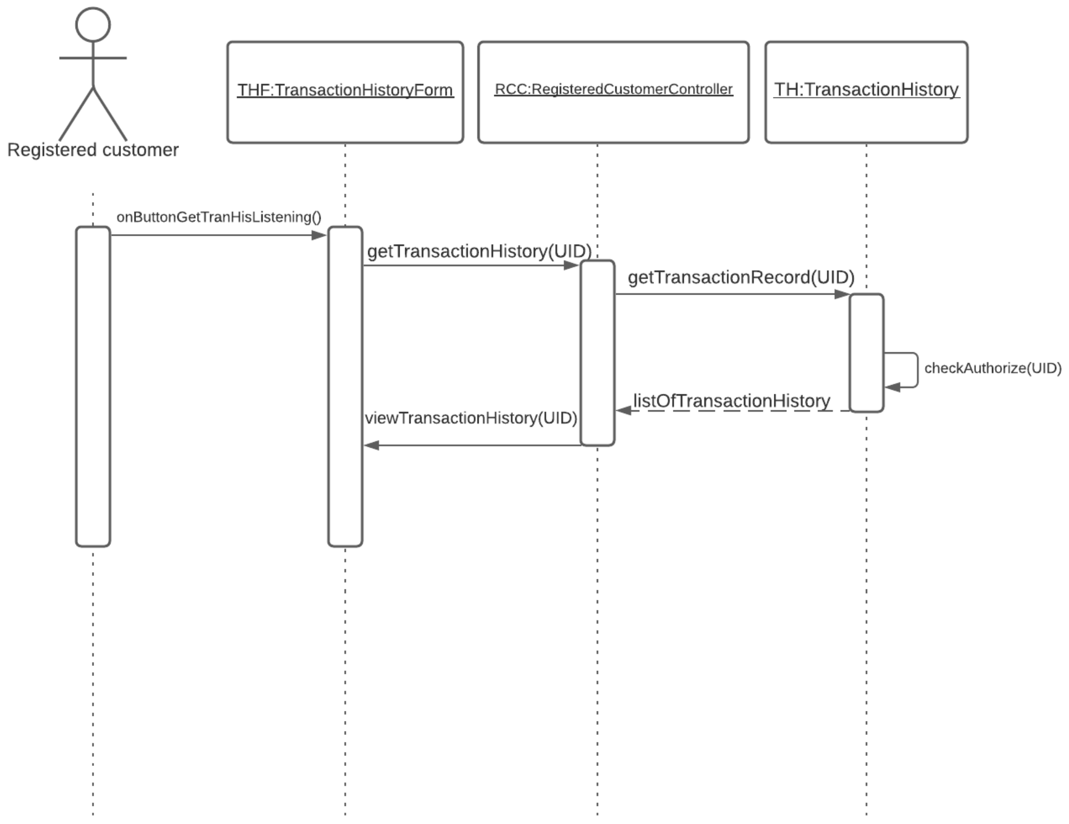

Registered customer
THF:TransactionHistoryForm
RCC:RegisteredCustomerController
TH:TransactionHistory
onButtonGetTranHisListening()
getTransactionHistory(UID)
getTransactionRecord(UID)
checkAuthorize(UID)
listOfTransactionHistory
viewTransactionHistory(UID)

Figure 23: View transaction history's sequence diagram

Description:

1. `Registered customer` call `onButtonGetTranHisListening()` of TransactionHistoryForm object
2. `TransactionHistoryForm` calls the `getTransactionHistory` method of the `RegisteredCustomerController` object and the `RegisteredCustomerController` object call `getTransactionRecord` method of `TransactionHistory` object.
3. The `TransactionHistory` object request information from `Database` and return list of Transaction history then render to UI

View order history statistics

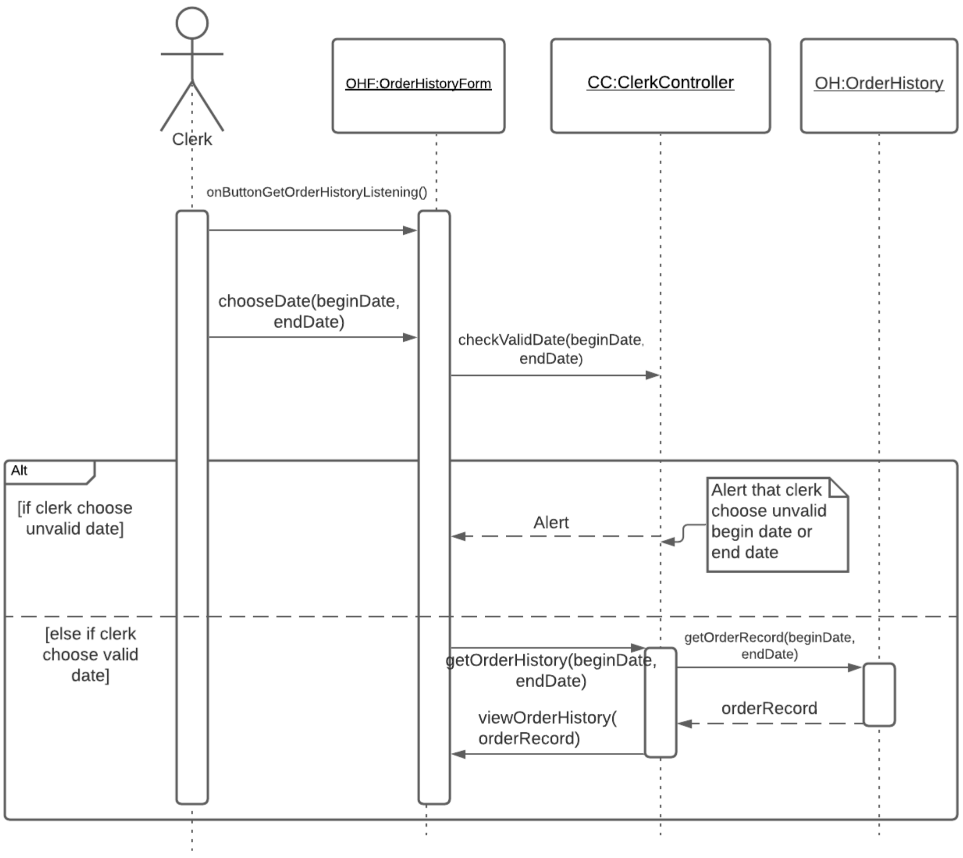

Clerk
OHF:OrderHistoryForm
CC:ClerkController
OH:OrderHistory
onButtonGetOrderHistoryListening()
chooseDate(beginDate, endDate)
checkValidDate(beginDate, endDate)
Alt
[if clerk choose unvalid date]
Alert
Alert that clerk choose unvalid begin date or end date
[else if clerk choose valid date]
getOrderHistory(beginDate, endDate)
getOrderRecord(beginDate, endDate)
orderRecord
viewOrderHistory(orderRecord)

Figure 24: View order history statistics's sequence diagram

Description:

1. `Clerk` call ***onButtonGetOrderHistoryListening*** method of `OrderHistoryForm` object to render History Statistics UI
2. Clerk call ***chooseDate*** method of `OrderHistoryForm` object then `OrderHistoryForm` call ***checkValidDate*** method of `ClerkController` to check if the begin date and end date clerk choose is valid
3. If `Clerk` choose unvalid date `ClerkController` return alert to `Clerk`.
4. If `Clerk` choose valid date `OrderHistoryForm` calls **getOrderHistory** method of `ClerkController` object and `ClerkController` object call **getOrderRecord** method of `OrderHistory` object. Then `ClerkController` object calls ***viewOrderHistory*** method of `OrderHistoryForm` to make statistics and render to UI

View feedback

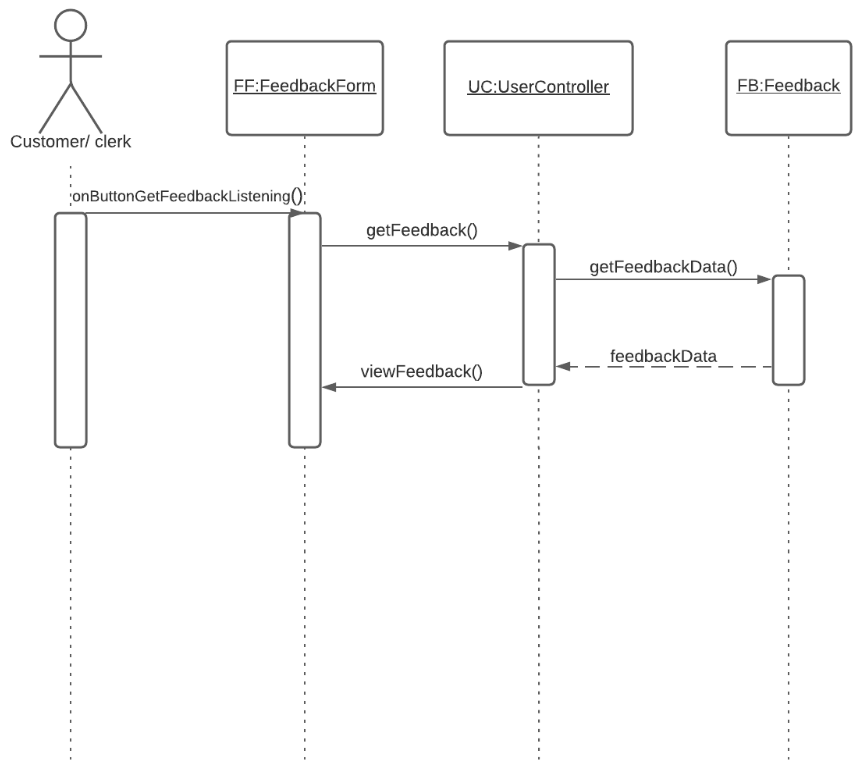


Figure 25: View feedback's sequence diagram

Description:

1. `Customer` or `Clerk` call ***onButtonGetFeedbackListening*** method of `FeedbackForm` object
2. `FeedbackForm` object calls the `getFeedback` method of the `Usercontroller` include `Clerkcontroller` and `Customercontroller`.
3. The `UserController` object calls the `getFeedbackData` method of the `Feedback` object and request information from `Database` and return list of Feedback then render to UI

3.5.8 Update menu

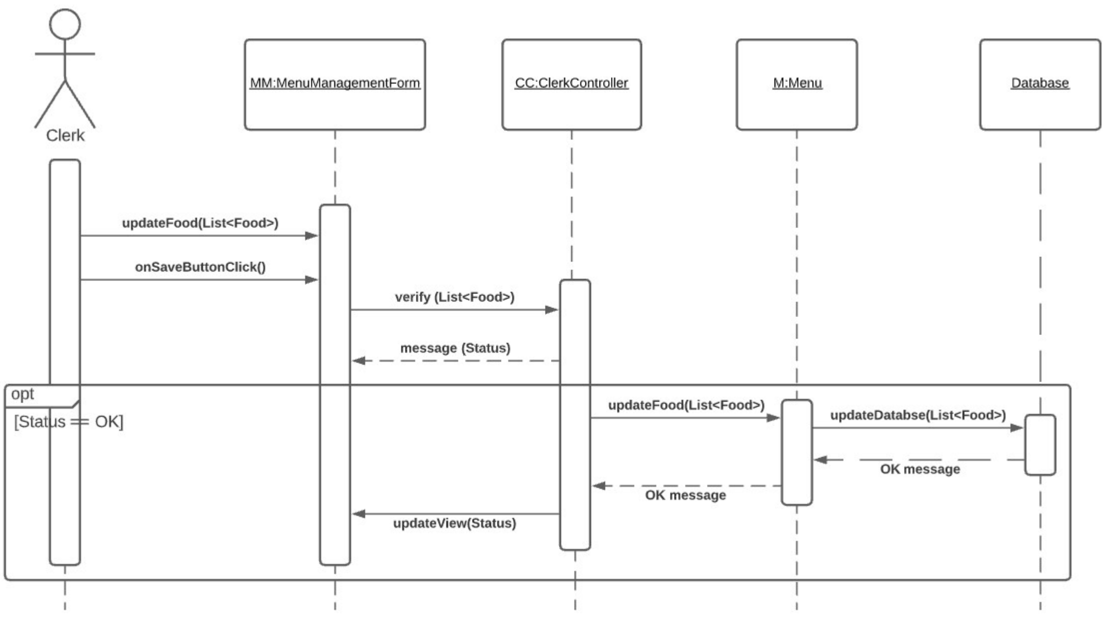


Figure 26: Update menu's sequence diagram

Description:

1. `Clerk` perform task such as add new foods, delete old foods, update attribute of current foods (price, quantity, side dish) in an instance Interface of the UI class, supplying required information.
2. `Clerk` saves these information, then instance Interface checks with an `Menu controller`for these information and `Menu controller` will return a status.
3. If status is OK, these information will be update to the `Menu` and `Menu` after change is displayed on Interface. If status is fail, a Notification is displayed.

3.5.9 Process order

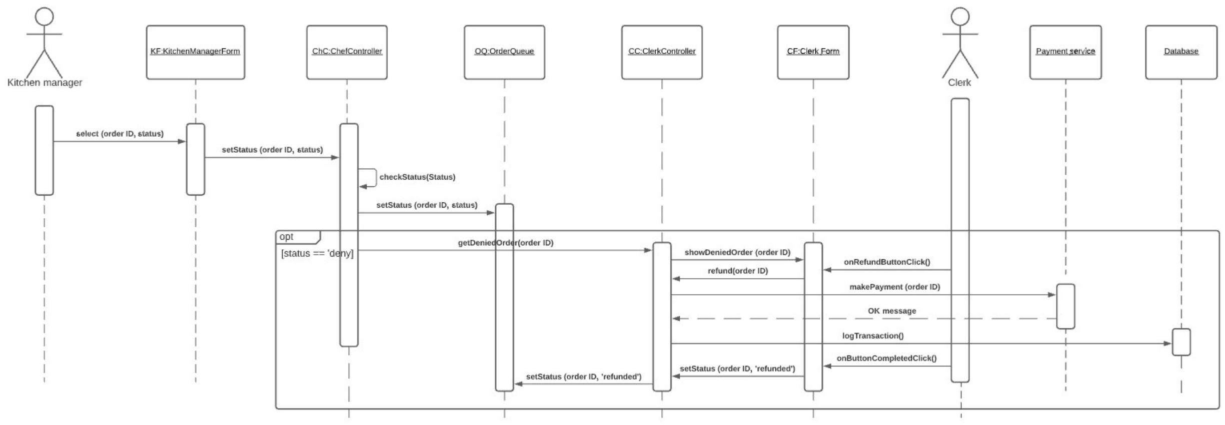


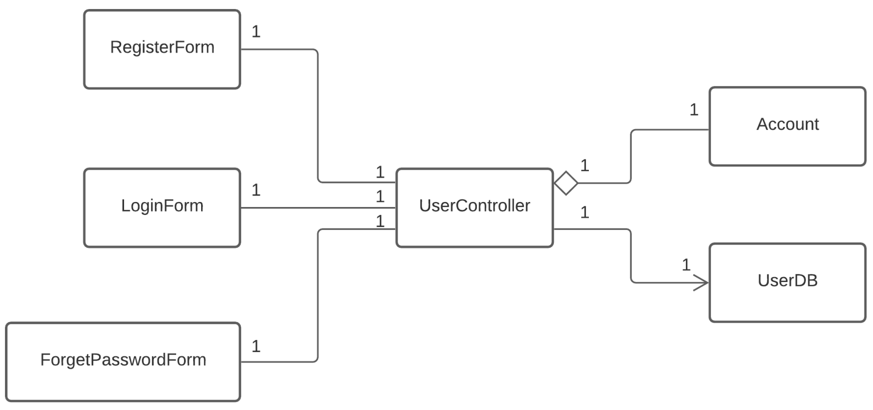
Figure 27: Process order's sequence diagram

Description:

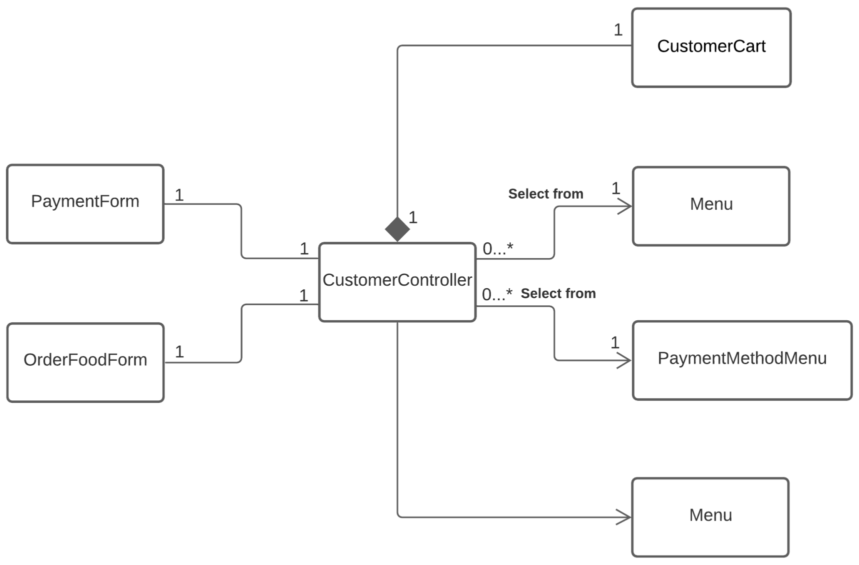
1. `Kitchen manager` selects orders and set status ("accept" or "deny") for these orders on Interface
2. Interface sends these information to Order controllers
3. If status is "accept", `Kitchen manager` marks accepted orders as completed after completing them on Interface, then Interface calls `Orders controller` to update status of these orders. If status is "deny", `Orders controller` sends denied orders to `Clerk`, `Clerk` then refunds these orders by calling payment service. `Payment service` call `Database` to log transactions and then return message to clerk. Finally, `Clerk` updates status of these orders.

3.6 Class diagram

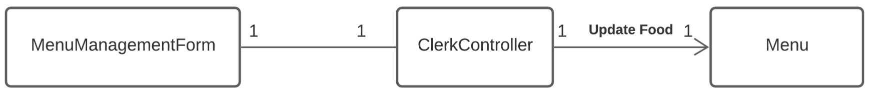
3.6.1 Login



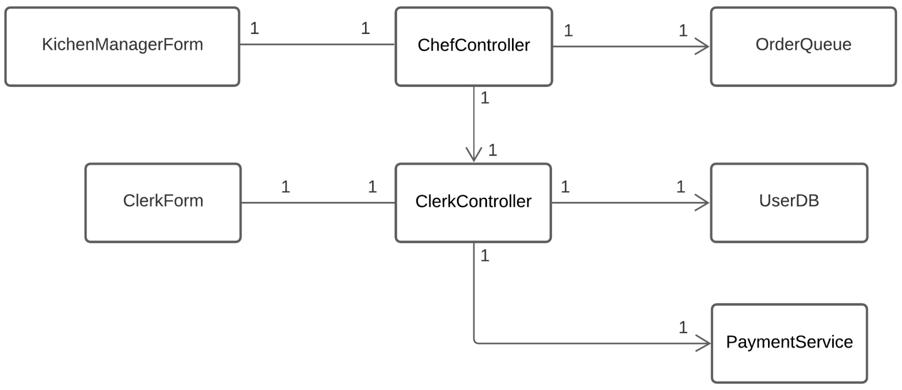
3.6.2 Order and pay



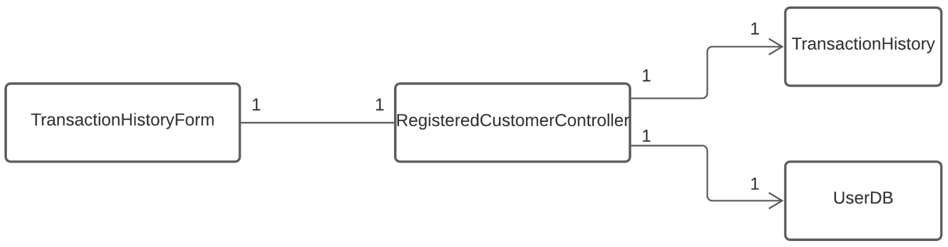
3.6.3 Update menu



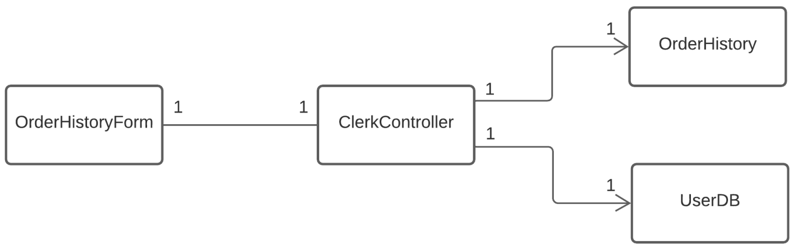
3.6.4 Process order



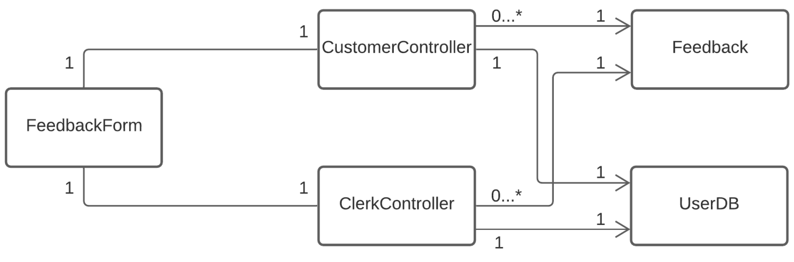
3.6.5 View transaction history



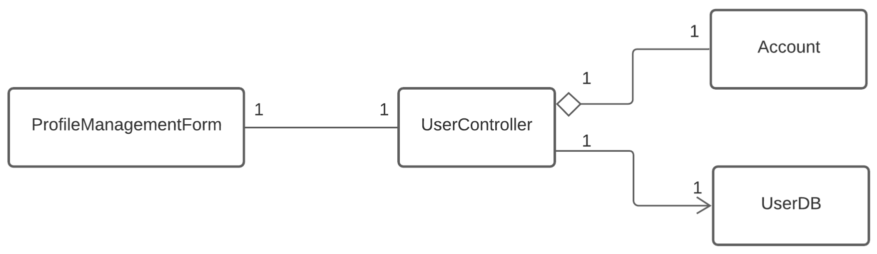
3.6.6 View order history



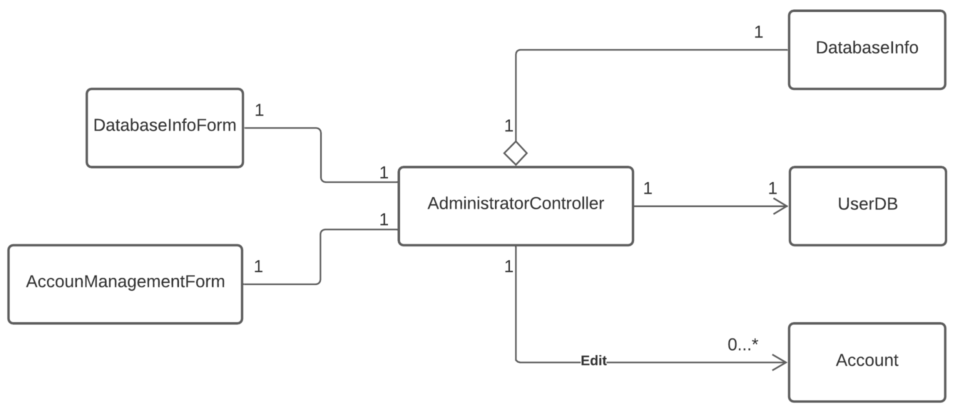
3.6.7 Feedback



3.6.8 Update profile



3.6.9 View and management profile



3.7 Class detail

3.7.1 Login

Account

- Email : String = "Unknown"
- Password : String = "Unknown"

User

# Phone: String = "Unknown"
# Username: String = "Guest"

+ updateProfile(name: String, phone: String): void

LoginForm

+ onButtonLogin(email: String, password: String): Void
+ renderLoginpage(msg: String): Void
+ renderHomepage(UserName: String): Void

Mail system

+ postLink(email: String): Void

UserController

+ checkLogin(email: String, password: String): Void
+ checkRegister(email: String, userName: String, phoneNumber: String, password: String, repeatPassword: String): Boolean
+ checkForgetPassword(email: String): Boolean
+ checkChangePassword(email: String, password: String, forgetPassword: String): Boolean

Database

+ checkInfo(email: String, password: String): Boolean
+ checkInfo(email: String, userName: String, phoneNumber: String): Boolean
+ checkInfo(email: String): Boolean
+ updateInfo(email: String, password: String): Boolean
+ updateInfo(email: String, userName: String, password: String, phoneNumber: String): Boolean

RegisterForm

+ registerPage.open(): Void
+ onButtonRegister(email: String, userName: String, password: String, repeatPassword: String, phoneNumber: String): Void
+ renderRegisterpage(msg: String): Void
+ renderAnnouncepage(msg: String): Void

ForgetPasswordForm

+ forgetPasswordPage.open(): Void
+ onButtonRetrievePassword(email: String): Void
+ renderForgetPasswordPage(msg: String): Void
+ renderAnnouncePage(msg: String): Void
+ changePasswordPage.open(): Void
+ onButtonChangePassword(email: String, password: String, repeatPassword: String): Void

3.7.2 View information

Feedback

- feedbackRecord: String
- feedbackID: Integer
- feedbackDate: Date

+ sortFeedback(): Void
+ getFeedbackData(): FeedbackRecord[0...* order]
+ setFeedbackData(in _feedbackRecord: FeedbackRecord): Boolean
+ addFeedback(in _feedbackRecord: String): Boolean
+ deleteFeedback(in _feedbackRecord: String): Boolean
+ setFeedbackDate(in _feedbackDate: Date): Boolean
+ getFeedbackDate(): Date

OrderHistory

- listOfOrder: OrderRecord[0...* order]

+ getOrderRecord(): OrderRecord[0...* order]
+ setOrderRecord(
in _orderID: Integer): OrderRecord
+ sortOrderHistory(): Void
+ addOrderRecord(in _orderRecord): Boolean
+ deleteOrderRecord(
in _orderRecord: String): Boolean

ClerkController

- totalMethod: Integer
- listOfPaymentMethod: PaymentMethod[0...* unorder]

+ checkValidDate(beginDate, endDate): String
+ getOrderHistory(beginDate, endDate): Boolean
+ getDeniedOrder(orderID: Integer): void
+ refundOrder(orderID: Integer): void
+ setStatus(orderID: Integer,
status="refunded": String): void
+ verify(foods: Food[...*]): String

OrderHistoryForm

+ onButtonGetOrderHistoryListening: void
+ chooseDate(beginDate, endDate): boolean
+ notify(_alert: string):
+ countOrder(): void
+ viewOderHistory: void

TransactionHistoryForm

+ onButtonGetTranHisListening(UID): void
+ viewTransactionHistory(UID): void

FeedbackForm

+ onButtonGetFeedbackListening: void
+ viewFeedback: void

TransactionHistory

- listOfTransaction: TransactionRecord[0...* order]

+ sortTransactionHistory(): Void
+ getTransactionRecord(in _UID: integer): TransactionRecord[0..*unorder]
+ setTransactionRecord(in _TransactionID: integer): Boolean
+ addTransactionRecord(in _TransactionRecord: String): Boolean
+ deleteTransactionRecord(in _TransactionRecord: String): Boolean

3.7.3 Account management:

AccountManagementForm

-gridUserView: GridView
-gridRestaurantView: GridView
resName, resAddress: TextView
clerkName, clerkPassword, clerkEmail: TextView
chefName, chefPassword, chefEmail: TextView
status: TextView

+ renderGridUsersView(users[0..*]: Userinfo)
+ renderGridRestaurantView(
restaurant[0..*]: Restaurant)
+ updateStatusView(status: String)
+ onButtonDeleteUser(uid: Integer): void
+ onButtonDeleteRestaurant(id: Integer): void
+ onButtonCreateCustomer(name, email,
password: String): void
+ onButtoncreateRestaurant(
address, resName,
clerkName, clerkEmail, Clerkpsword,
chefName, chefEmail, chefpsword: String): void
+ onButtonFindRestaurantListening(
name , address: String): void
+ onButtonFindUserListening(
name, email, uid)

ProfileManagementForm

- textStatus: TextView
- name, email, password, age: TextView
- buttonUpdateProfile: Button

+ updateStatusView(status: String): void
+ onButtonUpdateInfoClick(name, email, password)

DatabaseInfoForm

- GridDatabase: GridView

+ updateDatabaseView(Database): void
+ onButtonGetDatabaseListening()

UserController

+ updateUserInfo(name, email, password: String): String

AdminstratorController

-AuthorUID: int

+ findUserInfo(name, email:
String role: Role): UserInfo [0..*]
+ findRestaurant(address,
name: String): Restaurant[0...*]
+ deleteUser(uid: Integer): void
+ deleteRestaurant(rid: Integer): void
+ createCustomer(name, email, password)
+ createRestaurantInfo(
resName, address,
clerkName, clerkEmail, clerkPassword,
chefName, chefEmail, chefPassword
)
+ getDatabase(UID: int)

UserDB

+ updateUserInfo(uid, name, email,
password: String): String
+ getRestaurant(id: int): Restaurant
+ findUser(name, email: String,
role: Role): User [0..*]
+ findRestaurantViaID(id: Integer): Restaurant
+ findRestaurant(address, name): Restaurant[0..*]
+ deleteCustomer(uid: Integer): void
+ deleteRestaurant(uid: Integer): void
+ deleteClerk(uid: Integer): void
+ deleteKitchenManager(uid: Integer): void
+ createCustomer(name, password,
email : String): void
- createRestaurantInfo(address, resName,
clerkName, clerkEmail, clerkPassword,
chefName, chefEmail, chefPassword: String): String
- createRestaurant(address, name: String): void
- createClerk(name, password, email: String): void
- createKitchenManager(name,password,
email: String): void
- checkRestaurantExist(address, email,
email: String):Boolean
- checkEmailExist(email: String): Boolean

DatabaseDB

+ getDatabaseInfo(UID): DatabaseInfo
- checkPermission(UID): Boolean

3.7.4 Order and pay

OrderFoodForm

- imageList: Image[0...* unorder]
- quantityBox: TextBox
- sideDishesBoxs: CheckBox[0...* unorder]
- noteBox: TextBox
- methodBox: SelectBox
- addToCardButton: Button
- confirmOrderButton: Button

+ onFoodImageClick(in _FID: Integer): Void
+ onTextBoxListen(in _msg: String): Void
+ onCheckBoxsListen(in _sideDishes: SideDish[0...* unorder]): Void
+ onSelectBoxListen(in _method: String): Void
+ onAddToCartButtonClick(): Void
+ onConfirmOrderButtonClick(): Void
+ notify(in _msg: String) Void
+ renderFoodInfoUI(in _foodObj: Food): Void
+ updateUI(): Void

PaymentForm

- paymentButton: Button
- paymentMethodBox: SelectBox
- userInfoForm: TextBox[0...* unorder]

+ onPaymentButtonClick(): Void
+ onSelectBoxListen(in _PID: Integer): Void
+ onUserInfoFormListen(in _transUserInfo: String[0...* unorder]): Void
+ renderPaymentUI(): Void
+ renderPaymentMethodUI(): Void
+ notify(in _msg: String): Void

CustomerController

+ selectFood(in _FID: Integer)
+ selectQuantity(in _FID: Integer, in _num: Integer)
+ addOption(in _FID: Integer, in _sideDishes: SideDish[0...* unorder], in _note: String)
+ selectOrderMethod(in _FID: Integer, in _method: String)
+ addToCart(in _FID: Integer)
+ confirmOrder()
+ pay()
+ selectPaymentMethod(in _PID: Integer)
+ input(in _transUserInfo: String[0...* unorder])

Menu

- totalFood: Integer
- listOfSideDish: SideDish[0...* unorder]
- listOfMainFood: MainFood[0...* unorder]

+ getFood(in _FID : Integer): Food
+ addSideDish(in _dish: SideDish): Boolean
+ deleteSideDish(in _name: String): Boolean
+ addMainFood(in _food: MainFood): Boolean
+ deleteMainFood(in _name: String): Boolean
+ getSideDish(): SideDish[0..* unorder]
+ getMainFood(): MainFood[0..* unorder]
+ updateMainFood(in _FID: Integer): Void

CustomerCart

- totalPrice: Integer
- tax: Integer
- listOfMainFood: MainFood[0...* unorder]

+ validOrder(out _msg: String) : Boolean
+ getTotalPrice(): Integer
+ checkValid(in _isEnough: Boolean) : Boolean

PaymentMethodMenu

- totalMethod: Integer
- listOfPaymentMethod: PaymentMethod[0...* unorder]

+ getMethodInfo(in _PID: Integer) : PaymentMethod

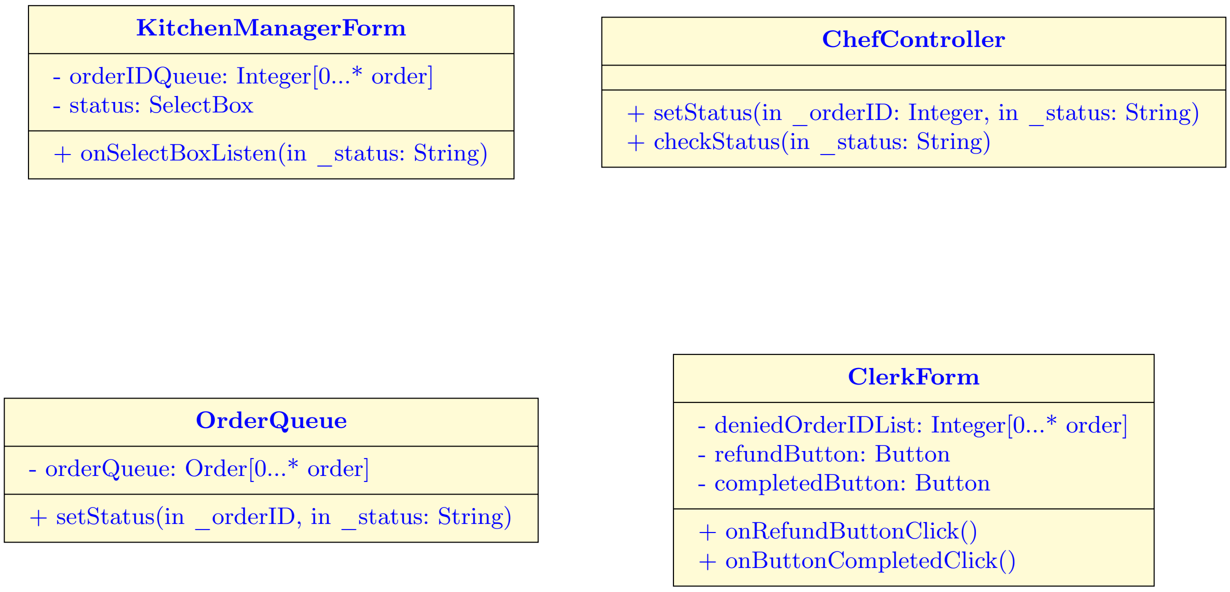
3.7.5 Update menu

MenuManagementForm

- imageList: Image[0...* unorder]
- addFoodButton: Button
- deleteFoodButton: Button
- updateFoodButton: Button

+ onAddFoodClick()
+ onDeleteFoodClick()
+ onUpdateFoodClick()

3.7.6 Process order



KitchenManagerForm

- orderIDQueue: Integer[0...* order]
- status: SelectBox

+ onSelectBoxListen(in _status: String)

ChefController

+ setStatus(in _orderID: Integer, in _status: String)
+ checkStatus(in _status: String)

OrderQueue

- orderQueue: Order[0...* order]

+ setStatus(in _orderID, in _status: String)

ClerkForm

- deniedOrderIDList: Integer[0...* order]
- refundButton: Button
- completedButton: Button

+ onRefundButtonClick()
+ onButtonCompletedClick()

4 Architecture

4.1 Architecture description

4.1.1 SPA introduction

Single page Application is a web application help enhance user experience by using HTML5 and AJAX. When loading any web page, SPA will load a single HTML page, then based on user request, SPA will continue to load other HTML in that same page.

To put it simply, the entire web resource including CSS, Javascript, master layout or web page structure files will be loaded for the first time when we start browsing a certain website A. Next time, when switching to another page, the client will send ajax requests to get the necessary data (usually the content). This provides a better web user experience, reduces the time it takes to reload the entire cumbersome web page, and saves bandwidth and waiting time. This is in stark contrast to the traditional website where the entire web page has to be reloaded every time the page turns.

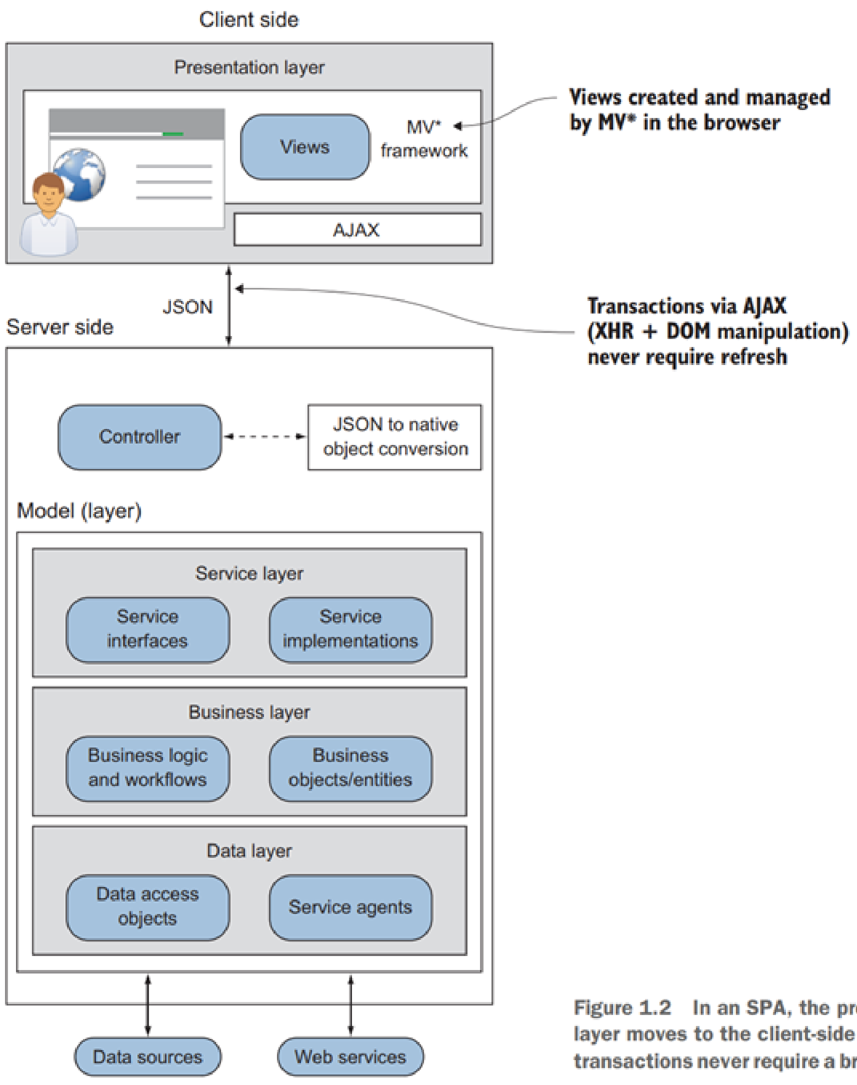


Figure 1.2 In an SPA, the presentation layer moves to the client-side code, and transactions never require a browser refresh.

Service layer	Service layer is an architectural pattern, applied within the service-orientation design paradigm, which aims to organize the services, within a service inventory, into a set of logical layers.
Business layer	This is the place to meet the data manipulation requirements of the GUI layer, process the data source from the Presentation Layer before it is transmitted to the Data Layer and saved to the DBMS. This is also the place to check constraints, data integrity and validity, perform calculations and handle business requirements. In POS system, Business layer process order, payment and send record to Database
Data layer	This layer has the function of communicating with the DBMS such as performing tasks related to storing and querying data (search, add, delete, edit, ...).

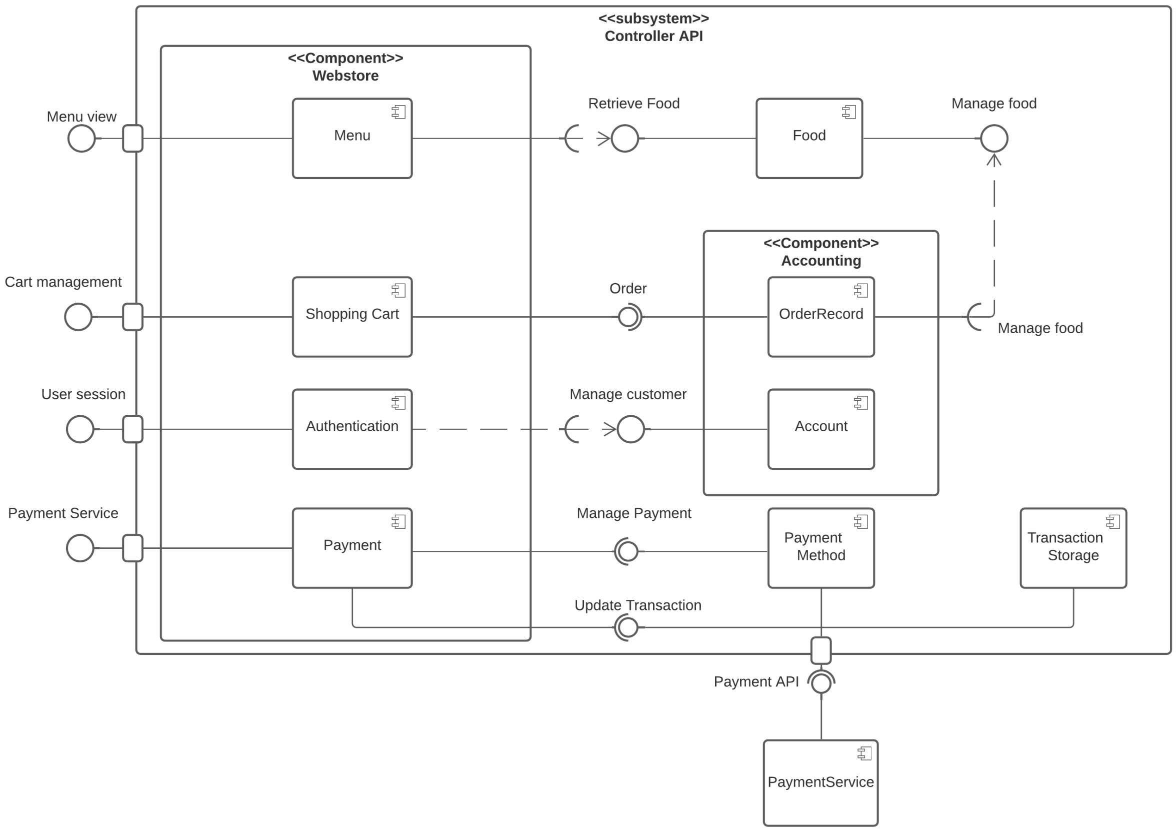
4.1.2 Advantage:

Better mobile experience	For POS system, most customers interact with the system through their mobile devices such as mobile phones, tablets, . . . , so using Single Page Application (SPA) will make the mobile experience better because the page load speed will be faster. it is also suitable when meeting the Nonrequirement of handling 300 orders a day.
Limit the query to the Server	The server will not send any more HTML to the client because the client has already downloaded it all from the beginning. The server sends the structure of the page and your browser renders the user interface (UI) on that structure. It also saves time and costs for businesses when deploying infrastructure.
Easier to target a specific object	The server will not send any more HTML to the client because the client has already downloaded it all from the beginning. The server sends the structure of the page and your browser renders the user interface (UI) on that structure. It also saves time and costs for businesses when deploying infrastructure.
Increase Website's credibility	This is the advantage of having only one page because every link points to the home page.

4.1.3 Disadvantage:

Limit content detail of a page	One of the disadvantages of a single-page site is that the content cannot be as specific and detailed as a multi-page site. But, we aim for convenience, speed, not lengthy content.
Limit the query to the Server	There are advanced SEO techniques (Search Engine Optimization) that certainly cannot be used on a single page. One of those techniques is the technique of structuring your website into Categories and Subcategory to show the best content to users and help your site be divided according to credibility.

4.2 Component diagram:



Details:
Controller API subsystem:

- This is used to communicate with the interface to receive requests from customer and manage communication with the PaymentService.
- This system consists of the Webstore subsystem, the Food component, the Payment Method component, the Transaction Storage component, and the Accounting subsystem.
- This system provides an interface for menu viewing, cart management, user session management, and payment management.
- This system also requires the Payment API interface to receive services from the PaymentService.
- Food: Component food provide Retreive food interface for Menu and provide Manage food interface for Order Record
- Payment service: Payment service provide Payment API interface for Payment Method

- Transaction storage: Transaction storage provide Update Transaction interface for Payment component

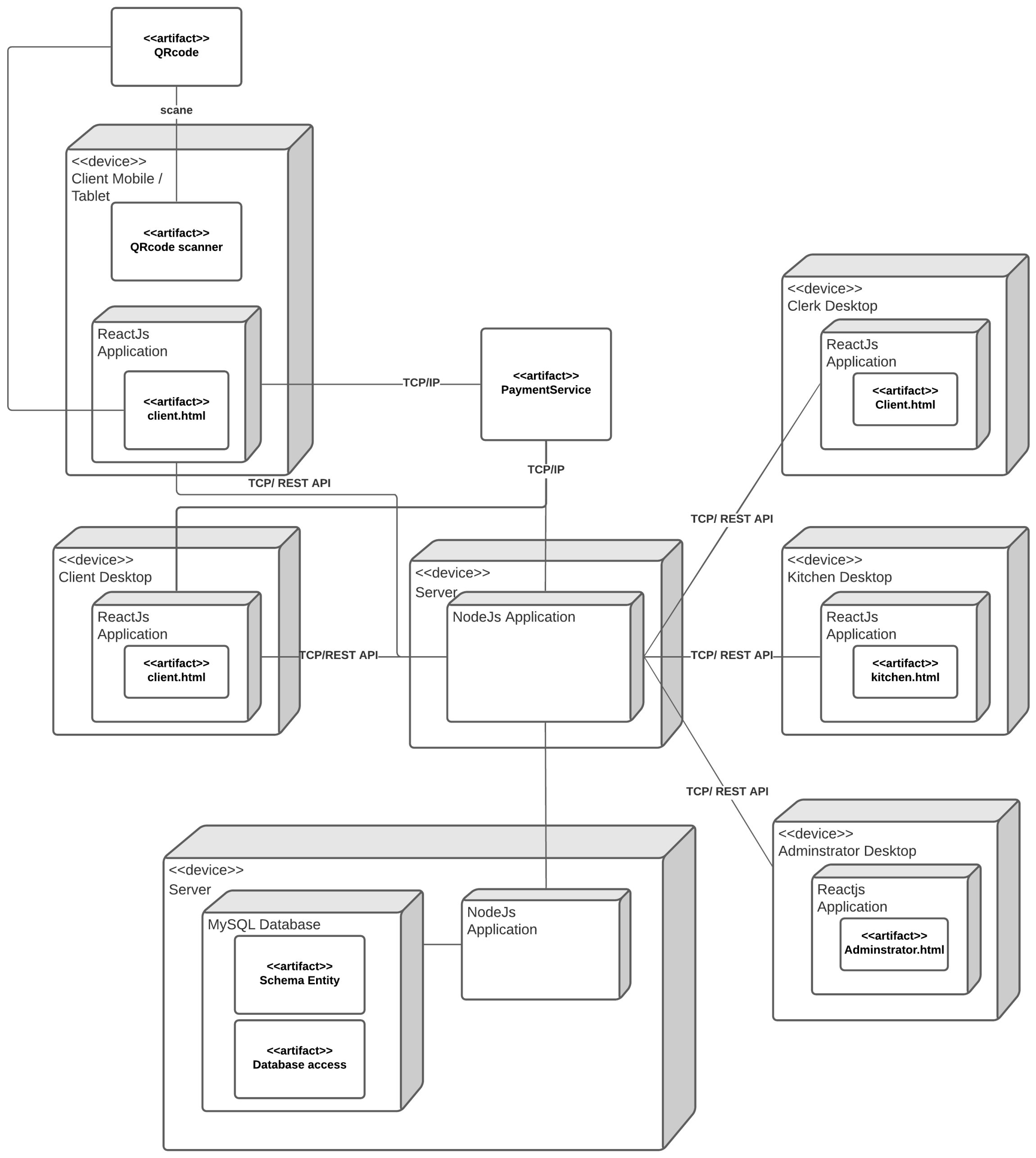
Webstore component:

- This is used to manage the process from order food to receiving food.
- This consists of the Menu, Shopping Cart, Authetication and Payment component.
- This system provides an interface for processing order.
- This system provides Order and requires Retrieve Food, Manage customer, Manage Payment and Update Transaction

Accounting Component:

- This is used to manager order record and account.
- This system consists of the OrderRecord component, Account component.
- This system provides an interface for managing customer.
- This system also requires order record and food's management.

4.3 Deployment diagram:



Description:

- On the client side, the system will be set up on 5 devices, including: Client's Mobile/ Tablet, Client, Clerk, Kitchen and admin desktop. The reactJS application will be run on the user side to handle some simple operations.
- On the system side, An intermediate server is used to provide the page for the Client and handle the business logic. The other server is used to set up the database. A NodeJS application will be run on each Server to handle the request streams from the system.
- Devices will be connected to each other by calling APIs via TCP/IP protocol.

- The QR code, containing the direct link to the restaurant, will be scanned using a scaneable app installed on the Client's Mobile/Tablet device.
- The system uses a payment service outside the system (eg, a bank, ...)

References

[1] Ian Sommerville. *Software Engineering*. Pearson, 2014.

[2] *Chapter 4. Use-Case Diagrams :: Part II : Structural Modeling :: Learning UML :: Programming :: eTutorials.org*. Accessed: 20/09/2021. Available at:

http://etutorials.org/Programming/Learning+uml/Part+II+Structural+Modeling/Chapter+4.+Use-Case+Diagrams/

[3] *The «include» and «extend» Relationships in Use Case Models*. Accessed: 20/09/2021. Available at:

https://www.karonaconsulting.com/downloads/UseCases_IncludesAndExtends.pdf?fbclid=IwAR2alWmo4pxVT2aECXvovmUKr6QICXBPsFwIvmbKpnxjwMrtrDM_8GOLM3o

Scott, E. A. (2016). Spa design and architecture: Understanding single-page web applications. Manning.