Group 11
Blazn_Seven

Milestone 5: Updated Design and Near-Complete Implementation

1. Glossary

AddCustomers: This is a class in the database package that deals with adding a new customer to the database of the customers.

AddDishes: This is a class in the database package that deals with adding a new dish to the database of the dishes.

addEmail: This is a function that takes in the email as a string. It takes in email as the parameter and it returns nothing.

AddLocation: This is a class in the database package used to insert location information for a user.

addLocations: This function is used to add a new location. The parameters are user_id, house_num, street, city, provice, post_code and they are for the location information.

AddMenu: This is a class in the database package used to add menu of a restaurant.

AddMenu: This is a function used to add menu information using restaurant_id and menuName which are the parameters.

AddOrderLines: This is a class in the database package that deals with tracking the number of orders placed for meals.

AddRestaurants: This is a class in the database package that adds restaurants to the system database.

AddUser: This class in the database package that is used to add a user, this should be before add customer and restaurant.

addUser: This is a function that is used for inserting a user to the database using username, userPassword, status as parameters.

CCNumberVerification: This is a class in the payment package that verifies the validity of the credit card number entered during payment. It has 2 global variables called CCNumber which is of type string and STD_CCN_DIGITS which is of type int.

Connect: This is a function that is of type Connection, this function connects to the database.

Credit: This is a class in the payment package that takes in the card details and checks if the date is a valid date necessary for use.

Customer: This is a class in the Entities package that has the customer name, phone number, email, and other identification details.

CVVCodeVerification: This is a class to verify the validity of CVV Code of the Credit Card.

DateVerification: This is a class in the payment package that is used to verify the validity of Expiry Date of the Credit Card.

displayAllMenus: This is a function displays menus in the database on the console. The function queries the sql database for the name of all food in the database. The function returns nothing.

DisplayDishes: This is a class in the database package that is used for the view of the different dishes available.

displayDishes: This is a function that prints all the dishes unto the console. This function returns nothing.

displayItem: This function prints out all items for a given menu. The function has menuID as the only parameter.

DisplayMenus: This is a class in the database package that shows all the menus on the console.

DisplayRestaurant: This is a helper class in the database package that helps to check every restaurant in the sql database.

displayRestaurantFromID: This is a class function that displays the restaurant derived from the ID using restID which is the only parameter.

execute: This is a function that returns a list of Ids. It takes in the input, connection, getInt as its parameters.

executeName: This is a function that is called when the output is a list of string. It has input, connection, getString as the parameters.

ExecuteNoInput: This is class in the Search_Sort package is called when the search object is null.

ExecuteWithInput: This is a class in Search_Sort package that works when the search object is not empty.

findRestaurantSameCity: This is a function dimply used to get the restaurant nearby (in the same city) a customer. It gets the restaurants found into a 2-d array and using customer_id as its only parameter.

Food: This is a class in the Entities package that has meal names and prices and preparation time.

FOS: The Food Ordering System is a system that provides an easy way to access different restaurants and their different menu and helps to place an order for the particular meal.

getCustomerAddress: This is a function that is used to find the customer address, so we can use google function to calculate the distance between them. It uses customer_id as its only parameter and it returns nothing.

getDishesID: returns the dish Id.

getDistance: This class in the Google package is using Google API to get distance between 2 positions.

getDistanceString: This is a function that returns the string of distance between origin and destination. It takes origin and destination as the parameters.

getEmail: this is a function that returns the email inputted.

getEmail: This function returns a string of the email of the user.

getFirstName: This function returns the first name of the user.

getFloatDistance: This is a function that returns the float of a distance between origin and destination. It takes origin and destination as the parameters.

getID: This function returns the customer id.

getLastName: This function returns the last name of the user.

getName: This function returns the name of a particular food item.

getName: This is a function in the ExecuteWithInput class that generates the complete name of the food the customer searched for. This function has Query, connection, search, getString as its parameters and its returns the string generated from the database.

getPhoneNumber: This function returns a string of the phone number of the user.

getPrefFood: This function returns the food preparation.

getPrepTime: This function returns the preparation time for a given food item. It returns nothing.

getPrice: This function returns the price.

setName: This function sets a new name to new food item. It returns nothing.

getUser_id: This function returns the user id.

getRestaurantId: This is a helper function to check the restaurant_id by using license_id. It returns an integer of the restaurant id and has license_id as its only parameter.

getRestaurantName: This is a function that gets the restaurants name using the restID which is the only parameter.

GoConnection: This is a class in the database package that is used to make a connection to the sql db.

insertDatabase: This function has the ability to insert customer, calls method from another package.

isValid: This is a Boolean function that takes in the string email as a parameter and this function makes sure that the email is of a valid type.

Login: This is a class in the database package, and it main purpose is to make a log in function, help to check if the username and password that the user input are correct or not.

login: This is function that checks the username and password, it has the username and password as the parameters. This function returns nothing. --?

messageCode: This is a function that returns a randomly generated message/code of type string of 7 characters.

NearbyRestaurant: This is class in the database package used for finding the nearest restaurants.

queryForRestaurantName: This is a helper function that select the query for the restaurant name.

restaurantNearBy: This is a function to return the restaurant name and distance for a customer by a 2-d array. It has customer_id as its only parameter and returns an array list of the restaurants nearby.

SearchDishes: This is a class in database package used to query the dishes table for a dish with the name matching the string passed.

SearchMetric: This is a search filter that filters through the system database depending on the customer's interest. This search metric could either filter via distance metric, ratings metric, price metric, and waiting time metric.

SearchRestaurant: This is a function to search a restaurant by a name that given by users using restaurantName as a parameter. The search cannot be empty or null. This function returns nothing.

SearchRestaurants: This is a class in the database package used to search a restaurant by name.

Send: This is a function that takes in the returned string from the messageCode function and the email. It sends the messageCode and sends it to the email. This functions returns nothing.

setCustomerInfo: This function is used to add a customer's information to the database. It contains firstName, lastName, phone number, email and prefFood as its parameters.

setDishInfo: This function is used to add a new dish information. The parameters are menus_id, dish_name, dishe_price, pptime and they are the dish information.

setFirstName: This function has first name as the only parameter and sets the first name to the user.

setLastName: This function has the last name as the only parameter and sets the last name to the user.

setNewPrefFood: This function has a food as the only parameter and sets the preparation time to the new food.

setOrderLineInfo: This is a function sets the order line information. It returns nothing. This function has order_id, dishes_id, quantity, pricePerUnit, priceTotal, discount_total as the parameters.

setPhoneNumber: This function has a new number as the only parameter and sets the new phone number to the user.

setPrepTime: This function has newPrepTime as the only parameter. It sets the preparation time and returns nothing.

setPrice: This function sets a new price to the food item. It returns nothing and throws an error if the price is set at negative.

setRestaurantInfo: This is a function that is used to insert the parameters for setting a restaurant information using restName, licenseId, openTime, closeTime, phone_num, emailAddress as the parameters.

SignUp: This is a class that deals with the verification of the email provided by the customer. It sends a random code to the customer's provided email address.

Sort: This is a class in Search_Sort package that helps to filter and sort searches and results either by rate, distance, price, waiting time.

SortByDistance: This function sorts result by distance from the user using the search as parameter.

SortByPrice: This function sorts results by the price either ascending or descending order depending on the preference of the customer. This function has the array list of input result as the parameter.

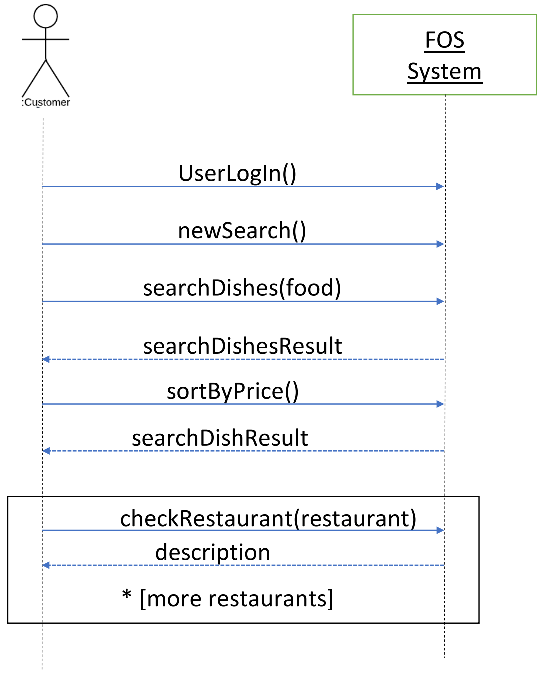
SortByRate: This function sorts result by rate, using search as a parameter.

SortByWaitingTime: This function sorts results by the waiting time for each dish from fastest time to latest time depending on the preference of the customer. This function has search as the parameter.

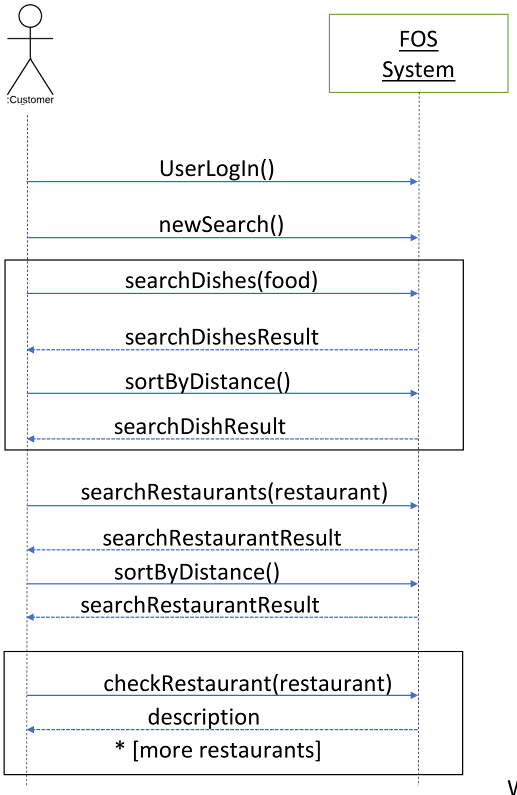
System Diagram: this is a model used to visually express the dynamic forces acting upon the components of a process and the interactions between those forces.

System Sequence Diagram (SSD): This is a sequence diagram that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.

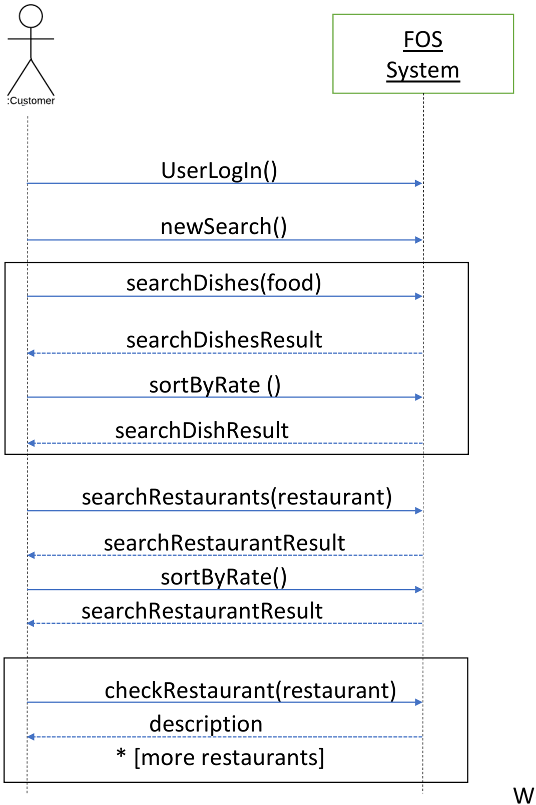
2. System Sequence Diagrams



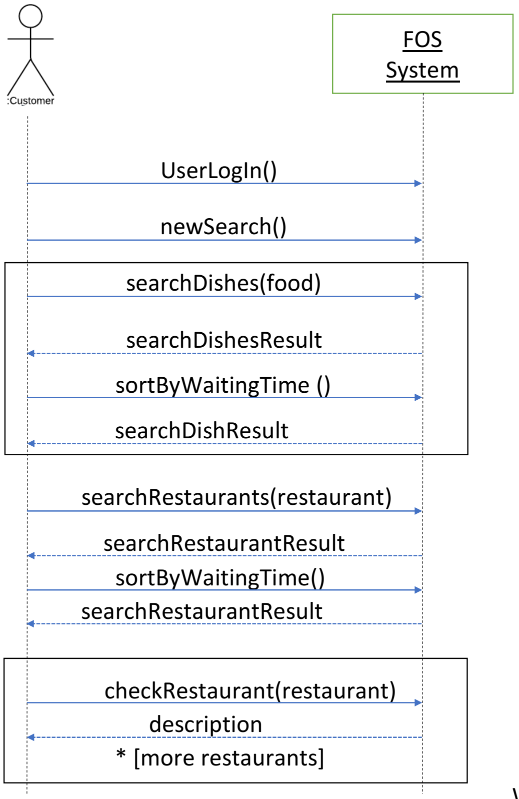
Word


:Customer
FOS System
UserLogIn()
newSearch()
searchDishes(food)
searchDishesResult
sortByDistance()
searchDishResult
searchRestaurants(restaurant)
searchRestaurantResult
sortByDistance()
searchRestaurantResult
checkRestaurant(restaurant)
description
* [more restaurants]

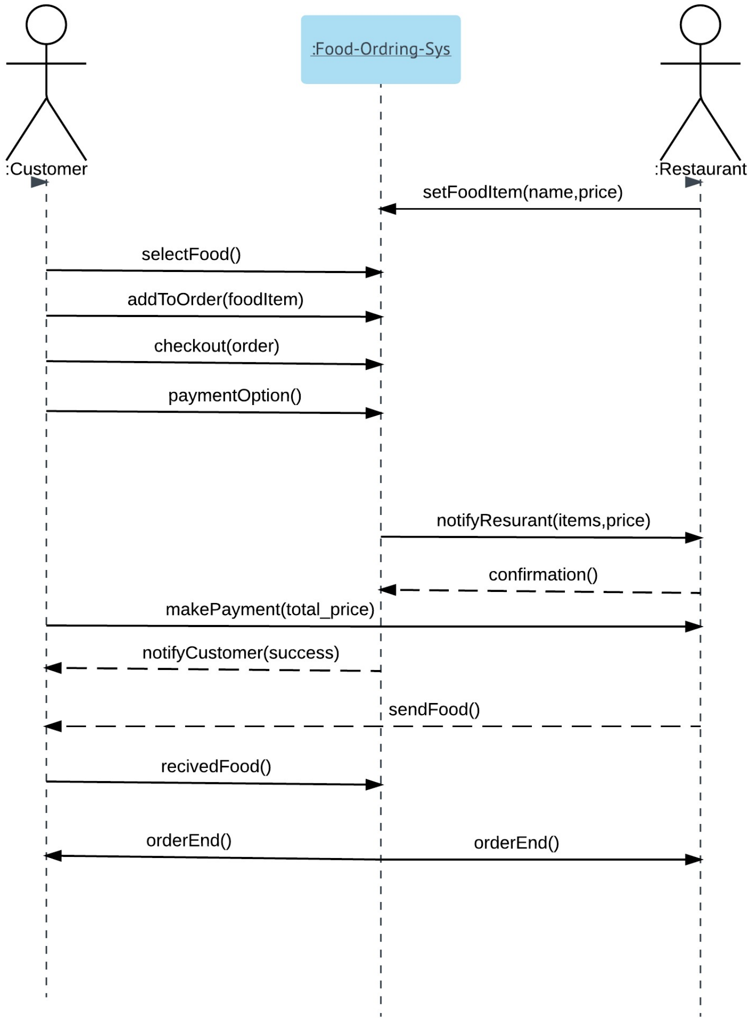
Word


:Customer
FOS System
UserLogIn()
newSearch()
searchDishes(food)
searchDishesResult
sortByRate ()
searchDishResult
searchRestaurants(restaurant)
searchRestaurantResult
sortByRate()
searchRestaurantResult
checkRestaurant(restaurant)
description
* [more restaurants]

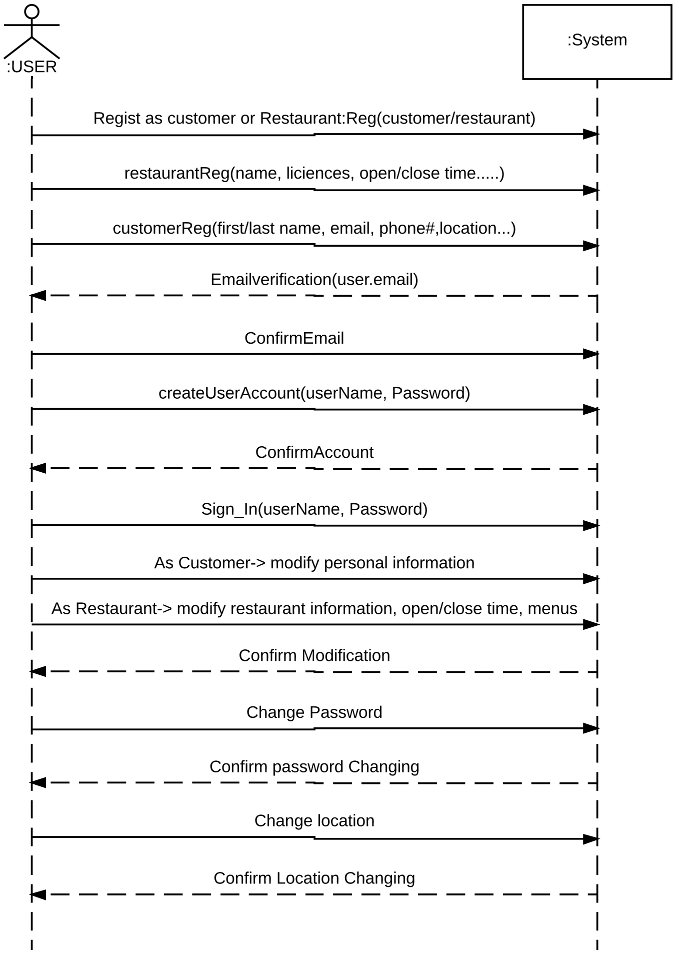
Word


:Customer
FOS System
UserLogIn()
newSearch()
searchDishes(food)
searchDishesResult
sortByWaitingTime ()
searchDishResult
searchRestaurants(restaurant)
searchRestaurantResult
sortByWaitingTime()
searchRestaurantResult
checkRestaurant(restaurant)
description
* [more restaurants]

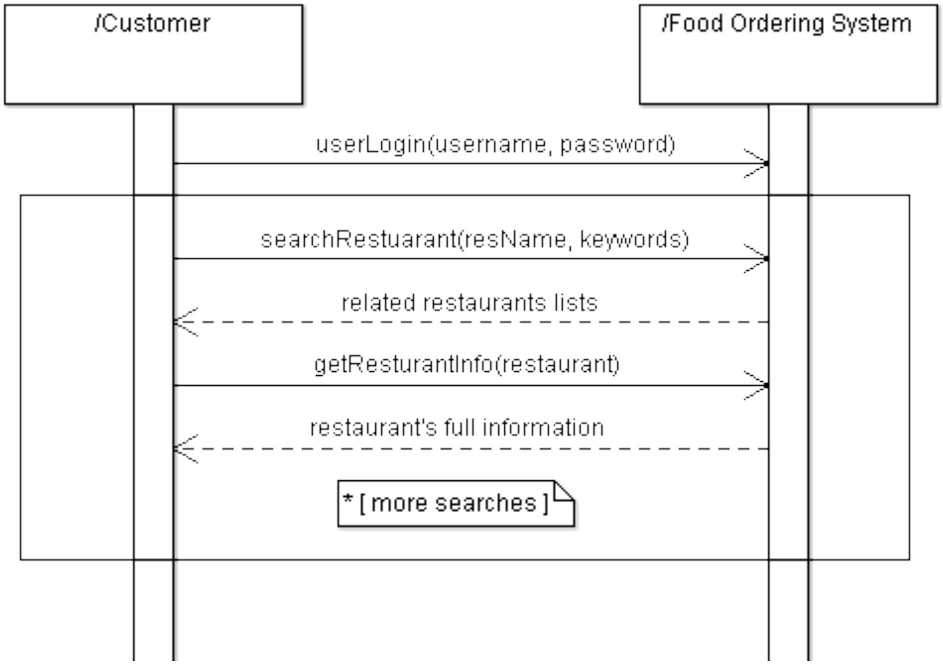
Word



www.lucidchar.com


:USER
:System
Regist as customer or Restaurant:Reg(customer/restaurant)
restaurantReg(name, liciences, open/close time.....)
customerReg(first/last name, email, phone#,location...)
Emailverification(user.email)
ConfirmEmail
createUserAccount(userName, Password)
ConfirmAccount
Sign_In(userName, Password)
As Customer-> modify personal information
As Restaurant-> modify restaurant information, open/close time, menus
Confirm Modification
Change Password
Confirm password Changing
Change location
Confirm Location Changing

www.lucidchart.com


/Customer
/Food Ordering System
userLogin(username, password)
searchRestuarant(resName, keywords)
related restaurants lists
getResturantInfo(restaurant)
restaurant's full information
* [more searches]

3. Operation Contracts

Operation: confirmEmail()
Cross References: Uses Case: Account Creation
Pre conditions: NIL
Post conditions: -An email has been sent to the new user
-User can use information in the email to confirm account creation

Operation: customerReg(**firstName**: String, **lastName**: String, **email**: userEmail**, phoneNum**: userPhoneNumber, **location**: userLocation)
Cross References: Use Case: Account Creation
Pre conditions: NIL
Post conditions: -Customer c was created (instance creation)
-c was associated with Users (association formed)
-A confirmEmail() operation created

Operation: validateSearch(**queryedItem**: string)
Cross References: Use Cases:
Restaurant Search – Check's whether customer's search is correctly entered
Pre conditions: **-**NIL
Post conditions: **-**Message returned that search was successful. Food Ordering System application will proceed to query the database.

Operation: searchDatabase(**queryedItem**: string)
Cross References: Use Cases: Restaurant Search – Searches the FOS Database queried restaurant.
Pre conditions -NIL
Post conditions: **-**Message returned that search was successful (or unsuccessful).
-The database will send the restaurant's information to the application. The application will format the results in an easy to use UI.

Operation: restaurantReg(**name**: string, **licence**: string, **open_time**: string, **close_time**: string)
Cross References: use case: Account Creation
Pre conditions: NIL
Post conditions: -Restaurant r was created (instance creation)
-r was associated with Users (association formed)
-A confirmEmail() operation created

Operation: modify_information()
Cross References: uses cases: User Account Creation and Log In
Pre conditions: - User is logged into their account
Post conditions: - User can go through all the personal information of themselves
- User can change their password in this operation
- User can change their location in this operation
- User can change their personal information in this operation

Operation: selectFood()
Cross References: Uses Cases: Placing an Order
Pre conditions: - search instance s exists
Post conditions - A order item instance oi was created
- oi was associated with the current Search based on searchSpecification match

Operation: checkout(**order**)
Cross References: Uses Cases: Placing an Order
Pre conditions: - Customer has selected food
Post conditions: - An Order instance o was created (instance creation)
- o was associated with the menu database (association formed)
- order_total in o was initialized (attribute modification)

Operation: addToOrder(**foodItem**)
Cross Reference: uses cases: Placing Orders
Pre conditions: - NIL
Post conditions: - A food item has been added the customer's chart.
- An order price total been initialized or updated.

Operation: makePayment()
Cross Reference: Use Cases: Placing Orders
Pre conditions: - A customer has confirmed the payment options and price
Post conditions: - An order has been processed
- System will notify the restaurant to prepare the food and transfer the money to the restaurant
- The restaurant will prepare the food in the order after the transaction complete

Operation: SortByPrice()
Cross References: Uses Cases: Search for Price
Pre conditions: - User is logged in
Post conditions: - A list of applicable restaurants shown to users

Operation: SortByWaitingTime()
Cross References: Uses Cases: Search for Waiting Time
Pre conditions: - User is logged in
Post conditions: - A list of applicable restaurants shown to users

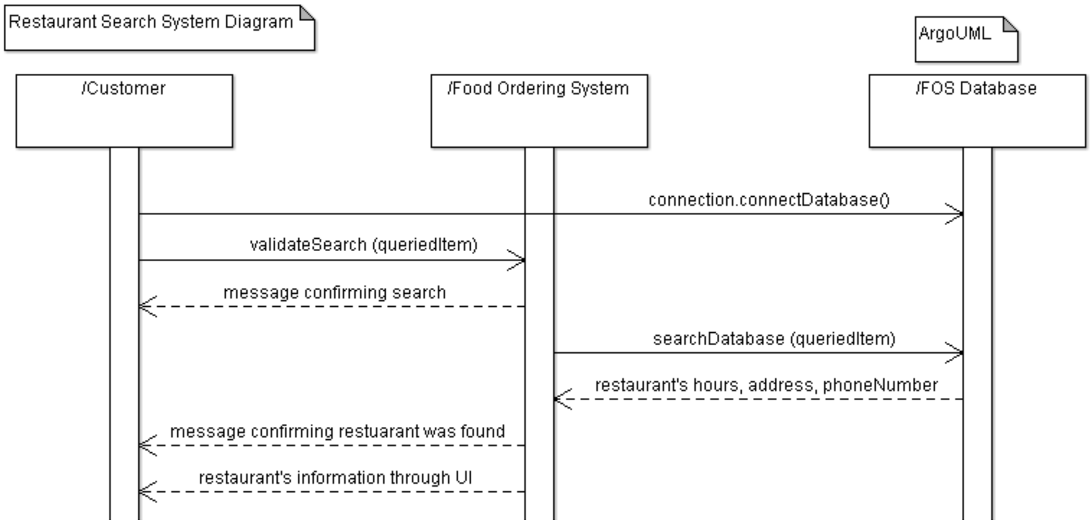
Operation: setFoodItem(**name:** string, **price:** float)
Cross Reference: Uses Cases: Placing an Order
Pre condition: -Item must have valid input values
Post condition -A Food instance f was created (instance creation)
-f was associated with Restaurant (association formed)

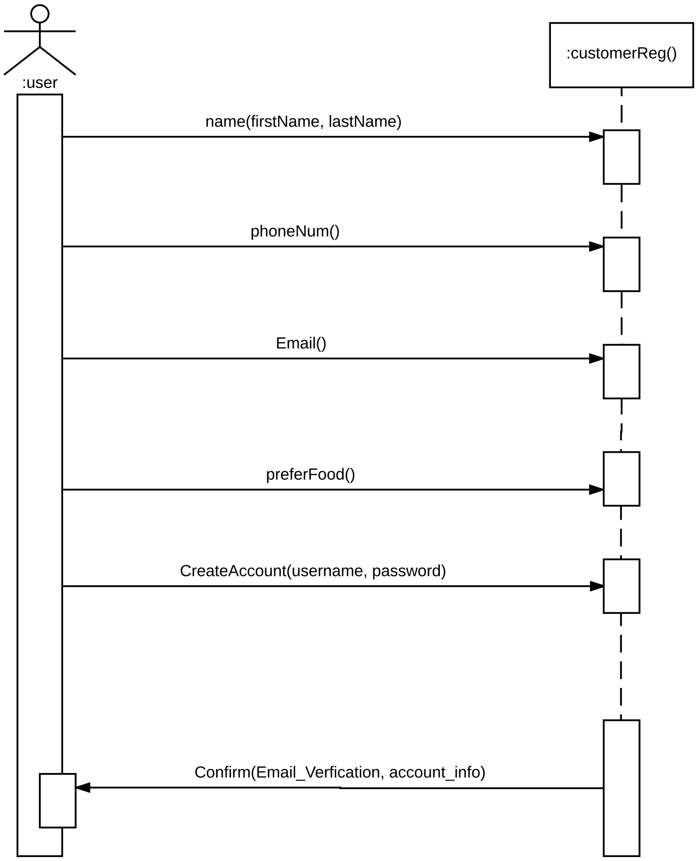
Operation: sign_in(**username:** string, **password:** string)
Cross Reference: Uses Cases: Placing an Order, Search for Waiting Time,
Search for Price, Search for Rating, Search for
Pre condition: -User exists in the database
Post condition: -User is logged in if information matches what is in the database

4. System Operations

SortByDistance() operation->sort the restaurant/dish list by the distance between customer and restaurant
SortByRate() operation->sort the restaurant/dish list by the restaurant rate
SortByWaitimeTime() operation->sort the restaurant/dish list by the average waiting time of the restaurant
SortByPrice() operation->sort the dish list by the price of the dishes
NearbyRestaurants() operation->find the restaurants located at the same city with customer
bubbleSort() operation->called by Sort function, sort the given list with bubble sort
Reg() operation->choose register as customer or restaurant
customerReg() operation-> register as customer
RestaurantReg() operation-> register as restaurant
sign_in() operation-> allowed user to log in the system
personal() operation-> allowed user modify their profile.
makePayment(total_price) **->**operation user pays for their order, with the price from the food items
addToOrder(foodItem) operation **->** puts a selected food item into the users open order
checkOut(order) operation -> user starts the process of purchasing their order

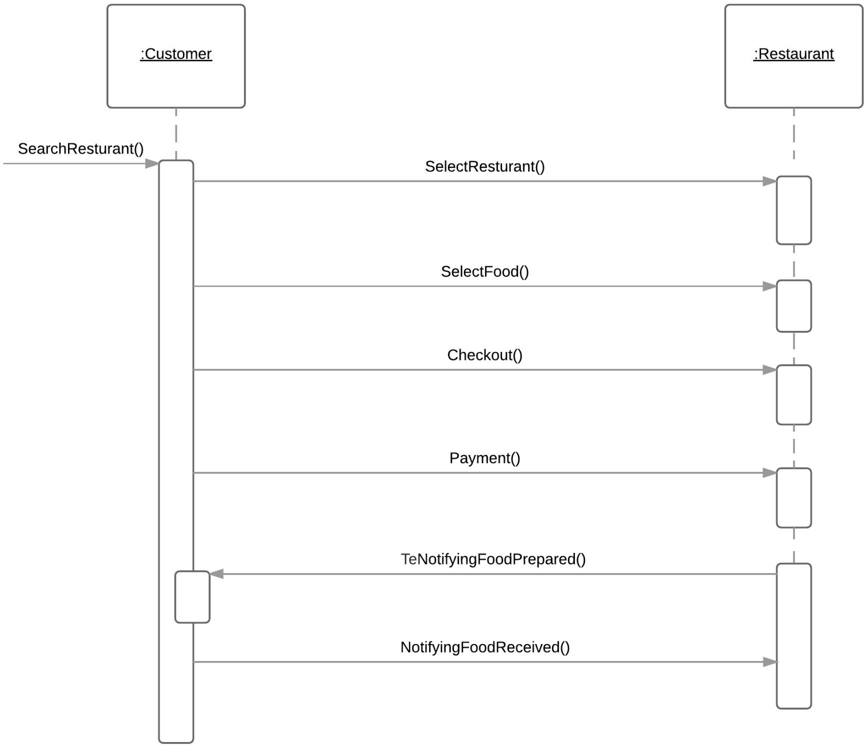
5. Sequence Diagrams


Restaurant Search System Diagram
ArgoUML
/Customer
/Food Ordering System
/FOS Database
connection.connectDatabase()
validateSearch (queriedItem)
message confirming search
searchDatabase (queriedItem)
restaurant's hours, address, phoneNumber
message confirming restuarant was found
restaurant's information through UI


:user
:customerReg()
name(firstName, lastName)
phoneNum()
Email()
preferFood()
CreateAccount(username, password)
Confirm(Email_Verfication, account_info)

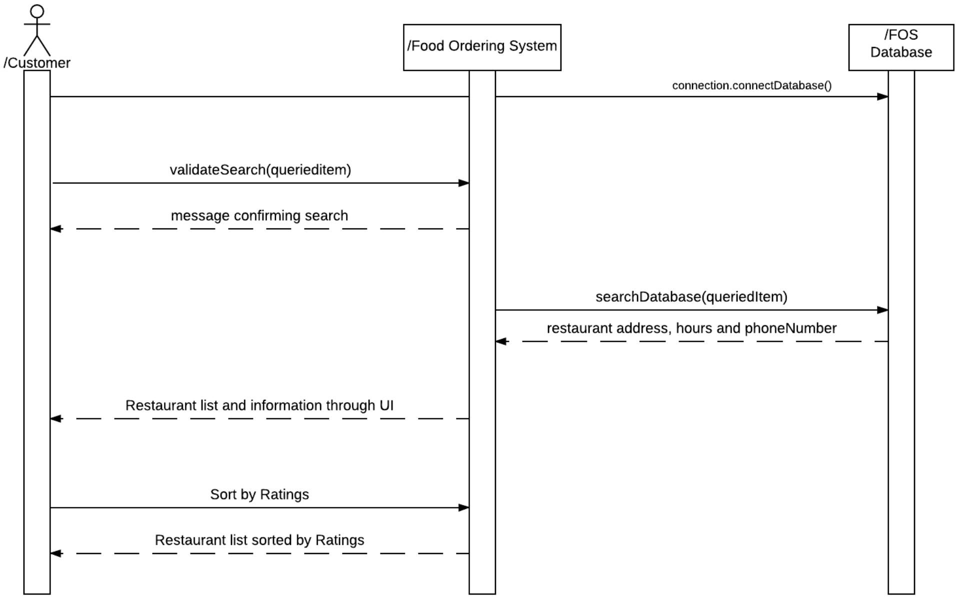
www.lucidchart.com

Place an Order Squence Diagram

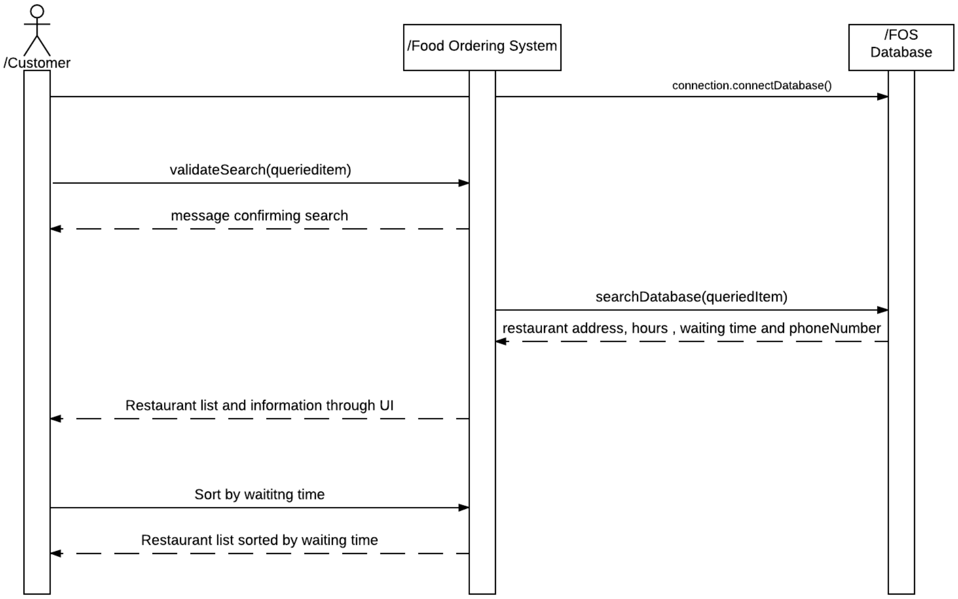


www.lucidchar.com

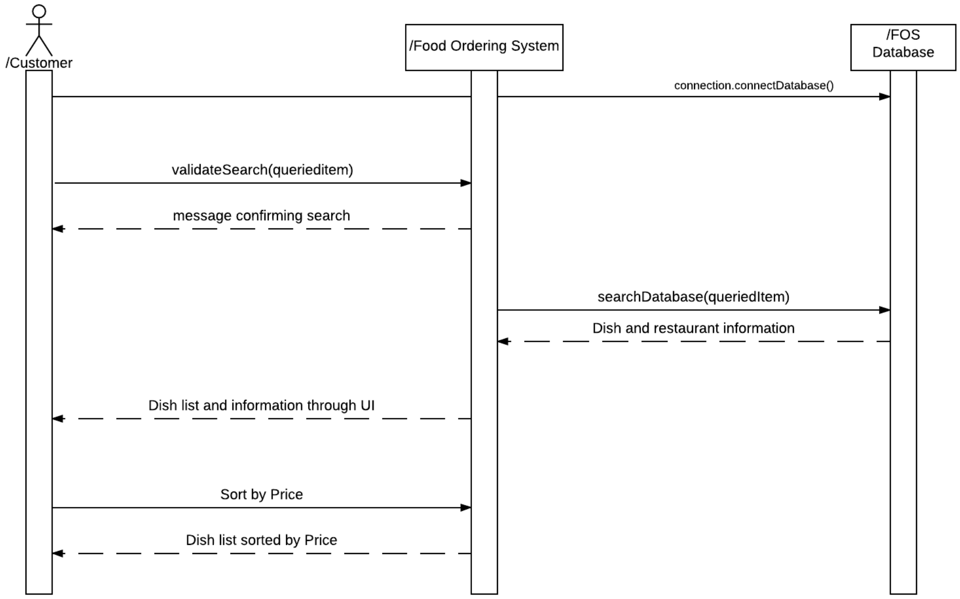
Sort by Ratings Sequence Diagram


/Customer
/Food Ordering System
/FOS Database
connection.connectDatabase()
validateSearch(querieditem)
message confirming search
searchDatabase(queriedItem)
restaurant address, hours and phoneNumber
Restaurant list and information through UI
Sort by Ratings
Restaurant list sorted by Ratings

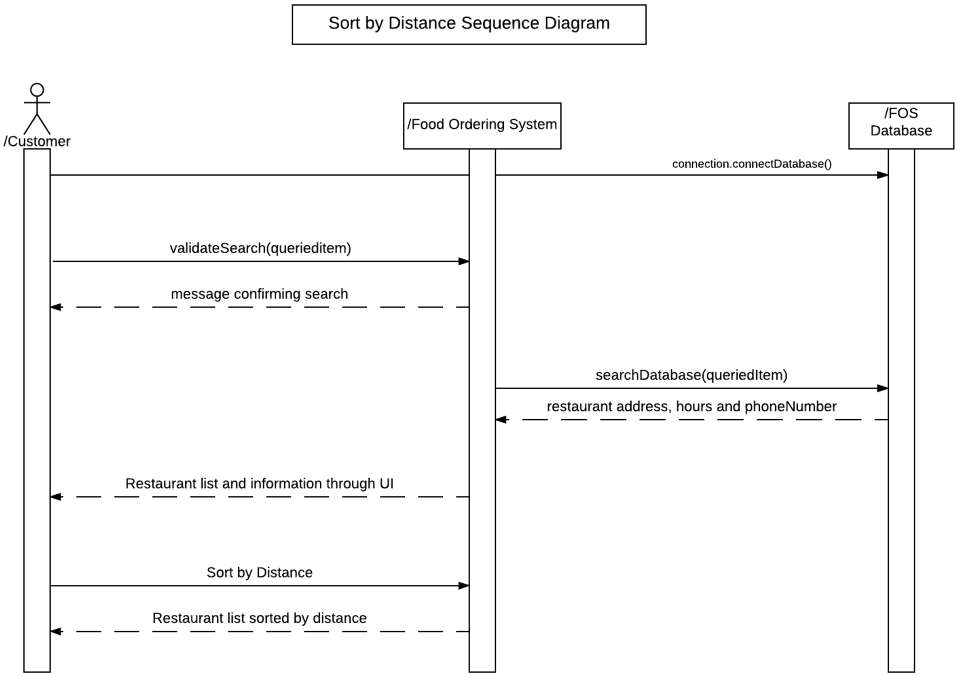
Sort by Waiting Time Sequence Diagram


/Customer
/Food Ordering System
/FOS Database
connection.connectDatabase()
validateSearch(querieditem)
message confirming search
searchDatabase(queriedItem)
restaurant address, hours , waiting time and phoneNumber
Restaurant list and information through UI
Sort by waititng time
Restaurant list sorted by waiting time

Sort by Price Sequence Diagram

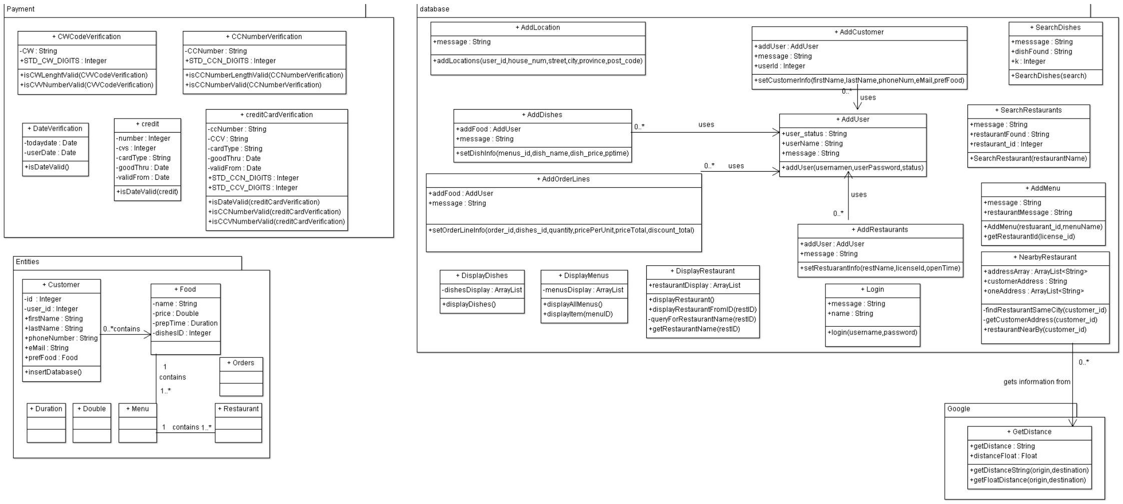

/Customer
/Food Ordering System
/FOS Database
connection.connectDatabase()
validateSearch(querieditem)
message confirming search
searchDatabase(queriedItem)
Dish and restaurant information
Dish list and information through UI
Sort by Price
Dish list sorted by Price

www.lucidchart.com


Sort by Distance Sequence Diagram
/Customer
/Food Ordering System
/FOS Database
connection.connectDatabase()
validateSearch(querieditem)
message confirming search
searchDatabase(queriedItem)
restaurant address, hours and phoneNumber
Restaurant list and information through UI
Sort by Distance
Restaurant list sorted by distance

www.lucidchart.com

6. Class Diagram



7f. User Manual

See attached text file

7g. Meetings After Milestone 4

Meeting October 17th
Attendance: Aparna Angu (ara561), Josh Kocur (jak472), Hao Li (hal356), Yinsheng Dong(yid164) and Duke Rong(yur013)
Start Time: 3:30
Duration: 90 minutes
Summary: This meeting we talked about programming concerns as well as addressed UI decisions.

Meeting October 26th
Attendance: Aparna Angu (ara561), Josh Kocur (jak472), Emmanuel Oriade (eoo983), Hao Li (hal356), Yinsheng Dong (yid164) and Duke Rong(yur013)
Start Time: 3:30
Duration: 90 minutes
Summary: This meeting we talked about where we were at with the project and what, in terms of programming we needed for the next milestone.

Meeting Nov 3rd
Attendance: Aparna Angu (ara561), Josh Kocur (jak472), Hao Li (hal356), Yinsheng Dong (yid164) and Duke Rong (yur013)
Start Time: 3:30
Duration: 30 minutes.
Summary: This was our shortest meeting. Duke raised some concerns with our database and, we decided that it would be best to meet again on the following Tuesday to address them

Meeting Nov 7th
Attendance: Aparna Angu (ara561), Josh Kocur (jak472), Hao Li (hal356), Yinsheng Dong (yid164) and Duke Rong (yur013)
Start Time: 3:30
Duration: 90 minutes
Summary: This meeting we addressed conceptual problems with our database. Yinsheng made a couple of small tweaks thereafter.

Meeting Nov10th
Attendance: Josh Kocur (jak472), Emmanuel Oriade (eoo983), Hao Li (hal356), Yinsheng Dong (yid164) and Duke Rong (yur013)
Start Time: 3:30
Duration: 90 minutes
Summary: This meeting was brief in topic. We planned our work for over the break and ensured that everyone understood their expectations

Meeting November 21st
Attendance: Josh Kocur (jak472), Emmanuel Oriade (eoo983), Hao Li (hal356), and Yinsheng Dong (yid164)
Start Time: 5:30
Duration: 120 minutes
Summary: We talked about last minute requirements and put finishing touches on the milestone.

7h. Git Logs

See attached file

8. Project Plan(s)

Task	**Group Member(s)**	**Time Taken**
Glossary	Emmanuel	3 hours
Database Implementation	Josh, Yinsheng	50 hours. (40% Josh, 60% Yinsheng)
Google Maps inclusion	Yinsheng	30 hours
Operation Contracts	All Members	5 hours. (Equal Work)
Class Diagram	Aparna	5 hours
SSD's Rough Copy	Aparna	5 hours
SSD's Good Copy	Eric, Duke, Yinsheng	6 hours (2 Hours Each)
SD Diagrams	Duke, Aparna, Eric, Yinsheng	8 hours (2 Hours Each
Database Search /Sort Functions	Duke	15 Hours
Putting together and finishing smaller parts of Milestone 5	Josh	10 hours
Presentation	All Members (Equal Work)	7
User Interface	Eric	30 Hours.
Use Case Diagram,	Ridwan	4 Hours
Project Logo	Ridwan	1 Hours
Milestone 2, 3, 4 write up and minor task completion	Josh, Aparna	12 Hours (85% Josh, 15% Aparna)
Domain Model	Eric	3 Hours
Prototype Read me	Duke	1 Hour
Meeting tracking, summation	Josh	2 Hours
Milestone 2, 3 Group decisions	All Members (Equal Work)	6 hours
Placing an Order	Yinsheng	10 Hours
Last Minute Code Tweaking	Ridwan, Eric, Yinsheng	15 Hours (Eric 40%, Yinsheng 40%, Ridwan 20%)

New Task	**Group Member(s)**	**Expected Time**
Polish UI	All members	20 Hours
Clean up entities for ease of use	Josh	10 Hours
Optimize the searches	Duke, Yinsheng	15 Hours
Improve Email verification	Emmanuel	15 Hours
Testing	All	10 Hours