IdealMeal

Restaurant Automation

Group 7

Akshat Parmar
Nianyi Wang
Ebad ullah Qureshi
Harman Kailey
Jianhong Mai
Kebin Li
Khizer Humayun
Shuren Xia
Tyler Hong

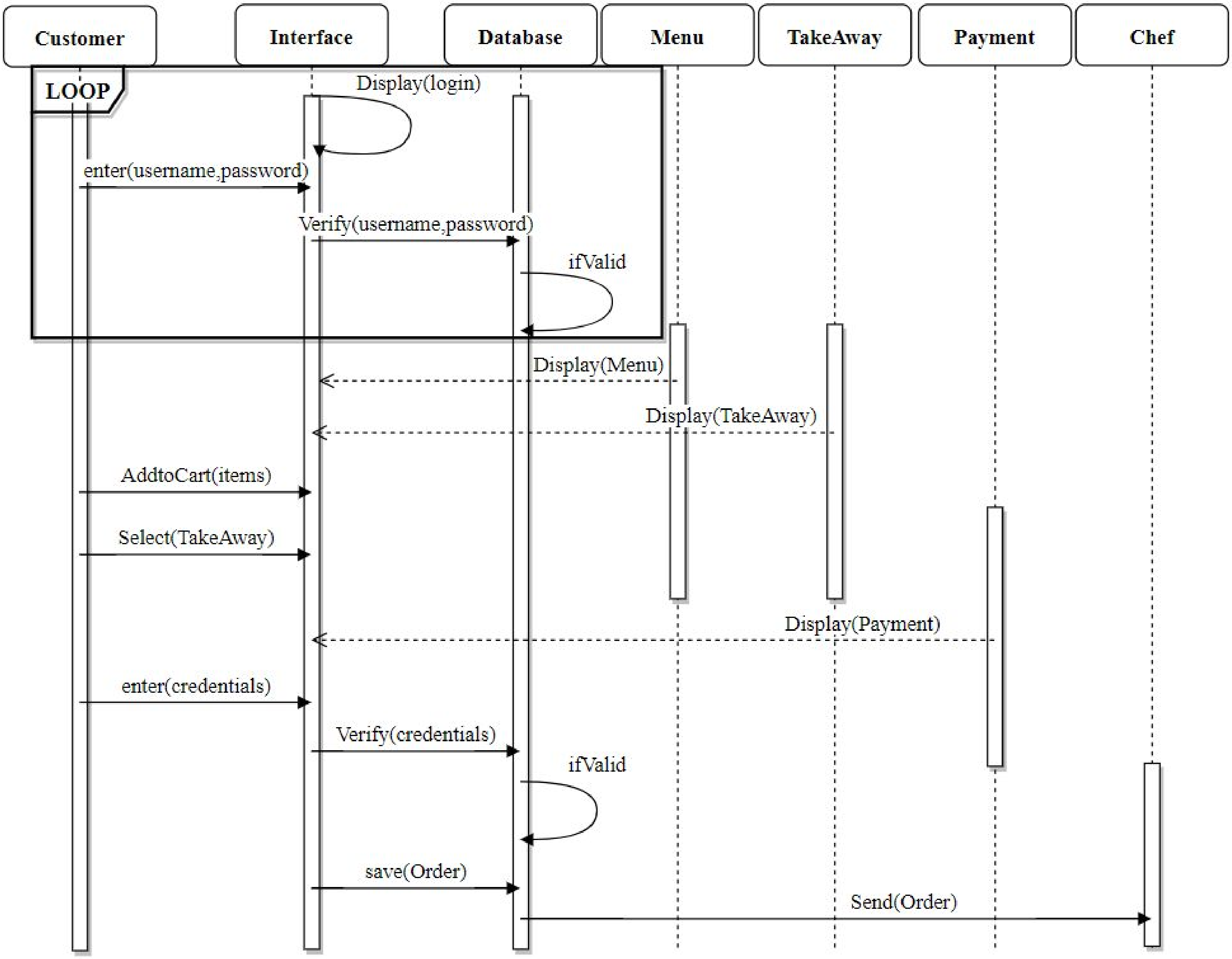
REPORT 2

Individual Contributions Breakdown

Topics	Customer		Chef			Server		Manager	
	Khizer	Ebad	Mai	Wang	Li	Shuren	Harman	Akshat	Tyler
Individual Contributions Breakdown									
Customer Statement of Responsibilities	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Glossary of Terms	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
System Requirements (Functional and non-Functional)	14.7%	14.7%	14.7%	14.7%	14.7%	12%			14.7%
User Interface Requirements	20%	20%	20%			20%			20%
Functional Requirements Specifications	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
User Interface Specifications	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Project Management	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Domain Analysis	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Project Size Estimation	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Interaction Diagrams	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Class Diagram and Interface Specification	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
System Architecture and System Design	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
Algorithms and Data Structures	33.3%	33.3%			33.3%				
User Interface Design and Implementation	33.3%	33.3%			33.3%				
Design of Tests	33.3%	33.3%			33.3%				

8 Interaction Diagrams

Use Case 1: TakeAway

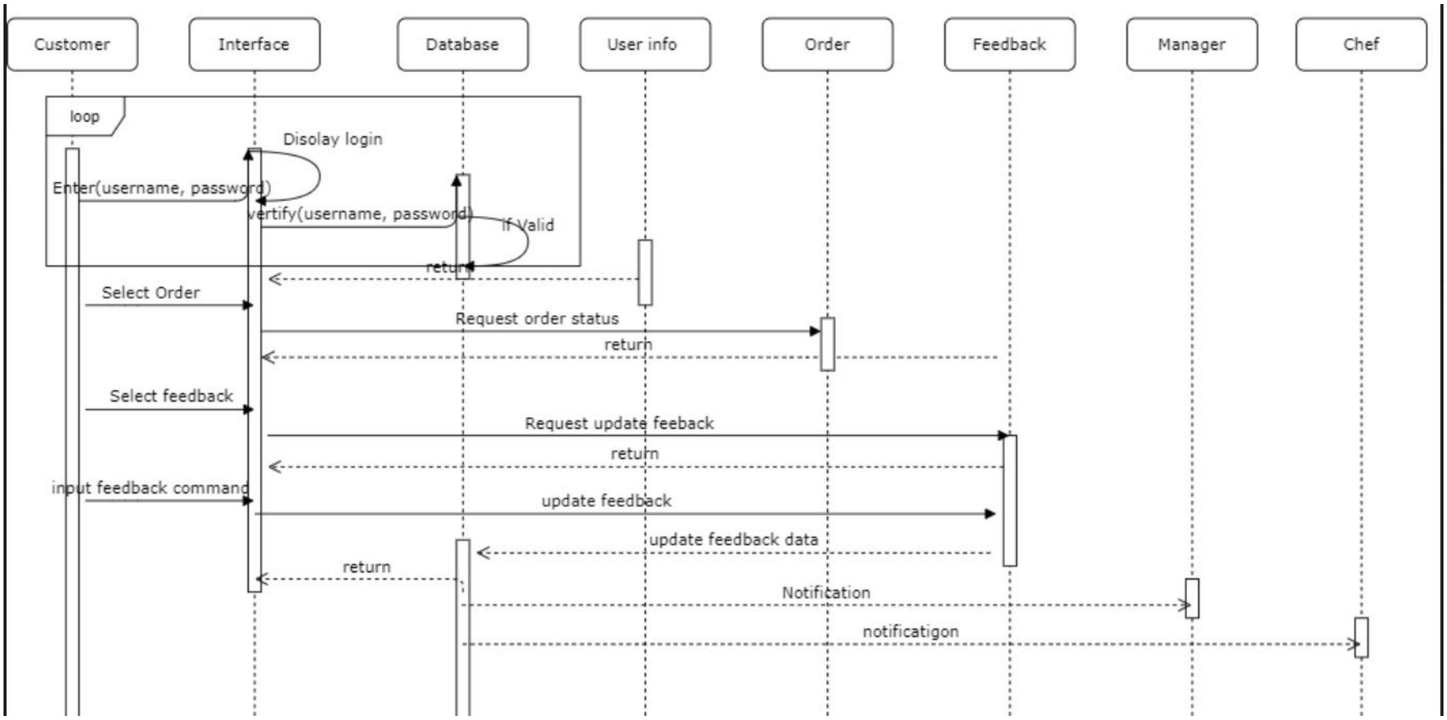


This diagram is for UC1-TakeAway which shows how to place an order for pickup/takeaway. Once users log into their account, their credentials will be verified by the system through the database. If their credentials are valid, the system will prompt the user to the next page which contains menu, reservation, and takeaway options along with other features, which are disregarded for this use case. To place an order for takeaway, users have to add items to cart from the menu by clicking the “Add to cart” button. The takeaway option will then allow users to place an order for pickup. However, in order to complete the takeaway, the system will prompt the user to the payment page through the interface. Users will then input their credit/debit card credentials which will be

verified through the database. After confirmation, the system will upload the order to the database. Subsequently, order will be sent to the Chef interface through the database.
One design principle employed in this sequence is the High Cohesion Principle. The High Cohesion Principle says that an object should not take on too many computational responsibilities. This principle is utilized in that each class only takes on responsibilities that have to do with its specific functionalities and does not take on more than it can handle at a time.

Another design principle used was the Expert Doer Principle: module that knows should communicate information to those that need it. Expert doer principle can be seen in the diagram where each class sends information directly to where it is needed, for instance, payment sends information directly to the interface not to database or any other class.

Use Case 16: Feedback

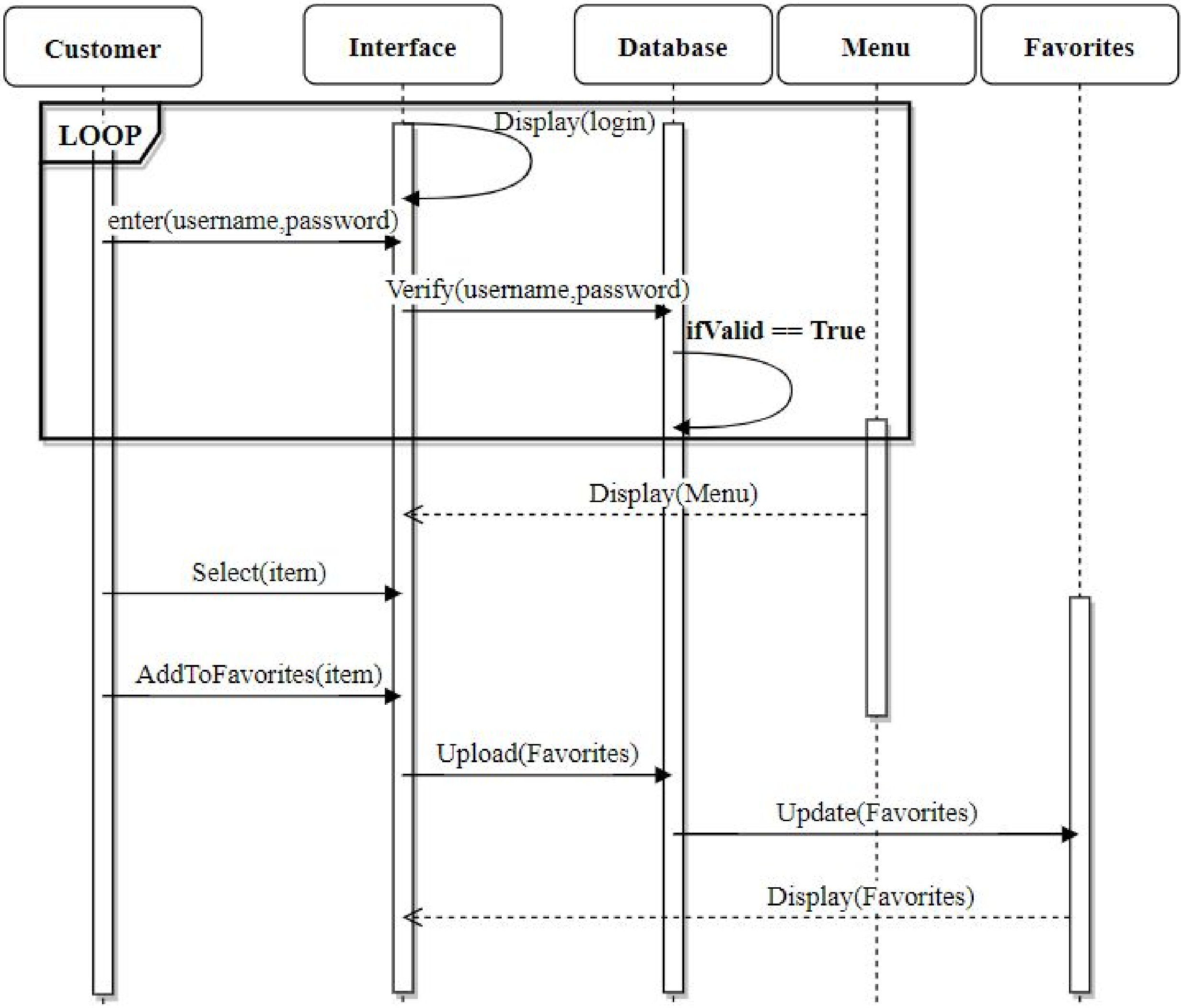


This diagram is for UC16-Feedback which allows the user to update their user customer experience of their order dishes. Once users log into their account, their credentials will be verified by the system through the database. If their credentials are valid, the system will prompt the user to the next page which has the main screen. And customers now can select the order status icon to access their order status. And the order status page will be displayed on the interface. At that moment, customers can type in their feedback in the input window and the feedback info will be saved into our database system. Both Manager and Chef will get notification on the feedback on their food.

The first principle of interface is the Liskov Substitution Principle. The Interface has a login authentication algorithm to keep track of the user's account. If the user login the UI interface, the main screen will have icons which can redirect you to other pages of your account status.

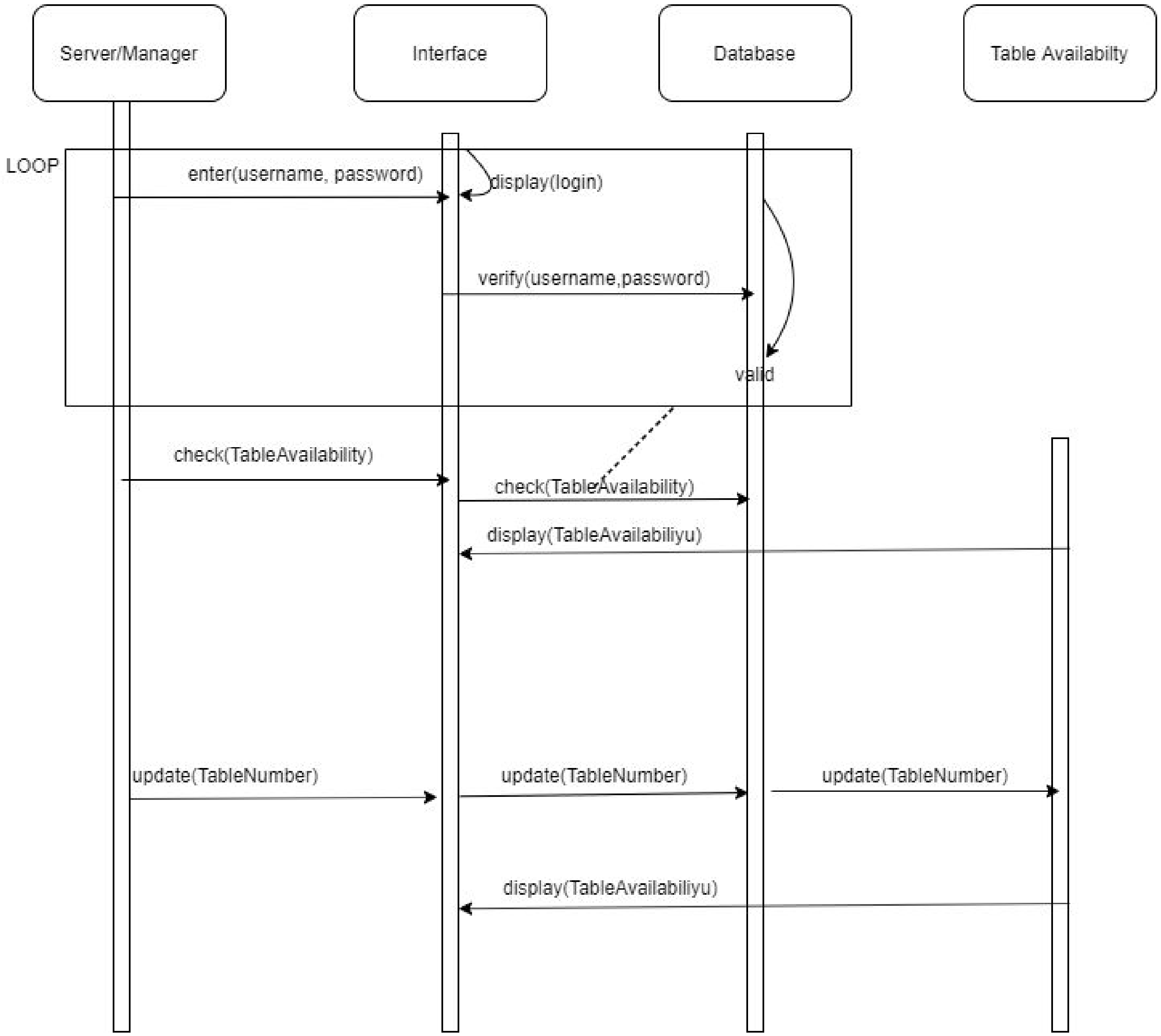
The second principle is the Single Responsibility Principle. We may each method like order status and user Info as simple as possible. So they don't need to have too much massive computation on their modules.

Use Case 18: Favorites



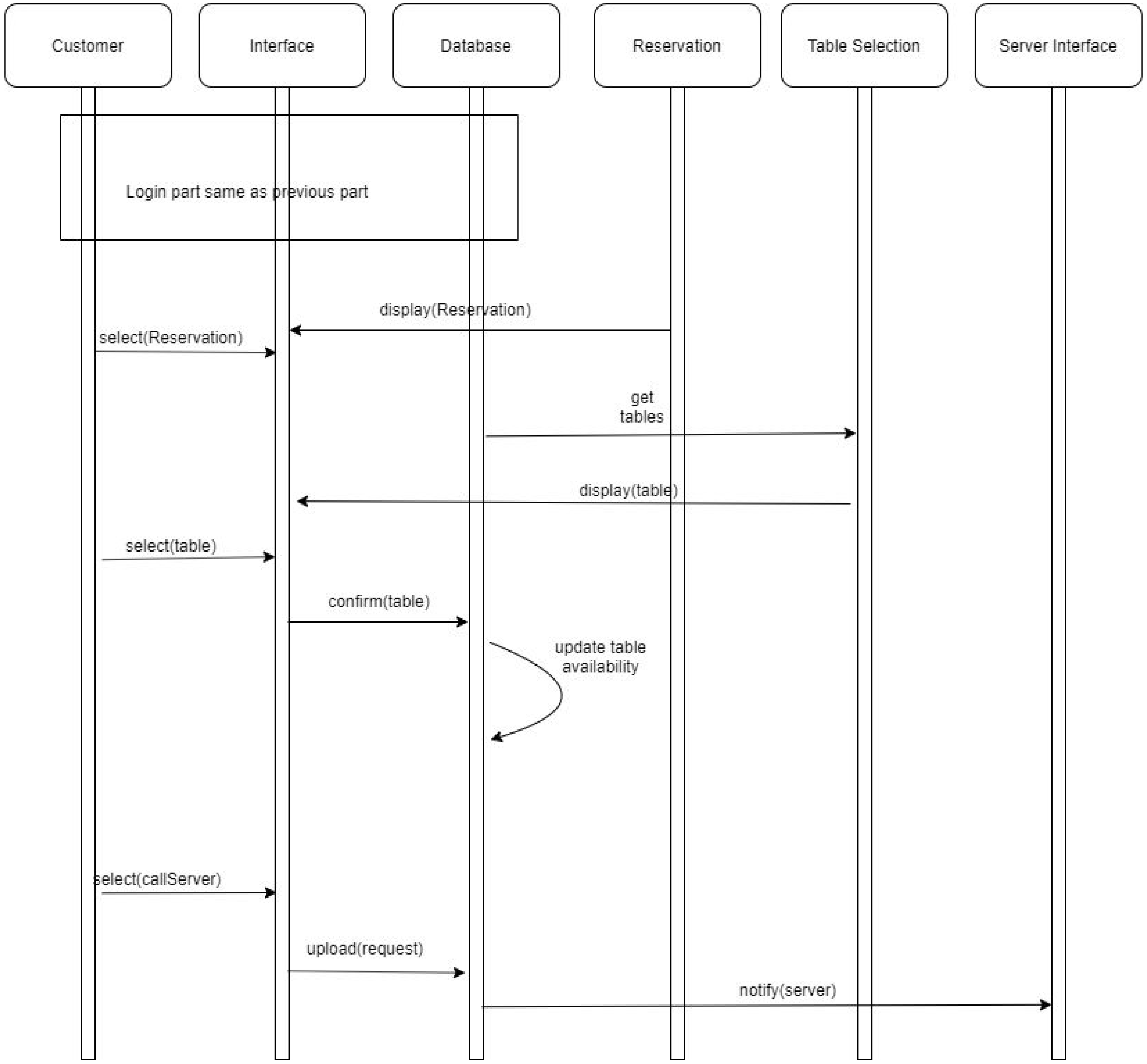
This diagram is for UC18-Favorites which shows how a customer can add an item to favorites from a menu. First and foremost, favorites feature will only work for registered customers and not guests. In order to access Favorites, the user has to log into their account. Once the user logs in, the system will prompt the user to the main page where the user can see different options such as Menu, Favorites, and IMRewards through the interface. If the favorites list is empty, the user can select menu and add items to favorites by clicking the "Add to favorites" button. The favorite item will be uploaded to the database where each customer's information will be stored. After the database is updated, the user will be able to see their favorite items in the Favorites option on the main page.

Use Case 3: Table Availability

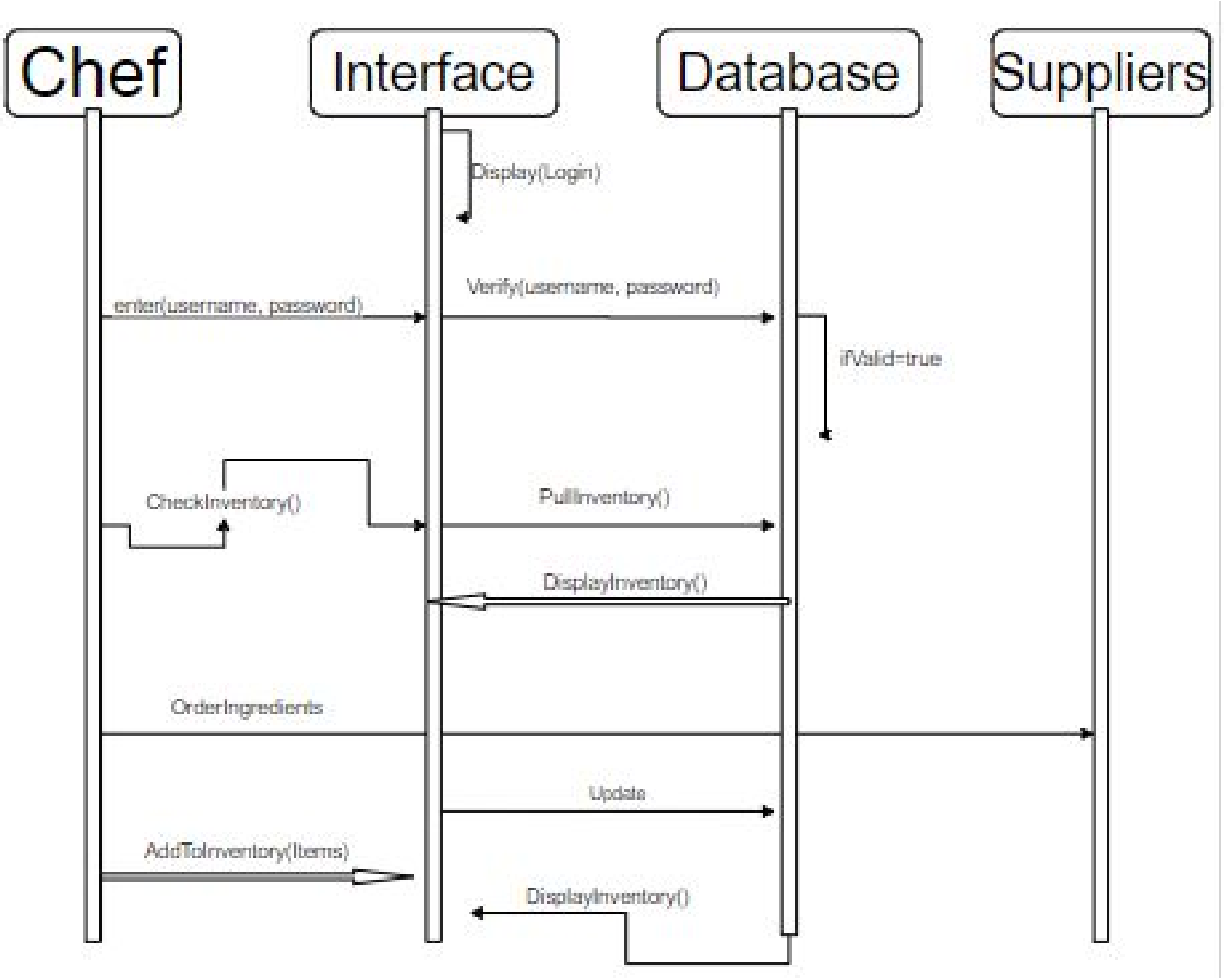


This diagram is for UC-3 Table Availability which shows how the server and manager to view available tables and edit their availability.

Use Case 5: Call Server



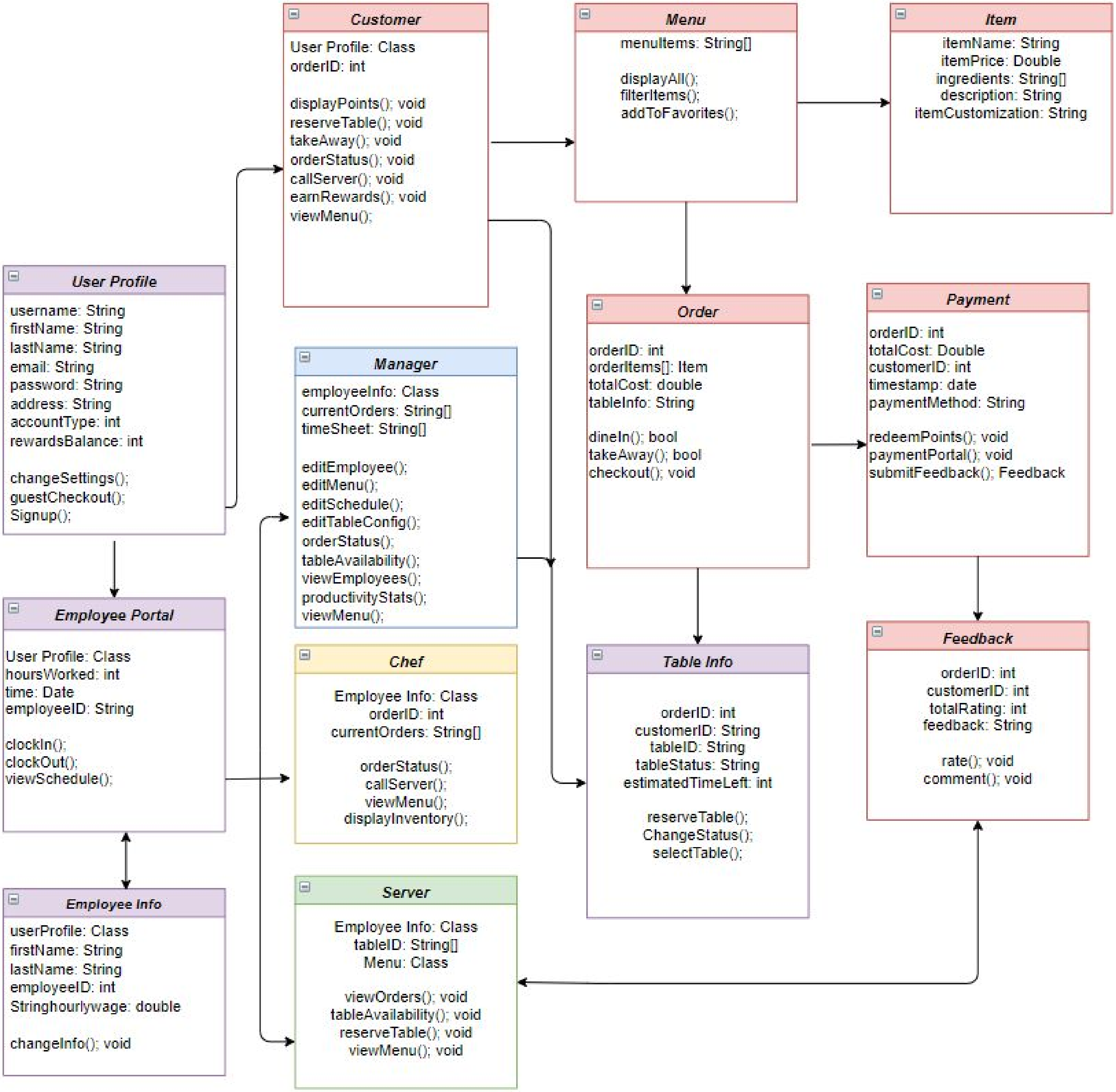
Use Case 17 : Inventory



The above table is the diagram for UC-17. It shows the functions which a chef can use to check the Inventory. The chef will need to log in to check the most uptodate inventory. The database will update the stock of ingredients in the restaurant for all changes from a chef. Once the chef orders ingredients from suppliers, they have the option to update or load the ingredients into the inventory of the database.

8 Class Diagram and Interface Specification

8.1 Class Diagram



Red = customer module; Blue = manager; Yellow = Chef Mod; Green = Server Mod; Purple = Combined https://drive.google.com/file/d/1FasWuvdIuBEPXVaOkBpL9dDegV3ZQ3Zh/view?usp=sharing

8.2 Data Types and Operation Signatures

Class: User Profile

Attributes:

- username: String - username chosen by the user during signup
- Firstname: String - first name as given by the user during signup
- lastname: String - last name given by the user during signup
- email: String - email provided by the user that can be used to send updates and promotions
- password: String - password chosen by user, conforming to certain security requirements
- address: String - mailing address of the user
- accountType: int - account type of the user; manager, server, chef, customer -> 1,2,3,4
- rewardsBalance: int - balance of points accumulated by user, all users get points employee or otherwise

Methods:

- changeProfile(); void - allows user to change account information shown above
- signup(); User Profile - allows a new user to sign up and create a new User Profile class
- guestCheckout(); - allows a user to checkout without having to sign up

Class: Employee Portal

Attributes:

- User Profile: Class - class created by all users during signup, includes firstname, lastname, username, password, address, accountype, rewards balance
- hoursWorked: int - tracking of amount of hours worked by the employee
- Time: date - date and time is needed for employees to be able to clock-in or clock-out
- employeeID: int - unique ID provided to each employee

Methods:

- clockin(); - employee clock-in button
- clockOut(); - employee clock-out button
- viewSchedule(); - allows employee to see current schedule and available shifts
- editAvailability(); - allows employee to change their current availability

Class:Employee Info

Attributes:

- userProfile: Class - class created by all users during signup, includes firstname, lastname, username, password, address, accountype, rewards balance
- employeeID: int - employee ID provided by employer for unique identification

Methods:

- changeInfo(userProfile); void - allows employees to change their account information

Class: Customer

Attributes:

- User Profile: Class - class created by all users during signup, includes firstname, lastname, username, password, address, accounttype, rewards balance
- orderID: int - ID given to order that the customer has placed

Methods:

- displayPoints(); - show how many points user has available
- reserveTable(); - allows user to place a reservation for a table
- takeAway(); - allows customer to skip reserving a table and order food for pickup
- orderStatus(); - provides customer with updates on their order process
- callServer(); - alerts server to tend to customer
- viewMenu(); - allows customer to see the full menu and proceed to ordering

Class: Manager

Attributes:

- employeeInfo: Class - class containing information for every employee such as wages and hours worked
- currentOrders: String[] - list of all current ongoing orders in the restaurant

Methods:

- editEmployee(); - allows manager to edit employee information such as wages
- addEmployee(); - allows manager to add a new employee to system
- editSchedule(); - allows the manager to populate schedule with shifts worked by the employees for the week
- viewSchedule(); - shows the schedule
- editTableConfig(); - allows manager to change the layout of tables within the restaurant
- orderStatus(); - allows manager to check the status of an order contained in list of currentOrders
- tableAvailability(); - shows chart of tables and their current availability status
- viewEmployees(); - allows manager to view all employees and their information
- productivityStats(); - allows manager to see statistics regarding productivity and sales data
- viewMenu(); - allows the manager to view menu of items

Class: Chef

Attributes:

- Employee Info: Class - class containing information for every employee such as wages and hours worked
- orderID: int - identification number of a particular order placed by customer
- currentOrders: String[] - list of all orders currently in process
- Menu: Class - allows chef to access the menu and add items to order for customer

Methods:

- orderStatus(); - shows the status for an order
- callServer(); - notifies the server to report to the chef when food is ready or assistance is needed
- viewMenu(); - allows chef to view the menu in detail
- displayInventory(); - allows chef to see the inventory and quantity of each item

Class: Server

Attributes:

- Employee Info: Class - class containing information for every employee such as wages and hours worked
- currentOrders: String[] - list of all orders currently in process
- Menu: Class - allows server to access the menu and add items to order for customer

Methods:

- viewOrders(); - allows the server to view orders that are currently being attended to
- tableAvailability(); - enables server to see which tables are available
- reserveTable(); - allows server to place a reservation on a table for the customer
- viewMenu(); - allows server to view the menu

Class: Menu

Attributes:

- Item: Class - item with all characteristics such as description and price
- menuItems: String[] - list of all menu items

Methods:

- displayAll(); - show all items on the menu
- filterItems(); - filter items according to certain parameters
- addToFavorites(); - add an item to user favorites
- displayFavorites(); - show only items in user favorites

Class: Order

Attributes:

- orderID: int - unique ID for each order placed in the restaurant
- orderItems: Item[] - list of items ordered by the user
- totalCost: double - total price of the items purchased by the user

Methods:

- checkout(); - takes user to the payment page
- addItem(); - adds items to the orderItems list
- removeItem(); - removes item from the orderItems list

Class: Payment

Attributes:

- orderID: int - unique ID for each order placed in the restaurant
- totalCost: double - total cost for the order
- customerID: int - unique ID for the customer in store (only if user is registered with system)
- Timestamp: date -
- paymentMethod: String

Methods:

- redeemPoints(); void
- paymentPortal(); void
- submitFeedback(); Feedback

Class: Table Info

Attributes:

- orderID: int
- CustomerID: int
- tableID: int
- tableStatus: String
- estimatedWait: int

Methods:

- reserveTable(); - The customer selects a table for reservation.
- changeStatus(); - The status of a table will change status upon selection.

Class: Feedback

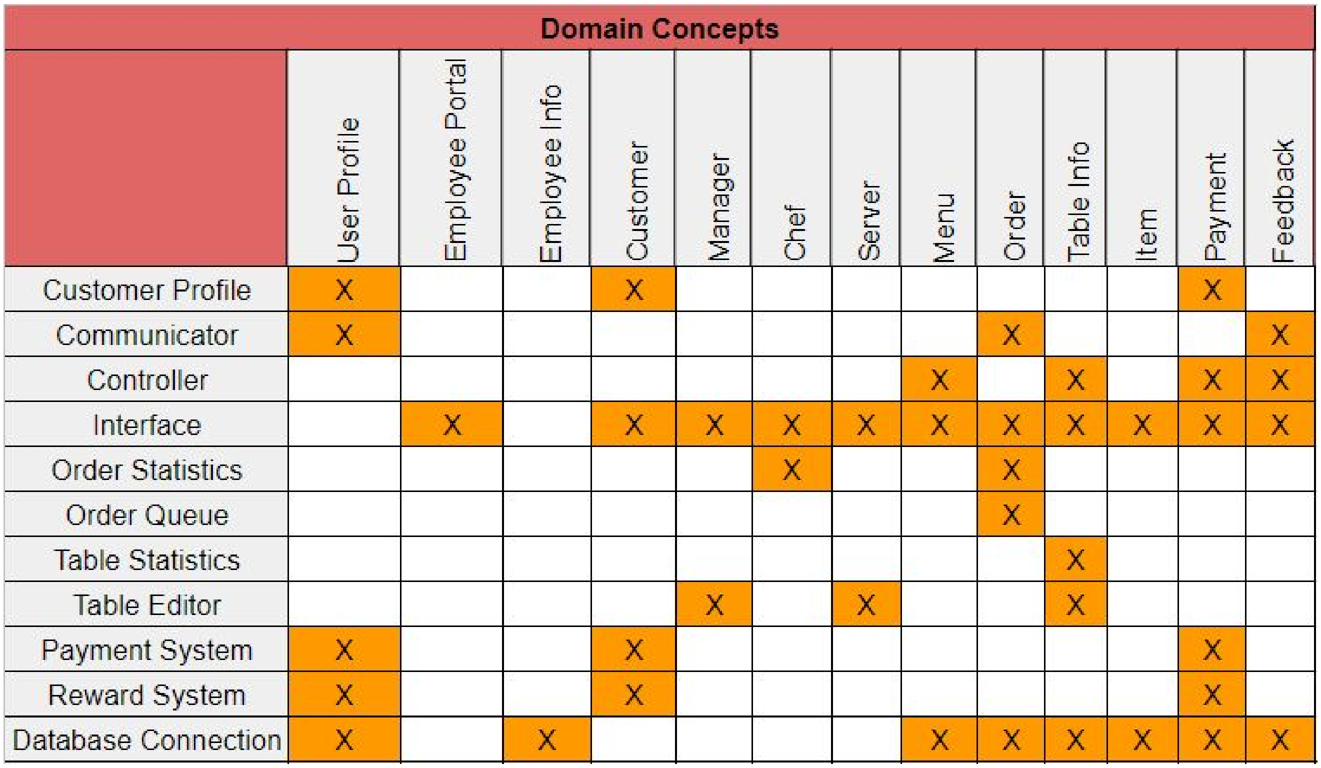
Attributes:

- orderID; int
- customerID; int
- totalRating: int
- Feedback: String

Methods:

- rate(); void
- comment(); void

8.3 Traceability Matrix



	Domain Concepts												
	User Profile	Employee Portal	Employee Info	Customer	Manager	Chef	Server	Menu	Order	Table Info	Item	Payment	Feedback
Customer Profile	X			X								X	
Communicator	X								X				X
Controller								X		X		X	X
Interface		X		X	X	X	X	X	X	X	X	X	X
Order Statistics						X			X				
Order Queue									X				
Table Statistics										X			
Table Editor					X		X			X			
Payment System	X			X								X	
Reward System	X			X								X	
Database Connection	X		X					X	X	X	X	X	X

Customer profile

- *User profile:* Allows customers to view and edit account specific data.
- *Customer:* This is the customer main page where one can see the menu and other features of the app.
- *Payment:* Allows customers to select preferred payment method.

Communicator

- *User profile:* Communicates user account information through the database for authentication.
- *Order:* Communicates the order placed by the customer to chef via database.
- *Feedback:* Transfers the feedback by the customer to restaurant's staff.

Controller

- *Menu:* Controller requires the users to select items from the menu.
- *Table info:* Controller requires that a valid table is selected.
- *Payment:* Controller requires the customer to select valid payment type.
- *Feedback:* Controller asks the user to provide feedback.

Interface

- *Employee Portal:* Employee portal is accessible via the interface.
- *Customer:* Customer portal which includes menu, favorites etc. is accessible via the interface.
- *Manager:* Manager portal is accessible via the interface.
- *Chef:* Chef portal is accessible via the interface.

- *Server:* Server portal is accessible via the interface.
- *Menu:* The restaurant's menu is accessible via the interface.
- *Order:* Order is placed and the status is shown through the interface.
- *Table info:* Table status and selection is checked/done via the interface.
- *Item:* Items in a menu are shown through the interface.
- *Payment:* Payment information is provided to the customer via the interface.
- *Feedback:* Feedback is provided by the customer via the interface.

Order Statistics

- *Chef:* Order status is in the hands of the chef to mark it complete once ready.
- *Order:* Food status is initiated once the order is placed.

Order Queue

- *Order:* Ordering a food item adds it to the queue.

Table Statistics

- *Table info:* Allows users to select a table and view its status.

Table Editor

- *Manager:* Table status can be edited by the manager.
- *Server:* Table status can be edited by the server.
- *Table info:* Allows users to select a table and change its status.

Payment System

- *Customer and user profile:* The purchase is logged to the customer's account after the payment is made.
- *Payment:* The system uses the selected payment method in order to complete payment.

Reward System

- *Customer and user profile:* Customer portal allows registered customers to see their reward points while the rewards information is stored into the user's profile
- *Payment:* Rewards are earned after the payment is made.

Database Connection

- *User profile:* User account information will be stored on the database.
- *Employee Info:* Employee information such as clock in/out will be stored on the database.
- *Menu:* Menu will be stored on the database so that users can access it.
- *Order:* Order information such as status and queue is accessible by the database.
- *Table info:* Table information is stored on the database.
- *Item:* Each item with detailed information will be stored inside of a menu on the database.
- *Payment:* Saved and preferred payment methods will be stored on the database.
- *Feedback:* Feedback by the customers will be stored on the database.

9 System Architecture and System Design

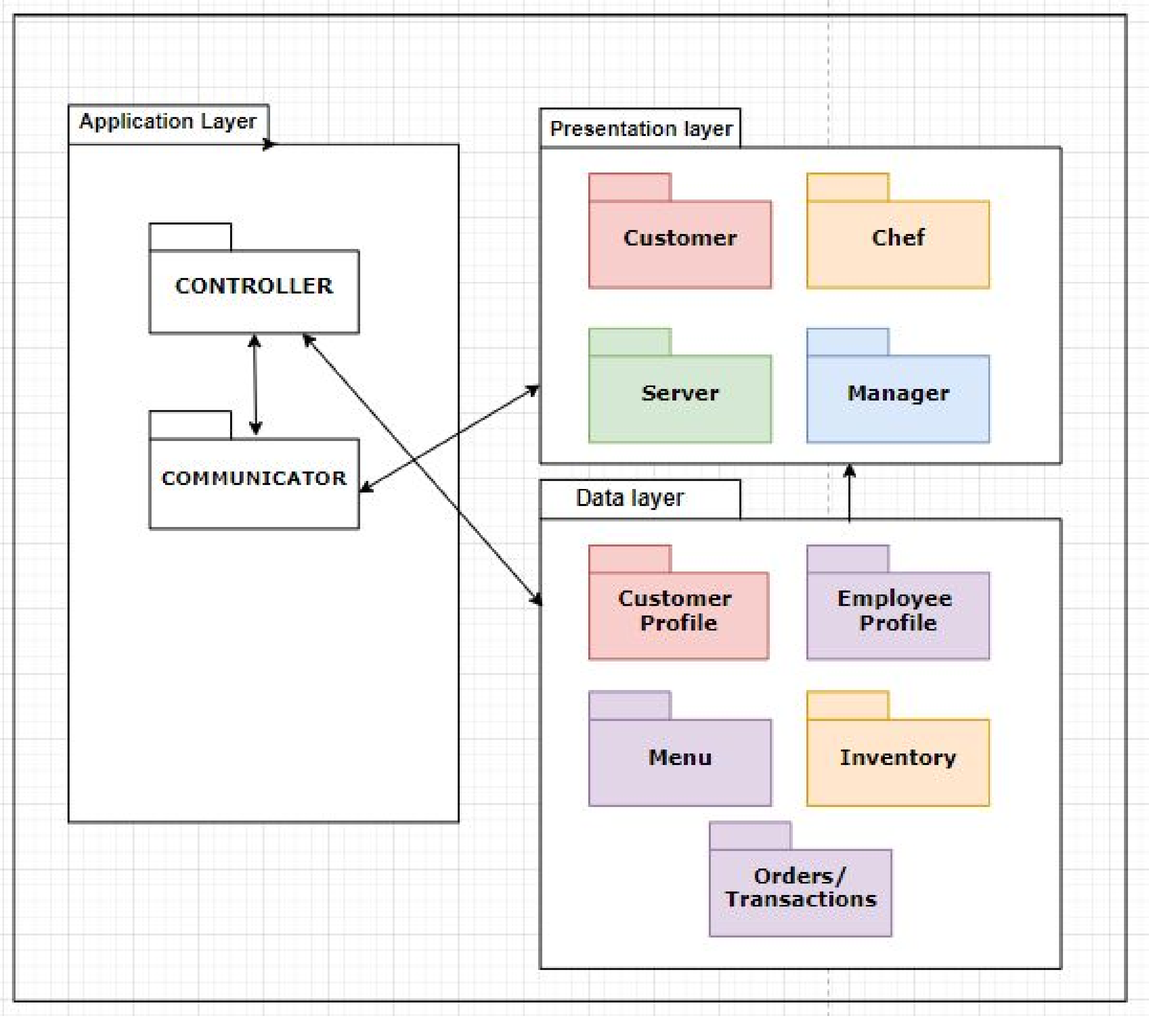
9.1 Architectural Styles

Our main architectural style will be based on a layered pattern architecture which consists of presentation layer (UI layer), application layer (service layer), and data layer (persistence layer). The presentation layer is seen by the users on the front-end, both customers and restaurant staff. Application and data layers are on the back-end and keep track of user-entered information, as well as respond to commands from the presentation layer.

Another architectural style for the backend used will be component-based. Each module in the restaurant will essentially serve as its own component. For example, a manager will require a different set of functions and needs from the program such as employee work hours or overall statistics about the restaurant as compared to other modules of the restaurant.

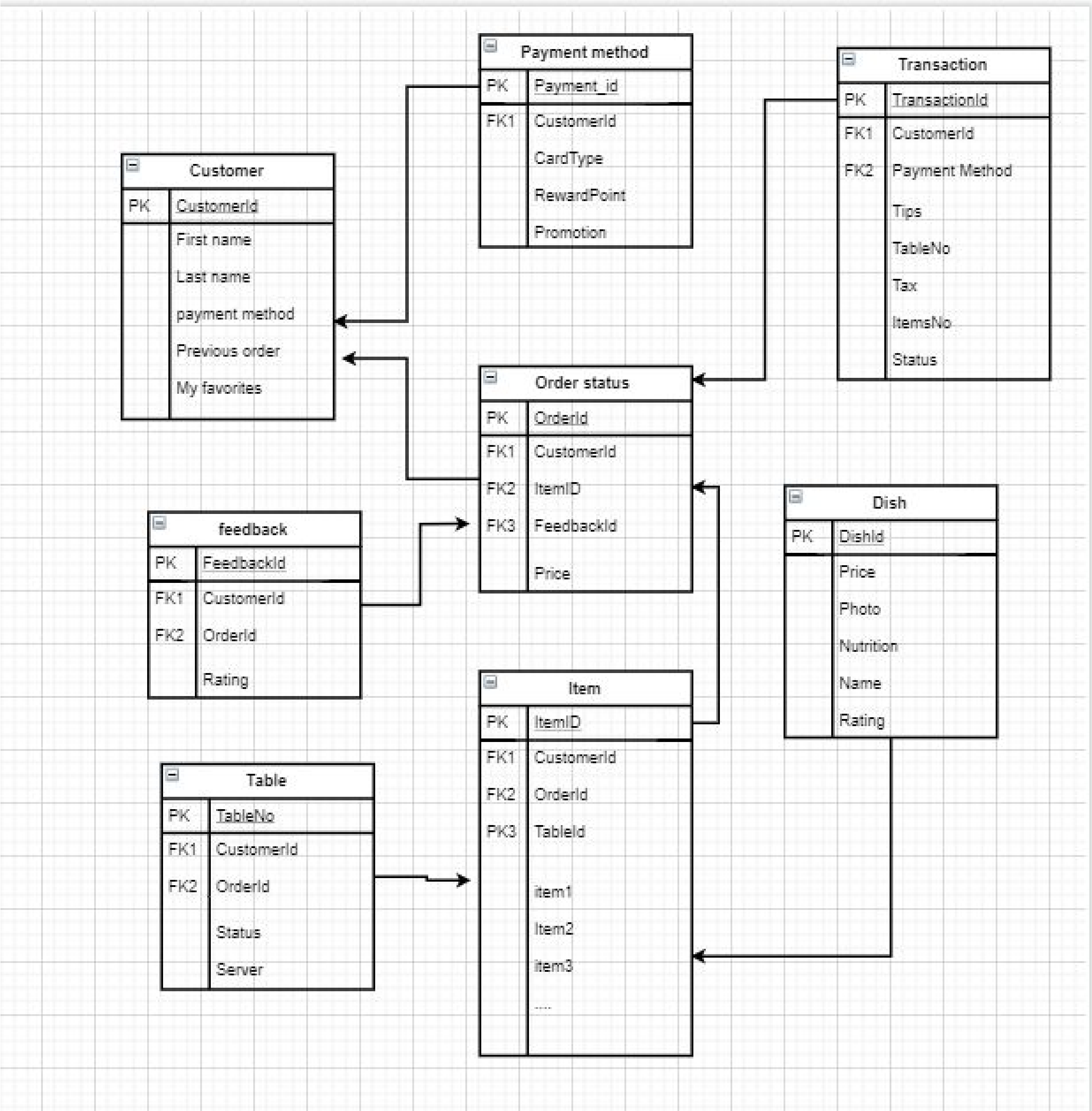
The third type of architectural style used is database-centric. An important part of creating the automation of the restaurant is to utilize a database system to store/retrieve information. All modules of the restaurant will be utilizing a database either to store/retrieve menu, orders, dishes, tables, employee information, customer information etc. depending on who (which module) accesses it. By utilizing a database-centric architecture it creates an easy way to access the information a user needs.

9.2 Identifying Subsystems



As mentioned earlier, our architecture is based on a layered pattern consisting of three major layers: application layer, presentation layer, and data layer. The application layer consists of a controller and communicator. Controller controls all the major operations between the data layer and the system such as accessing inventory, menu etc. and sending that information to the system to display it to the user through the presentation layer. Communicator allows communication between the individual modules and their systems. For instance, when a customer places an order, it is sent to the chef system via a communicator. The presentation layer consists of four independent modules: customer, chef, server, and manager. Each module will display its own functionalities and unique screens. Lastly, the data layer is basically the database where all the information from inventory to user information etc. will be stored and can be retrieved when needed.

9.3 Persistent Data Storage



Persistent storage is any **data storage** device that retains **data** after power to that device is shut off. It is also sometimes referred to as non-volatile **storage**. We will use MySQL for our database system. We will set up the database instances with SQL workbench and connect it to the server. The Server lately can access our data with the GET and POST requests. Our software does need to store persistent data that needs to outlive a single execution of the system. Each user's (customer) information such as username, password, favorites, rewards, and feedback will be stored on the database which can be retrieved whenever a customer logs in to their account. For an employee the software needs to store and retrieve the employee's first and last name, username and password, salary, time worked, work schedule, etc. The restaurant itself will also have a lot of information consisting of inventory and other information which the manager needs to know will be stored/retrieved on/from the database.

9.4 Network Protocol

Based on our requirements and expectations, we decided to build up a server to connect the different parts such as manager, chef, and customer. We decided to use the local internet to communicate since we want to build an application in the restaurant. We are going to use the OSI network model. The network protocols are TCP/IP, etc. The advantages of wifi are high speed by avoiding the traffic from uploading and downloading to cloud. What is more, the WIFI communication has very high security and high reliability. Besides, we only need to have local communications between the different application terminals, so this network is the best choice. HTTP is a very common Network. For instance, if the client orders nourishment in the wake of clicking affirm, the request will be sent by means of HTTP POST solicitation to the server which will react back with data including assessed hold up time, which is vital for the customer to show to the client. Another model is the point at which the client looks for data about what number of remuneration focuses the client earned, a GET demand from the customer to the server will be given mentioning the client's prize focuses balance, the server will process this solicitation and in its reaction body will incorporate the equalization.

9.5 Global Control Flow

Customers will have access to our application by logging in if they have previously made an account or continue as a one time guest. Customers will get to choose a table or reserve one if needed. Once the customers have been seated, the functions of the application are as follows: user will be able to view a menu, select items from menu, proceed to their cart, comment on their order for any allergies/dietary restrictions, an option to view the progress of their orders, cancel an order, lastly, a payment page. This procedure of having customers logging in and inputting their needs makes our system execute in a “linear” fashion. Every time a customer logs into the app they will have to go through the same steps.

On the other hand, the servers and chefs will be handled on a more event-driven system. They will be on pause, until an order or table is ready. Only once the initial action of the customers putting in an order into the system will the servers and chefs be put into action. Once the chef and server have completed their tasks, then they will return to their loop of waiting for their next input.

Time dependency is one of the biggest issues between customers and chefs during busy hours. As mentioned above, customers will be able to view the progress of their orders, we will be adding an option for the chef interface. It will be a function which allows chefs to notify the customers that they have started making their orders, this function will leave the customers time to cancel their orders before a chef starts to make a dish.

9.6 Hardware Requirements

This software will be compatible with iOS and Android operating systems. For iOS, since XCode recommends development on the latest iOS firmware (iOS 13), we will be supporting iOS 13 and later, which will cover 50% of the iPhone users as far as March 2020. For Android OS, we are aiming to support Android 9 Pie (API 28) and later because it is currently supporting 41.22% of the Android users. Software interaction will take place via touchscreen interface. The color display resolution will range from 1330 x 700 pixels upto 1792 x 828 pixels. A minimum storage capacity of 80 MB must be available for the user to download and install the applications and its packages. The application will communicate on a client-server based model running over the restaurant Wifi and will make push-pull requests with the server, manager, customer. The application also requires GPS access which is available in most phones in 2020.

10 Algorithms and Data Structures

10.1 Algorithms

Table Reservation Algorithm:

When a customer makes a reservation , they will be requested to enter their name and party size. The customer's party size will be stored in a queue, considering first in first out order. The algorithm will search for an available table that suits the party size and the floor map will highlight the table assigned to the customer. As a customer successfully makes a payment and the server marks the table ready, the table status will change to available for that table (using the table's identification number). The algorithm will utilize two arrays that will help it determine which tables are available. The first array will store the identification numbers of the available tables and the second array will store the identification numbers of the tables that are occupied. The algorithm will check the first arrays with the customer's party size and if there is an available table that fits the party size, then the algorithm will assign that table to the customer. However, if there is no available table, then the customer will be told to wait and another algorithm will be used which will calculate the estimated wait time that fits the customer party size accordingly.

Wait Time Algorithm:

As tables become occupied and the customers are requested to wait, they are shown an estimated wait time on their screen (interface) for the next free table. An array will be needed to store the reservation/check-in time of the customer in order for an algorithm to calculate the estimated wait times. The algorithm will calculate an average dine time for customers and a buffer will be added for larger party sizes. In our system, small party sizes will be shown the average dine time with no buffer added. For medium party sizes (3-4 people), there will be a small buffer added to the average dine time to calculate the estimated wait time and so on.

Total Cost Algorithm:

After a customer adds all their items to cart and confirms their order, an algorithm must be utilized that calculates the total cost. This algorithm will include a linked list or array (not decided yet) with the items that the customer has ordered and add up the cost of each element, including the tax. This algorithm will help calculate the total cost of the food that each customer has purchased.

Financial Record (Statistical data for manager) Algorithm:

There will be times when the manager wishes to view the financial status of the restaurant. The financial tracker will be very simple. For the current scope of the project, the restaurant will have a balance of money owned. When customers purchase food, the amount owed will be directly added to the restaurant balance. When the manager makes a purchase, the amount will be deducted from the restaurant's balance. Similarly, employees paycheck will also be deducted from the restaurant's balance.

10.1 Data Structures

Table Reservation Data Structure:

In order to designate tables to the customers at the restaurant, two arrays and one queue must be used. One of the arrays will hold the identification numbers of tables that are free to be used by the customers. This array will be sorted in ascending order. Similarly, another array will be used that will hold information about tables that are unavailable to the incoming customers. Each element in this array will also be in ascending order. Through the use of these two arrays, algorithms can easily search which tables are available and compare the customer' party sizes with the total number of people that can be seated at each corresponding table. As the table statuses change from available to occupied and vice versa, the identification numbers of the corresponding tables will be removed from one of the arrays and added to the other array, indicating the status change. In addition, a queue will be used for storing the customer's party sizes. It will be a way to determine which customers arrived first to the restaurant because the queue will be first in first out order (FIFO). As more customers visit the restaurant, their party sizes will be added to the end of the queue and will not be matched with an available table until the party sizes in front have been handled.

Wait Time Data Structure:

As customers enter the restaurant and are added to the end of the queue, another sorted array will be needed in order to calculate the estimated wait time. This array will have elements in ascending order that represent the time a customer has made a reservation for or has checked in at. The elements of this array can be compared to the average dine in times corresponding to the party size to determine estimated wait time.

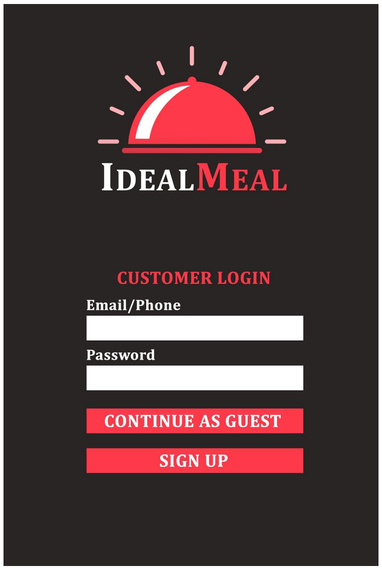
Total Cost Data Structure:

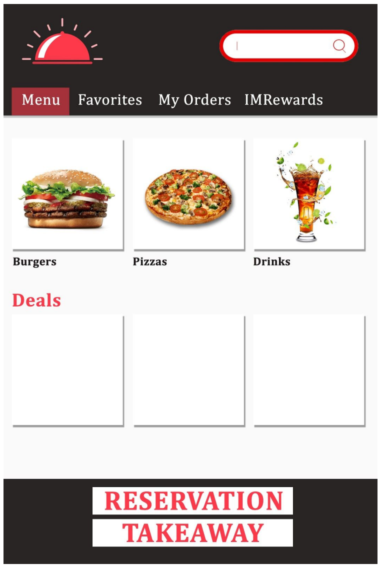
When a customer selects items from the menu, these items and their costs will be stored in a linked list/array. However, a linked list is suitable to use in this case because a customer can delete a dish anywhere in the list, unlike an array. In an array, there is less flexibility and more difficult to add or remove items in any position. Arrays have fixed sizes and to increase their sizes utilizes a lot of storage and time. Thus, a linked list works well for the software system in terms of ordering because a customer has the option to make changes to the order and the item can be located anywhere in the list. In addition, this linked list will help determine the total meal cost. When customers have confirmed their orders, all the elements in the linked list will be summed together to determine the total cost of the customer's meal.

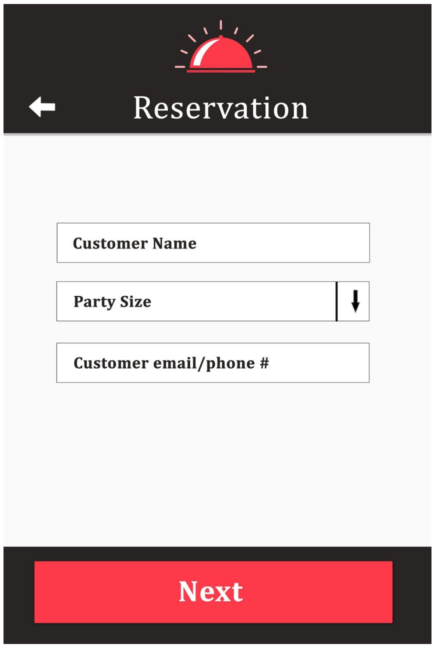
11 User Interface Design and Implementation

Customer

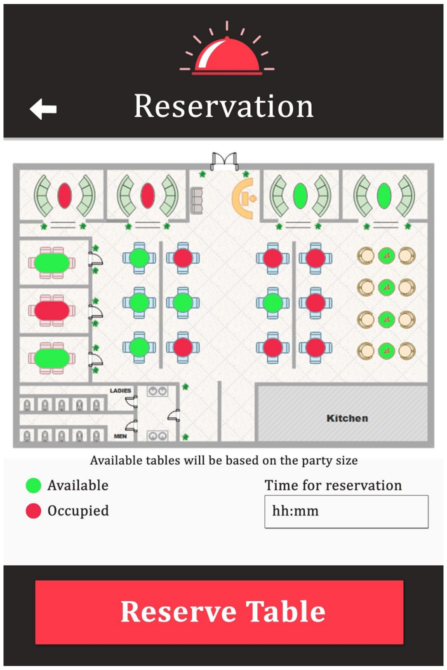
Our initial screen mock-ups from the first report are sufficient since we believe they optimize user effort. However, as reservation mockup was not shown before, it will be discussed here.







1. Customers sign into their account or as a guest.
2. The customer chooses reservation by clicking on the RESERVATION button.
3. The customer will then be prompted to put their information which includes their name, party size and phone# or email (optional).



Available tables will be based on the party size

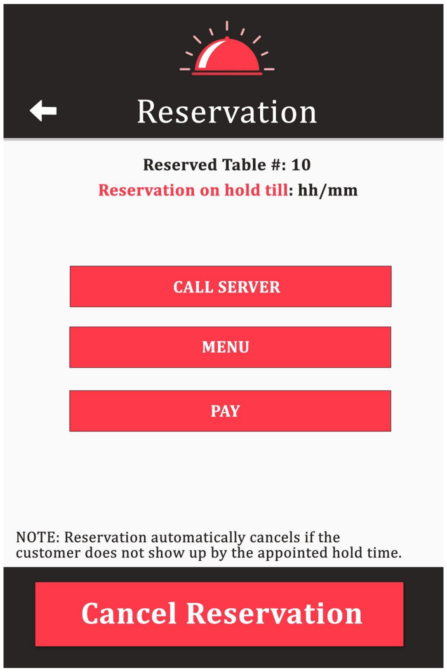
🟢 Available

🔴 Occupied

Time for reservation

hh:mm

4. On the basis of party size, our application will show only those tables to the customers that fit their party size and the rest of the tables will be grayed out. The customer at this point will choose from the available tables by simply clicking on the table and then entering the time for reservation under the "Time for reservation" static button.
5. To reserve table after inputting information, click reserve table.



Reserved Table #: 10

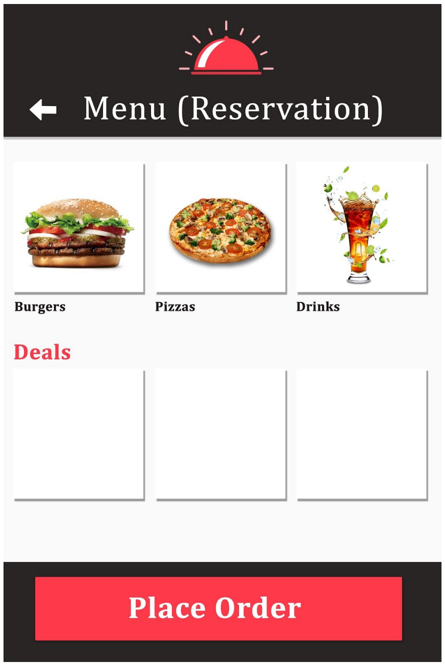
Reservation on hold till: hh/mm

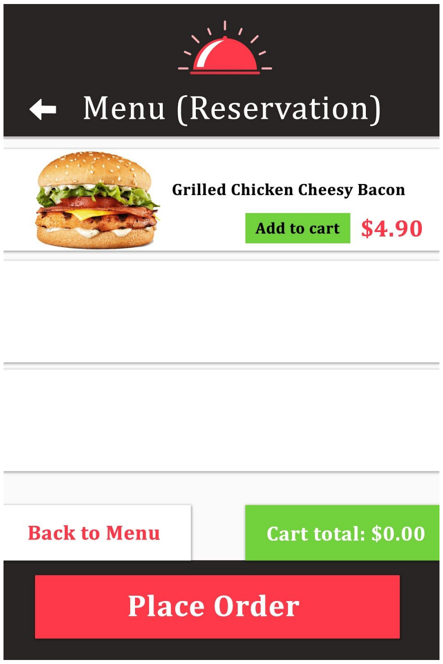
NOTE: Reservation automatically cancels if the customer does not show up by the appointed hold time.

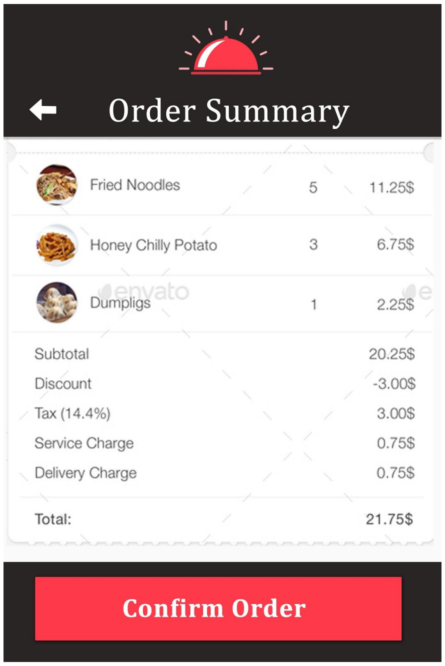
At this step, the customer can cancel their reservation or it will automatically cancel after the hold time is over. Also, the three buttons (Call server, menu, and pay) will be disabled before the customer checks-in in the restaurant.

However, if the customer checks-in on time and confirms their reservation with the waiter/server, the customer can then use the "Call Server," "Menu," and "Pay" buttons.

To order food, customers choose the Menu option and are prompted to Menu.

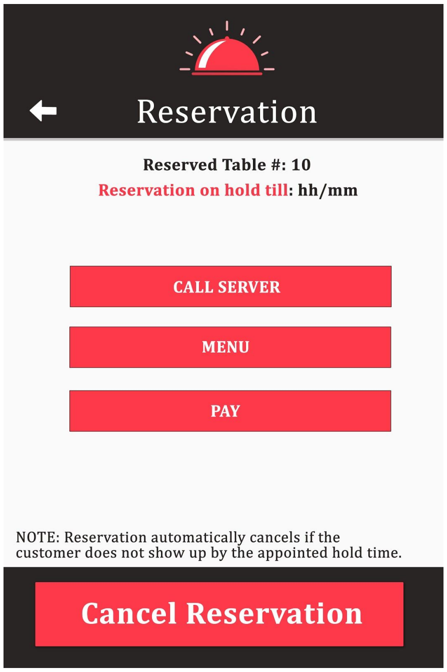






Fried Noodles	5	11.25$
Honey Chilly Potato	3	6.75$
Dumpligs	1	2.25$
Subtotal		20.25$
Discount		-3.00$
Tax (14.4%)		3.00$
Service Charge		0.75$
Delivery Charge		0.75$
Total:		21.75$

After adding items to cart, customers place and confirm their order and are prompted back to the Reservation page where they can order more food by going to the menu or use other options such as call server or pay.



Reserved Table #: 10
Reservation on hold till: hh/mm

NOTE: Reservation automatically cancels if the customer does not show up by the appointed hold time.

Chef

We realize the chef part needs the most optimization on the user's effort because the major function of the UI interface for chef part is to display information. During the working time, chefs are mostly focusing on cooking, they won’t have much time to interact with the UI interface. So we add the new automation features in the Chef interface. When it is set to cooking mode, the incoming order will be automatically separately in different suborders and then assigned to different chefs with specific patterns. And the main window will show up on the UI interface will be the name of the current cooking dish and its special requirements comment with large font size. And there will be a large button near each dish to display its finishing time. Whenever the dish is done, the chef just needs to click on the button, and it will automatically notify the server, and the finishing dish will be removed from the window and turn to the next one. According to that, the user effort of the chef is just one click per dish.

12 Design of Tests

12.1 Test Case

The following are our unit test case:

TC-1 : Login test
TC-2 : SignUp test
TC-3 : Incoming Order Notification
TC-4 : Reservation test
TC-5 : Reward Point system test
TC-6 : Favorites
TC-7: Feedback
TC-8: TakeAway

<table>
<tr><td colspan="2"><b>Test Case Identifier: TC-1</b>
<b>Use Case tested: UC-19 (Login)</b>
<b>Pass/Fail Criteria:</b> Test pass if the user can successfully log in our IDEAL MEAL system. Tests fail if the user is not able to log in the system.
<b>Input data:</b> Username, password</td></tr>
<tr><td><b>Test Procedure</b></td><td><b>Expected result</b></td></tr>
<tr><td>Step 1: User type in the a valid username and password
Step 2: User type in the a invalid username and password</td><td>Access successfully, UI interface displays the welcome window.
Access denied, UI interface displays “username and</td></tr>
</table>

	password incorrect”.

Test Case Identifier: TC-2

Use Case tested: UC-2 (SignUp)

Pass/Fail Criteria: Test pass if the user can successfully create an account if the email is not being used. Test fail if the user is not able to create an account

Input data: email, other user profile

Test Procedure	**Expected result**
Step 1: User type in a new email and other user profile	Access successfully, UI interface displays the welcome window.
Step 2: User type in a used email and other user profile	Access denied, UI interface displays “email has been used”.

Test Case Identifier: TC-3

Use Case tested: UC-14 (Order Notification)

Pass/Fail Criteria: Test pass if the chef server can receive a new order coming notification. Tests fail if no notifications show up on incoming order.

Input data: new placing order

Test Procedure	**Expected result**
Step 1: User placing a valid order on the customer side.	Chef server received the order and popped up a message box and played sounds to notify the chef.
Step 2: User placing in an invalid order on the customer side	No notification showing up.

Test Case Identifier: TC-4

Use Case tested: UC-4 (Reservation)

Pass/Fail Criteria: Test passes if the customer inputs valid name and party size.

Test fails if the customer inputs invalid name and party size.

Input data: Customer name:string, party size: int

Test Procedure	**Expected result**
Step 1: Customer selects reservation	Application will go to the next page and ask the user

	(customer) to enter their Name and Party Size.
Step 2: The customer inputs their Name and Party Size correctly and clicks Next.	The customer will be prompted to the next page where they can choose a table for reservation.
Step 3: The customer inputs their Name and Party Size incorrectly and clicks Next.	System will pop up a message “Invalid Name/Party Size, please re-enter information.”

Test Case Identifier: TC-5

Use Case tested: UC-12(EarnRewards), UC-20 (IM Rewards)

Pass/Fail Criteria: Test passes if the customer can check, earn, and spend their RewardPoints. Tests fail if the customer is not able to do either one of checking, earning, or spending their RewardPoints.

Input data: IMRewards:button

Test Procedure	**Expected result**
Step 1: Customer logs into their account and selects the IM Reward program	The interface will display the current number of reward points for the users and the benefit for each Reward level.
Step 2: Customer logs in as a guest and selects the IMReward option.	The interface will display an error “Feature only available for registered customers.”
Step3: Customer completed a valid order in their account	The customer accounts will receive several number of reward points based on the cost of the order.
Step4: Customers choose to redeem reward points when they are on the payment page.	The interface will deduct the total cost based on the points the user redeems. And the customers account will lose the corresponding reward points.

Test Case Identifier: TC-6

Use Case tested: UC-18 (Favorites)

Pass/Fail Criteria: Test passes if the customer can use Favorites option. Tests fail if the customer is not able to use the Favorites option.

Input data: Favorites:button

Test Procedure	**Expected result**

Step 1: Customer logs into their account and selects the Favorites feature.	The interface will display the list of items the customer has chosen to mark as favorite.
Step 2: Customer logs in as a guest and selects the Favorites feature.	The interface will display an error “Feature only available for registered customers.”

Test Case Identifier: TC-7

Use Case tested: UC-16 (Feedback)

Pass/Fail Criteria: Test passes if the customer can add feedback on their order and the restaurant staff can view it. Tests fail if the customer is not able to save feedback on their order or restaurant staff are not able to access the feedback system.

Input data: feedback

Test Procedure	**Expected result**
Step 1: Customers click on their current order and type in their feedback under the comment box.	The database will record these feedback and then the interface will shows “thank you for your feedback”
Step 2: Restaurant staff log in their account and then click on order lists, filter by feedback order and date	The interface will list all the orders with recorded feedback on that date.

Test Case Identifier: TC-8

Use Case tested: UC-1 (TakeAway)

Pass/Fail Criteria: Test passes when the customer clicks TakeAway option on the main page after adding items to cart. Test fails if the customer clicks TakeAway and has not yet added items to cart.

Input data: TakeAway:button

Test Procedure	**Expected result**
Step 1: Customer adds item to cart from the menu and clicks TakeAway.	The interface will prompt the user to Checkout page.
Step 2: Customer clicks TakeAway button without adding items into cart.	The interface will display a message “Cart is empty.”

12.2 Test coverage

The current tests are meant to cover the cases that can be presented to the Client (IdealMeal) themselves, and do not yet require the components of the other modules to be integrated. The goal of the first demo is to show IdealMeal the progress that has been made, and give them an idea of our vision for their software goals. The tests have thus been designed in such a way to show IdealMeal the most important aspects of the Software. As we develop more of our classes and methods, we will add and adjust test cases as needed. Testing will be done for as many possible cases that a class could go through. The test procedures for the classes will have the format of both a pass procedure and a fail procedure in response to a user input. If the test fails due to a faulty user input, the system will prompt the user to try again and correct the input.

12.3 Integration Testing Strategy

Generally, there are four approaches for the integration testing strategy, Big band, Top down, Bottom up and sandwich.

Since we divided our project into 4 different parts as Customer, Server, Chef, and Manager. We are willing to do bottom up testing. Bottom Up is an approach to Integration Testing where bottom level units are tested first and upper-level units step by step after that. This approach is taken when a bottom-up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases. According to that, we will do those 4 bottom level parts (the UI interface of each) of our projects first. Whenever it works well, we will turn to test the higher parts which is the communication between those interfaces.

The nonfunctional system requirements will also be tested on their own for each module. For example, a nonfunctional requirement of the Chef to be able call a server with an on-screen button that does not require going through many steps. So, when designing the interface, the chef module will test their interface to adhere to this user story. Likewise, each module will verify their programs are operating system independent. Once the system is integrated as a whole, there will be further testing to verify each of the modules adheres to its nonfunctional and system requirements.

13 Project Management and Plan of Work

13.1 Merging the Contributions from Individual Team Members

Compiling the report was not a problem for us as we work in a shared document where each group contributes their part to the report. However, formatting and consistency was an issue as different members used different fonts or table formats. These issues were resolved by having one member (Khizer Humayun) keep track of the entire report from consistency to formatting and quality. The UML diagrams and sequence diagrams, however, are different from one another as each group made theirs and it was difficult to remake it to keep the format the same.

13.2 Project coordination and progress report

Customer module:

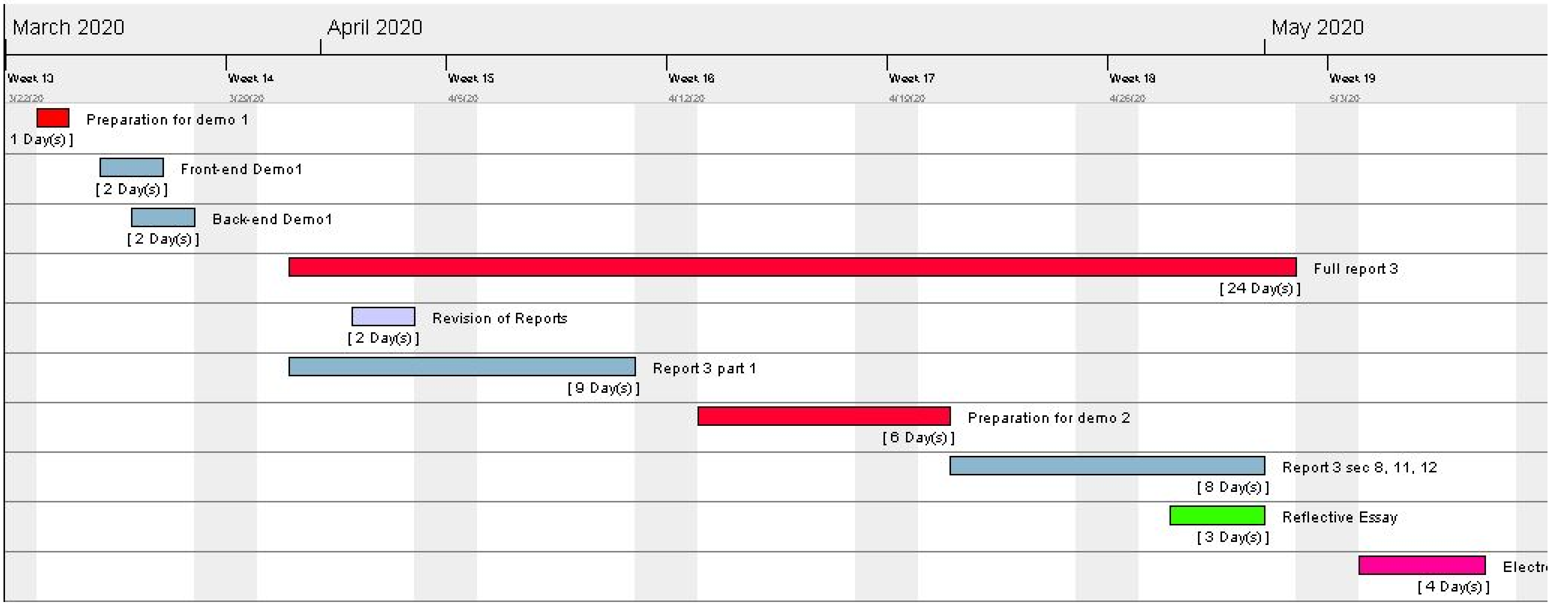
The use cases that our group aims to complete for demo are:

1. UC-19: Login
2. UC-13: ViewMenu (Which is the main page)
3. UC-1: TakeAway
4. UC-10: Payment

What is already functional, what is currently being tackled?

We are currently working on the frontend development of our user interface using React Native. We have also discussed building on top of a previous project as has been suggested to us. The project from 2019 (TurboYums) meets some of our goals. We are also looking into backend development using Javascript.

13.3 Plan of work



13.4 Breakdown of Responsibilities

Customer

Khizer Humayun and Ebad ullah Qureshi will be responsible for developing, coding, and testing the following classes/modules:

- Controller
- Database Connection
- Customer Profile
- Menu

- Payment System
- Feedback
- Customer Interface
- Table Info
- Reward System
- Communicator

Chef

Jianhong Mai, Nianyi Wang, and Keblin Li will be responsible for developing, coding, and testing the following classes/modules:

- Controller
- Database Connection
- Employee(Chef) Interface
- Order Statistics
- Order Queue
- Communicator
- Employee(Chef) Profile

Manager

Tyler Hong and Akshat Parmar will be responsible for developing, coding, and testing the following classes/modules:

- Controller
- Database Connection
- Employee(Manager) Interface
- Table Statistics
- Table Editor
- Menu
- Employee(Manager) Profile

Server

Shuren Xia and Harman Kailey will be responsible for developing, coding, and testing the following classes/modules:

- Controller
- Database Connection
- Employee(Server) Interface
- Table Statistics
- Table Editor
- Employee(Server) Profile
- Payment

14 References

Turbo Yums
https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2019-g13-report3.pdf

Why W8
https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2018f-g4-report3.pdf