Food Delivery Subsystem

Ari Torczon[#1], Hans Jeremy[#2], Jeffrey To[#3], Eric Jiang[#4], William Pong[#5]

Computer Science Department,
California State Polytechnic University Pomona
Pomona, United States
[1]aatorczon@cpp.edu
[2]hjeremy@cpp.edu
[3]jlto@cpp.edu
[4]eljiang@cpp.edu
[5]williamp1@cpp.edu

***Abstract*— We are going to update a restaurant's food delivery system and develop it into an online system. This system's functional requirements should be capable of collecting a customer's order through online means or having a staff input their order, their personal and payment information, and their reviews while tracking the order and delivery. To build this system, we are using Java Swing for the graphical user interface and MySQL for the database management. This paper will communicate the development, analysis, and design of this food delivery system.**

***Index Terms*— delivery, java, SQL, database**

I. Introduction

A Food Delivery subsystem can be employed in a variety of ways such as creating a customer account, ordering food, checking the order and delivery, leaving a review, and seeing previous reviews. The objective of this subsystem is for the better management of the orders and deliveries of restaurants. This includes customers being able to create orders and track them, while employees can better coordinate multiple orders and change their status based upon fulfilling them as well as allowing drivers to also update the status to inform customers of their order status.

A. Problem Description

Contemporary technological advancements have allowed people to connect in easier ways. This includes communication with businesses, including restaurants. Online systems can now take orders from customers through the press of a digital button and even from the comfort of their homes. This eliminates the necessity of undergoing the process of listing your order while a waiter or waitress writes it down. Food delivery is also part of the modern lifestyle and some prefer having the convenience of food being delivered to them. It is then necessary to address this convenience of online systems and food delivery to stay competitive in the market. A restaurant wants to implement a digital system for its delivery services that allows customers to make orders and check on their deliveries while employees are able to organize and stay updated on the orders and deliveries. Without an online system, customers would have to make a call to the restaurant and list or explain their orders while the employees would have to manually note each item which can waste time and cause errors in communication if something is misheard. Allowing customers to input their own orders removes the hassle of timely communication with the employees and the risk of miscommunication as customers can make and review their own orders online without the help of employees. The customer can also save their personal and payment information for ease of access the next time they order. Customers are also going to be able to keep track of their own orders. Employees are also going to be able to monitor multiple orders and deliveries in the online system without the inconvenience of physically searching for written orders while delivery drivers can access a customer's information easily for their location.

B. Proposed Solution

An implementation of the restaurant's system into a digital food delivery system will create better management and provide easy access to the orders and deliveries for the staff while granting better communication between employees and customers. Customers will be able to search for food items from a selection on the online menu and take their

own orders, check on the status of their orders, and rate or review the restaurant's food and services. The staff will be able to do the same as the customers to still keep the original status quo of allowing customers to order at the establishment or through phone calls. Staff additionally can handle multiple orders while being able to update the status of the orders and deliveries. The employees will also be able to have access to the customer's contact information and address if they create an account for better communication when certain errors happen or to know the locations of the customers for an efficient route. Employees with the specific role of being a delivery driver will also have the capability of viewing ready-to-deliver orders while also being able to update those orders for both other employees and customers to check on.

II. Analysis

A. System Requirements

A food delivery system has many factors that are important to its functional requirements. But the most important factor in these requirements is structured around the customers themselves. The customer is able to search and browse through the online menu of the system and when they are done, they can finalize their order. But when they do, they will have to sign in their customer account into the system. If the customer does not have an account, they are able to create an account. Creating an account requires the customer to input their name, contact information, payment information, and one or multiple addresses. Each requirement for a customer account is needed as contact information can be for employees to communicate to customers about certain errors or problems, payment information is saved to complete their order, and multiple addresses can be for when a customer orders at home or at other locations. By creating this account, a customer only needs to register their information once and the system will record this information for convenient accessibility for employees. After signing into an account, the order can be reviewed by the customer and finalized with payment of the order. This order is then recorded in the system which then the staff will be able to view and prepare the order. When the orders are completed, staff are able to update the status of the orders to "ready to deliver" so delivery drivers will be capable of seeing the orders that are ready to be picked up. After the order is picked up, the employees at the restaurant or the delivery drivers will be able to update the progress of the delivery changing the status from "ready to deliver" to "picked up" and ending the delivery with "delivered." Throughout this process, following the order placement, the customer or staff member has the option to cancel the orders changing the status to "canceled." After the order has been delivered, the customer is able to leave a review about the quality of the food, service, and delivery while also being able to add comments about their experience.

B. System Actors and Stakeholders

Classification of the actors and stakeholders of our system is important to show who will benefit from this implementation.

The staff and delivery drivers are interacting with the system themselves by either viewing orders or updating delivery statuses making them operational stakeholders. Only the staff can create orders for customers. Staff and delivery drivers are also part of the organization of the restaurant which makes them internal stakeholders.

Customers will also interact with this system by ordering their food and checking the status of their order which makes them operational stakeholders. Customers are not part of the organization of the restaurant which makes them external stakeholders as they do not have access to information an employee would have.

The restaurant owner or owners and investors are executive stakeholders as they do not directly interact with the system but they still financially benefit from it. But the investors and owner/s, are still a part of the restaurant meaning they are internal stakeholders. Since the scope of the system only extends to the delivery and ordering systems they are not included as a direct actor for the system.

C. Use Cases

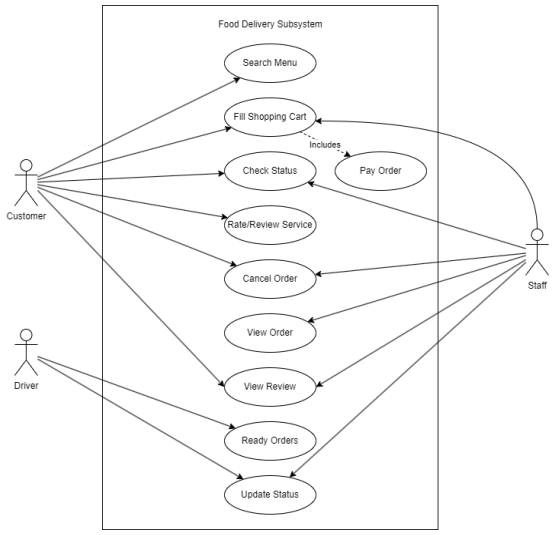

Fig. 1 A Use Case Diagram

After analyzing the actors and stakeholders of the system, we are able to focus on the actors and the operational stakeholders that are going to interact with this system and develop use cases that the actors can use. Starting with a customer actor interaction, the customer would start to use the system by browsing the menu for the food they want. This interaction creates the use case "Search Menu" which describes how the customer searches for specific food items from the restaurant's menu. After the customer decides on a food item to order, they can add it to their cart which leads to the "Fill Shopping Cart" use case. After the customer is done adding food items to their order, the previous use case extends to a "Pay Order" use case for the customer to pay for the order. While waiting for their order, the customer is allowed to check on their order which creates the "Check Status" use case that sees the order's status as it is still being made, already on delivery, or delivered. The customer is also able to cancel their order at any time which generates the "Cancel Order" use case. The customer is also able to see reviews about the quality of the food or service from other customers with the "View Review" use case. These use cases are also accessible to the staff actors as well so the staff can take orders from customers or cancel them for them. However, leaving a review for the restaurant is exclusive to the customer through the "Rate/Review Service" use case. The staff will also have the exclusive use case to view any customer's order with the View Order use case. A delivery driver and staff will be able to update the status of an order in the "Update Status" use case. The delivery driver has the "Ready Orders" use case to view all orders available to pick up.

D. Fully Developed Use Case

Use Case Name:	*Check Status*	
Scenario:	Check the status of an online order delivery	
Triggering Event:	A customer or staff wants to check on the status of a delivery	
Brief Description:	The customer can obtain the order's status of ready to deliver, picked up, or delivered and the system will send the appropriate information. The staff who placed the order whether it was through a phone or in-person order can also check the status of the order.	
Actors:	Staff, Customer	
Related Use Cases:	Could be invoked by canceling orders, updating status, and ready orders	
Stakeholders:	Customers, staff, delivery drivers	
Preconditions:	An order must exist inside the system that has a status.	
Postconditions:	Reveals the status of an order which have 4 states	
Flow of Activities:	Actor	System
	1. Customer or staff want to check on a delivery of an order	1.1 System looks up delivery status of order 1.2 System returns delivery status
Exception Conditions:	1.1 Order does not exist	

Fig. 2 A Fully Developed Use Case Description Table of the Check Status Use Case

We decided to expand and get more in-depth with one of the use cases. We chose the use case Check Status to expand into a fully developed use case description table. The scenario of this use case is when an actor wants to check the status of an online order delivery. The actors that are able to use the use case involve the staff members and customers. Stakeholders that benefit from this use case are also customers and staff but include delivery drivers as well. The triggering event that will start this use case would be when a customer or staff member presses the View Order button on the system's main page. Before the use case continues though, an order must have been made by either a customer or staff member to be able to check its status. After the preconditions and triggering event have been met, the flow of activities between the actor and the system will start where the actor will ask for the status of their order and the system will search for the status of that order and display for the actor to see. Once the status of the order has been displayed, the use case has ended. Some exception conditions

to expect are mainly if the order request does not exist.

E. Activity Diagram

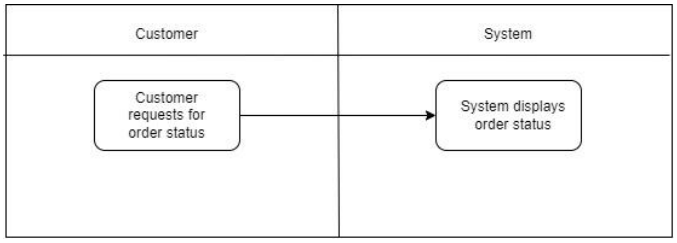

Fig. 3 An Activity Diagram of the Check Status Use Case

After creating the fully developed use case, we needed a visualization of how the system would interact with the user. An activity diagram can help conceptualize the process of the Check Status use case. Starting this activity involves the customer or staff requesting a specific order's delivery status. This request will then go to the system and it will retrieve the status of that order and display it for the user.

F. System Sequence Diagram

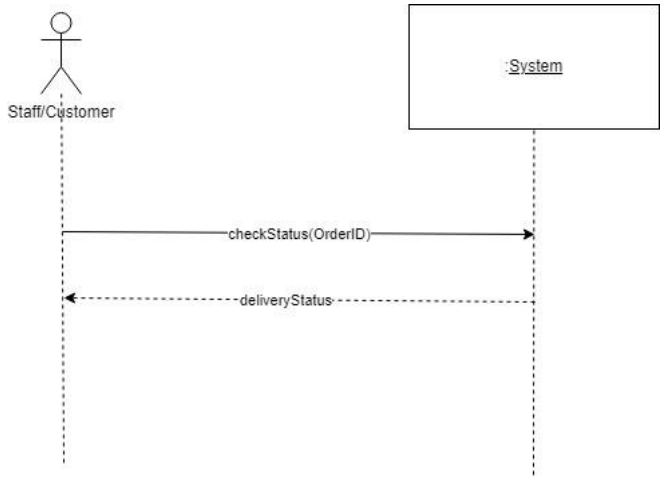

Fig. 4 A System Sequence Diagram of the Check Status Use Case

After developing the flow of activity from the fully developed use case table into an activity diagram, we can develop it further into a system sequence diagram. This system sequence diagram allows us to view the specific methods and variables that would be used in the interaction between a user and the system. The user would provide a specific identification for the order to check its status and the system will give the variable of the delivery status of the order to the staff member or customer.

G. Domain Model Class Diagram

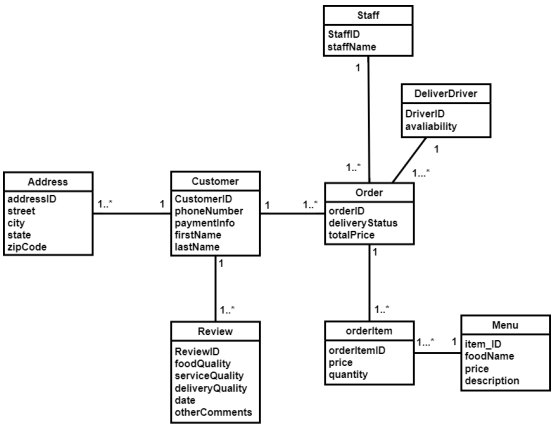

Fig. 5 A Domain Model Class Diagram

From the use cases we created, we can produce a domain model class diagram by determining who uses the use cases and what attributes or variables will the use case use and gather. From the Search Menu use case, we are able to discern that we will need a customer class as the customer actor is the one to use it. The Search Menu use case also implies that there should be a menu class as there needs to be a class for each of the food items on the menu. This menu class also is out of scope for this system and therefore will often already have pre-existing items inside the database for our system to interact with. The Fill Shopping Cart use case and Pay Order use case show us that a staff class and customer class are needed as this use case can be used by the staff as well as customers. Those use cases also indicate that it will need an order class for the customer's order. Since each order may contain more than one item, an additional class would be created for each item inside the order itself which would be named orderItem. Furthermore, the order that contains all the items need to be pulled from the pre-existing menu class which is how we derive the existing connection between orderItem and menu classes. From the Check Status use case, we can recognize that the order class we made will require a deliver status attribute that the Check Status use case can obtain. From the Rate/Review Service use case, a review class can be made to store the data of the customer's reviews. Cancel Order, View Order, and

View Review use cases do not have anything to analyze as the classes and attributes that can be inferred from them have already been made from the previous use cases. The last use cases to analyze are the Ready Orders and Update Status use cases which tell us that we need a driver class for the driver actor using both use cases. An address use case can be made to signify that a customer can have multiple addresses so they choose which location to deliver to.

III. Design

A. *Database Schema*

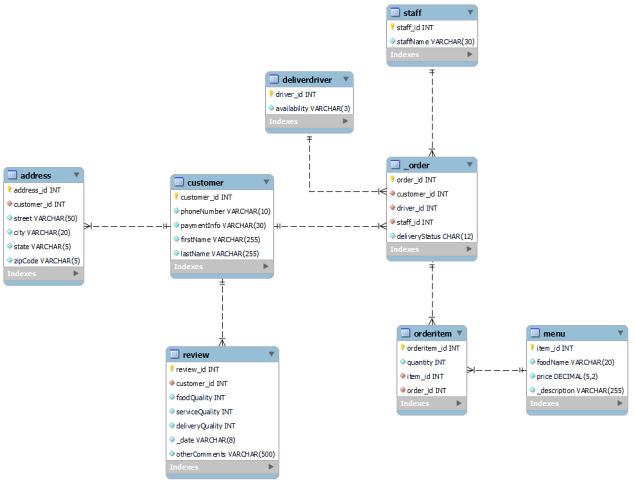

Fig. 6 A Database Schema

The shift from the analysis phase into the design phase requires the shift from the domain model class diagram into a database schema. To start the transition into a database schema, each class in the domain model class diagram would require a new attribute for identification. This ID will be their primary keys. The next step is to update each attribute into a variable type, where most non-primary key attributes are strings. Only identification attributes are placed as integers. Specific attributes like totalPrice had been removed from this schema because of the third normal form as totalPrice relies on the price of the orderItem and Menu classes.

B. *Graphical User Interface*

The next step in designing the implementation of this system requires the user to be able to interact with it. For the user to be able to interact with the system, a graphical user interface is important as the user will have a visual component to be able to discern how to either input data or use the system.

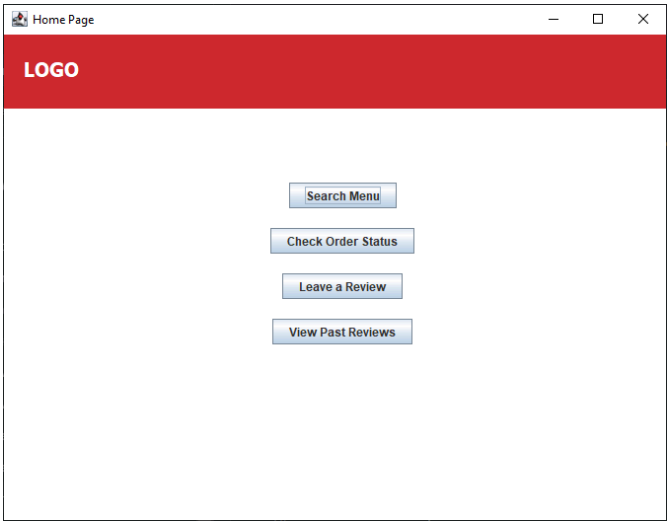

Fig. 7 The GUI for a customer's main page

For the graphical user interface of the customer's main page, the customer has options to interact with the interface through the different buttons which call on some main functions. The "Search Menu" button is for the customer to open a new window to choose their food and then complete their order. The "Check Order Status" button allows the customer to check their delivery order through a window that displays the status of the order. The "Leave a Review" button is for customers to be able to leave a review about the restaurant's quality of service and food. The "View Past Reviews" button opens a new window for the customer to see previous reviews other customers have made about the system.

The staff also have their own main page that has similar elements to the customer page. The staff's user interface looks overall the same and has the equivalent buttons of "Search Menu" and "View Past Review." There are still some differences as the staff has access to all of the orders made by the customers. This is accessed through the "View Orders" button which will open a window with all of the orders. Additionally, the status of the orders can be changed to update its current progress.

Finally, the last main page is for the delivery drivers. There are only two buttons on this page as drivers are only able to access two functions. The first button is the "Ready Orders" button which will

open up a window that shows all orders with the status "Ready to deliver." The driver then can select which orders they will deliver. The second button is the "Update Status" button which will open a window of the current orders the drivers picked up for delivery. The driver can then update the status to "delivered" if the delivery has been completed.

C. Development

An iterative development plan was how we proceeded with creating this system. We first went through an analysis of how the system would work which created the different diagrams such as the use case diagram and the domain model class diagram. We used the use case diagram to figure out specific functions the different users would be able to use and which users were allowed access to different data. The domain model class diagram helped with managing the data that would need to be collected by the system. This allowed us to frame how our classes would save the data.

After implementing the diagrams and analysis into the design of the system, we shifted toward an agile development process as there were some unexpected features we did not plan for. Some features included a sign-in and create account page for different users which we did not plan for in our use cases. This was necessary to collect specific data for the attributes we had with actor-specific classes.

IV. Findings and Suggestions

A. Findings

While analyzing and building this system, we found that this system is easily adaptable to other subsystems within a restaurant such as the admin systems for upkeep of the restaurant inventory as well as the menu creation. An application of this system in a restaurant will allow customers to order for themselves without communicating with a waiter or waitress but also keep the familiarity of ordering from waiters by allowing staff members to input orders. If this system can become a part of a bigger and more complex system, collecting data on certain choices customers make can help improve sales and other aspects of the restaurant.

B. Suggestions

With the creation of this system, we realized we had some missing aspects we could have added but did not have enough time. Some of these aspects involve our created classes. For the customer class, we had an attribute for payment information that was difficult for us to implement into the actual online system. This was because we had just created the payment information as a string instead. After implementing this into the system, we realized that the payment information should be converted into a class because customers could have multiple forms of payment information. The payment information could also have been separated into different attributes instead of the debit or credit card numbers or the expiration dates. We also forgot to implement use cases and functions to be able to update or delete a customer's, staff's, and driver's accounts. This was important so customers could change or add payment info or their physical addresses.

V. Conclusion

A. Difficulties

While we worked on analyzing, designing, and implementing the system, we experienced many hurdles to overcome. One challenge was considering all forms of communication that can be introduced to create an order from a customer and decided to shift the system to an online ordering and in-person ordering system. Our solution to this was combining the online ordering to have the same GUI so the employees and customers would order through the same front end.

Another difficulty was how we collaborated. We collaborated through Github for version control but we were using different IDEs to write the code for the creation of the system resulting in many incompatibilities in terms of file structure and compilation. While this wouldn't usually result in a problem if we were still using the same coding language, the connection of the SQL database was different as well as how Java Swing interacted in the differing IDEs. Each of us had problems connecting to the database and we were not able to resolve these issues to be able to test the databases. Having different IDEs made things difficult for us

as we needed different solutions to connect to the database.

B. What We Learned

After this project, we learned how to analyze real-world problems and design solutions for them in the context of creating systems. Through analyzing the problems, we can use our learned methods and systemically create diagrams to dissect the problem into smaller problems to solve. This analysis phase created a stronger foundational understanding of how systems work and their interactions with data consumption and storage. The diagrams coupled with the understanding of how iterative development worked helped design the implementation of a small part of the bigger solution for those problems to then assemble into a more whole and complete system that will help to solve the problem at large.