Group 7

Final Report – V2.0
UCI CS296P – Fall 2018

Robo-waiter

Chenyang Ren
chenyar@uci.edu

Chun-Kai Chan
chunkac@uci.edu

Shu-Fan Lin
shufanl1@uci.edu

Yi-Dar Liu
yidarl@uci.edu

Yutong Li
yutol12@uci.edu

December 12, 2018

Table of Contents

A. REVISION HISTORY

Date	Comment	Version
12/08/18	Merge requirements document and design document	0.1
12/10/18	First version	1.0
12/12/18	Group members' feedback was reviewed and either incorporated or dismissed.	1.1
12/12/18	Finalize document	2.0

B. INTRODUCTION

This document details the requirements and design of Robo-waiter, a system that receives orders from customers and allows employees to manage orders at a restaurant.

Robo-waiter consists of two subsystems: a chatbot and an order management system (OMS). The chatbot receives text messages from customers, interprets the sentences, and responds with either questions or confirmations. Once an order is confirmed, the chatbot passes the order information to the OMS. Employees can access the OMS to view/process the order through a browser. The goal of Robo-waiter is to make ordering food an enjoyable experience.

The document is organized as six sections:

A. **Revision History:** Provides the date of and the reason for the modification of this document.
B. **Introduction:** Provides a brief description of the system as well as the document structure.
C. **Executive Summary:** Provides an overview of the system.
D. **System Requirements:** Provides the requirements for the system.
E. **System Design:** Describes the architecture and design of the system.
F. **Glossary of Terms:** Defines technical terms mentioned in this document.

C. EXECUTIVE SUMMARY

C.1 Overview

Many restaurants face the following problems:

- Hiring a waiter is expensive, which may lead to customers' long wait of ordering food.
- A waiter may misunderstand customers' requests and order the wrong food.
- For non-native English speakers, sometimes it is difficult to communicate with a human waiter.

Robo-waiter solves these problems. It consists of two subsystems: a chatbot and an order management system (OMS). The chatbot lets customers order food using text messages. Customers can also view the menu and ask for a product's information, e.g., price and ingredients. Once an order is confirmed, it will be fed into the OMS. Through the system, employees can view/process the order. By viewing all upcoming orders, chefs are able to manage the food production process more efficiently. We believe Robo-waiter can help improve the satisfaction of customers, especially international ones, and order fulfillment speed.

In summary, Robo-waiter has the following potential benefits:

- **For All Customers:** Order food quickly and correctly
- **For International Customers:** Order food comfortably as well
- **For Chefs:** Prepare food more efficiently

Technically, the chatbot uses Facebook's messaging platform Facebook Messenger as the user interface, uses Google's chatbot platform Dialogflow to handle natural language processing (NLP), and uses MySQL as the database to store necessary information. After a user inputs a standard English sentence in the text box of Facebook Messenger, the chatbot will use NLP features to parse the utterance, determine the intent, and take the corresponding action. There are two types of actions:

- **Query Information:** Look for the information in the database and return it if found.
- **Order Food:** Create a new order and ask clarifying questions. Once the order is confirmed, it will flow into the database.

The OMS is a web app developed using Python and Django. It displays order information to employees and allows them to modify order status. The payment process remains the same.

In order to illustrate the functions of Robo-waiter, we use an imaginary burger store – MCS Burger, which offers burgers, fries, and drinks – as an example. Please refer to the appendix for its menu.

C.2 Users

The users of Robo-waiter contain the following categories:

- **Customers:** People who would like to order food at a restaurant.
- **Chatbot Administrators:** Employees who monitor the usage of chatbots and provide assistance if necessary.
- **Cashiers:** Employees who accept payments from customers.
- **Chefs:** Employees who prepare food to fulfill orders.
- **Deliverers:** Employees who deliver food to customers.

C.3 Major Features

The following are the major features of the system and the rationale for each:

- **Facebook Messenger Interface:** Can receive orders via text message. All customers can have a greater control of their orders, and international customers can order food comfortably.
- **Natural Language Processing (NLP):** Can understand various expressions of the same request. Customers can have more flexibility to express their needs.
- **Flexible Ordering Process:** Supports ordering multiple items in one sentence, adding customizable options, modifying an item, removing an item, and cancelling an order. Customers can modify their orders easily.
- **Slot-filling System:** Will ask clarifying questions if a request is ambiguous. It is particularly useful for new customers.
- **Order Management System (OMS):** Provides a web-based tool for employees to manage orders. Chefs can be fully aware of what is being ordered and optimize their production processes accordingly.

C.4 Assumptions

The following are the assumptions made about the existing world:

- Customers can read and type standard English.
- Customers are familiar with Facebook Messenger.
- The chatbot solution is cheaper than hiring more employees to receive orders.

D. SYSTEM REQUIREMENTS

This section provides the requirements for Robo-waiter. It serves as a reference for the system designers and programmers during the development of the final version of the system.

This section is organized as follows:

1. **Application Context:** Provides the application context.
2. **Functional Requirements:** Describes the proposed software product, including its capabilities and attributes.
3. **Non-functional Requirements:** Lists the software qualities.
4. **Risks:** Lists the known risks to the project.
5. **Implementation Phases:** Lists priorities of the system's functionality.
6. **Future Directions:** Lists future directions of improving the system.

D.1 Application Context

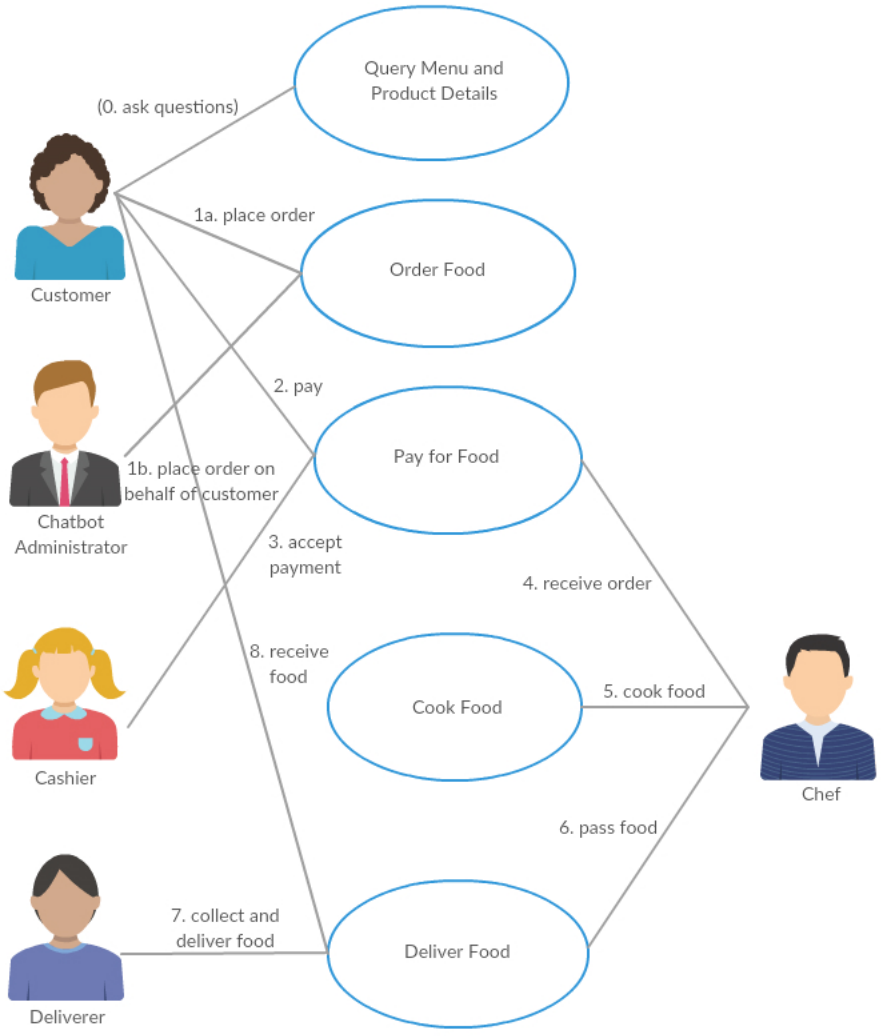
Robo-waiter consists of two subsystems: a chatbot and an order management system (OMS). Customers can converse with the chatbot through Facebook Messenger, and employees can access the OMS through any web browser. Both subsystems run on in-store tablets and require access to the Internet.

D.2 Functional Requirements

This subsection is organized as follows: First, we show a use-case diagram and an example to illustrate the usage of Robo-waiter. Next, we provide functional requirements which are categorized by user.

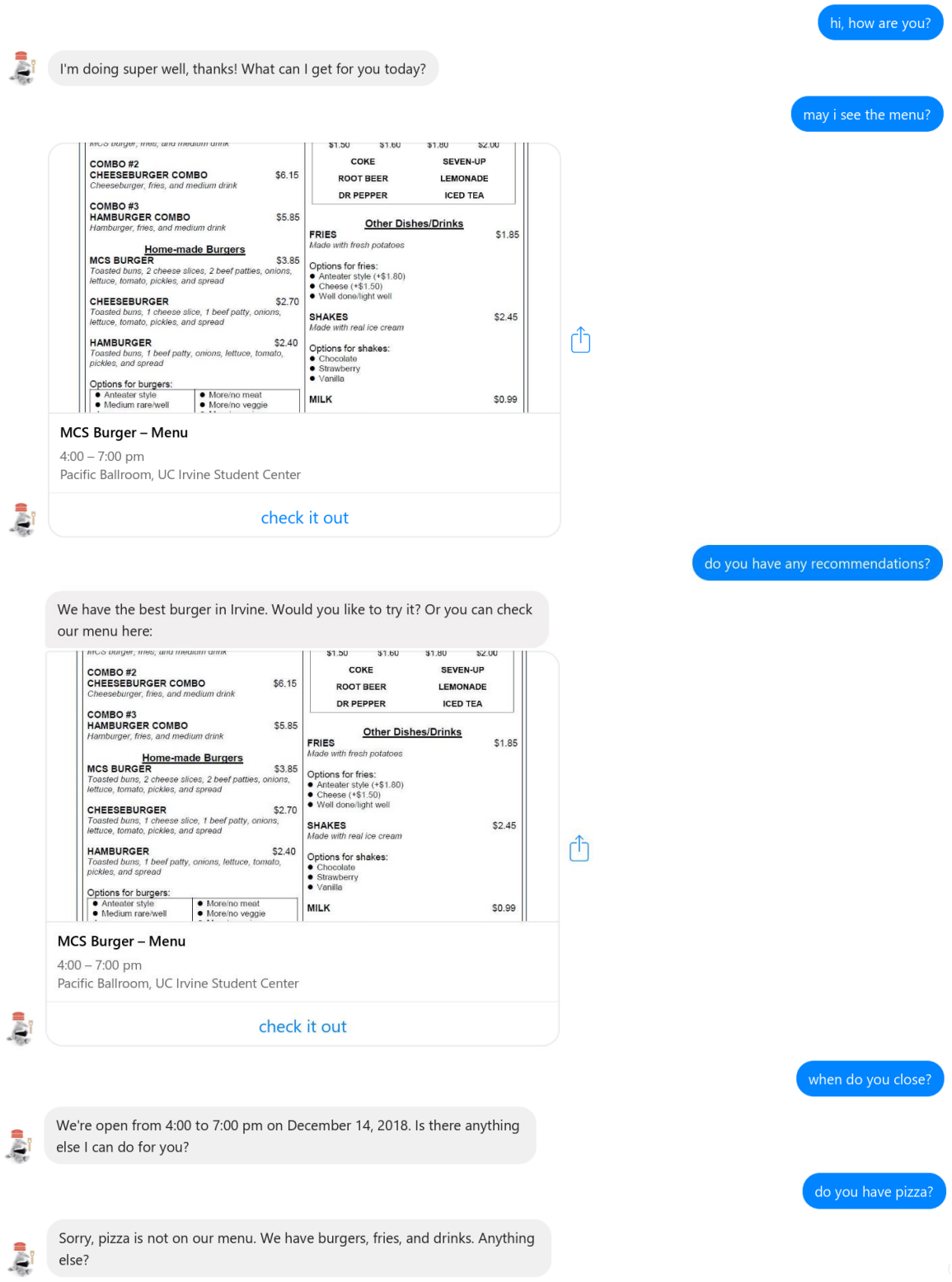
D.2.1 Use-Case Diagram

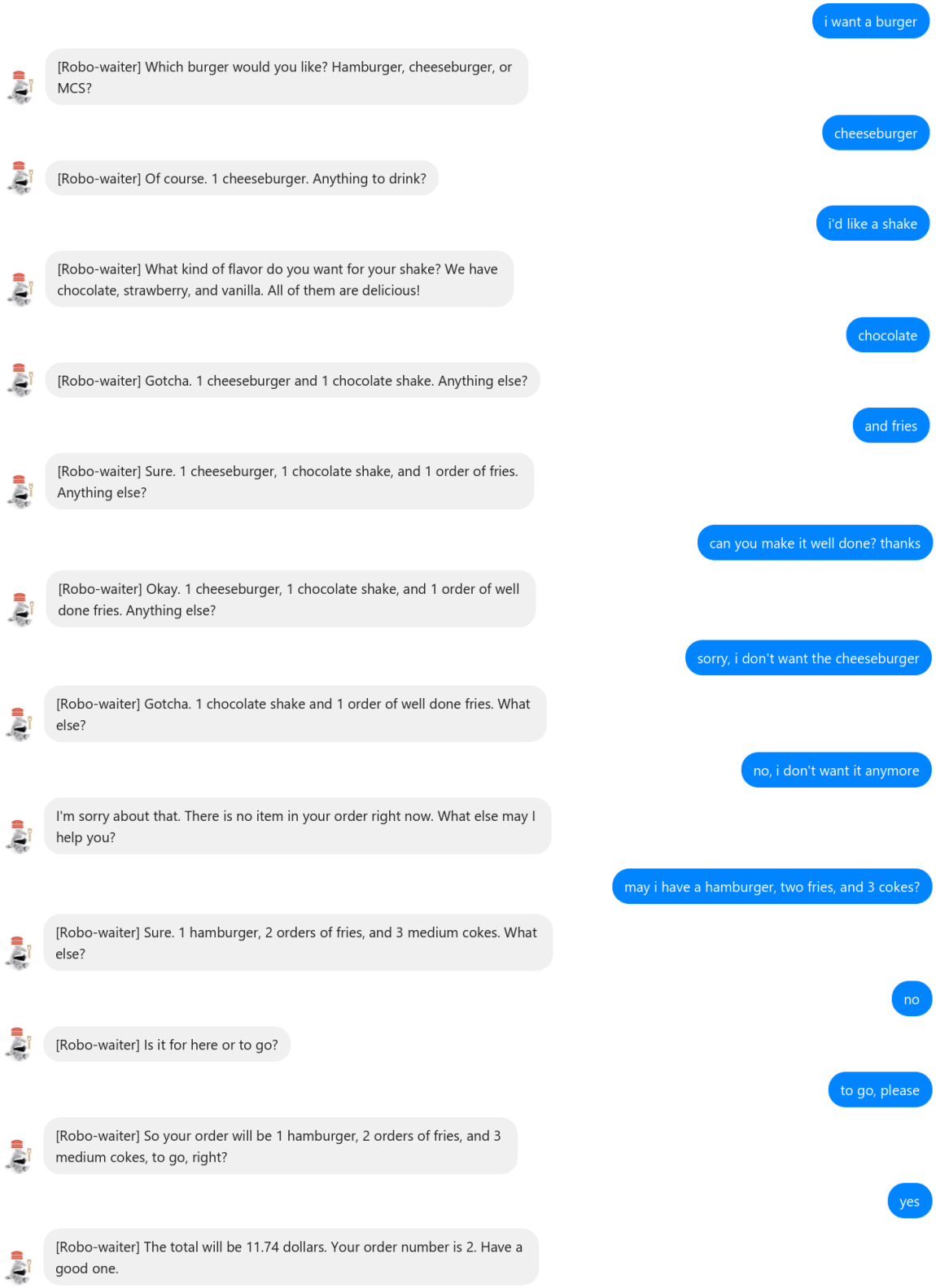
The following use-case diagram shows the relationships between the users and the different use cases in which the users are involved. We will explain the diagram with an example in the next part.



D.2.2 Usage Example

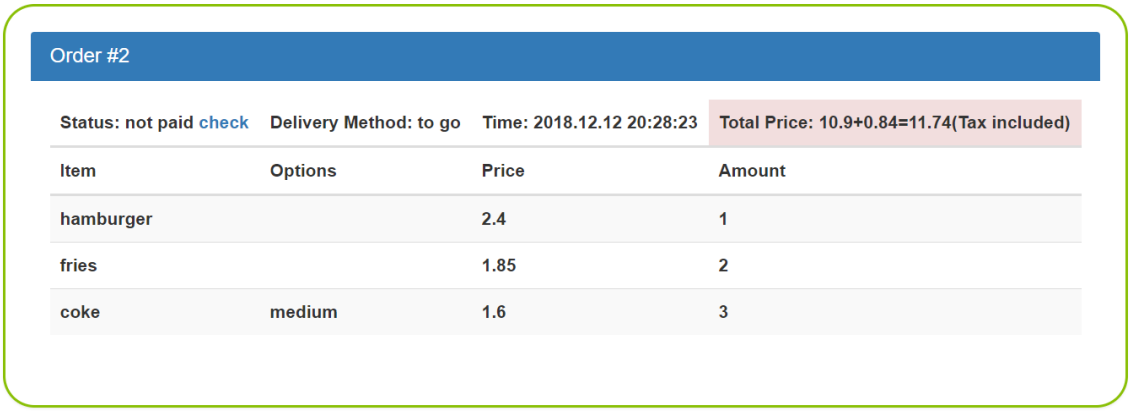
The following is an illustrative conversation between a customer and the chatbot. It is intended to show the functions of our system:





If customers have any difficulty conversing with the chatbot, a chatbot administrator will step in to offer help.

Once a customer finishes ordering food, the order will be fed into the OMS. The following table shows employees' view of the system:



Order #2

Status: not paid check	Delivery Method: to go	Time: 2018.12.12 20:28:23	Total Price: 10.9+0.84=11.74(Tax included)
Item	Options	Price	Amount
hamburger		2.4	1
fries		1.85	2
coke	medium	1.6	3

When the customer reports the order number and completes the payment at the counter, the cashier will look up the order in the system and modify the status from "not paid" to "not cooked." Next, chefs see this order on their screens and start to prepare the food. When the food is ready, the chefs will modify the status from "not cooked" to "not delivered" and pass it to a deliverer, who will call the order number. Once the customer picks up the food, the deliverer will modify the status from "not delivered" to "delivered" and complete the order fulfillment process.

D.2.3 Product Features

The features for each subsystem/user group are as follows:

Subsystem: Chatbot/User Group: Customer

- Can converse with customers via text messages
- Understand standard English sentences frequently spoken during conversations of ordering food
- Show the menu
- Show recommendations
- Show the requested information of the store, e.g., open hours and payment method
- Show the requested information of menu items, e.g., ingredients and price
- Accept orders for all of the items in the menu
- Deny orders for items not in the menu
- Ask clarifying questions if an order is ambiguous (slot-filling system)
- Allow modifications on the items
- Allow an item to be added to the order
- Allow an old item to be replaced by a new item in the order
- Allow an item to be removed from the order
- Allow cancellation of an order
- Allow selection and modification of the delivery method
- Place an order in the OMS and display the entire order, the delivery method, the price, and the order number to the customer after the order is confirmed

Subsystem: OMS/User Group: Employee

- Provide different access rights for different roles (Cashier, Chef, and Deliverer)
- Show the order details
- Allow modification of the order status
- Allow search of an order by keyword

D.3 Non-functional Requirements

Robo-waiter has the following qualities:

- **Ease of Use:** Most customers can immediately know how to use it or how to respond without pre-training. Only 10% of customers need a chatbot administrator's assistance.
- **Efficiency:** It is able to respond to a customer's request quickly. The average response time is 2 seconds, and the maximum response time is 3 seconds.
- **Accuracy:** It is able to parse a customer's utterance correctly and take the correct action. The overall accuracy of a response is close to 75%.
- **Reliability:** It can run smoothly without crashing.
- **Scalability:** It can receive hundreds of orders per day.
- **Security:** Employees cannot modify order content; instead, they can only modify the order statuses which are appropriate for their roles.

D.4 Risks

The following are known risks to the project:

- **Financial Risk:** We need to upgrade our tools to the paid editions to be able to use some key functions.
- **Business Risk:** Other people may have launched a similar or better product.
- **Legal Risk:** If our Facebook Messenger chatbot can be accessed by the public, we have to obey the regulations enforced by Facebook, such as data protection laws. If we violate one of them, we may be sued by the company.
- **Ethical Risk:** Robo-waiter may be seen as discriminating against illiterate, blind, and old people. In this case, chatbot administrators will assist them to order food.
- **Project Management Risk:** We may not have enough time to complete the project before the deadline (December 14, 2018).
- **Development Risk:** Our NLP model is unable to handle all user inputs correctly.
- **Deployment Risk:** The chatbot uses an external messaging platform as the user interface, and the OMS is hosted in a cloud platform. If one of the services terminates, we will need to seek an alternative platform.

D.5 Implementation Phases

D.5.1 Chatbot

Implementation of the chatbot is prioritized as follows, beginning with the highest priority feature:

Must Have (first):

- Can converse with customers via text messages
- Understand standard English sentences frequently spoken during conversations of ordering food
- Accept orders for all of the items in the menu
- Allow selection and modification of the delivery method
- Place an order in the OMS and display the entire order, the delivery method, the price, and the order number to the customer after the order is confirmed

Should Have (second):

- Deny orders for items not in the menu
- Ask clarifying questions if an order is ambiguous
- Allow modifications on the items
- Allow an item to be added to the order

Nice to Have (third):

- Show the menu
- Show recommendations
- Show the requested information of the store, e.g., open hours and payment method
- Show the requested information of menu items, e.g., ingredients and price
- Allow an old item to be replaced by a new item in the order
- Allow an item to be removed from the order
- Allow cancellation of an order

D.5.2 OMS

Implementation of the OMS is prioritized as follows, beginning with the highest priority feature:

Must Have (first):

- Show the order details
- Allow modification of the order status

Should Have (second):

- Provide different access rights for different roles (Cashier, Chef, and Deliverer)

Nice to Have (third):

- Allow search of an order by keyword

D.6 Future Directions

D.6.1 Chatbot

The following are future directions of improving the chatbot:

- The chatbot can support multiple languages.
- Customers can converse with the chatbot via voice. (The chatbot can be installed in a Furby-like robot instead of a tablet.)
- Customers can choose to pay via chatbot (maybe through credit card).
- Customers can create aliases for their orders and directly use aliases to order food.
- Customers can grade user experience or leave comments after ordering food.
- Customers can look up a word through the chatbot.
- Chefs can prevent an item from being ordered if they run out of its ingredients.

D.6.2 OMS

The following are future directions of improving the OMS:

- The OMS can be automatically reconciled with the payment system and identify any amount gap.
- The OMS can automatically assign orders to chefs.
- The OMS can utilize order data to generate management reports.
- The OMS can be integrated with the inventory management system so that an ingredient will be automatically purchased when its inventory level falls below a threshold.

E. SYSTEM DESIGN

This section describes the architecture and design of Robo-waiter. It is intended to provide guidance to the development team.

This section is organized as follows:

1. **Design Principles:** Describes the underlying principles of our system design.
2. **System Architecture:** Shows the architecture, components, and data flows of our system.
3. **Component Design:** Describes the design and its rationale of each component.
4. **System Interfaces:** Explains how to connect all components together.
5. **Requirements Matrix:** Shows how each requirement is satisfied by our design.

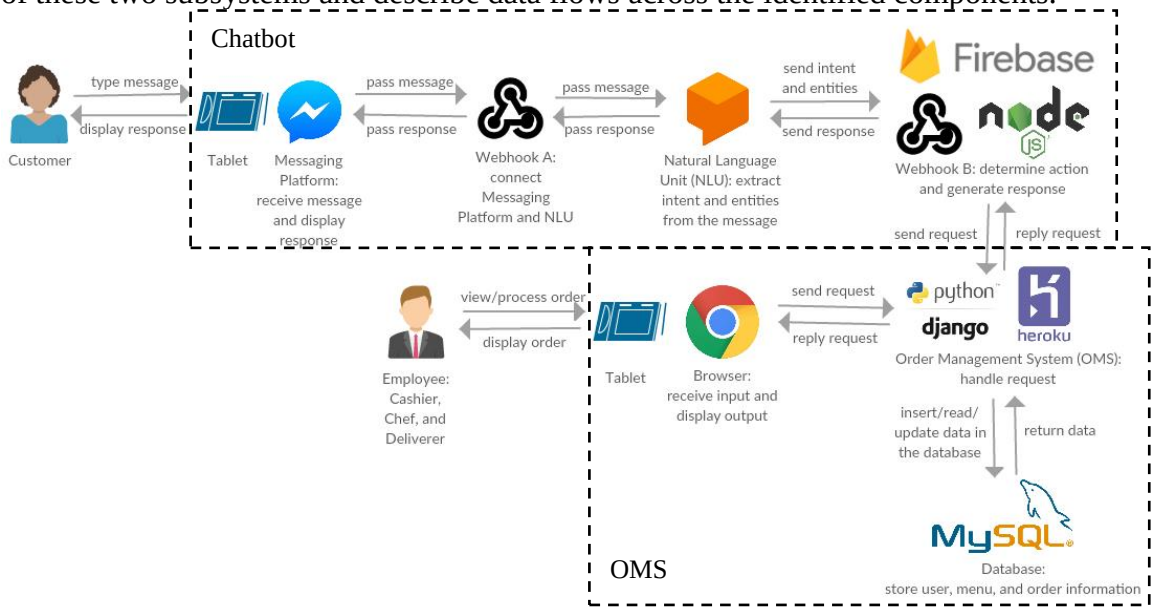
E.1 Design Principles

The principles of our system design are as follows:

- **User-oriented:** When customers order food through our system, we want them to feel like conversing with a human being. Besides, we also want our system to help employees manage orders more efficiently.
- **Easy to Implement or Familiar to Team Members:** We chose solutions which are either developer-friendly or familiar to our team members in order to complete the project within the time limit.
- **Free to Implement:** We only use free tools to implement our system due to our budget limitation.

E.2 System Architecture

This subsection will give a holistic view of our system. The following diagram shows the architecture and data flows of our system, which consists of two subsystems: a chatbot and an order management system (OMS). In this subsection, we will identify components of these two subsystems and describe data flows across the identified components.



E.2.1 Chatbot's Components

The chatbot lets customers place orders via text messages. It contains the following components:

- **Messaging Platform:** Acts as a user interface which receives user input and displays the response to users.
- **Natural Language Unit (NLU):** Extracts intents and entities from user input. It is the core component of the chatbot.
- **Webhooks:** The chatbot has two webhooks. Webhook A connects the messaging platform and the NLU, and Webhook B connects the NLU and the OMS. The latter is also responsible for determining actions and generating messages in response to user input.

E.2.2 OMS's Components

The OMS is a web app that enables employees to view order details and modify order status. It consists of the following components:

- **User Interface:** Displays order information and allows users to search an order and modify order status. It can be accessed through a web browser.
- **Application Server:** Receives and handles requests from clients (browsers and the chatbot). It is the integral part of the OMS.
- **Database:** Stores user, menu, and order information.

E.2.3 Data Flow of the Chatbot

Suppose that a customer types a text message in the text box of the messaging platform installed in a tablet. Webhook A relays the message from the messaging platform to the NLU, which will extract the intents and entities. The NLU sends the extraction results to Webhook B, which will determine the action based on the received data. There are two possible actions:

- **Generate a Response Based on Data from the Database:** Webhook B sends a query request to the OMS, which will perform the query. After retrieving the data from the database, the OMS will return it to Webhook B, which will utilize the data to compose a response and send it back to the NLU. The NLU passes the response to Webhook A, which will continue to pass it to the messaging platform. Finally, the messaging platform displays the response to the customer.
- **Insert a Confirmed Order into the Database:** Webhook B sends the request to the OMS, which will store the order into the database.

E.2.4 Data Flow of the OMS

Suppose that an employee accesses the OMS through a web browser installed in a tablet. The client (i.e., the browser) sends a GET request to the application server, which will retrieve order information from the database. Next, the application server returns the data to the client. Finally, the browser displays to the employee the webpage containing the order information. If an employee modifies the status of an order, the client will send a PUT request to the application server, which will update the order in the database.

E.3 Component Design

This subsection will discuss about how we implement each component mentioned in the last subsection. We will start from the back-end, i.e., the database and the applications, and then move towards the front-end, i.e., the user interface.

E.3.1 Database

We utilize a MySQL database to store required information because our team members have experience with it. Besides, MySQL is open-source and free.

Our database consists of six relations. The following tables show their information and attributes. The UserProfile relation stores user information. The ItemMenu and OptionPrice relations store menu information. Because available options depend on product category instead of individual product item, we create a Category attribute to connect the two relations. The OrderDetail, OrderItem, and OrderItemOption relations store order information. They are connected by the OrderNumber and ItemNumber attributes. The reason we use VARCHAR to store all attributes (except Picture) is to simplify the data processing of the OMS.

Relation: **UserProfile**
Purpose: Store user information
Primary Key: UserName

Attribute	Data Type	Length	Definition	Example
UserName	VARCHAR	20	User name	Helenkeller
Password	VARCHAR	20	Password associated with the user name	Abcd1234
Name	VARCHAR	20	Actual name of the user	Helen Keller
Group	VARCHAR	20	Role of the user; it can be Cashier, Chef, or Deliverer	Chef

Relation: **ItemMenu**
Purpose: Store product information
Primary Key: Item

Attribute	Data Type	Length	Definition	Example
Item	VARCHAR	20	Name of the product item	Fries
Price	VARCHAR	20	Base price of the product item (without any option); for the Drink category, base price is the price of medium size	$1.85
Ingredient	VARCHAR	80	Content of the product item	Made with fresh potatoes
Picture	Blob		Image of the product item	
Category	VARCHAR	20	Used to look up the OptionPrice relation; it can be Burger, Fries, Shake, Drink, or Others	Fries

Relation: **OptionPrice**
Purpose: Store price adjustment associated with each option
Primary Key: Category + Option

Attribute	Data Type	Length	Definition	Example
Category	VARCHAR	20	Product category associated with the option; it can be Burger, Fries, Shake, Drink, or Others	Fries
Option	VARCHAR	20	Available option under this product category; for example, the Fries category's option can be Anteater Style, Cheese, Well Done, or Light Well	Anteater Style
PriceAdjustment	VARCHAR	20	Adjustment for the base price if the option is specified; positive value means markup, and negative value means discount	$1.80

Relation: **OrderDetail**
Purpose: Store order information
Primary Key: OrderNumber

Attribute	Data Type	Length	Definition	Example
OrderNumber	VARCHAR	20	System-generated number that represents a confirmed order	5
DeliveryMethod	VARCHAR	20	Delivery method of the order; it can be For Here or To Go	To Go
Time	VARCHAR	20	Time when the order is confirmed	11/10/18 17:23
Status	VARCHAR	20	Status of the order; it can be Unpaid, Uncooked, Undelivered, and Delivered	Unpaid
Cashier	VARCHAR	20	User name of the employee who processes the payment of the order	Maryjane
Chef	VARCHAR	20	User name of the employee who cooks the food	Williamheinger
Deliverer	VARCHAR	20	User name of the employee who delivers the food to the customer	Amberschinkel

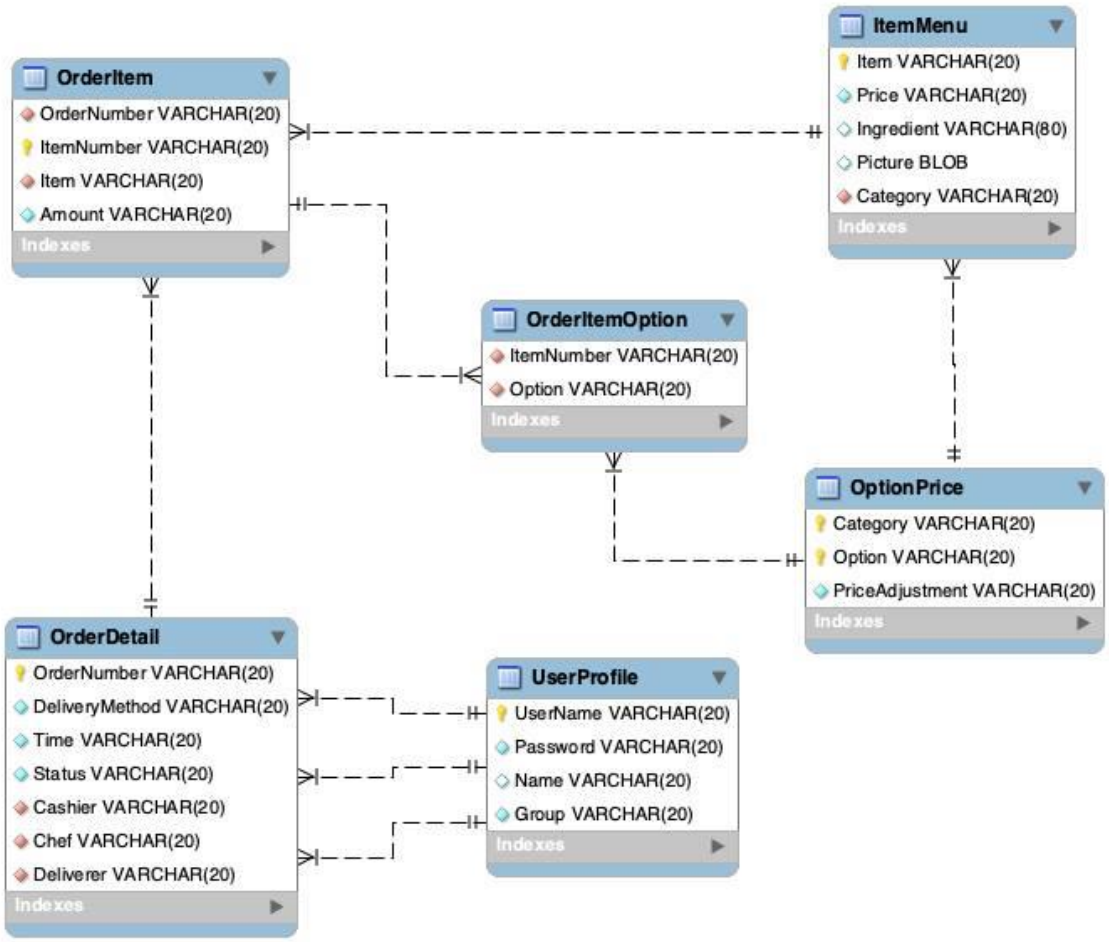
Relation: **OrderItem**
Purpose: Store items in each order
Primary Key: ItemNumber

Attribute	Data Type	Length	Definition	Example
OrderNumber	VARCHAR	20	System-generated number that represents a confirmed order	5
ItemNumber	VARCHAR	20	System-generated number that represents an item in an order	18
Item	VARCHAR	20	Name of the product item in the order	Fries
Amount	VARCHAR	20	Quantity of the product item	2

Relation: **OrderItemOption**
Purpose: Store options associated with each item in an order
Primary Key: ItemNumber + Option

Attribute	Data Type	Length	Definition	Example
ItemNumber	VARCHAR	20	System-generated number that represents an item in an order	18
Option	VARCHAR	20	Desired option associated with the item	Anteater Style

The relationships between the relations can be illustrated by the following relational diagram. We can calculate the total price of an order by looking up the Price attribute in the ItemMenu relation for each associated Item in the OrderItem relation and the PriceAdjustment attribute in the OptionPrice relation for each associated Option in the OrderItemOption relation.



Legend:

- : Primary key
- : Cannot be null
- : Can be null
- : Foreign key
- : 1-n relationship

E.3.2 Natural Language Unit (NLU)

The chatbot needs to understand various expressions of intents, so it has to be implemented using a chatbot platform with natural language processing (NLP) features. After evaluating all free NLP tools including Wit.ai and LUIS.ai, we chose Dialogflow (originally Api.ai) due to the following reasons:

- It is the most popular free chatbot platform, so we can find many related examples and discussions on the Internet.
- It is easy to integrate with multiple platforms, including Facebook Messenger.
- It implements the slot-filling system, which makes it easy to collect missing information.
- It is more developer-friendly than Wit.ai while requiring more coding than LUIS.ai, so it strikes a nice balance between speed and complexity.

The following figure shows how Dialogflow handles user input:

1. **User Utterance:** First, it receives user utterance.
2. **Intent Matching:** Next, based on the words in the utterance, it uses a proprietary machine learning algorithm to calculate a confidence value that ranges from 0.0 (completely uncertain) to 1.0 (completely certain) for each intent. If any confidence value is higher than a given threshold, Dialogflow selects the intent with the highest confidence value; otherwise, it selects the default fallback intent. Entities are also extracted in this phase.
3. **Response:** Finally, Webhook B generates a response based on the selected intent and the extracted entities. The details will be discussed in the next subsection.

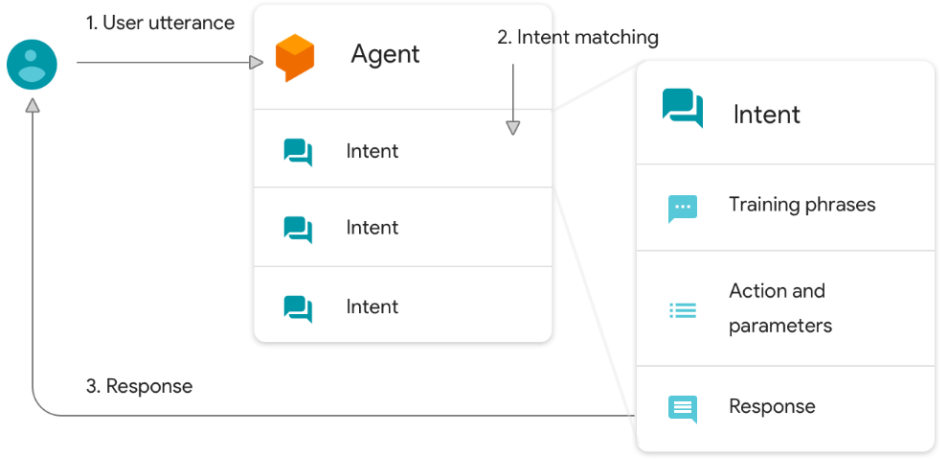


Figure 1. Example of how Dialogflow handles a user utterance.

Source: https://dialogflow.com/docs/agents

We use a hybrid method to handle intent matching. It means that during the intent matching phase, Dialogflow tries to find an exact match first; if none of the training phrases perfectly matches the utterance, then the machine learning algorithm is employed to select the intent. The threshold of the default fallback intent is set to be 0.3, which is recommended by Dialogflow.

In order to let Dialogflow identify entities, we built an entities structure in Dialogflow based on our menu (which is in the appendix). The basic framework of entities is as follows: (The @ symbol indicates a set of words. An example is provided in the curly brackets.)

- @all-product
  - @all-item
    - @food
      - ✓ @burger {anteater style hamburger with no onion}
      - ✓ @fries {well done fries}
    - @drink
      - ✓ milk
      - ✓ coffee
      - ✓ cocoa
      - ✓ @shake-option shake {chocolate shake}
      - ✓ @normal-drink {coke}
      - ✓ @drink-option @normal-drink {medium coke}
    - @combo {hamburger combo}
  - @item-amount @all-item {2 hamburgers}
- @all-option
  - @burger-option {anteater style}
  - @fries-option {well done}
  - @shake-option {chocolate}
  - @drink-option {medium}
- @item-amount
  - @number-equivalent {a}
  - @sys.number {1}
- @product-attribute {price}
- @delivery-method {for here}

In order to train Dialogflow's machine learning algorithm, we collected English sentences frequently spoken during conversations of ordering food and input them into Dialogflow as training phrases. The intents, the training phrases (selected), and the corresponding responses in our NLP model are shown in the following table:

Intent	Training Phrases (Selected)	Entities	Response
Welcome/ Greetings/ Query Name	Hey you Hi, how are you? Who are you?		Give a reasonable, polite, and friendly response followed by "What can I get for you today?"
Query Menu	May I see the menu?		Show the menu
Query Recommendation	Do you have any recommendations?		Show the recommendations
Query Hours/ Payment	When do you close? How should I pay?		Provide store-level information
Query Product/ Ingredients/ Price/ Options/ Picture	Do you have {coffee}? What are the {ingredients} in a {cheeseburger}? What is the {price} of a {cheeseburger}? What {kind} of {shake} do you offer? What does {MCS burger} {look like}?	{@all-item} {@product-attribute}	Show the requested information of the product. If the product is not available, then answer "We don't have that, sorry."
Place Order	I want {a hamburger}. I'd like to order {2 cheeseburgers}. Give me {an anteater style MCS burger}.	{@all-product}	If the request is ambiguous, ask clarifying questions; otherwise, create a temporary order containing the requested item(s).
Clarify Burger/Shake	{cheeseburger}. {chocolate}.	{@burger-option} /{@shake-option}	Add the clarified item to the order
Modify Item	I want {anteater style} {fries}. I {don't want lettuce} on that {hamburger}.	{@all-item} {@all-option}	Modify the item
Add Item	And {fries}. Can I have {one} more {cheeseburger}?	{@all-item} {@item-amount}	Add the item to the order
Replace Item	I want {hamburger} instead.	{@all-item}	Replace the old item with the new item
Remove Item	Sorry, I don't want {hamburger}.	{@all-item}	Remove the item from the order
Cancel Order	No, I don't want it anymore.		Delete the order
Finalize Order	No. That would be all.		Ask the delivery method if not yet specified
Choose Delivery	{To go}, please. {For here}, please.	{@delivery-method}	Repeat the item(s) in the order
Confirm Order	Yes. That's correct.		Tell the customer the total price and the order number. Insert the order data into the database.

E.3.3 Webhook B (Response Generator)

In order to generate more dynamic responses, we implemented response handlers in Webhook B instead of using Dialogflow's built-in tool. In terms of Dialogflow's language, response handlers in a webhook are called "fulfillment." When implementing fulfillment, we considered two options: Node.js and Python. We chose Node.js because Dialogflow's fulfillment templates are written in Node.js. Although fulfillment templates written in Python can be found on the Internet, they are written by people outside Dialogflow and are not compatible with the latest version of Dialogflow.

The Response column in Page 23's table lists the response handlers we have implemented in the webhook. They can be categorized into two groups:

- **Query Information:** In the fulfillment code we store part of the menu information, which can be used to provide quick response. If the requested information is not in our code and requires query into the database, the webhook sends a JSON message to the OMS. For example, the following is a message used to query a product's picture:

```
{
    "intent": "QueryPicture",
    "name": "cheeseburger"
}
```

  After obtaining the data (also in JSON) from the OMS, the webhook uses it to compose a response and passes the message to the NLU.

- **Order Food:** When the Place Order intent is initiated, an Order context is created to save the items in the order. Each item is represented by a structure of name, category, options, and amount. Once the delivery method is chosen and the order is confirmed, the webhook sends to the OMS a JSON message containing the order information. The message looks like this:

```
{
    "intent": "PlaceOrder",
    "order": [
            {"name":"cheeseburger",
             "category":"burger",
              "options": ["anteater style", "more cheese", "no onion"], "amount": 1},
            {"name":"fries", "category":"fries", "options": [], "amount": 3},
    ] ,
    "delivery": "for here"
}
```

  The OMS calculates total price and assign an order number, returning the information to the webhook. After obtaining the data (also in JSON) from the OMS, the webhook uses it to compose a response and passes the message to the NLU.

Because Dialogflow provides step-by-step instructions about how to host a webhook on the Firebase server, we deploy the webhook to Firebase. (Both Dialogflow and Firebase are owned by Google.)

E.3.4 Application Server of the OMS

The OMS is a web app hosted on a cloud platform. We selected Python and Django to develop the system because our team members are reasonably familiar with the language and the tool. Besides, Python and Django allow fast implementation and provide many useful built-in modules.

The following table shows the view and functions of each user group of the OMS:

User Group	Order Status in the View	Functions
Cashier	Unpaid	Search an order; change an order's status from Unpaid to Uncooked
Chef	Uncooked	Search an order; change an order's status from Uncooked to Undelivered
Deliverer	Undelivered	Search an order; change an order's status from Undelivered to Delivered

We achieve this goal by doing the following things:

- **Implement a User Authentication System:** Employees have to create accounts in the OMS before logging in the system. Their IP addresses are saved in a dictionary data structure.
- **Implement Different Functions to Handle Requests from Different User Groups:** The functions include viewing orders, searching an order, and modifying an order's status.

The OMS also handles requests from the chatbot, including:

- Query the requested information of a product in the database
- Store a confirmed order into the database and return the total price and the order number

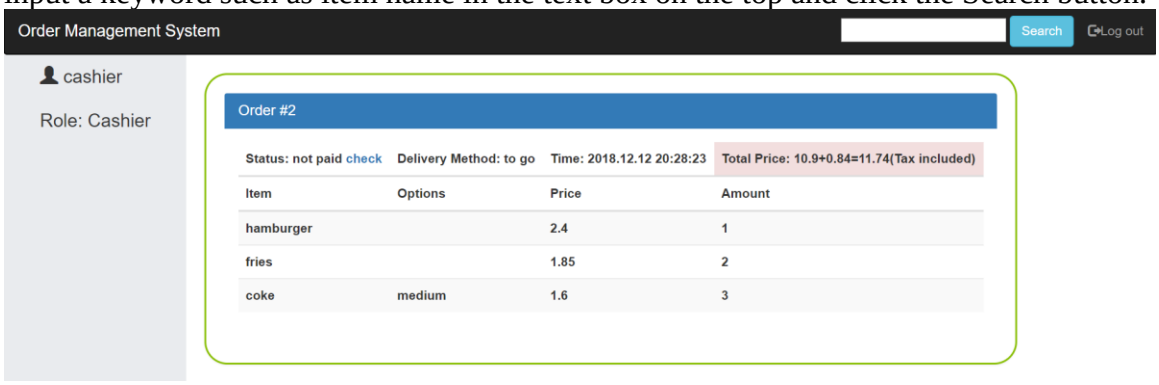
Initially, sockets were used to connect the chatbot and the OMS, but we found that Webhook B, hosted on Firebase, was unable to send messages to a locally hosted app. Therefore, we turned to cloud solutions. After trying three major cloud platforms – Heroku, AWS, and Azure, we decided to deploy the OMS to Heroku for the following reasons:

- **Quick Deployment:** Using Heroku, we can complete the deployment just by pushing our app in a git repository. AWS and Azure's deployment processes are more complicated and difficult.
- **Free of Charge:** Heroku offers a free edition; on the contrary, AWS and Azure will charge after the trial period. The drawback of the free edition is that the server will shut down if there is no traffic during the past 30 minutes. Nevertheless, we think it is sufficient for our demonstration purpose.

E.3.5 User Interface of the OMS

We think the best way to manage tabular information like orders is to provide a web-based interface. Therefore, we decided to implement the OMS as a web app.

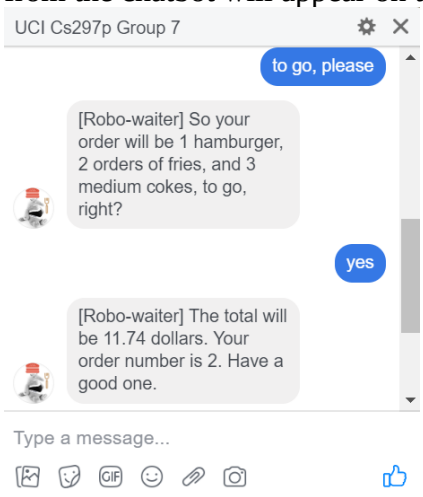
We use CSS, JavaScript, and HTML to implement the user interface. The following screenshot is a sample view of a cashier. It only exhibits orders which are not yet paid. Employees can click the Check hyperlink (in blue) to indicate that the order has been processed by themselves. For example, after cashiers receive the payment of an order and click Check, the order will disappear from their views, and then chefs will see this order emerge on their webpages. If employees want to look for a particular order, they can input a keyword such as item name in the text box on the top and click the Search button.



E.3.6 User Interface of the Chatbot (Messaging Platform)

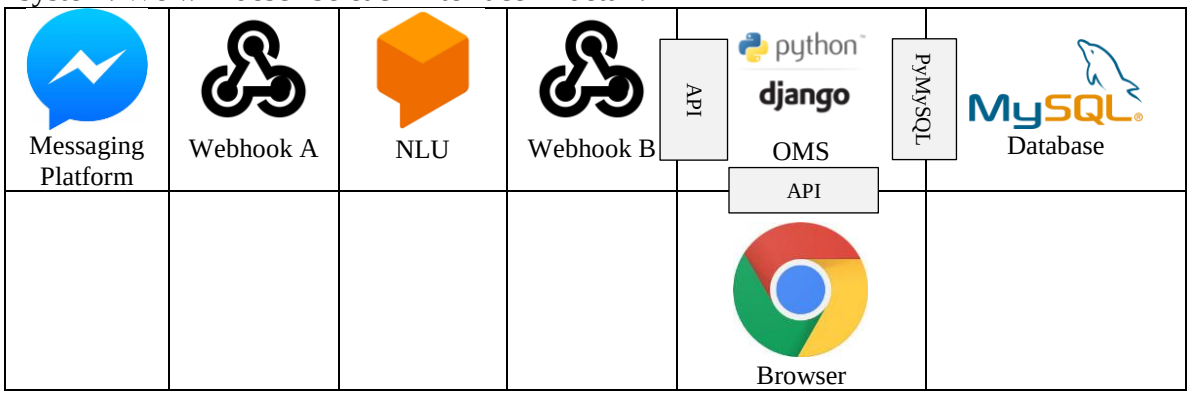
We decided to use Facebook Messenger as the user interface of the chatbot because it is one of the most popular messaging platforms around the world. However, because Facebook has enforced strict policy since this year (2018), if we wanted to make the chatbot accessible to the public, we would have to go through a business verification process which may last weeks to months. Therefore, we still keep it private for now.

The user interface of the chatbot looks like below. After a customer types a message in the text box at the bottom and presses the Enter button on the keyboard, the response from the chatbot will appear on the left of the above dialogue.



E.4 System Interfaces

This subsection will show how we connect the components mentioned in the last subsection. The following figure illustrates the interfaces connecting components of our system. We will describe each interface in detail.



E.4.1 Interfaces of the Chatbot: Webhooks

When a customer sends a text message via Facebook Messenger, we want it to be automatically delivered to the NLU. Similarly, when the NLU receives a request which needs to access the database, we want it to be automatically delivered to the OMS, which executes the query. Therefore, we need to set up webhooks to connect the components.

Webhook A, which connects Facebook Messenger and the NLU, can be easily set up using Dialogflow's built-in integration tool. Webhook B, which connects the NLU and the OMS, is discussed in E.3.3 Webhook B (Response Generator).

E.4.2 Interfaces of the OMS: API and PyMySQL

Since the OMS is hosted on a cloud platform, it can provide the Web API service by assigning a URI to each function. If a client (the chatbot or a browser opening the OMS) wants to use a function of the OMS, it can access the corresponding URI. After receiving the request, the OMS will call the function to fulfill it and return the result in JSON.

In order to simplify the data flow, we assign the OMS as the sole contact window of the database. The OMS is connected with the MySQL database through a Python module called PyMySQL. After the connection is established, the OMS can perform operations on the database using SQL. The data returned by the database is in a Python list data structure, which will be converted into a JSON message before sent to the client.

E.5 Requirements Matrix

The following table demonstrates how each requirement in D.2.3 Product Features is addressed by our system design:

Subsystem	Requirements	Design Reference
Chatbot	• Can converse with customers via text messages	E.3.6 User Interface of the Chatbot (Messaging Platform)
	• Understand standard English sentences frequently spoken during conversations of ordering food	E.3.2 Natural Language Unit (NLU)
	• Show the menu • Show recommendations • Show the requested information of the store, e.g., open hours and payment method • Accept orders for all of the items in the menu • Deny orders for items not in the menu • Ask clarifying questions if an order is ambiguous (slot-filling system) • Allow modifications on the items • Allow an item to be added to the order • Allow an old item to be replaced by a new item in the order • Allow an item to be removed from the order • Allow cancellation of an order • Allow selection and modification of the delivery method	E.3.3 Webhook B (Response Generator)
	• Show the requested information of menu items, e.g., ingredients and price • Place an order in the OMS and display the entire order, the delivery method, the price, and the order number to the customer after the order is confirmed	E.3.1 Database E.3.3 Webhook B (Response Generator) E.3.4 Application Server of the OMS
OMS	• Provide different access rights for different roles (Cashier, Chef, and Deliverer) • Show the order details • Allow modification of the order status • Allow search of an order by keyword	E.3.4 Application Server of the OMS E.3.5 User Interface of the OMS

F. Glossary of Terms

The following are a list of technical terms mentioned in this document, ordered alphabetically:

- **Action:** The steps that a chatbot is capable of committing as a response to the corresponding intent.
- **Application Programming Interface (API):** A set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication among various components.
- **Application Server:** A software framework that provides both facilities to create web applications and a server environment to run them.
- **Binary Large OBject (BLOB):** A collection of binary data stored as a single entity in a database management system. Blobs are typically images, audio or other multimedia objects.
- **Cascading Style Sheets (CSS)**: A style sheet language used for describing the presentation of a document written in a markup language like HTML.
- **Chatbot:** A computer program or an artificial intelligence which conducts a conversation via auditory or textual methods.
- **Client:** The process that initiates the communication in the context of a network communication session between a pair of processes.
- **Dialogflow:** A platform that gives you the ability to build chatbots with natural language processing (NLP) support.
- **Django:** A free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern.
- **Entity:** A modifier of an intent. For example, if a user types "show me yesterday's financial news", the entities are "yesterday" and "financial". Entities are given a name, such as "dateTime" and "newsType". Entities are sometimes referred to as slots.
- **Facebook Messenger:** A messaging app and platform which is used by more than 1 billion users around the world.
- **Firebase**: A mobile and web application development platform developed by Firebase, Inc. It provides multiple services, which include Firebase Hosting. The service is a static and dynamic web hosting service and supports hosting static files such as CSS, HTML, JavaScript and other files, as well as support through Cloud Functions.
- **Fulfillment**: Code that's deployed as a webhook that lets your Dialogflow agent call business logic on an intent-by-intent basis.
- **GET:** A request method supported by HTTP used by the World Wide Web. By design, the GET request method requests a representation of the specified resource.
- **Hypertext Markup Language (HTML)**: The standard markup language for creating web pages and web applications.
- **Hypertext Transfer Protocol (HTTP):** The Web's application-layer protocol.
- **Intent:** The user's intention. For example, if a user types "show me yesterday's financial news", the user's intent is to retrieve a list of financial headlines. Intents are given a name, often a verb and a noun, such as "showNews."
- **JavaScript:** A high-level, interpreted programming language.

- **JavaScript Object Notation (JSON):** An open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).
- **MySQL:** An open-source relational database management system (RDBMS).
- **Natural Language Processing (NLP):** A subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.
- **Natural Language Unit (NLU):** A component in the chatbot which handles NLP.
- **Node.js:** An open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.
- **Order Management System (OMS):** A web app that enables employees to view order details and modify order status. It is a subsystem of our project.
- **Python**: An interpreted high-level programming language for general-purpose programming.
- **PUT:** A request method supported by HTTP used by the World Wide Web. By design, the PUT method requests that the enclosed entity be stored under the supplied URI.
- **Robo-waiter:** A system that receives orders from customers and allows employees to manage orders at a restaurant.
- **Server:** The process that waits to be contacted to begin a network communication session between a pair of processes.
- **Slot-filling System:** A method of requesting the information from the user and extracting them as entities by pertaining the list of already acquired entities and asking for missing ones.
- **Uniform Resource Identifier (URI):** A string of characters that unambiguously identifies a particular resource.
- **Utterance:** Anything the user says. For example, if a user types "show me yesterday's financial news", the entire sentence is the utterance.
- **VARCHAR:** A set of character data of indeterminate length.
- **Webhook:** A method of augmenting or altering the behavior of a web page, or web application, with custom callbacks. These callbacks may be maintained, modified, and managed by third-party users and developers who may not necessarily be affiliated with the originating website or application.

References:

- Wikipedia
- *Computer Networks - A Top Down Approach*, Kurose and Ross, 7th Edition, Pearson
- https://chatbotsmagazine.com/chatbot-vocabulary-10-chatbot-terms-you-need-to-know-3911b1ef31b4
- https://medium.com/activewizards-machine-learning-company/a-comparative-analysis-of-chatbots-apis-f9d240263e1d

Appendix: MCS Burger's Menu

Luxury Combos

COMBO #1
MCS COMBO $7.30
MCS burger, fries, and medium drink

COMBO #2
CHEESEBURGER COMBO $6.15
Cheeseburger, fries, and medium drink

COMBO #3
HAMBURGER COMBO $5.85
Hamburger, fries, and medium drink

Home-made Burgers

MCS BURGER $3.85
Toasted buns, 2 cheese slices, 2 beef patties, onions, lettuce, tomato, pickles, and spread

CHEESEBURGER $2.70
Toasted buns, 1 cheese slice, 1 beef patty, onions, lettuce, tomato, pickles, and spread

HAMBURGER $2.40
Toasted buns, 1 beef patty, onions, lettuce, tomato, pickles, and spread

Options for burgers:

- Anteater style
- Medium rare/well done
- Cut in half
- Extra toasted/lightly toasted/untoasted buns
- More/no cheese
- Cold/grilled cheese
- More/no meat
- More/no veggie
- More/no onion
- More/no lettuce
- More/no tomato
- More/no pickle
- More/no salt
- Add mustard/ketchup

Combo Drinks

Small	Medium	Large	Extra Large
$1.50	$1.60	$1.80	$2.00

COKE **SEVEN-UP**
ROOT BEER **LEMONADE**
DR PEPPER **ICED TEA**

Other Dishes/Drinks

FRIES $1.85
Made with fresh potatoes

Options for fries:

- Anteater style (+$1.80)
- Cheese (+$1.50)
- Well done/light well

SHAKES $2.45
Made with real ice cream

Options for shakes:

- Chocolate
- Strawberry
- Vanilla

MILK $0.99

HOT COCOA $1.60

COFFEE $1.25