Meal by Words

Nina Duan, Shiyi Zhang, Lisa Xiong

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of taking restaurant orders verbally. Inaccurate orders and long wait times are critical problems that discourage customers from ordering at quick-service restaurants. With the traditional ordering method, customers often need to increase their volume or even yell just so that the server hears their order correctly. Some restaurants provide touch screens as an alternative which has the potential of spreading harmful bacteria and diseases. To tackle these problems, we propose to create an ordering system that uses the speech recognition technology to place orders for our customers.**

***Index Terms*—Design, verification, sensor, embedded system, signal processing, speech recognition, tokenization, natural language processing, cloud database, user interface**

I. Introduction

THE understaffing issue of some fast food restaurants results in either exhaustive customer wait times, or burnt out employees shuffling between the kitchen and the counter. The current approach of having cashiers take orders also heavily relies on employee training, which consumes both time and money.

Some fast food restaurants installed touch screen ordering kiosks to solve these problems; however, after the pandemic, the public raised health concerns about unsanitized shared utilities to a new level. Fox News reported that a swab test on eight random McDonald's restaurants in the United Kingdom resulted in the discovery of infection-causing bacteria in every test. The existing ordering kiosks, with thousands of people making orders on them by touching, will carry lots of bacteria if not sanitized properly on a regular basis.

Kiosks that can take verbal orders will reduce the burden on fast food restaurant staff and the customers' health concerns. The goal of our project is to build a voice-operated ordering kiosk that processes orders efficiently. The entire process, not including paying, will be completed verbally. The confirmed order will be sent to the kitchen automatically, allowing the staff to start preparing as soon as an order has been placed. The ultimate goal is to create an ordering experience that mirrors the one a human employee offers, but with greater efficiency and accuracy.

II. Use-Case Requirements

Available operations: A regular fast-food order interaction includes requesting for desired items and proceeding to checkout. The system should support not only these two operations, but also two additional features – removing item(s) and changing quantities – to ensure that erroneously recognized and/or incomplete orders will not be sent to later stages of our system.

Service time: According to SeeLevel HX's 2016 research, the average service time in the U.S.-based fast-food restaurant is around 200 seconds. Our system should offer a similar if not better experience. Hence, we expect the entire ordering process for one customer to be completed in less than 200 seconds.

Latency: The kitchen staff should be able to see a newly placed order within 1 second after the customer checks out. This will ensure customers' food is prepared in a timely manner.

Voice reception: To accommodate fidgeting customers, the microphone should be able to receive inputs from a horizontal span of 120°, from a distance between 0m to 1m. The microphone and speech processing algorithms should accommodate all human voice frequencies (80Hz ~ 260Hz), at normal conversational volume (60dB ~ 80dB).

Noise tolerance: Since most fast-food restaurants have noisy environments, our system should be able to operate in a considerable level of background noise. To ensure a satisfactory experience, the system should reach 100% order accuracy by the time a customer checks out.

Power conservation: Our system aims to consume an appropriate amount of power to promote an environmentally-friendly ordering experience. Therefore, it should only process audio inputs when a customer is present. Otherwise, it should remain in a low-power mode that only monitors whether a person has approached.

Accessibility: Our system should accommodate children and those with disabilities. Factors to consider include the customer's height, how an approaching person is detected, and through what ways a customer can interact with our system. Specifically, the system should accommodate all customers with heights >= 0.7m and customers in wheelchairs.

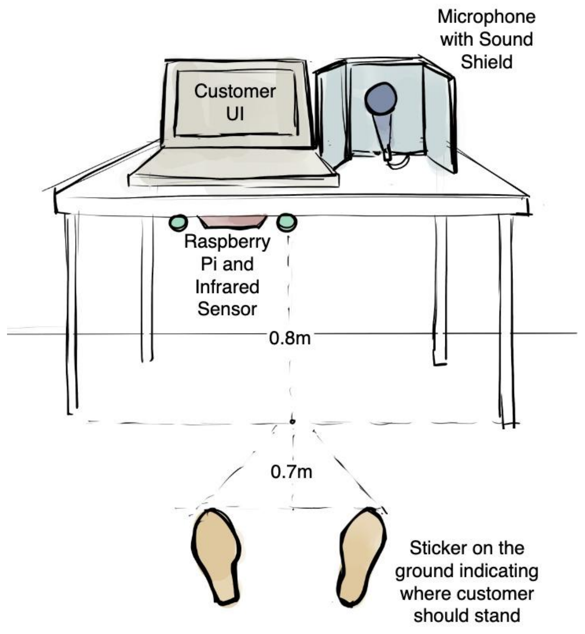
Process termination: If the customer leaves the kiosk, the system should be able to initiate a timeout. After the timeout, it should delete the current, incomplete order and enter the low-power mode. The system should not initiate a timeout unless it detects inactivity of more than 2 minutes, as deleting a customer's order by mistake would result in a frustrating user experience.

III. Architecture and/or Principle of Operation

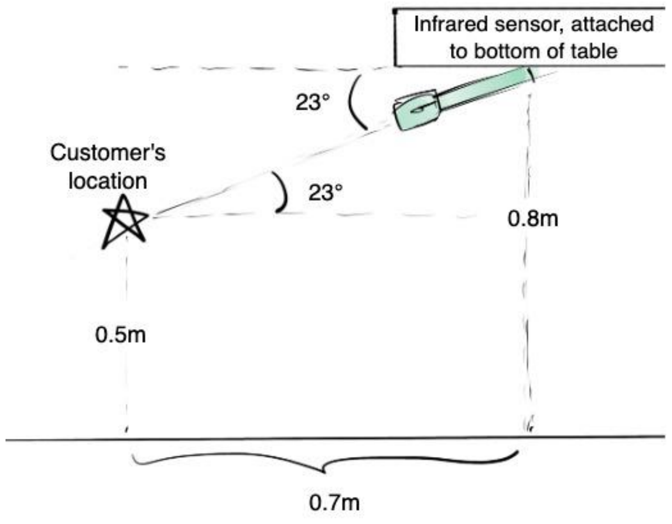
On a high level, our project can be divided into a hardware component and a software component. The hardware component manages the system's interaction with the physical world, receiving physical data such as infrared and sound waves. It provides the information our software component requires to satisfy our customers' needs. The software component receives the mentioned information, converts them into meaningful tokens, and stores them in a non-local location for safekeeping. The software component is also responsible for fetching and removing stored data in the future.

A customer's interaction with our system (Fig. 1) starts when they approach our kiosk. The footstep-shaped stickers on the ground guide them to stand approximately 0.7m away from the kiosk. This makes sure that they will be able to see the text on our customer-side UI while keeping a proper distance from the kiosk.

The infrared sensor, driven by a RPi 4 attached to the bottom of the kiosk table, is installed at an angle designed to pick up heat-bodies at a height of approximately 0.5m and a distance of approximately 0.7m. After the infrared sensor detects human presence, the system wakes up our backend speech recognition loop, which also runs on the same RPi.


Microphone with Sound Shield
Customer UI
Raspberry Pi and Infrared Sensor
0.8m
0.7m
Sticker on the ground indicating where customer should stand

(a)



(b)

Fig. 1. Kiosk drawing. (a) Overall kiosk setup. (b) Infrared sensor location and angle.

At the same time, the laptop screen will display our customer-side UI. Information this UI displays include the restaurant menu, a list of the customer's current order items and their respective prices, the order's total price, and helpful instructions to get the customer started. For a better user experience, the instructions will also be complemented with pre-prepared, synthesized voice lines.

A directional microphone, protected by a professional sound shield, is responsible for receiving the customer's voice input. This input stream is fed through a signal processing algorithm for noise reduction. The speech recognition module takes the sanitized signal and, using a trained machine learning model, converts it to a text string.

Our natural language processing algorithm parses the text string and extracts necessary order information, including the food item's name, the quantity of the item, and the customer's desired action (e.g. add, remove, change, etc.). Once the order information is processed, the customer-side UI will update to display the newly received item. If the customer voices a change and/or a removal, they will also be reflected on the UI in a timely manner. If the NLP module fails to recognize the item name, it will notify the UI of this failure. The customer-side UI will then display an error message and kindly ask the customer to repeat their request.

After the customer checks out, the entire order, currently stored as a local object, will be uploaded to the Redis cloud database. Information in this database is accessible to the staff-side UI. The database notifies the staff-side UI of the new order by publishing a notification message. The customer-side UI will also display the customer's unique order number, which they will use to pay for and pick up their order.

When notified, the staff-side UI, used by the restaurant's kitchen staff, queries the database for new order information. For ease of use, the UI sorts, in ascending order, existing orders based on the time they were placed (i.e. the oldest will rank first and the newest last).

The lifetime of an order ends when the kitchen staff removes it from the database. The staff-side UI provides a one-click functionality that allows the staff to do so after preparing the order.

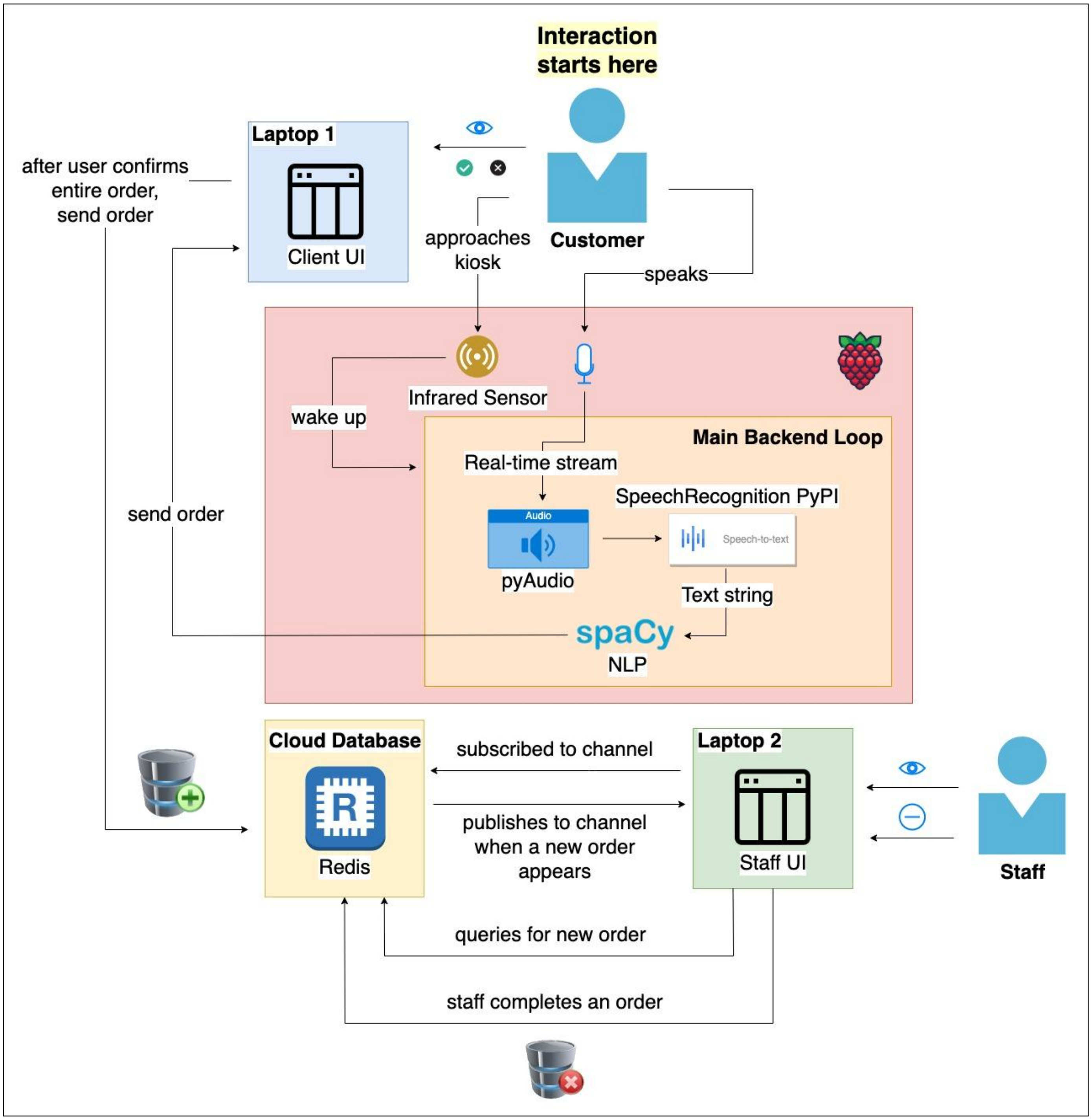


Fig. 2. System block diagram.

IV. Design Requirements

Customer detection: Rather than having a frustrated customer yelling at a screen that refuses to respond, we prefer that the system wake up more often than necessary. Therefore, we expect our system to detect every approaching person. When no one is around, a 70% detection accuracy (i.e. system stays asleep 70% of the time) is satisfactory.

Noise tolerance: In the article “Noise in Restaurants: Levels and Mathematical Model,” Dr. To states that a typical fast-food restaurant’s noise level falls in the range of 69.1dBA to 79.1dBA. Humans’ hearing accuracy drops significantly when the noise level is greater than 70dBA, measuring a mere 50% average in Raghavan’s paper. Therefore, we expect an 85% speech-to-text accuracy at a noise level < 70dBA and a 50% speed-to-text accuracy at a noise level >= 70dBA. This will ensure a relatively high success rate for matching customer speech and order items.

Data storage: As a general design rule, the representational gap between a physical order and its stored counterpart should be low. The stored information should be tolerant against unexpected events such as hardware failures, power outages, and network disruptions. The data should be accessible from multiple sources at the same time, as there may be multiple kitchen staff preparing orders at the same time. Following our latency requirement, the data storage model should also guarantee a <= 1s latency between order placement from customer-side UI and appearance in staff-side UI.

Customer-to-staff latency: To ensure that the kitchen staff sees a newly placed order within 1 second of the customer checking out, the staff-side UI should refresh as soon as the order appears in the data storage location. This means the staff-side UI either pulls for new data every second, or the data storage establishes a messaging system that notifies the staff-side UI of the change in real-time.

Backend-to-Customer Responsiveness: The backend must be able to recognize the end of the speech and deliver the speech to the speech recognition processor. After the speech is processed by the backend, it must immediately update the customer-side UI (within 1 second) so that the UI can confirm the receipt of the speech with the customer by displaying an icon, letting the user know that the speech has been received and please wait for further instructions. In addition, we will increase the customer-side UI rendering speed using Ajax to improve performance. This reduces traffic from and to the UI.

UI Readability: The customer-side UI should accurately display the customer's order items, their respective prices, and the order's total price. The texts should be clearly visible to people with 20/20 vision from at least 0.7m away. The staff-side UI should display orders in easily readable format. The staff should be able to read from the UI an item's name and quantity within 3 seconds. The UI should group items based on order rather than item type. The orders should be ranked based on order time (from oldest to newest). Orders that have been present for more than ten minutes will be emphasized by color-coding and enlarging fonts.

V. Design Trade Studies

Based on our use-case and design requirements, we made five key decisions.

A. Customer Detection Trade-Offs

By having an infrared sensor, the speech recognition system can go to sleep when there are no customers present. This not only conserves energy but also ensures that our system will not accidentally interpret background noises as meaningful information and act on them.

Passive infrared sensors (PIR) vs. active infrared sensors (AIR): While AIR sensors are more versatile than PIR sensors, as AIR sensors emit their own infrared radiation and therefore can detect objects and motion even in complete darkness, food-ordering systems operated on kiosks do not require such capabilities. In fact, fast food restaurants are usually brightly lit environments, so there is no need for that level of sensitivity. Therefore, we will be using PIR sensors, which detects the presence of a customer by measuring the changes in infrared radiation in the environment. The change in infrared radiation will trigger the PIR sensor and cause the RPi to output a signal.

Horizontal vs. tilted at an angle: Another factor to consider is how the sensors are positioned, whether horizontally or at an angle. We need to ensure that our system accommodates customers of different heights, as well as those with disabilities such as those in wheelchairs. If the sensors were positioned horizontally, there is a higher chance that people with heights below average might not be detected. Installing the sensors beneath the table and tilting them at an angle would increase the likelihood of detection, as the lower half of the customer's body is most likely to be in the detection zone.

B. Speech Recognition & Parsing Trade-Offs

To optimize the customer's experience, our system processes voice inputs as they are received. Ideally, the speech recognition system will be able to take in the customer's speech as audio and convert the information to a dictionary of menu items and quantities, which is then sent to the database. This feature will rely heavily on the efficiency and accuracy of our signal processing, speech recognition, and natural language processing (NLP) modules.

The Python programming language is compatible with a lot of external libraries containing speech-processing features which will come useful in our project. Our preferred tool for speech recognition is the SpeechRecognition library. It is a free, open-source library with ample documentation. Since the source code is available to the public, we can also add custom features on top of those the library provides. In addition, SpeechRecognition supports a wide selection of speech recognition engines such as CMU Sphinx, Google Speech Recognition, Microsoft Azure Speech, and Houndify API. This creates a safety net for when a particular engine fails.

After SpeechRecognition converts the audio input to text, our NLP system, implemented with spaCy, parses it into menu items and quantities. SpaCy, compared to Stanza and other NLP libraries compatible with Python, is better documented while maintaining a similar accuracy in token recognition and matching.

C. Frontend Support Trade-Offs

An open-source frontend library or framework will make our UI visually consistent and appealing. We debated between the React library and the Bootstrap framework but, ultimately, chose the latter. Bootstrap not only provides a library of off-the-shelf components and styles, but also various other supports that help boot up a standard project quickly. It allows rapid prototyping of a web application without the design work React requires. Our UIs don't have complicated functionalities that need heavy customization, so Bootstrap is an effective and efficient solution.

D. UI Design Trade-Offs

Digital menu vs. printed menu: We can either display the menu on a user interface or print and hang it on the wall above the kiosk. Printed menus can span a space larger than our UI screen (i.e. laptop screen), which is easier to read for customers. However, the advantages of digital menus outweigh this shortcoming. When the restaurant wants to add a new item or change item prices, digital menus only need minimal changes, while paper menus require reprinting. Digital menus are also more environmentally friendly, since they reduce paper waste and promote sustainability.

Visual vs. verbal instructions: To improve our system's accessibility, we will both display the order instructions on screen and use a voice synthesizer to play it verbally. Verbal instructions are more natural to our customers because they are what a human employee would give in at a regular kiosk. However, without a text display, people with hearing impairment may struggle to understand the instructions or navigate the UI, which leads to difficulty in making selection and extended service time.

On a side note, the staff-side UI is not among our top priorities in this project. Therefore, we will delay the exact design until we have made sufficient progress on other subsystems. For our MVP, the staff-side UI will remain a command line program that displays a list of orders, ranked based on placement time. Eventually, it will evolve into a Django web application retrieving information from our database.

E. Data Storage Trade-Offs

We mainly studied two options for customer data storage: local and cloud storage.

TABLE I. DATA STORAGE TRADE-OFFS

Criteria	**Storage Options**	
	Local Storage	***Cloud Storage***
Storage Formats	JSON file CSV file In-program global structures (e.g. map, set, dictionary, etc.)	NoSQL key-value pairs SQL tables (relational) Documents (JSON-like)
Difficulty	Low. Uses information team members have already learnt in other courses.	High. Requires extra ramp-up time.
Network	Doesn't require network access for storage. Requires network access for querying from a different machine.	Requires network access for storage and querying.
Storage Speed	Fast. Minimal latency.	Slow, but within sub-microsecond range.
Query Speed From Other Machines	Slow. Requires machines to send local data over the network.	Slow, but within sub-microsecond range.
Durability	Low. System failure results in data being inaccessible or permanently lost.	High. Cloud services provide fail-safe measures.

To reiterate, our main design requirements for data storage are 1) low representational gap between order and stored information, 2) high tolerance against hardware failures, 3) ability to access stored data from multiple machines, and 4) <= 1s latency between order placement from customer-side UI and appearance in staff-side UI.

Object-oriented programming is a good way to maintain a low representational gap between physical orders and the stored order information. Locally, JSON files, CSV files, and data structures such as dictionaries and sets can all store custom objects. On the cloud, this requirement disqualifies key-value storage, as the keys and values are generally both string or integer values. While the SQL table format supports mapping between tables (e.g. an entry in the Order table can contain a field that maps to an entry in the Item table), it establishes undesirably high coupling between them. Relational databases also maintain complicated internal data about the stored objects, which inherently makes insertions and deletions slower. In the end, the document format emerges as the most ideal cloud storage solution.

A major shortcoming of local storage is the vulnerability to hardware failures. If the hardware (e.g. RPi, laptop, etc.) that stores our order information drops offline, the data will be inaccessible until the system goes back on line. If a file becomes corrupted during reboot, we run the risk of losing the data entirely. By using cloud storage, we mitigate this risk by creating a backup copy in a remote location.

Cloud storage also inherently provides the ability to access data from other locations. It decouples our two UIs, the customer-side and the staff-side UIs. Both UIs will send and request information from the database. Local storage, on the other hand, requires our system to send new order information from the customer-side UI to the staff-side as we receive them.

Given the benefits of cloud storage, we believe that it is reasonable to pay the price of slightly higher access latency. The cloud database we have chosen to work with, Redis, generally guarantees a sub-microsecond range latency, which is significantly lower than the one second latency we are aiming for.

Redis is an open-source cloud database that 1) supports both the key-value and the document storage models, 2) allows customizations on data structures, and 3) provides a notification system useful for communication between our customer-side and staff-side UIs. The third functionality is particularly useful, since this eliminates the need of busy-polling the database for new order information.

VI. SYSTEM IMPLEMENTATION

The main processor behind our system is a RPi 4. It is responsible for driving the infrared sensor and the microphone, running the human detection algorithm, and parsing the voice inputs.

Each of the following subsystems will be tested individually before the final integration. Therefore, we will not discuss unit tests in the following Testing, Verification and Validation section.

A. Customer Detection

To detect the presence of a customer, two key hardware components are needed: a RPi 4 and a PIR sensor. The setup process involves connecting the VCC pin of the PIR sensor to a 5V pin (red wire), connecting the OUTPUT pin of the PIR sensor to pin 23 (yellow wire), and finally connecting the GND of the PIR sensor to a GPIO GND pin (black cable). When the sensor detects movement, the RPi 4 will receive a signal on GPIO and GPIO.input(PIR_PIN) == True. In the python script for the sensor, an endpoint should be set up through the HTTP library to communicate with the endpoint set up in the python script for the Django backend. The signal will then be transmitted from the sensor endpoint to the backend endpoint through an HTTP POST request. The timeout logic replies on whether a signal was recently received. Therefore, the backend will have a loop that constantly validates the timestamp of the most recently received sensor signal.

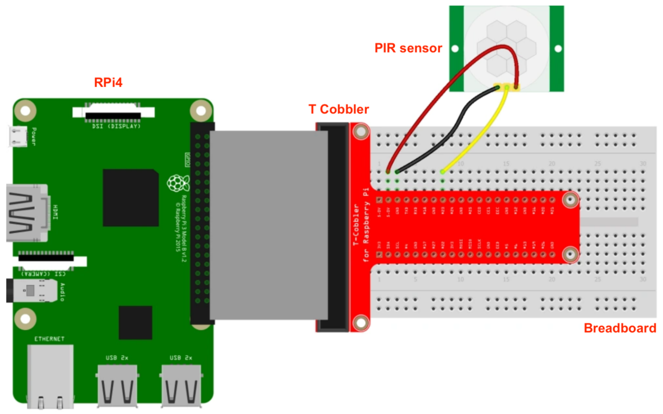


Fig. 3. PIR sensor, and how it connects to the RPi.

B. *Speech Recognition & Parsing*

Our system uses a USB directional microphone, Neat Bumblebee II, to capture voice inputs. To produce the best result, a professional sound shield, Moukey Microphone Isolation Shield, surrounds the microphone and limits its reception range to <= 90° vertically and <= 120° horizontally.

Input from the microphone is fed into a signal processing module, implemented with the open-source library PyAudio. The listen() function will capture signals from the microphone. This module uses Active Noise Reduction (ANR) to reduce background noise in real-time and relays the resulting stream to the speech recognition module.

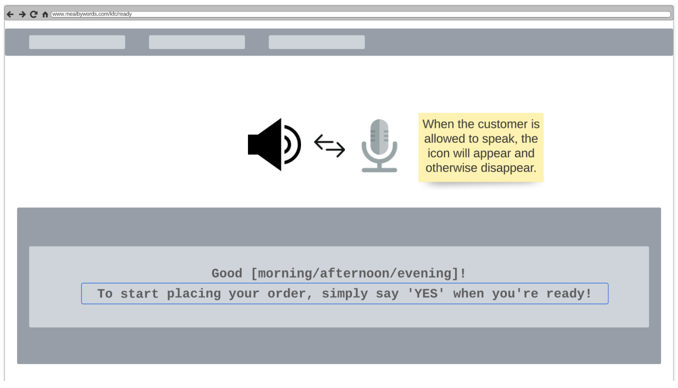
The speech recognition system is implemented with the SpeechRecognition Python library. It calls upon an existing, trained speech recognition engine and converts the input to text. A while loop makes sure that our system continues to take user input when the sound level is above a certain decibel.

The NLP system is implemented with the spaCy library. The NLP pipeline first numerizes the input text string (from the speech recognition module), converting any numeral words to numbers. Then, it will tokenize, add tags to, and identify named identities in the text. Lastly, we will establish grammar rules for the matchers to identify the menu items and their quantities.

C. *UI & Frontend Framework*

The UI regularly retrieves necessary variables from the backend and populates the reserved fields in the HTML file. To illustrate, when the speech recognition program finishes processing the ordered items, it should encode them into JSON format and send the data to the Javascript body in the order summary HTML file to be decoded. Javascript will fill the HTML fields with the decoded variables. Furthermore, transitioning from one page to another typically necessitates approval from the backend. The UI has no authority over it.

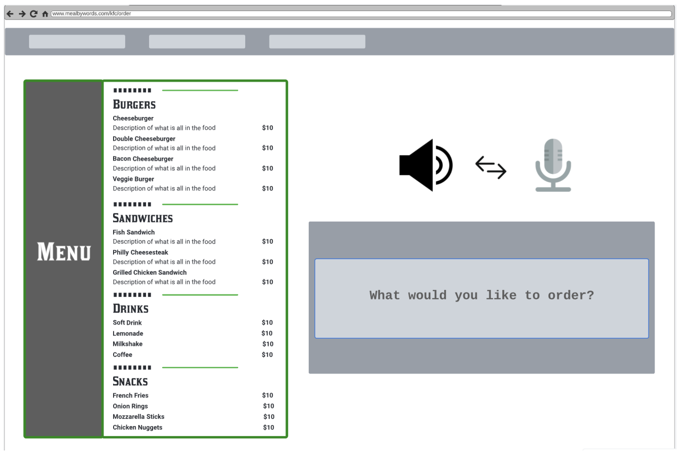
Customer-side UI: As the mockup (Fig. 4) shows, a microphone icon indicates that the customer should speak into the microphone. It will be replaced with a speaker icon when the UI is providing feedback (e.g., indicating an error has occurred when interpreting the customer's request). In the meantime, a pre-synthesized voice recording of the instruction will be played. A textbox will display the same message to accommodate those with hearing disabilities.


www.mealbywords.com/kfc/ready

When the customer is allowed to speak, the icon will appear and otherwise disappear.

Good [morning/afternoon/evening]!
To start placing your order, simply say 'YES' when you're ready!

(a)


www.mealbywords.com/kfc/order

MENU

BURGERS
Cheeseburger
Description of what is all in the food $10
Double Cheeseburger
Description of what is all in the food $10
Bacon Cheeseburger
Description of what is all in the food $10
Veggie Burger
Description of what is all in the food $10

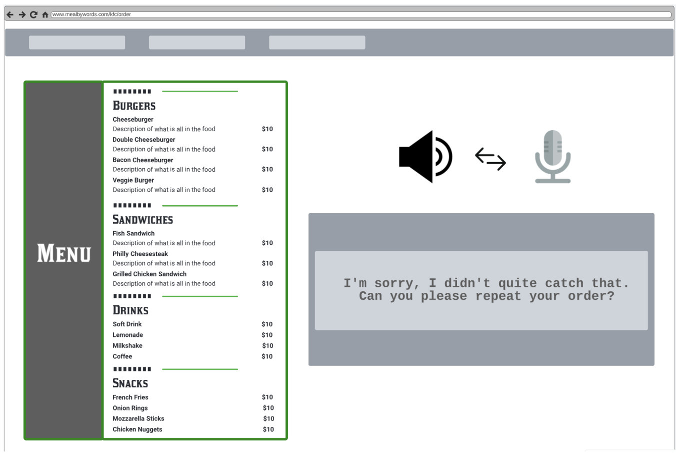
SANDWICHES
Fish Sandwich
Description of what is all in the food $10
Philly Cheesesteak
Description of what is all in the food $10
Grilled Chicken Sandwich
Description of what is all in the food $10

DRINKS
Soft Drink $10
Lemonade $10
Milkshake $10
Coffee $10

SNACKS
French Fries $10
Onion Rings $10
Mozzarella Sticks $10
Chicken Nuggets $10

What would you like to order?

(b)


www.mealbywords.com/kfc/order

Menu

Burgers

Cheeseburger	
Description of what is all in the food	$10
Double Cheeseburger	
Description of what is all in the food	$10
Bacon Cheeseburger	
Description of what is all in the food	$10
Veggie Burger	
Description of what is all in the food	$10

Sandwiches

Fish Sandwich	
Description of what is all in the food	$10
Philly Cheesesteak	
Description of what is all in the food	$10
Grilled Chicken Sandwich	
Description of what is all in the food	$10

Drinks

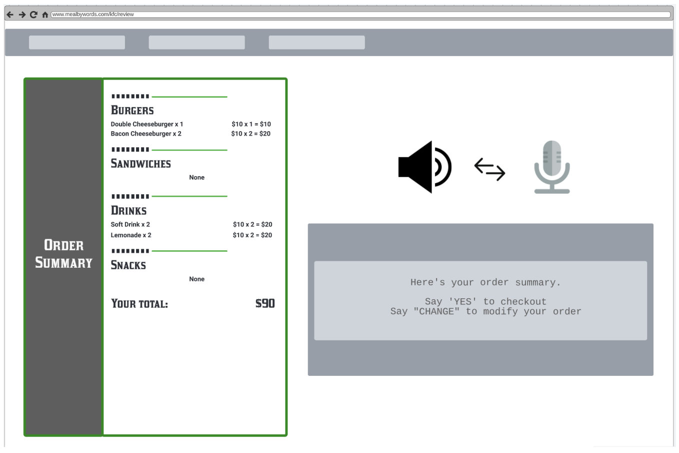
Soft Drink	$10
Lemonade	$10
Milkshake	$10
Coffee	$10

Snacks

French Fries	$10
Onion Rings	$10
Mozzarella Sticks	$10
Chicken Nuggets	$10

```
I'm sorry, I didn't quite catch that.
Can you please repeat your order?
```

(c)


www.mealbywords.com/kfc/review

Order Summary

Burgers

Double Cheeseburger x 1	$10 x 1 = $10
Bacon Cheeseburger x 2	$10 x 2 = $20

Sandwiches

None

Drinks

Soft Drink x 2	$10 x 2 = $20
Lemonade x 2	$10 x 2 = $20

Snacks

None

Your total: $90

```
Here's your order summary.

Say 'YES' to checkout
Say "CHANGE" to modify your order
```

(d)

Fig. 4. Customer-side UI mockup. (a) Welcome screen. (b) Display order instructions. (c) Ask the customer to repeat. (d) After checkout.

D. *Data Storage - Cloud Database*

Our system communicates with the open-source database Redis to store and retrieve customer order information. The Redis Object-Mapping (redis-om) module, an open-source library, maps object-oriented data models in Python to those on the Redis Stack. Redis also provides a publish-subscribe (Pub/Sub) functionality that allows the publisher to send messages to one or more subscribers.

The menu is declared as an immutable dictionary (MappingProxyType) and stored locally. A menu item's unique name acts as the key, and the corresponding value is the item's price. To ensure only menu items are used in order-related operations, an Enum type named MenuItem enumerates all the available item names.

Although Redis offers fast and reliable accesses, frequently retrieving data from the cloud will still stall the pipeline. In order to minimize communication with the database, order information will be kept in local storage before the customer checks out. The OrderLocal class (Fig. 4) is responsible for storing and managing information about an on-going order. Each order is uniquely identified by an orderNum. The time the order is created (system time, in seconds) will be documented as the orderTime, which will allow the staff-side UI to sort orders based on processed time. Menu items' names act as the keys to the OrderLocal class's items dictionary. Their corresponding values are the ordered quantity. The total price of the order will be updated as customers add or remove items. The price of an item can be found in the OrderLocal class's static variable, menu. Other than getters, the OrderLocal class provides three interfaces: addItem, removeItem, and checkout.

In the cloud, orders are stored in JSON format with models that are declared through redis-om. There are two models in total: Order and Item (Fig. 4). Item is an embedded model representing a menu item belonging to an Order. Order keeps track of its Items using a list attribute. In addition, Order also consists of attributes orderNum, orderTime, and totalPrice, which correspond to attributes with the same names in the OrderLocal class.

To upload a local order to the cloud, the downstream dependency only needs to call OrderLocal.checkout(). This method automatically parses the local order's contents and creates its cloud counterpart, Order.

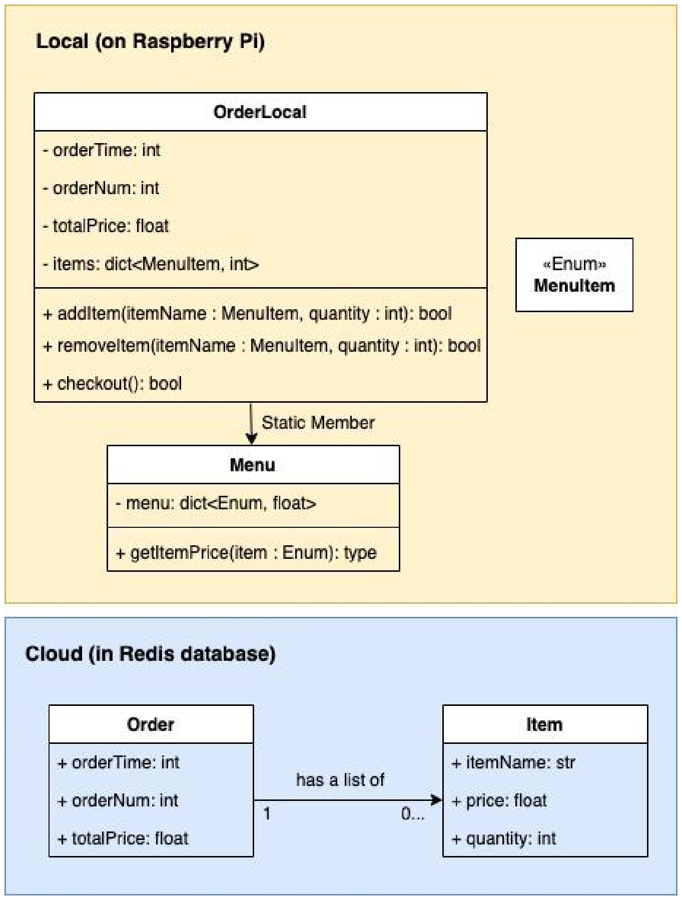


Fig. 5. The data model for representing a customer order.

Once the order has been stored to the database, the customer-side UI needs a way to notify the staff-side UI so that the staff-side UI can fetch the new order.

Redis's Pub/Sub functionality provides exactly this. The customer-side UI is set up as the publisher for the channel. When an order has been successfully added to the database, the customer-side UI will publish a message including the order's unique order number. The staff-side UI, subscribed to the same channel, can immediately query the database with the received order number and retrieve the new order information.

The staff-side UI's only other interaction with the database is order removal. A successful removal request completely erases the order information from the database.

VII. Test, Verification and Validation

Since our project is heavily customer-focused, we will invite 10 volunteers with varied gender, heights, accents to experience the ordering process. During testing, we will be keeping track of the following criteria:

A. Tests for Customer Detection

We will ask volunteers to approach the kiosk in various directions and linger for different amounts of time. The accuracy when a customer is present can be measured by the number of times the system wakes up over by the number of customer approaches we have tested. We expect the system to wake up 100% of the time.

Our volunteers will also try walking past the kiosk, standing far from it or tossing plushies at it to imitate the situation where no customer is approaching the kiosk but there are movements of humans and/or objects around it. The accuracy when a customer is not present can be calculated by the number of times the system wakes up divided by the number of actions the volunteers make.

B. Test for Service Time

The service time is measured from when the customer starts speaking to the time when they finish checking out. We will test how long the service time is for different volunteers with a varied number of order items (3, 5, 7, etc.). We hope to achieve an average service time of less than 200 seconds for a satisfactory user experience.

C. Tests for Noise Tolerance

To test the system's noise tolerance, we will generate background noise by two approaches. First, play recordings of white noises at the desired decibel level on loop while the volunteers are ordering. Second, take our system to a real restaurant setting, measure the noise level, and ask our volunteers to make orders there. Noise tolerance is essential for our system since the use setting is fast food restaurant, a relatively noisy environment.

D. Test for Latency

The latency is measured from the time when order is sent to the staff-side to when staff sees the actual order. We will record the system time, in milliseconds, that the order is sent and compare it with the system time, in milliseconds, that the order is received.

E. Tests for Process Termination

For checkout, we will require volunteers to use some common termination cues such as "that's it," "I'm done," and "finished." We will also ask the volunteers to check out with an empty or a filled order. The system should terminate the process but ignore the empty order (i.e. doesn't upload to the database). The system should successfully push a filled order to the database.

For timeouts, we will ask the volunteers to imitate different unexpected order terminations, such as approaching the kiosk but walking away after the system wakes up, walking away halfway through an order, and stopping to speak while still standing in front of the kiosk. In all cases, we expect the system to timeout after two minutes.

F. Test for Order Accuracy

We use recall as the measurement for order accuracy, that is, the number of correct item entries seen on the staff-side UI divided by the total number of entries the customer says. Our goal is to reach 100% order accuracy.

G. Test for Order Accuracy

Aside from the 10 volunteers testing the order processing, we will ask 10 more volunteers to observe and give feedback on both our customer-side and staff-side UI. They will rate, on a scale from 0 to 5, the customer-side and staff-side UIs for appeal and ease of reading.

VIII. Project Management

A. Schedule

We plan to complete our individual parts before the beginning of April. This will leave ample time for final integration and end-to-end testing. For a more detailed version, please refer to the Gantt Chart (Fig. 4, page 10).

B. Team Member Responsibilities

Tasks for team members are distributed based on each member's main focus. Nina is responsible for the database, microphone, and their integration with the RPi microcontroller. Lisa manages the speech recognition and NLP modules and creates the staff-side UI. Shiyi programs the infrared sensor against the microcontroller and creates the customer-side UI. Unfamiliar tasks or those that need more hands on (e.g. sound shield installation and end-to-end testing) will be group efforts.

C. Bill of Materials and Budget

For cost breakdown, see Table II, page 10.

D. Risk Mitigation Plans

Noise reduction: No one on the team has previous experience with sophisticated signal processing, and the noise tolerance requirement for the project is considerably strict. As a fallback plan, we will purchase a stretchable microphone holder that places the microphone closer to the customer's mouth.

NLP: The NLP system may fall short in parsing the input text to return the correct menu items or quantities. There are two tricky edge cases. First, the item name and quantity do not appear in the same sentence (e.g. "I want some fries, um, let me think. A large one is fine.") This is difficult for the NLP system since the Dependency Matcher relies on the dependency parser, which can only detect relationships between words in the same sentence. Second, the user does not specify quantity for menu items clearly (e.g. "I would like two hamburgers and fountain drinks.") It will be unclear for the NLP system whether the quantity "two" is referring to the hamburgers, fountain drinks, or both. The first possible mitigation is to set the default quantity to one and ask the customer to confirm their order item by item before check out. The second is, when the quantity is absent in the current sentence, to look for the first numeral word in the next sentence and make it the default for that item quantity.

Cloud storage: Using an open-source cloud database runs the risk of experiencing extremely high access latency and/or issues with server connection. An unresponsive system results in poor customer experience, which violates our use-case requirements. If the performance of our cloud storage component falls below the required 1 second, we will fall back to using local storage and sending raw order data between the UIs.

IX. Related Work

There has been an abundance of speech recognition and natural language processing developments in other markets, and some researches and products are focusing on integrating these technologies into the food service industry.

A research conducted in Dr. Mahalingam College of Engineering and Technology in India proposed a speech ordering system that allows the customers to place orders through phone communications. The research and our project have the common goal of establishing a speech-controlled food ordering system, sending orders to the staff's end after orders are completed. The research uses Naïve Bayes classification to convert text input to different entry intents of menu item names and quantities, and proceed with the intent that has the highest probability.

Speechly also has an API for taking customer orders by speech. The system can transcribe both real-time speech input or uploaded audio files, and recurrent neural networks are utilized to process the text.

X. Summary

With our speech-ordering kiosk, the fast food restaurant staff can now shift their primary focus to food preparation, and not worry about serving the customers coming in to order at any moment. The customers will no longer be concerned about touching a kiosk with potentially harmful bacteria attached to it. Our greatest challenge is meeting the 100% order accuracy at checkout to make sure that the customers' orders are all correct before sending them to the kitchen. We plan to ask for confirmation from customers before completing the order or even in multiple stages of the order, yet we are also aware that too many confirmation requests will potentially decrease ordering speed and customer satisfaction. Our team will do our best to find a balance between order accuracy and confirmation frequency.

Glossary of Acronyms

AIR - Active Infrared Sensor
MVP - Minimum Viable Product
NLP - Natural Language Processing
PIR - Passive infrared sensors
RPi - Raspberry Pi
UI - User Interface

References

[1] A. M. Raghavan, N. Lipschitz, J. T. Breen, R. N. Sami, and G. D. Kohlberg, "Visual speech recognition: Improving speech perception in noise through artificial intelligence2020," Otolaryngology--head and neck surgery : official journal of American Academy of Otolaryngology-Head and Neck Surgery, 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32453650/.

[2] G. Myrianthous, "How to perform real-time speech recognition with python," Medium, 23-Nov-2021. [Online]. Available: https://towardsdatascience.com/real-time-speech-recognition-python-assemblyai-13d35eeed226.

[3] P. Djuric, "How to use Redis Pub/Sub in Your Python application," Medium, 29-Dec-2021. [Online]. Available: https://blog.devgenius.io/how-to-use-redis-pub-sub-in-your-python-application-b6d5e11fc8de.

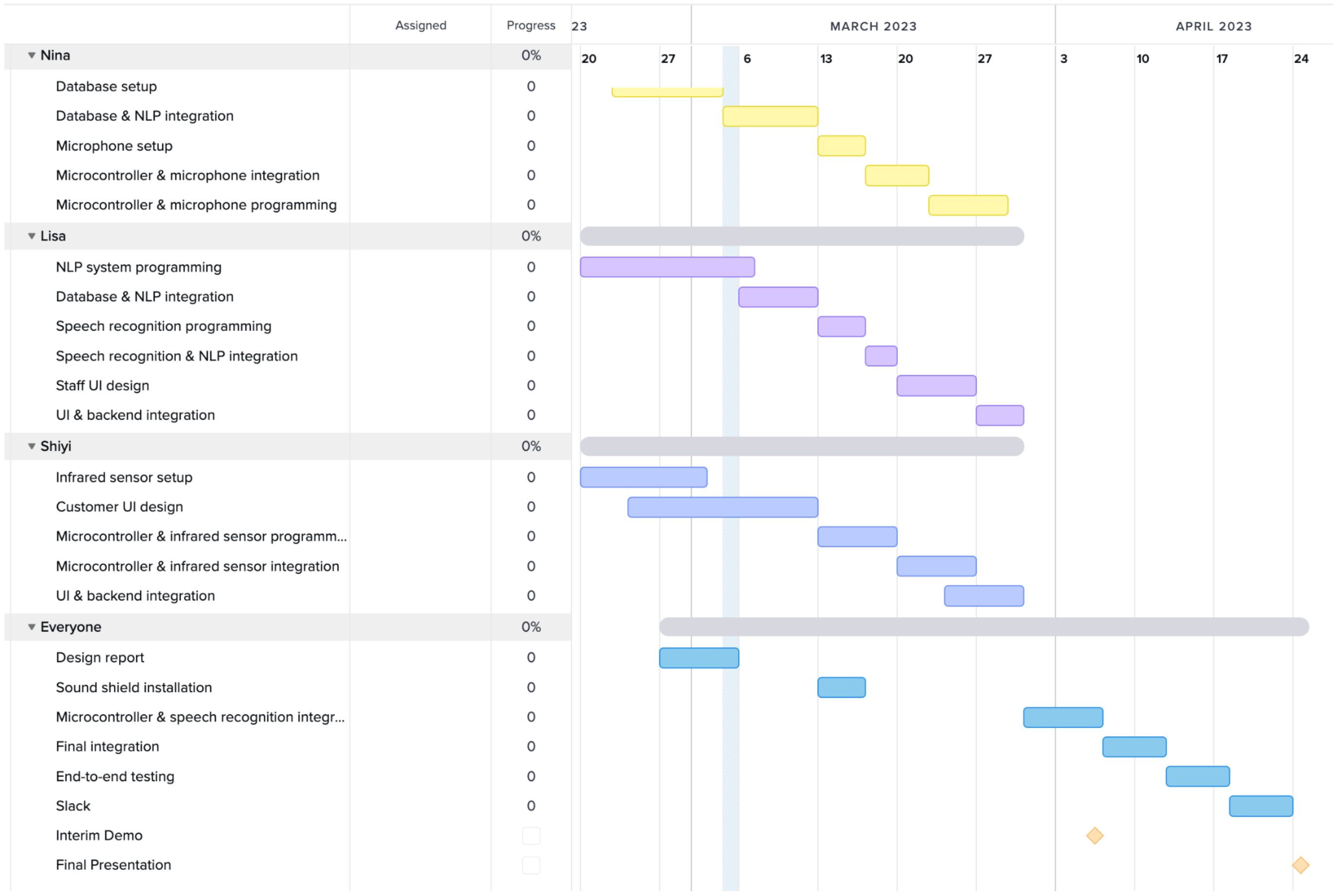
[4] "Speed of service," QSR magazine, 2016. [Online]. Available: https://www.qsrmagazine.com/content/speed-service.

[5] Sabarinathan, Swathi, V., & Pandi, D.M. "Smart Ordering System using Speech Recognition," International Journal of Creative Research Thoughts, 2-Feb-2020. [Online]. Available: https://www.semanticscholar.org/paper/Smart-Ordering-System-using-Speech-Recognition-Sabarinathan-Swathi/115e45fe61345a91d71ac4f7e8baa42e211f69f5#references.

[6] Speechly. 2023. [Online]. Available: https://www.speechly.com/solutions/quick-service-restaurants

[7] T. Sun, "McDonald's touchscreens test positive for traces of feces, deadly bacteria," Fox News, 28-Nov-2018. [Online]. Available: https://www.foxnews.com/food-drink/mcdonalds-touchscreens-test-positive-for-traces-of-feces-deadly-bacteria.

[8] W. M. To and A. Chung, "Noise in restaurants: Levels and Mathematical Model," Noise & health, 2014. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/25387532/.



	Assigned	Progress
▾ Nina		0%
Database setup		0
Database & NLP integration		0
Microphone setup		0
Microcontroller & microphone integration		0
Microcontroller & microphone programming		0
▾ Lisa		0%
NLP system programming		0
Database & NLP integration		0
Speech recognition programming		0
Speech recognition & NLP integration		0
Staff UI design		0
UI & backend integration		0
▾ Shiyi		0%
Infrared sensor setup		0
Customer UI design		0
Microcontroller & infrared sensor programm...		0
Microcontroller & infrared sensor integration		0
UI & backend integration		0
▾ Everyone		0%
Design report		0
Sound shield installation		0
Microcontroller & speech recognition integr...		0
Final integration		0
End-to-end testing		0
Slack		0
Interim Demo		
Final Presentation		

Fig. 6. Schedule with milestones and team responsibilities.

TABLE II. BILL OF MATERIALS

	Items				
Name	Microphone Isolation Shield	Cardioid Directional USB Condenser Microphone	PIR Motion Sensors	Raspberry Pi kit	Raspberry Pi 4
Model Number	L(29.9"x12.8")	24 bit/96 kHz Bumblebee II	HC-SR501	/	Model B 8GB RAM
Manufacturer	Moukey	Neat	Stemedu	SunFounder	Raspberry Pi
Unit Price	$49.99	$59.95	$9.99	$13.99	From ECE inventory
Quantity	1	1	1	1	1
Total (tax included)	$142.56				