Final Report
Food Delivery System

Computer Engineering Design Project (CEG4913)

University of Ottawa | Faculty of Engineering

Professor: Emil M. Petriu
Team Members: Amen Abshir (300035421)
Harouna Sylla (300086991)
Marc-Frédérick Tremblay (8603273)
Michelle Tang (300010861)
Santiago Cely (300038950)

Table of Contents:

1.0 Project Charter

1.1 Project Description

Food delivery automation is slowly being introduced in the food service industry where it will only grow on a larger scale as time progresses. The main idea is to introduce robots as a way of creating a more efficient environment for restaurants. The CEO of Yum Brands believes that Artificial intelligence could replace humans in the food services industry by the mid-2020s. The main purpose of this project is to develop a robot that will perform various tasks within a restaurant to improve efficiency, lower possible costs that arise from hiring staff members, provide a better experience for customers and decrease the possibilities of human error that result from different tasks within a restaurant. Integrating robots in a restaurant allows businesses to have a more stable scheduling process as predictive scheduling eliminates unaccounted-for complications that may arise from manual processes and human emergencies. The robot to be created will be able to navigate through a crowded restaurant and narrow spaces, decreasing the chance of human accidents. Restaurant automation also has the chance of carrying out tasks that would rather be time-consuming to create a more reliable environment all around. Restaurant automation in this project will also benefit restaurants by being more secure, as features will be added to ensure that any transaction involved in a credit card fraud will be immediately reported to the authorities.

1.2 Project Rationale

Following the pandemic, many restaurant owners are still struggling financially. Inefficiencies related to labour reduce considerably the profitability of such businesses. Integrating automation in the food service industry will improve operating efficiency by reducing cost and error rate which will in turn improve the overall customer satisfaction while yielding higher profit margins for the owners. Furthermore, it would reduce the circulation of servers in the dining room thus helping owners respect social distancing rules during this period of pandemic.

1.3 Expectations

The goal to be achieved within this project is the successful development of a food delivery robot to be integrated within restaurants performing various tasks while decreasing person-to-person interactions which results in a lower chance of contracting COVID and providing a safer environment. This project allows restaurants to utilize a newer method to improve efficiency, decrease some costs that owners face everyday while providing a unique and enjoyable experience for customers.

Being a part of an engineering program, we as students are always striving to learn and work with new technologies as well as acquire more experience in both the hardware and software side of engineering. This project allows us to strengthen our programming skills that were acquired in the past few years of our education along with familiarizing ourselves more with the fundamental hardware components and mechanical parts required to develop a functional robot that carries out the tasks it was set to perform.

Developing a robot is a concept that is new to the members of our team and is a great opportunity for us to learn and acquire new skills that can be applied later in our future careers as engineers. Robotics in the food industry is being utilized more as technology advances everyday. We will be using different programming skills that we were taught during our time in the program to create an android application that will be available to use by the customer and staff members of the restaurant. Our programming skills will also be beneficial while working on connecting the hardware aspects of this project to the "brain" of the robot, as well as finding quicker and more efficient solutions to any software obstacles we may face during the project. The concept of robotics is a way for us to learn how to not only work on the software knowledge obtained through the program, but apply the theories learned about computer architecture to practical applications to become more familiar with them for future references.

Due to the fact that robots are a large scale project and different sets of skills are necessary to successfully complete it, we as a team will be enhancing our communication skills as we are constantly required to consult with each other to ensure that we have a clear understanding of all the tasks that need to be completed, as well as provide help when further explanation is needed. Additionally, we will be enhancing our organizational skills through the scrum methodology we decided to employ as it allows us to have a guideline of the tasks to be completed at various dates. We hope to gain lots of experience and different skills through this project as well as create a great team dynamic to enjoy the process.

1.4 Clients and Participants

The food delivery robot to be implemented targets one client, restaurant owners, and one participant, customers of restaurants. As new advances in technology are seen everyday,

businesses such as restaurants are given the opportunity to incorporate AI technologies in order to increase efficiency, reduce labor costs while maximizing profits and decreasing human error that result from regular tasks. Statistics show that 74% of business owners confirm that automation saves and 58% believe that automation increases financial opportunities. With the implementation of a food delivery robot in a restaurant environment, owners are also able to decrease stress levels as automation allows them to work with a more predictable employment schedule while avoiding any complications that arise from human scheduling such as sick days, emergency situations and vacation days. It also frees servers from unstimulating and repetitive tasks, giving them more free time. This rise in productivity will stimulate the economy, offsetting the loss of jobs that is caused by automation by creating more engaging jobs. A food delivery robot also interacts with customers in several ways. The robot will bring food to the customer which will put many customers at ease during the pandemic we are facing as they will not have to interact with people who may be carrying COVID while unaware of it. Customers are also able to create an account and look at reviews from previous customers to get more of an understanding of what menu items they might enjoy.

2.0 System Requirements

The following functional and non-functional requirements are essential to the successful development of our food delivery robot along with meeting all of the project's expectations.

2.1 Functional Requirements

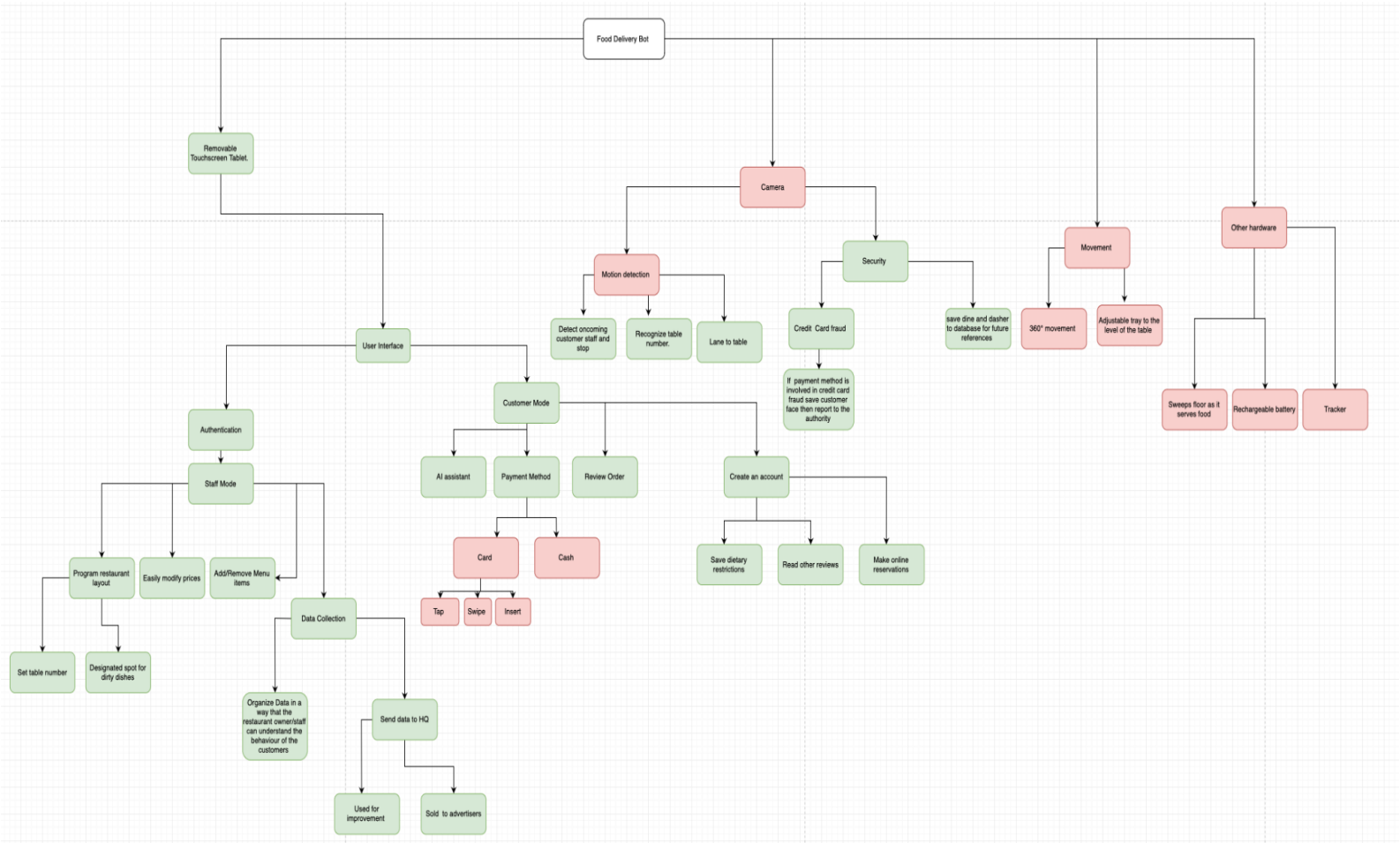

Figure1 : Diagram of functional requirement of project (some features might not be included in final model)

- Robot is able to detect oncoming staff and customers, and stop when necessary (no spillage from tray).

- Robot is able to recognize table numbers as well as bring/take food from table to kitchen and vice versa.
- Robot to be able to move with 360° motion. (left, right, forward, backwards).
- Robot must have a retractable tray that allows for an easy removal and serving of dishes.
- Robot to take photos of dine-and-dash customers while saving it for future reference.
- Robot to alert authorities if there is any indication that a credit card used in a transaction is involved in a credit card case.
- Robot must be able to accept different methods of payment from customers. (cash, coin or card).

2.2 Non-Functional Requirements

- Customers can easily create an account or continue as a guest while being able to rate menu items and add comments at the end of the dining experience.
- Staff mode will enable restaurant staff to easily modify menu items or prices as required.
- Staff mode allows restaurant staff to review data and comments by customers to further improve the customer experience in the future.

3.1 Software Design

3.1.1 Robot Operating System

The robot operating system or ROS is a free and open source software that defines the components, interfaces and tools for building advanced robots. This software helps us build motors, sensors and control systems and quickly integrate them using ROS tools such as topics and messages. This is primarily how each individual module communicates with each other.

3.1.2 Gazebo Simulation

To test our system we used Gazebo, a simulation software, to simulate the robot model and the environment it operates in. As our simulation environment we chose a cafe with moving actors, tables and a kitchen.

Figure 2 :The Gazebo simulation environment

3.1.3 Universal Robot Description Format

Universal Robot Description Format (URDF) is an xml file that describes the robot model and its sensors. Our robot is composed of a kobuki base, a vacuum cleaner like robot base, a hexagon tray containing a laser scan sensor, and a kinect depth RGB camera, and on top a load sensor that can detect when plates and drinks have been placed or removed on top of it.

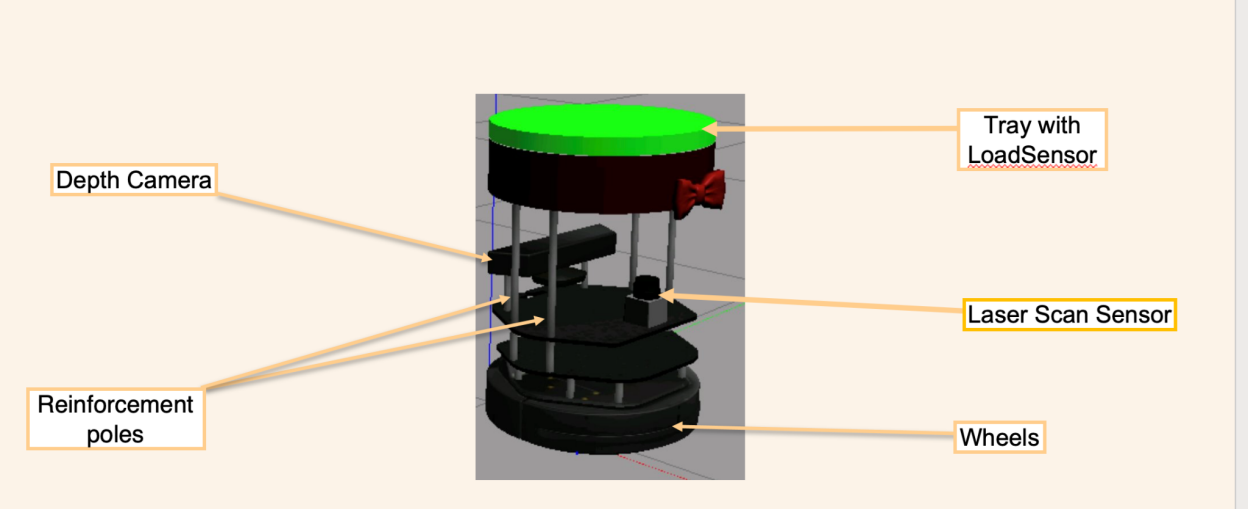

Figure 3: Robot model

3.1.4 Image Processing unit

The Image processing unit (IPU) is responsible for processing all the input from the robot's camera and analyzing it to detect objects such as tables, people, food plates … as well as to inform the high level controller of the current environment of the robot. The IPU has to detect when an object appears in the field of view of the robot and send a message to the high level controller using a ROS subscriber/publisher model. Additionally, The IPU needs to send a signal to the high level controller when an empty dish is identified, when a customer needs help, when a new customer is identified and when an object has fled the field of view of the robot. All these different features are impossible to implement without the use of a machine learning model that can efficiently label the images for us. Therefore we will be using a machine learning model to analyze our images and send it to our high level controller. We will use the YOLO convolution neural network architecture to classify these images and OpenCV to label the image.

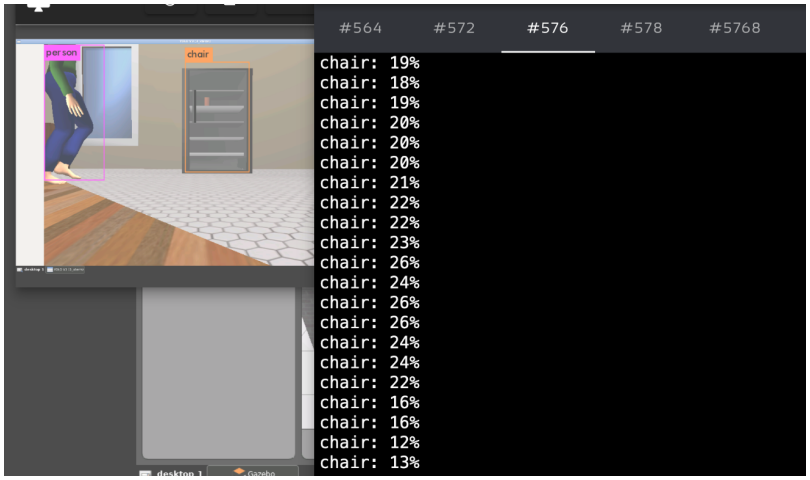

Figure 4: Object detection through the robot's camera

3.1.5 Localization and mapping

Before the robot can start taking orders, it has to map its environment and localize itself in this environment. To achieve this we use the gmapping package, a ROS wrapper for OpenSLAM's GMapping software, that takes as input raw laser scans and odometry data, and outputs a map with an occupancy grid. To localize the robot in our map we use the amcl package, a probabilistic localization system, that takes as input the laserscan data, the map and an initial position of the robot. It outputs an estimated pose of the robot. The localization and mapping is highly dependent on the robot's odometry, which always suffers from precision problems, because the wheels tend to slip and slide on the ground, the distance traveled is uneven compared to wheel slip. The error increases when the robot is on an uneven surface. As these errors accumulate and magnify over time, the odometer reading becomes less and less reliable. This means that the robot needs to be recalibrated regularly. [11][12]

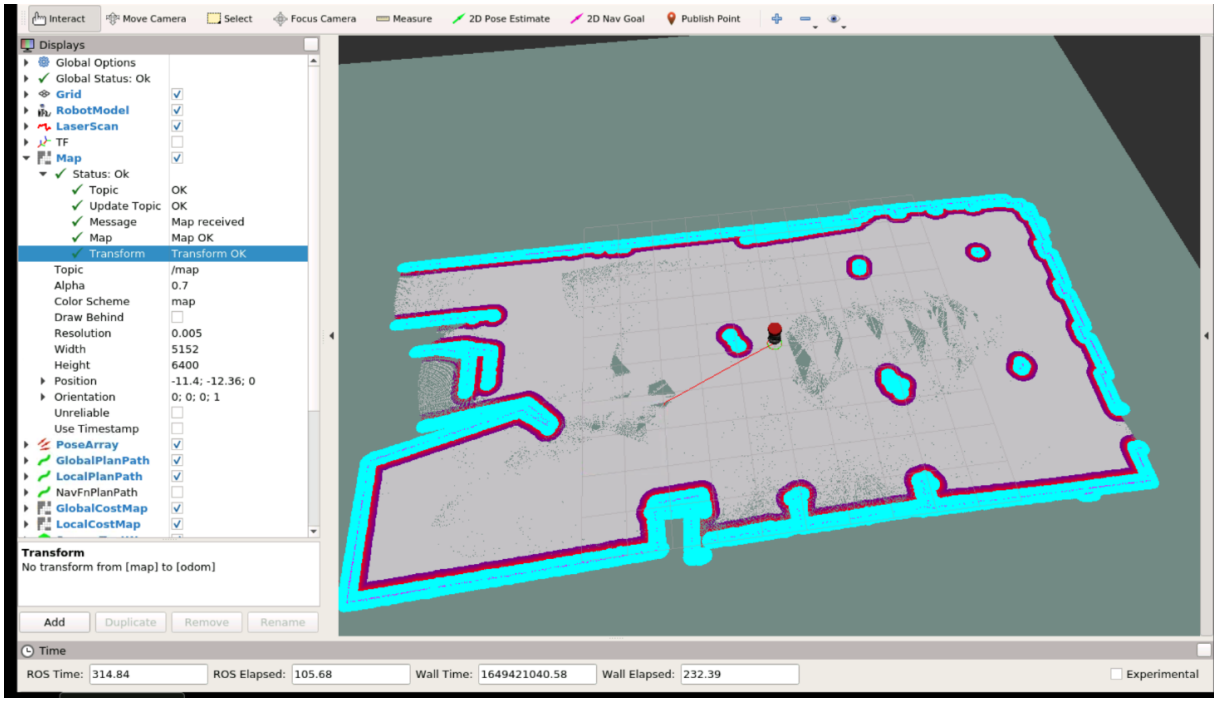

Figure 5: Map of the environment with area with high collision probability highlighted in blue and red

3.1.6 Path Planner

The robot has to move from point A to point B, may it be from kitchen to table, and vice versa, from charging dock to to the waiting area, from greeting customers to the tables … Therefore we have to plan the most optimal path and send it to the low level controller so that it can execute the specified movement. To achieve this we use ROS's move base package, which given a goal in the world it will reach it. It receives the goal in the form of a position + orientation. It also receives the odometry data, the map, the laserscan data, and the transforms (to locate the position of the sensors in relation to the base of the robot). The global planner charts the path to the final destination, the local planner charts the sequence of paths it has to take to get to the destination. The global costmap helps the robot avoid non moving obstacles in the

complete map, the local costmap helps the robot avoid moving obstacles in its immediate vicinity. The package outputs commands to the robot's wheel. [10]

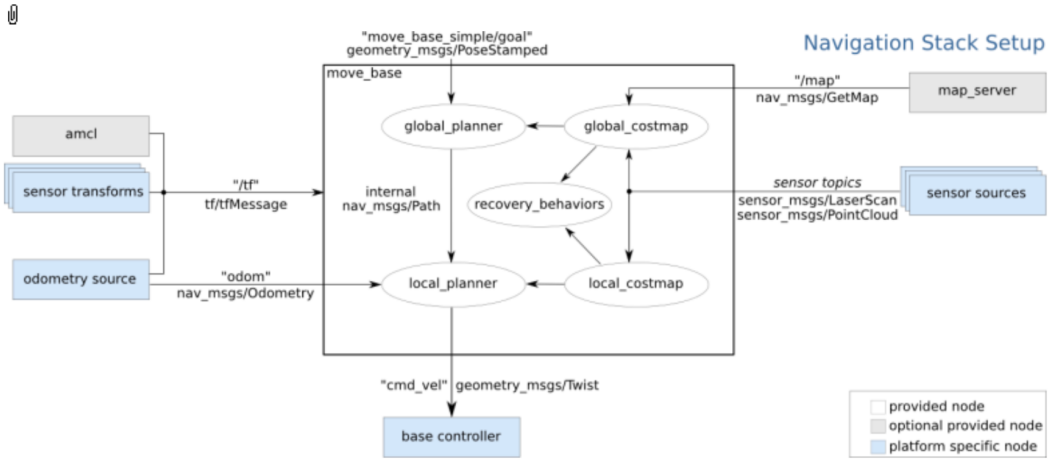

Figure 6: Diagram of the move base package

3.1.7 Natural language processing

The natural language processing (nlp) is the module that allows the robot to interact with customers and staff members. It's contextual based conversation. So the robot can understand the majority of the terms that are used in the restaurant area and provides an adequate response to his interlocutor.

The module is implemented using the **rasa** framework with built-in machine learning models. We first designed a chatbot which allows users to interact with the robot on a textual basis. This allowed us to use **rasa-x** to test the module. With **rasa-x** we were able to share the chatbot with random users through a web browser. This allowed us to strengthen the model of the system and to add other stories that we had not thought of beforehand.

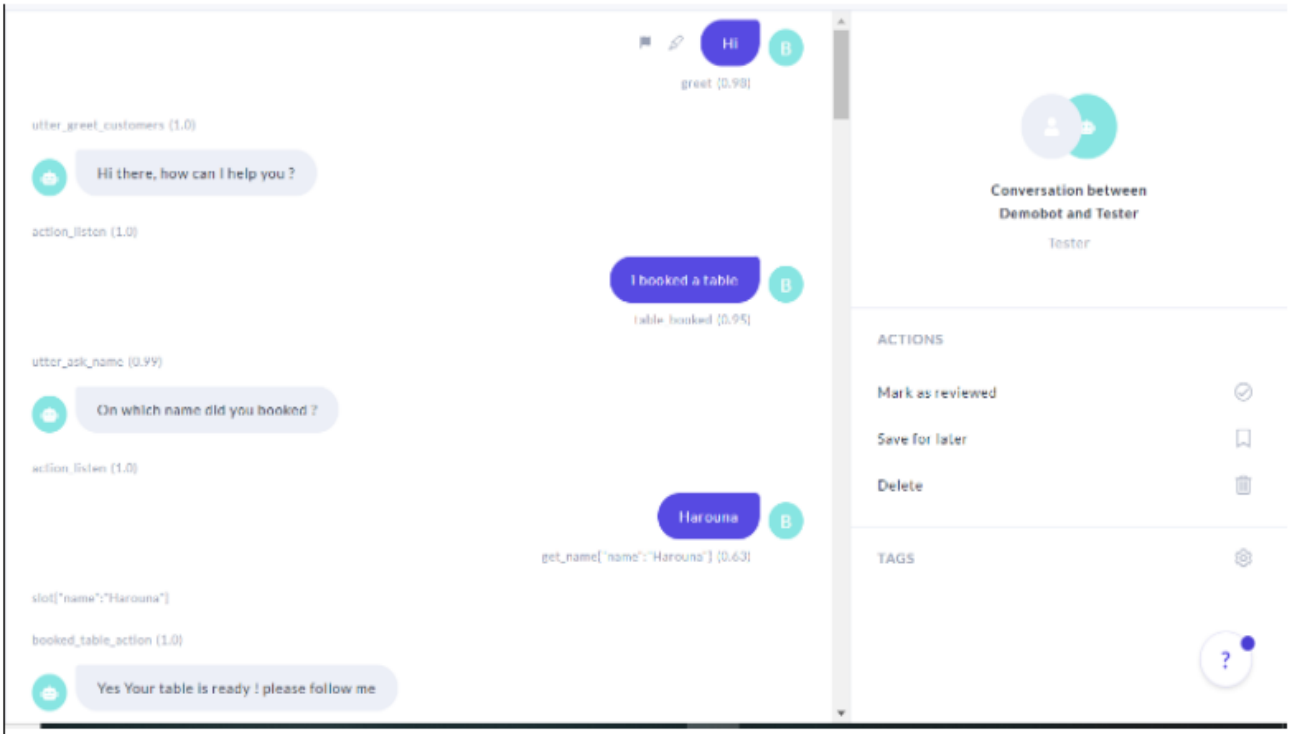

Figure 7: rasa-x interface: conversation between the chatbot and a user

After testing the chatbot with rasa-x and making improvements on the previous model we designed another module named **voice_bot** whose purpose is to convert the chatbot's output (text) into audio and play it to the user. Also it records the user's speech and converts it into text and then sends it to the chatbot. Thus the two modules together form our **vocal-based natural language processor.**

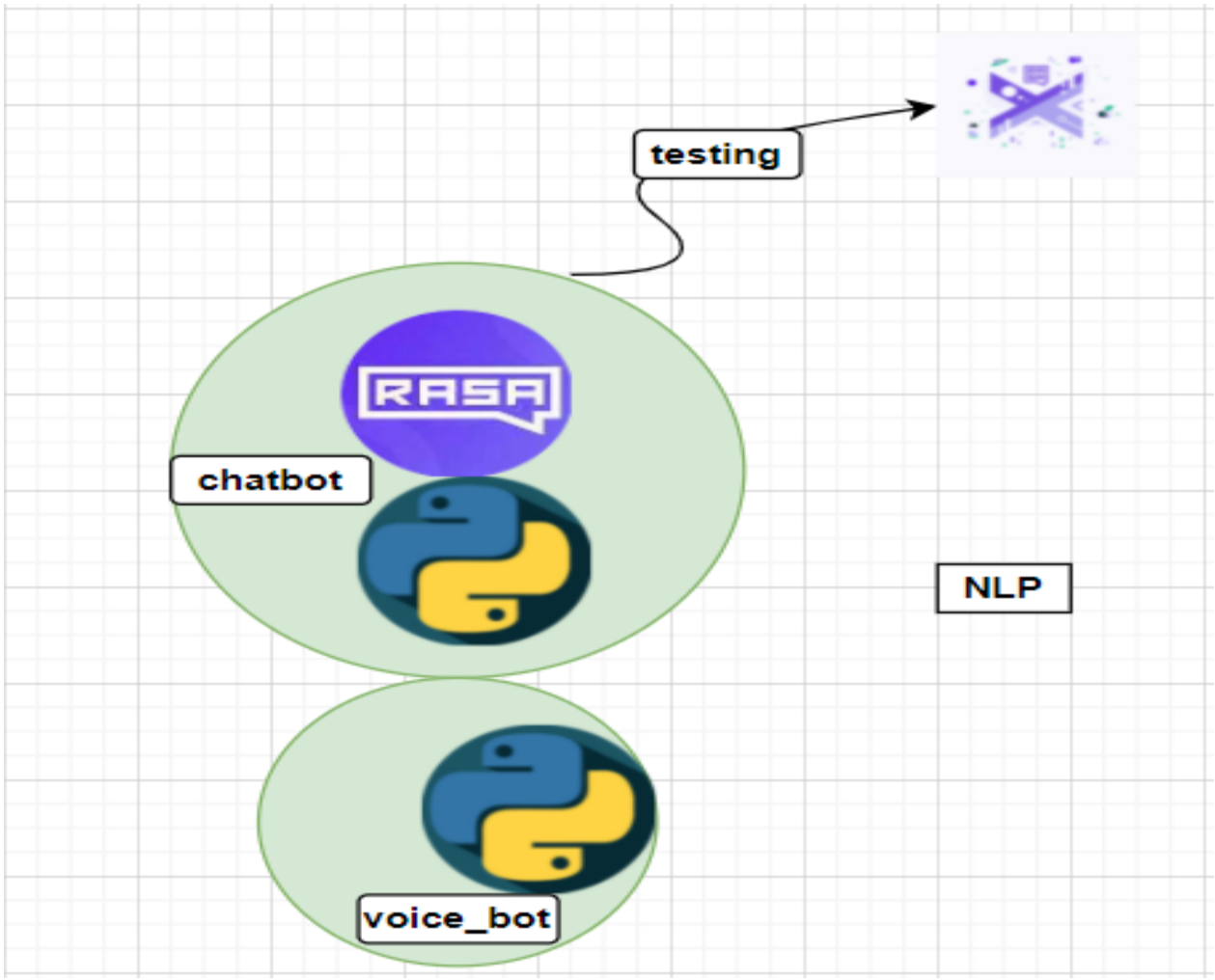

Figure 8: Nlp structure

The nlp module is integrated in the whole system through the **hlc** module which is connected to the database.

The nlp sends client requests or any other information that needs to be shared with otehrs modules to the **hlc** which in turn pushes data into the database. Then this data will be extracted by the appropriate module

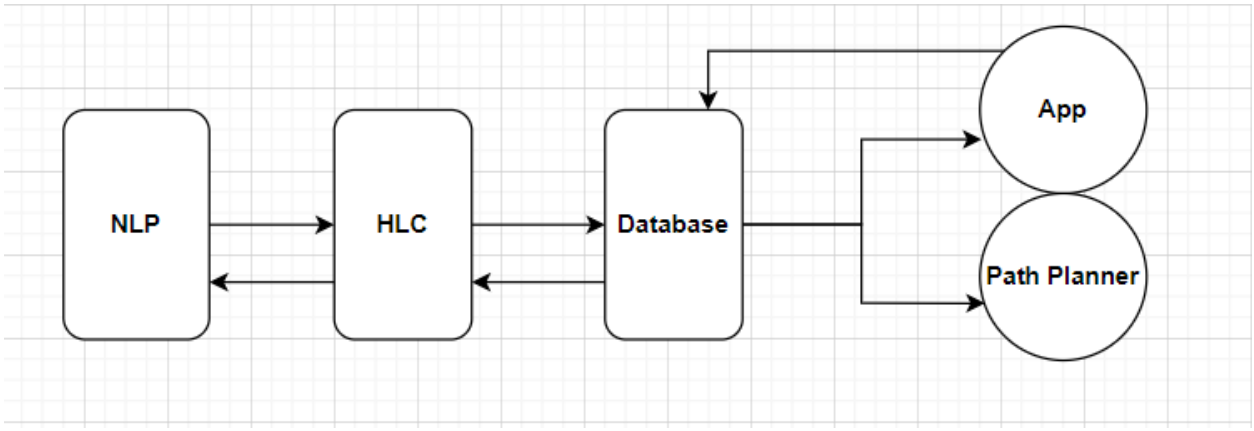

Figure 9: Nlp interaction with the other modules.

3.1.8 Software Application Front End

The interface of the frontend side of the project will be a restaurant-ordering e-commerce Android application. The application will use the Hybrid App Development, with the Native Android as the wrapper and Ionic's Capacitor as the frontend interface framework. This application will use the MVC framework model since the user will be interacting with the application to obtain menu items, order statuses, etc.

Ionic's Capacitor allows for a faster development as the application essentially becomes a web application that is coded in Javascript, HTML and CSS, that will be encapsulated by a native (Android) shell. Using this technology will greatly reduce development time and increase productivity.

The frontend will be connected to the backend, so the system can send and receive all the information required to facilitate a food-ordering experience (e.g., viewing menu items and categories, viewing the cart, etc). It will also allow restaurant staff to modify the menu and restaurant layout at will and to view analytics concerning the menu items and customers.

The frontend application will have two user interfaces: Customer-mode and Staff-mode. The Customer-mode is default; administrators must login to access Staff-mode.

Since the database does not store the datatypes as Objects, and returns them as Observables once queried, we will be defining Observable Object Interfaces for our datatypes. Services provide the modules with communication with other modules or the database. To help facilitate the development of the application, the application will be modularized. Each functionality of Customer mode and Staff mode will have its own module and thus page.

The following Object Interfaces are required:

- **CustomerAccount:** Where the information about customer accounts is saved. Used purely to increase customer experience by keeping track preferences, contact information and past orders, as well for analytics on administrator's side.
    - **customerId:** ID of customer
    - **name:** Name of customer
    - **email:** Email of customer account
    - **password:** Password of customer account
    - **preferences:** Dietary restrictions saved onto account (e.g., Nut-Free)
    - **pastOrders:** String array of IDs of past orders made by customer
    - **favourites:** Menu items favorited by customer

- **RestaurantLayout:**
    - **table:** Number that indicates the table number
    - **size:** Number that indicates the amount of people that can sit at the table
    - **available:** Boolean that indicates whether the table is available or not

- **MenuItem:**

- **itemId:** ID of the menu item
- **name:** Name of the menu item
- **category:** Dietary filters which the menu item belongs to (e.g., Vegan, Gluten-Free, etc)
- **price:** Price of the menu item
- **type:** Category which the menu item belongs to (eg Appetizer, Burgers, Fish)
- **image:** Path to the image for the corresponding menu item.

- **Order:**
  - **orderId:** ID of the order
  - **items:** String array of IDs of menu items ordered
  - **ready:** Boolean flag to indicate whether the order is ready or not.
  - **orderCompleted:** Boolean flag to indicate whether order has been delivered to the table or not.
  - **table:** Table number at which the customer sat.
  - **total:** Total amount due for the order.
  - **totalPaid:** Total amount paid by customer
  - **timePlaced:** Number in milliseconds of the time that the order was placed.

- **OrderToPay:**
  - **items:** String array of IDs of menu items ordered
  - **orders:** String array of IDs of orders associated to the table
  - **table:** Table number at which the customer sat.

        - **total:** Total amount due for all the orders.
        - **totalPaid:** Total amount paid by customer

    - **Order:**
        - **Request:** String indicating the request made by the customer
        - **Table:** Table number at which the customer sat.
        - **Acknowledged:** Boolean flag indicating whether the employee has seen the request or not.

The following Services will be created:

- **CRUD Service:** This is the service used by the customer module to create, read, update and delete documents in the database. It is called whenever access to the database is required. It is important to separate the service used by the customer and staff as later on, we will be implementing different access rules to the database depending on the user.
- **Events Service:** This is the service that will be used to communicate between modules. It will allow modules to subscribe to and emit content. This will be used by the **account** module to communicate the preferences added by the customer to the **browse-menu** menu as well as any items added in the **browse-menu** to the customer's order will be passed over to the **view-order** module.
- **Admin Service:** This is the service used by staff to create, update, read and delete documents in the database. It is called whenever access to the database is required.
- **Ionic-auth Service:** This is the service used by the customer and the employee to login to their accounts in a secure and efficient manner. The service uses Firestore authentication to

verify and retrieve credentials as well as user data from the database. Additionally, this service offers the customer the option to login with different partners such as Google, Microsoft, Twitter and GitHub.

The following modules will be created for the Customer interface:

- **browse-menu:** This module loads the menu from the database in live time and allows the customer to browse the menu, filter, and add items to their order. It must communicate with the view-order module to pass on items by using a Service to pass on the menu item ID. It has the following methods:

    - **constructor():** This initializes the services used by the module. The services include the **CRUD Service** and the **Events Service**.
    - **ngOnInit()**: This method is called upon initialization of the module when it is first loaded. It calls the **displayMenuItems()** function.

    - **displayMenuItems():** It will load all the menu items from the database and display them according to the "type" they are by calling the MenuItem Service.
    - **addToOrder(item):** This method is called when the Customer adds an item to the cart. It will communicate with the view-order module by using the **Events Service**.
    - **filterItems(category):** This method is called when the Customer filters the menu items by the preferences (eg Vegan, Halal, etc). It will reload the menu items

displayed by calling the MenuItem Service and query the database so only the items filtered will be displayed.

- **clearFilters():** This method removes the filters applied to the menu (if any)

- **customer-login:** This module allows Customers to login to view previous activity and edit profiles and can communicate with the browse-menu module. It has the following functions:

  - **constructor():** This initializes the services used by the module. The services include the **CRUD Service**, the **Events Service** and the **Ionic-auth Service** .
  - **ngOnInit():** Load the preferences set by the Customer into the **browse-menu filters()** function, so the Customer will have a pre-saved **browse-menu** experience. Additionally, it validates that the email entered corresponds to a domain and also that the password has at least 6 characters.
  - **signIn():** Uses ionic-auth service to authenticate the user. Redirects to the customer account if login successful or displays an error in the current screen if credentials are not found in the database.
  - **goToSingup()**: Redirects users to the registration page.
  - **updateAccount():** This will allow the user to update their contact information.
  - **viewPastOrders():** This will load a list of past orders saved on the Customer's account by calling the **CRUD Service**.
  - **logout():** Logs the user out.

- **googleLogin():** Uses Google login service to authenticate users in firebase. Redirects to the customer account if login is successful.
- **microsoftLogin():** Uses Microsoft login service to authenticate users in firebase. Redirects to the customer account if login is successful.
- **twitterLogin():** Uses Twitter login service to authenticate users in firebase. Redirects to the customer account if login is successful.
- **gitHubLogin():** Uses GitHub login service to authenticate users in firebase. Redirects to the customer account if login is successful.

- **pay:** This module allows Customers to pay for the orders they have placed. It has the following functions:

  - **constructor():** This initializes the services used by the module. The services include the **CRUD Service** and the **CART Service**.
  - **ngOnInit():** Uses the function **resetSettings()** to reset the settings of the previous user.
  - **setTableNumber():** Sets the table variable to the number specified by the user.
  - **resetSettings()**: Resets the settings of the previous user.
  - **displayBill():** This function uses the **CRUD Service** to display the bill for the chosen table. It gathers all orders associated with the table and then adds all the totals and the total paid by the customer. It displays all menu items in the orders and displays the amount that is left to pay.

- **payBill():** This function uses the **CRUD Service** to go through all the orders associated with the table and pay them accordingly to the amount that the user chose to pay.

- **registration:** This module allows customers to create an account using their email and password of their choice. It has the following methods:

  - **constructor():** This initializes the services used by the module. The services include the **Ionic-auth Service**.
  - **ngOnInit():** Validates that the email entered corresponds to a domain and also that the password has at least 6 characters.

  - **signUp():** Verifies the email entered does not correspond to another existing user and if it does not. It creates the account on the database and redirects the user to the customer account.
  - **goToLogin()**: Redirects users to the login page.
  - **setTableNumber():** Sets the table variable to the number specified by the user.

- **view-order:** This module communicates with the browse-menu module and takes the menu item IDs added to the cart. It then queries the database for the items with the matching IDs. It will display the items added to cart but not yet submitted on top, and the items that have been submitted (and thus, in progress).

- **constructor():** This initializes the services used by the module. The services include the **CRUD Service,** the **CART Service** and the **Events Service**.
- **ngOnInit():** This method is called upon initialization of the module. It will create a temporary Order Observable Object.
- **submitOrder():** This method uses the **CRUD Service** to create an order in the database.
- **displayLocalCart():** Creates a temporary list with all the items that the user submitted to the cart and displays it on the page
- **resetSettings():** Resets all settings in case the customer chose a wrong table or in case there is a new customer using the application.
- **pay():** This method navigates the user to the pay page.

The Staff interface will have the following modules:

- **admin-login:** This module allows Staff to login and access Staff mode. It has the following methods:

  - **signIn():** Uses ionic-auth service to authenticate the admin. Redirects to the admin page if login successful or displays an error in the current screen if credentials are not found in the database or if the credentials used do not have admin privileges.

- **OnInit():** Validates that the email entered corresponds to a domain and also that the password has at least 6 characters.

- **constructor():** This initializes the services used by the module. The services include the **Ionic-auth Service** and the FormBuilder service used to capture the input email and input password .

- **admin:** This module serves as a navigation page for the staff to redirect the user to different modules to access different functionalities. It will have the following methods:
  - **signOut():** This terminates the admin's session once the back button is pressed when on the admin navigation page or when the button 'logout' is pressed.
  - **ngOnInit():** Checks whether the current user logged in has admin privileges or not. Redirects to the browse-menu page if the user does not.
  - **ngOnDestroy():** Unsubscribe from elements that are not needed outside of this scope.
  - **viewUserRequests():** This function uses the **CRUD Service** to enable an observable on the 'Nlp' collection of the firestore database and then it displays the user requests generated by our natural language processing module integrated in the robot.
  - **openReqs():** Shows and hides the user requests

- **acknowledgeResquest(id):** Uses the **CRUD Service** to delete the user request from the database. Hence, marking it as acknowledged.
- **goToCurrentOrders():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the view-current-orders page
- **goToEditMenu():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the edit-menu page.
- **goToEditLayout():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the edit-layout page.
- **goToViewAnalytics():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the view-analytics page.

- **view-current-orders:** This module allows Staff to view current orders at each table and send out the menu items once they have been completed. It utilizes the **CRUD Service**. It has the following methods:
  - **constructor():** This initializes the services used by the module. The services include the **CRUD Service.**
  - **viewOrder(String orderID)**: This allows the user to view the current order at the specified table by using the CRUD Service to retrieve and display the order.
  - **sendItem(String itemID, String orderID):** This allows the user to send out the order to the customer.

    - **ngOnInit():** Subscribe to elements that are needed inside of this scope and verify the status of the current user logged in. If it has admin privileges, stay on page. Otherwise, redirect to the browse-menu page.
    - **ngOnDestroy():** Unsubscribe from elements that are not needed outside of this scope.

- **edit-menu:** This module allows Staff to modify and add menu items. It utilizes the CRUD Service. It has the following methods:
    - **constructor():** This initializes the services used by the module. The services include the **CRUD Service** and the **Ionic-auth Service.**
    - **goHome():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the admin page.
    - **ngOnDestroy():** Unsubscribe from elements that are not needed outside of this scope.
    - **ngOnInit():** Subscribe to elements that are needed inside of this scope and verify the status of the current user logged in. If it has admin privileges, stay on page. Otherwise, redirect to the browse-menu page.

- **edit-layout:** This module allows Staff to change the status of the different tables in the restaurant. It utilizes the CRUD Service. It contains the following methods:
    - **constructor():** This initializes the services used by the module. The services include the **CRUD Service** and the **Ionic-auth Service.**

    - **goHome():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the admin page.
    - **viewLayout():** It uses the **CRUD Service** to subscribe to the collection RestaurantLayout on the firestore database and retrieves all the tables with their corresponding size, availability and table number.
    - **changeAvailability():** This method uses the CRUD Service to update the status of the table on the database. It changes between available and unavailable.
    - **ngOnDestroy():** Unsubscribe from elements that are not needed outside of this scope.
    - **ngOnInit():** Subscribe to elements that are needed inside of this scope and verify the status of the current user logged in. If it has admin privileges, stay on page. Otherwise, redirect to the browse-menu page.

- **view-analytics:** This module allows Staff to view analytics about the restaurant and orders. It will have the following methods:
    - **viewData(String data):** This method takes the data that the user wishes to see (eg most popular menu items) and queries Firebase Analytics to display the data.
    - **goHome():** Verifies the status of the current user logged in and if it has admin privileges, it redirects the user to the admin page.
    - **ngOnInit():** Verify the status of the current user logged in. If it has admin privileges, stay on page. Otherwise, redirect to the browse-menu page.

The UI design of the Customer interface and the Staff interface were designed with user needs in mind and the principles of UI design. They differ in that the Staff interface will have bigger and more accessible buttons as we can safely assume that Staff will need more “direct” access and do not have as much time as a Customer. The Customer on the other hand, can spend much more time browsing and exploring the app. We plan on using familiar widgets such as buttons and modals to increase user adaptability and usability.

The UI design of the Customer interface is shown in the following Figures below:

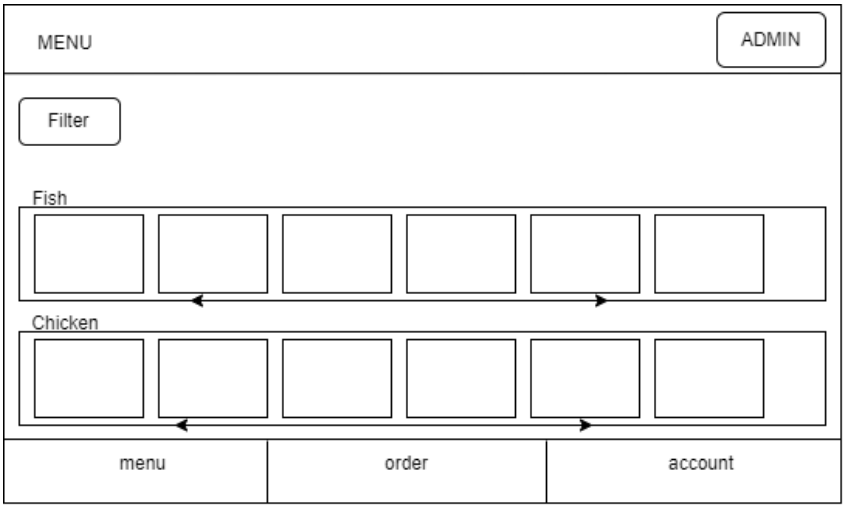

Figure 10: UI Design for Browsing Menu

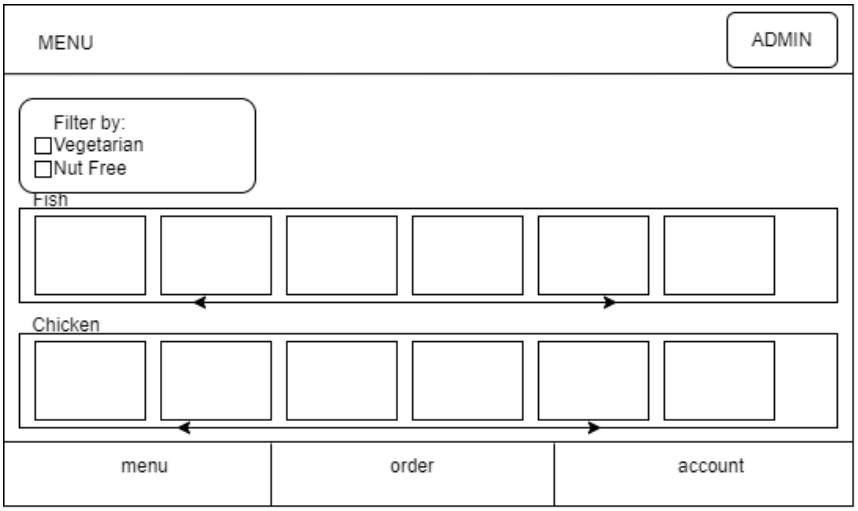

Figure 11: UI Design for Browsing Menu and Filtering Products

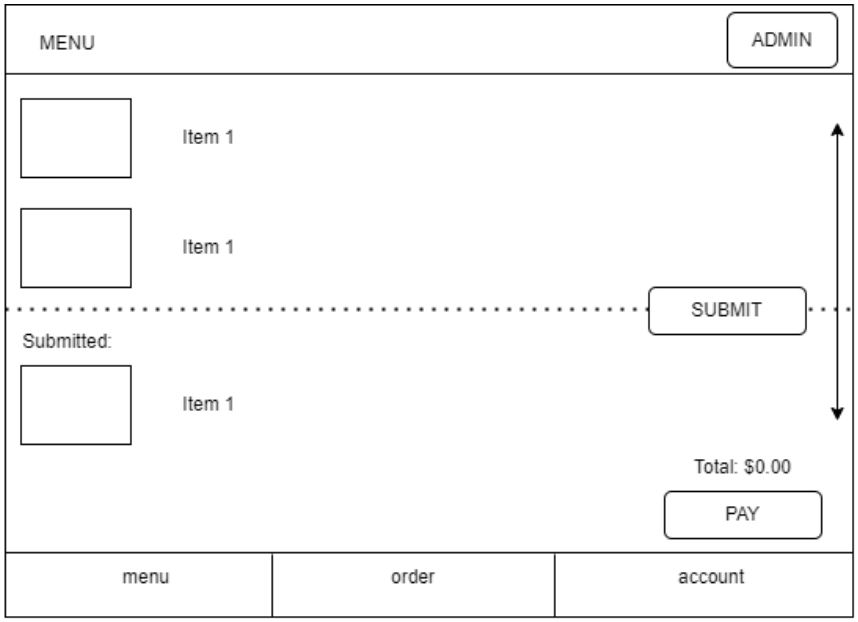

Figure 12: UI Design for Viewing Current Order and Submitting Items in Order

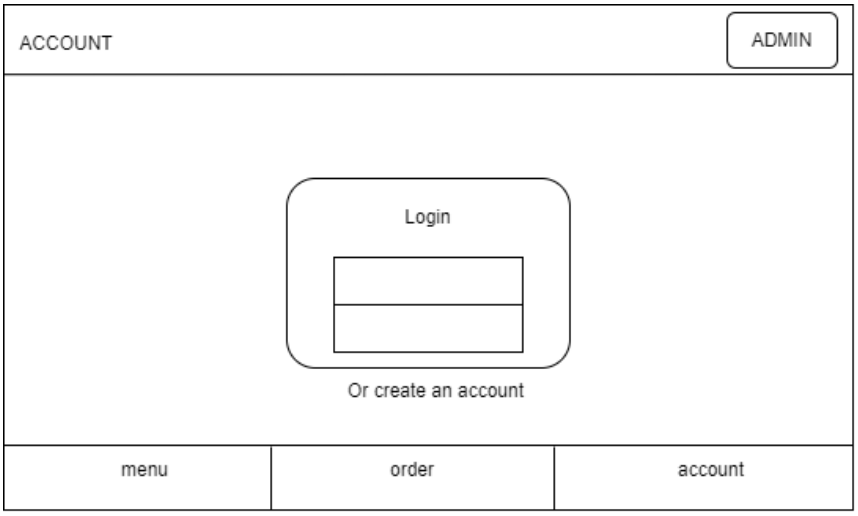

Figure 13: UI Design for Logging into User Account

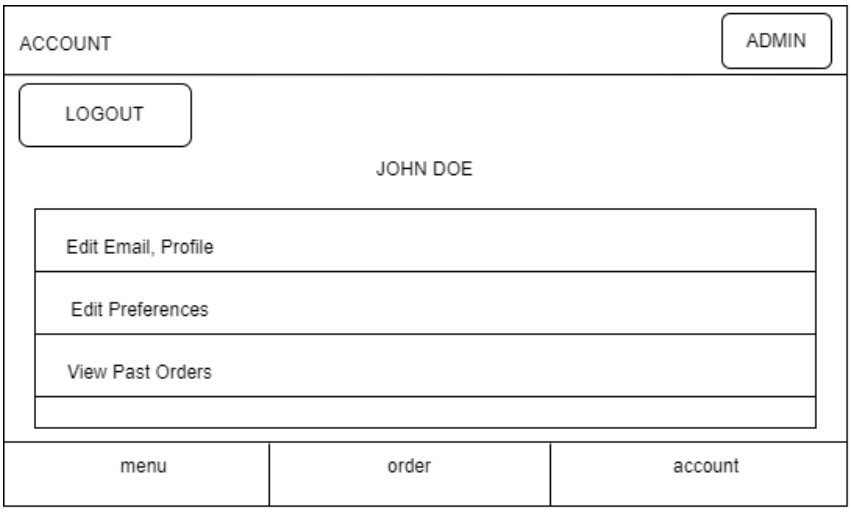

Figure 14: UI Design to View User Account

Note that staff are able to access staff mode by pressing on the “ADMIN” button at the upper left corner of the Customer Interface.

The UI design of the Staff interface is shown below:

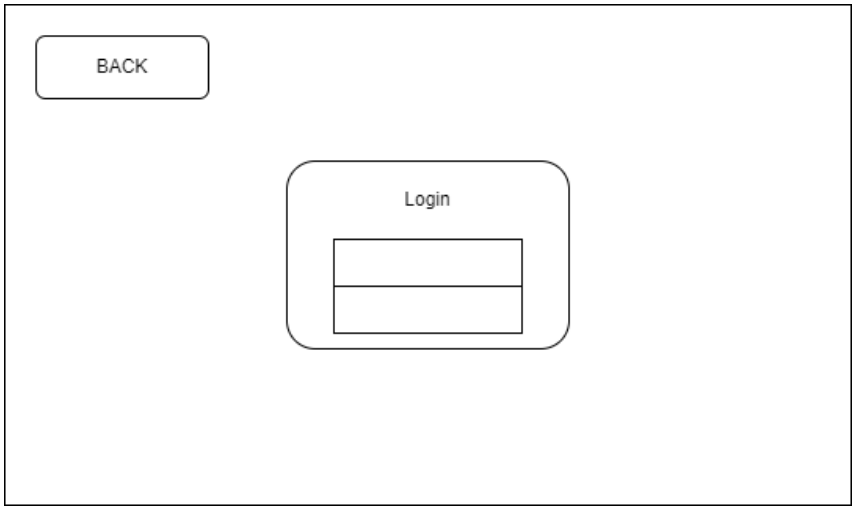

Figure 15: UI Design for Logging into Admin Mode

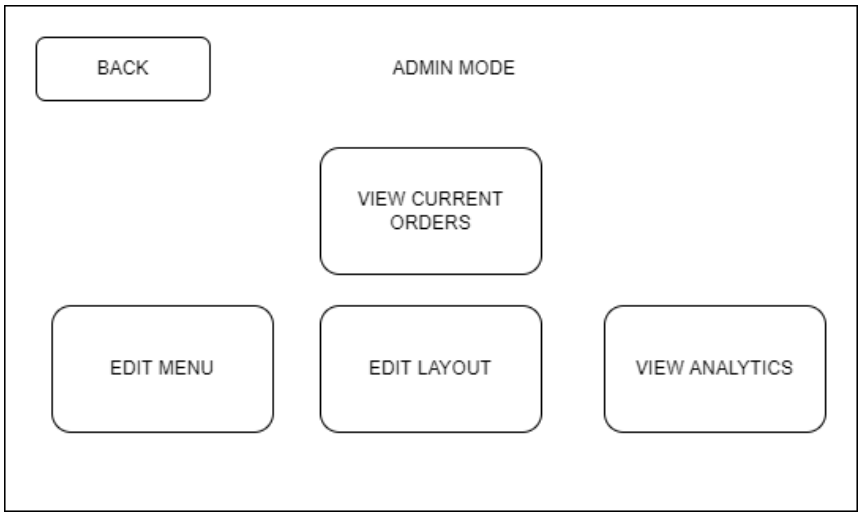

Figure 16: UI Design for Admin Navigation

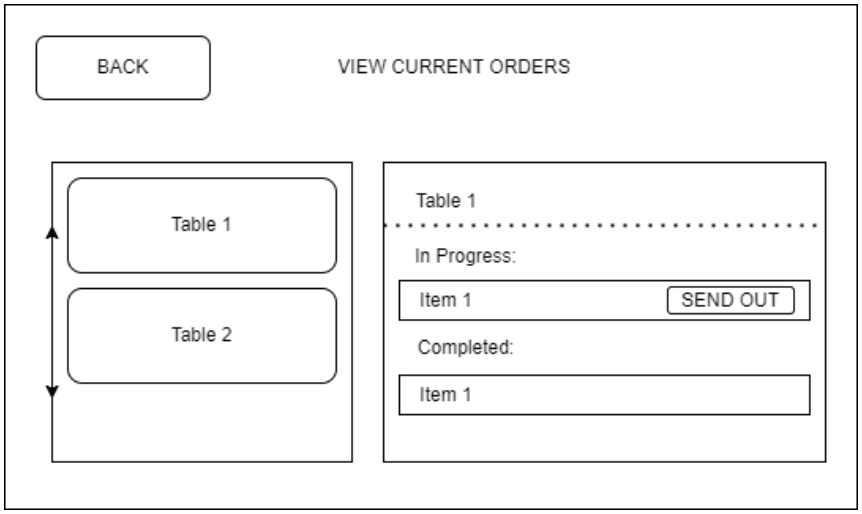

Figure 17: UI Design for Admin to View Current Orders and Send Items Out

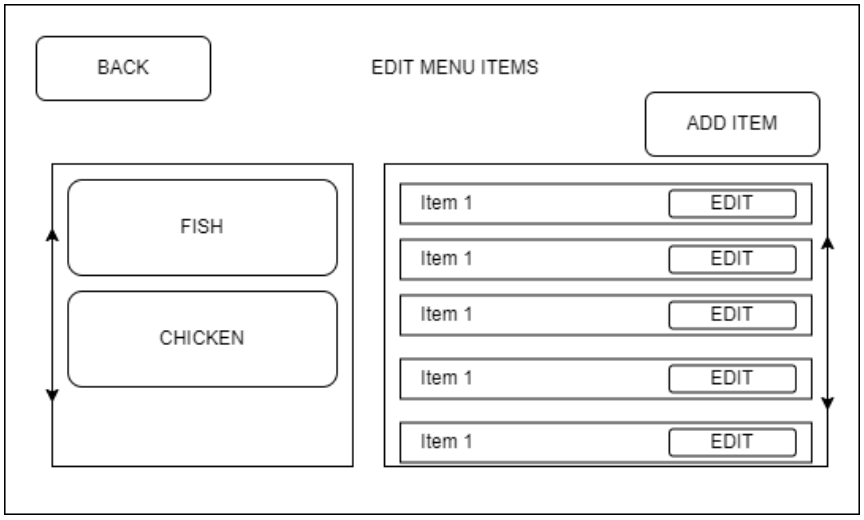

Figure 18: UI Design for Admin to Modify Menu

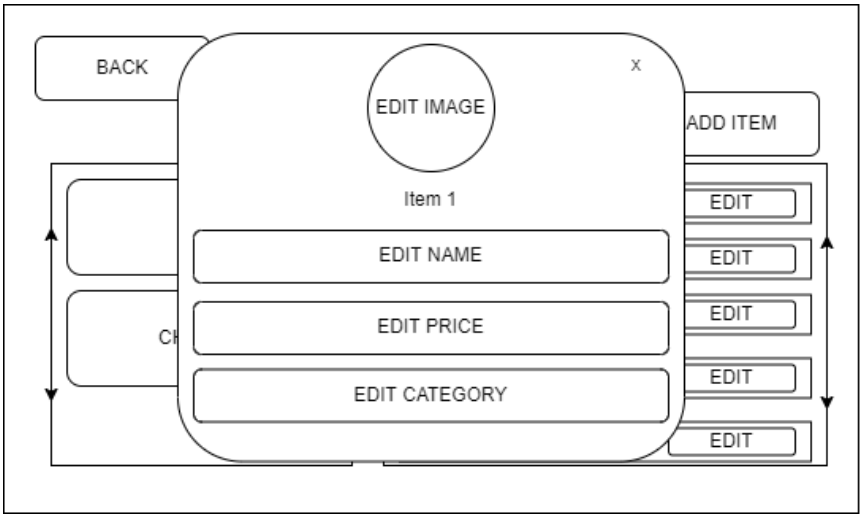

Figure 19: UI Design for Admin to Modify/Add Menu Item

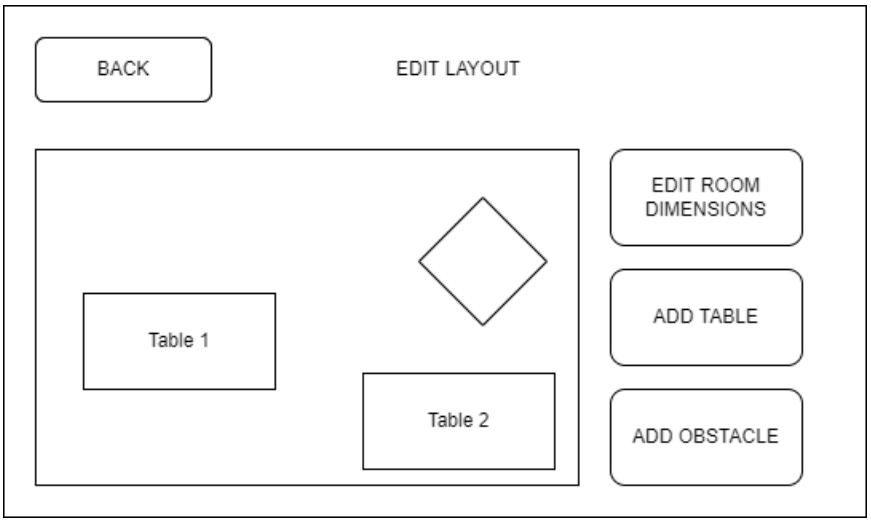

Figure 20: UI Design for Admin to Modify Restaurant Layout

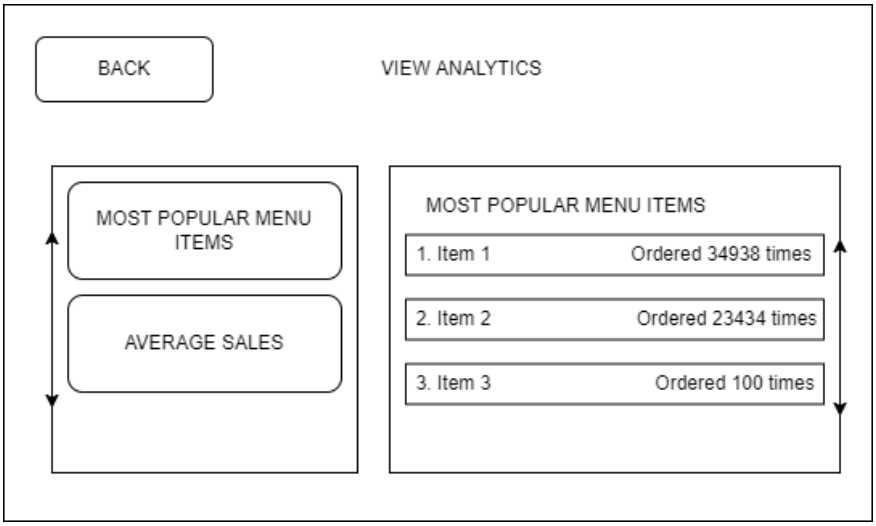

Figure 22: UI Design for Admin to View Analytics and Example Viewing Most Popular Items

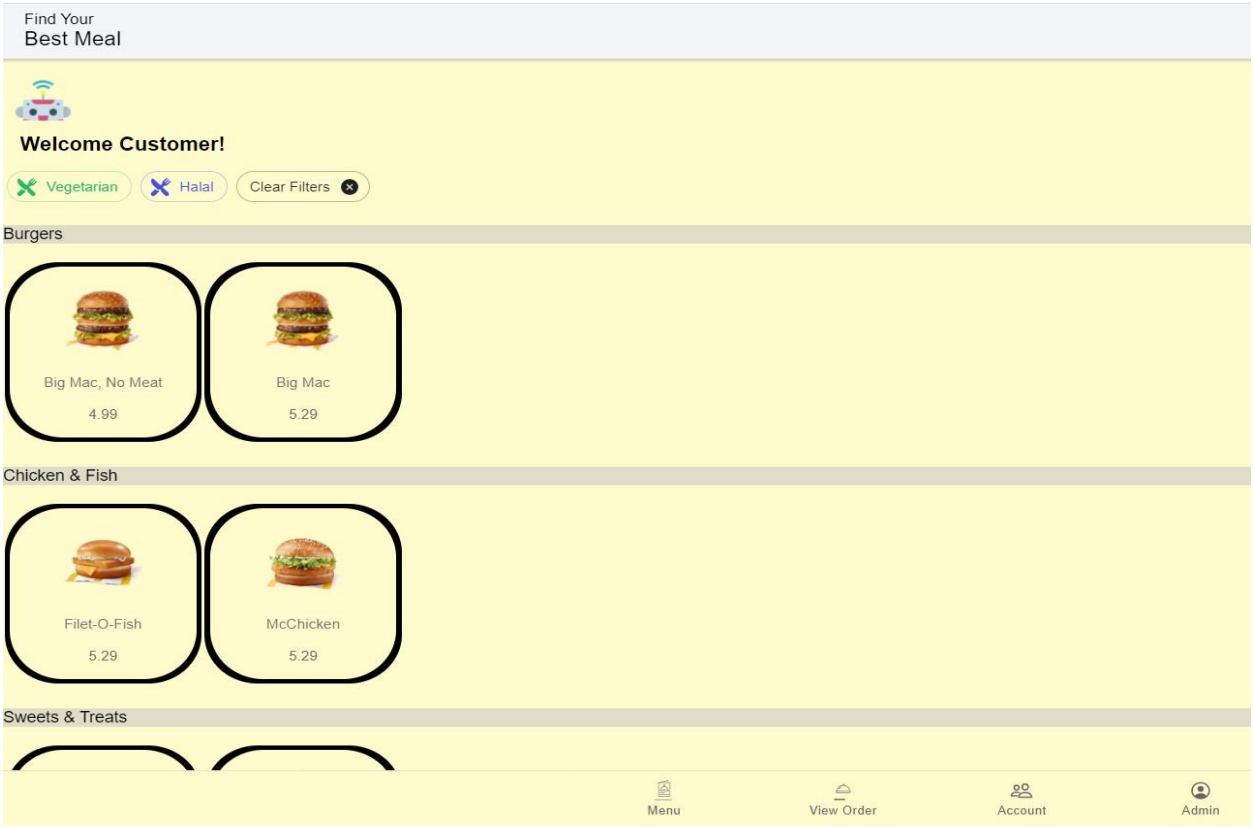

Figure 23: Current Implementation of Menu

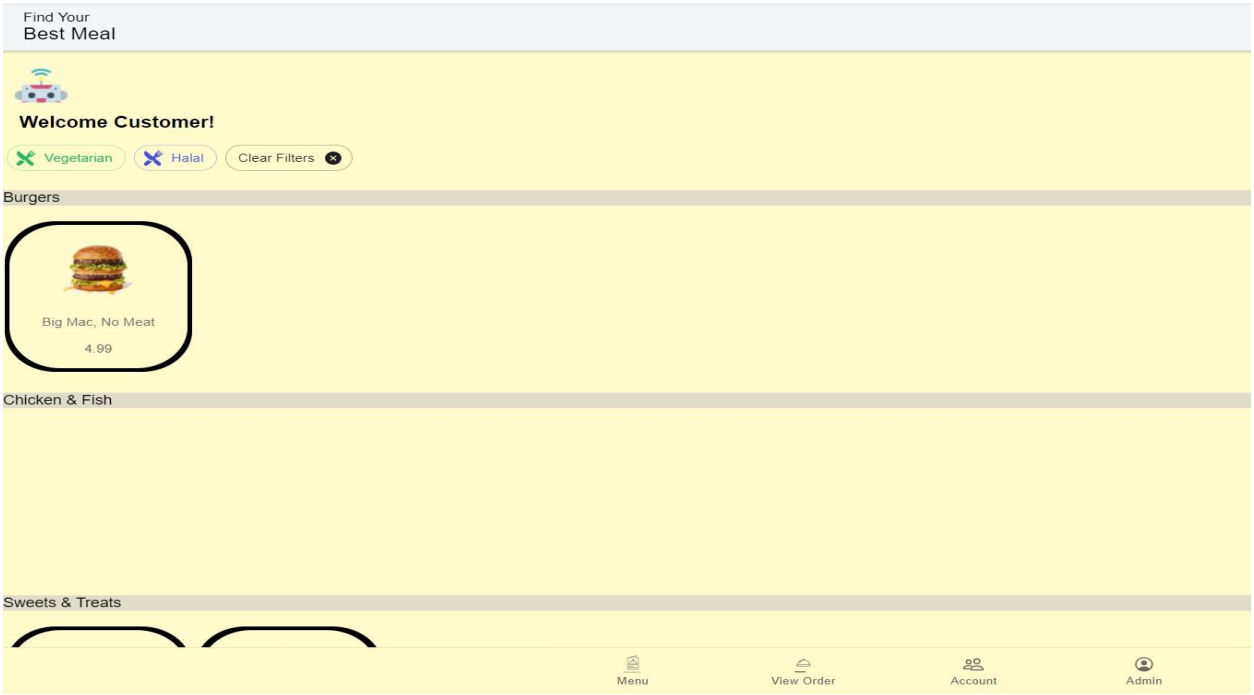

Figure 24: Current Implementation of Menu when filters are applied

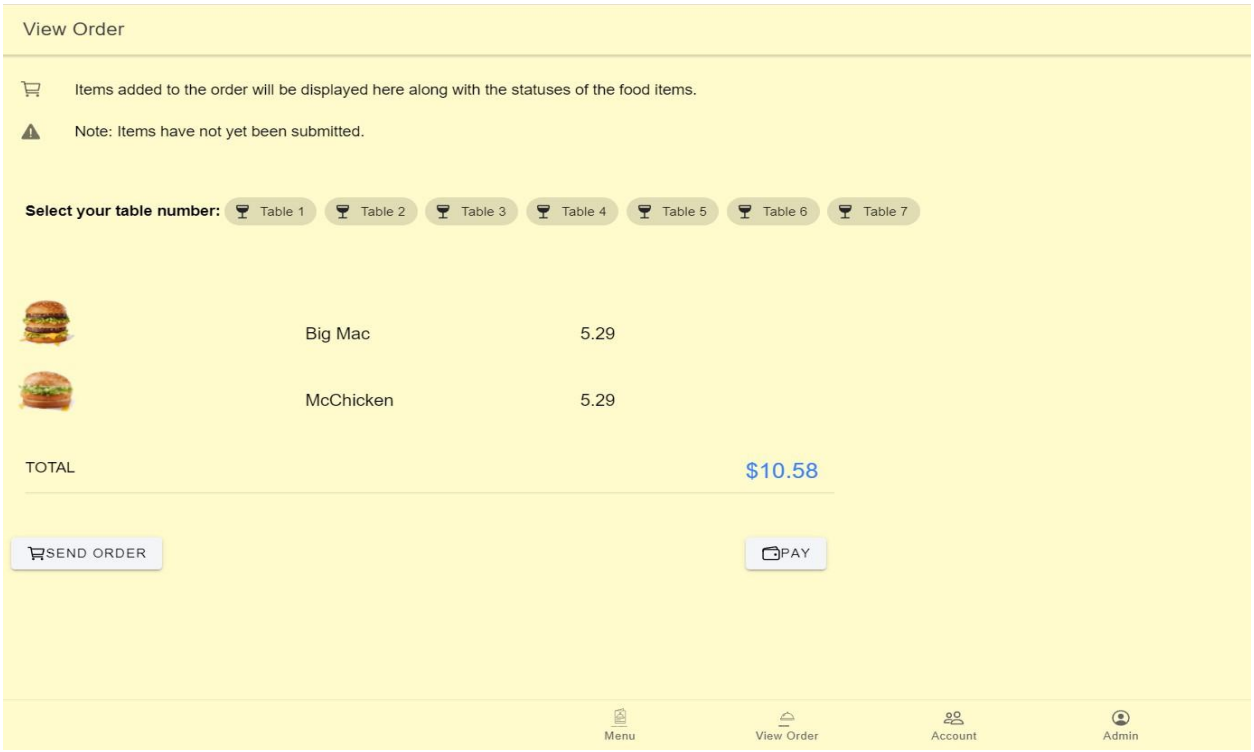

Figure 25: Current Implementation of View Order

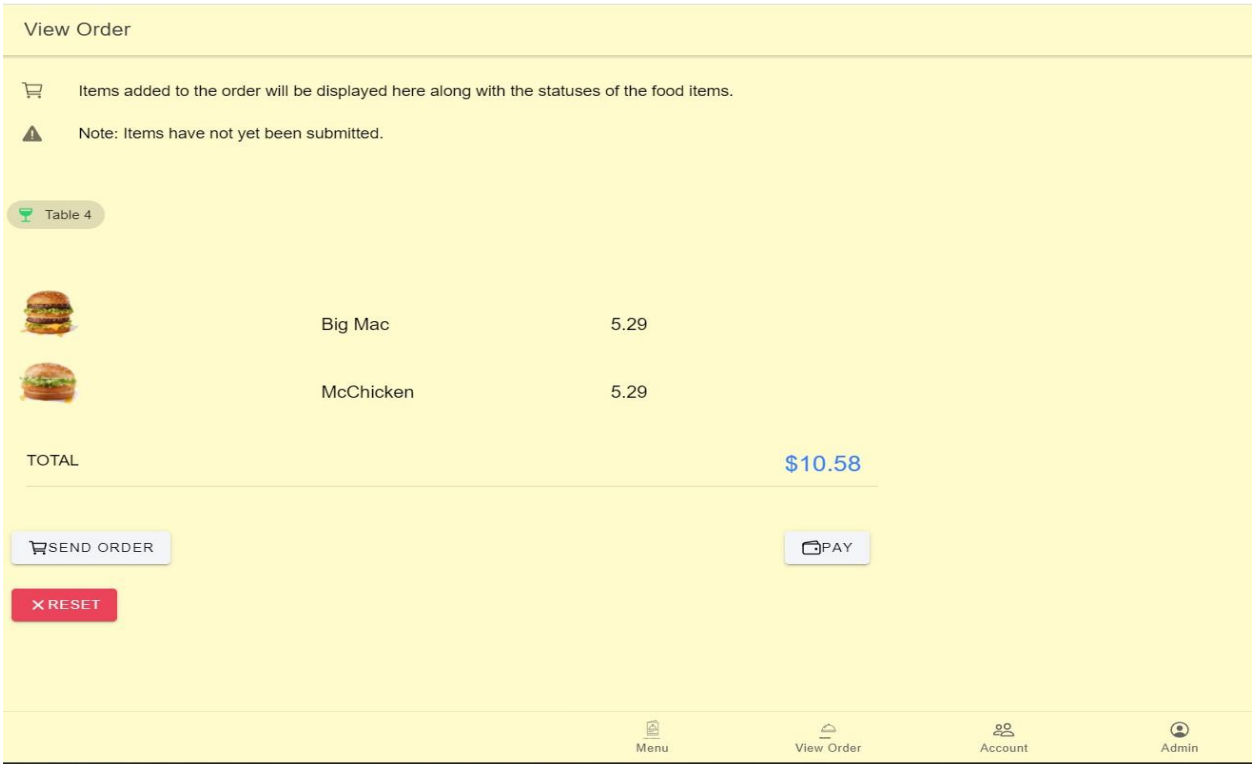

Figure 26: Current Implementation of View Order when table selected

Figure 27: Current Implementation of Pay

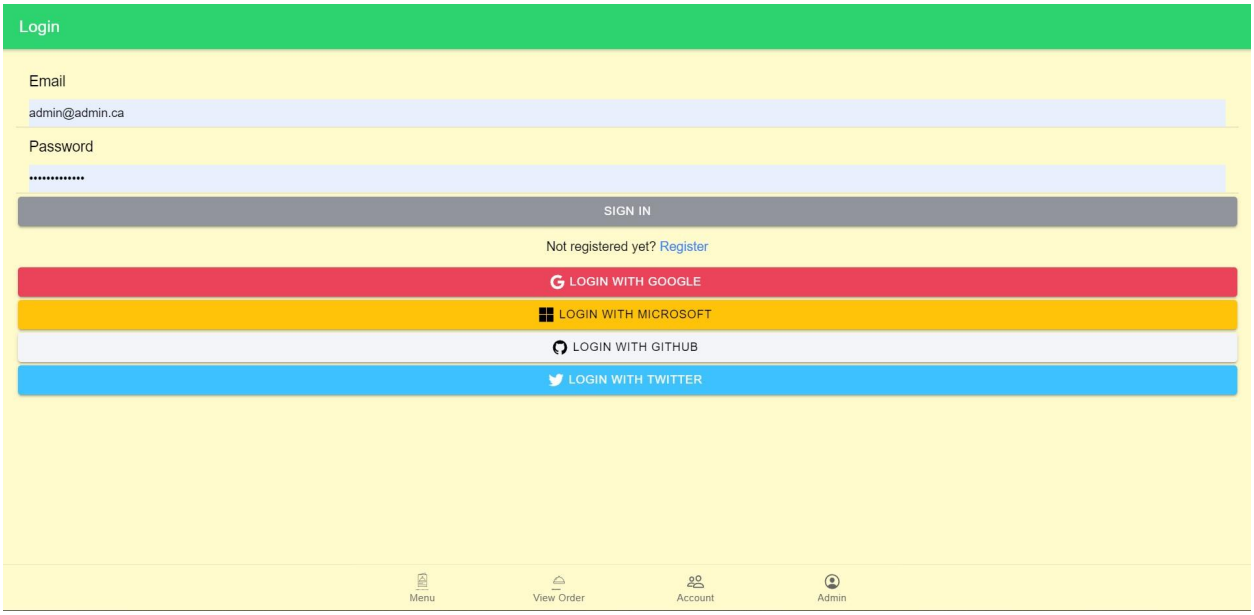

Figure 28: Current Implementation of the Customer Login page

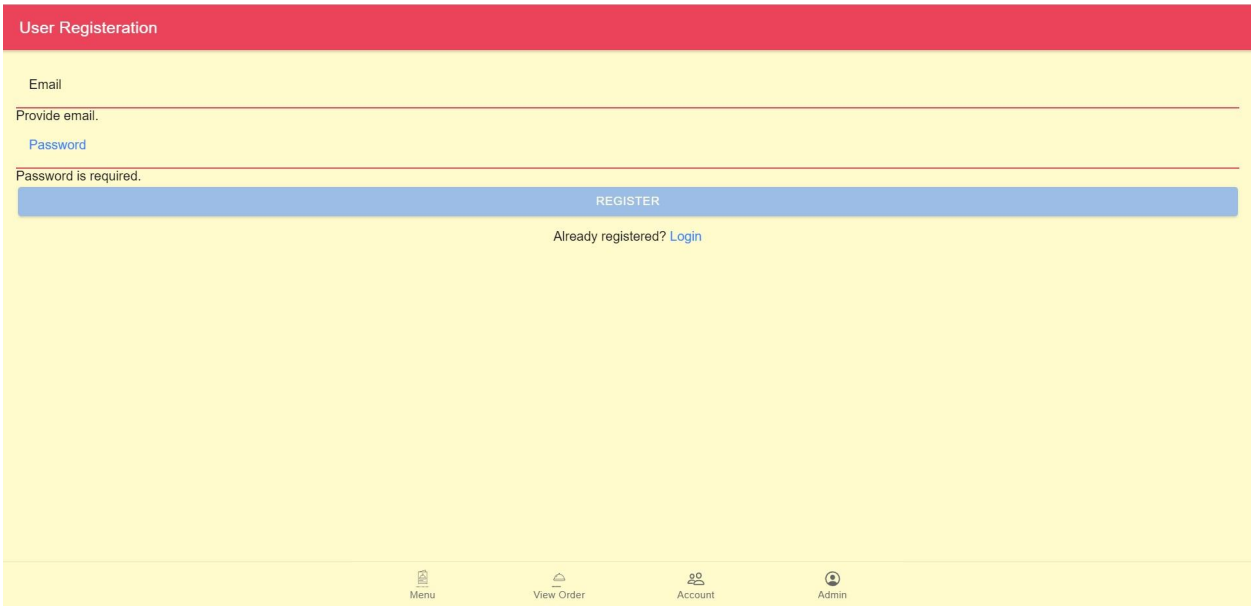

Figure 29: Current Implementation of the Registration page

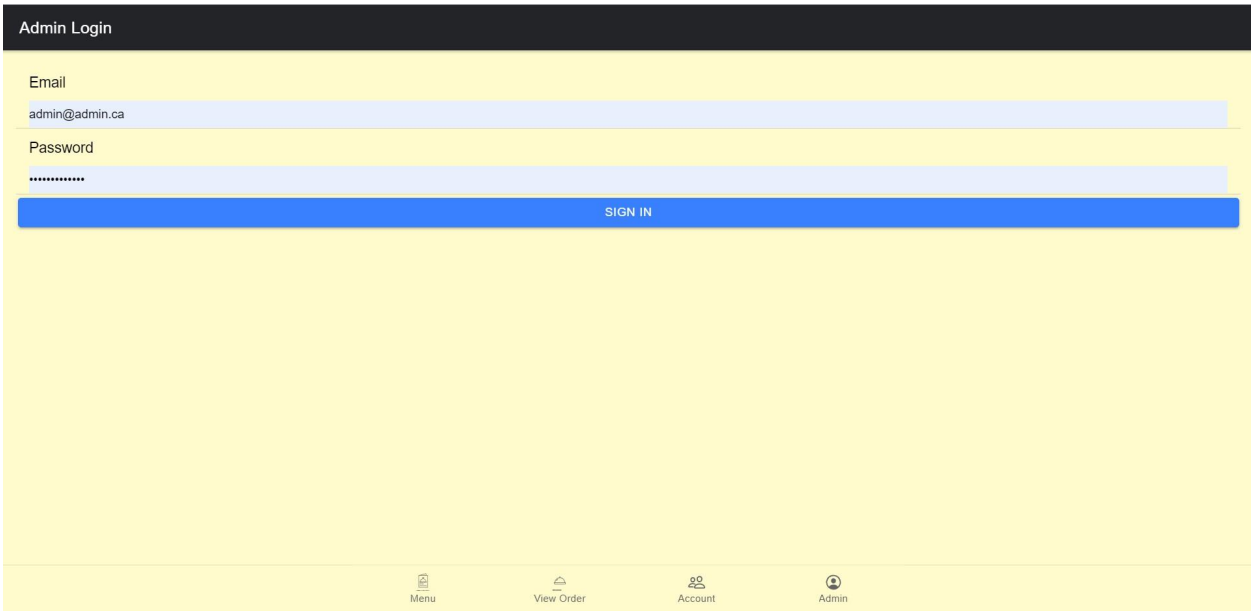

Figure 30: Current Implementation of the Admin Login page

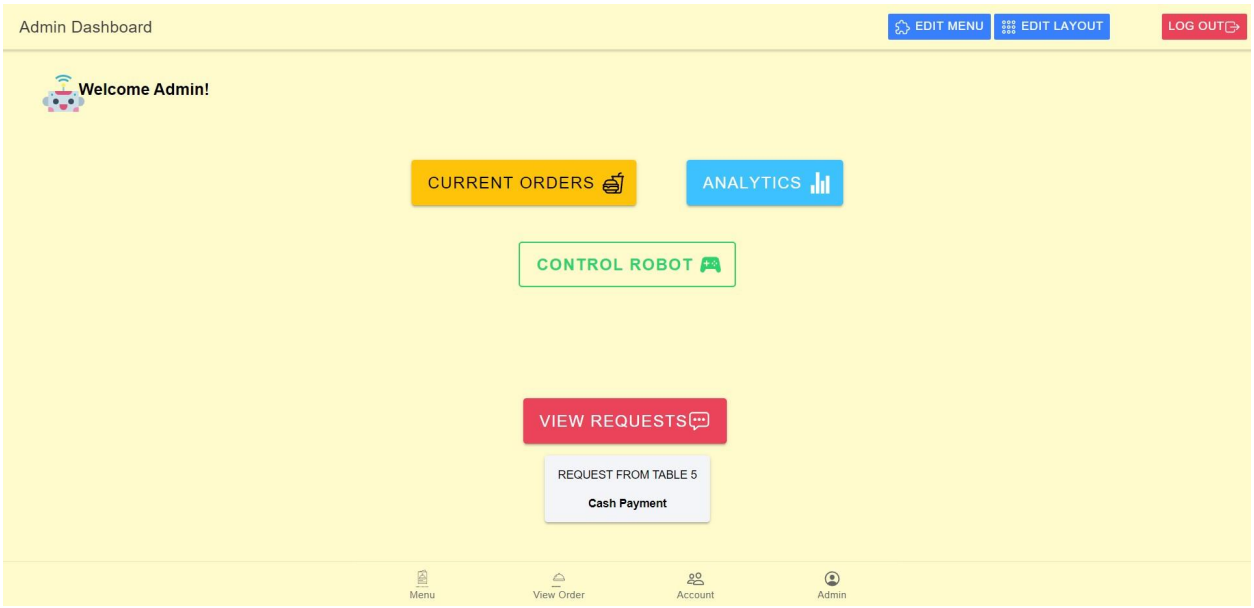

Figure 31: Current Implementation of the Admin page

Figure 32: Current Implementation of the Edit Menu page

Figure 33: Current Implementation of the Edit Menu page, adding an item

Figure 34: Current Implementation of the Edit Layout page

Figure 35: Current Implementation of the View Current Orders page

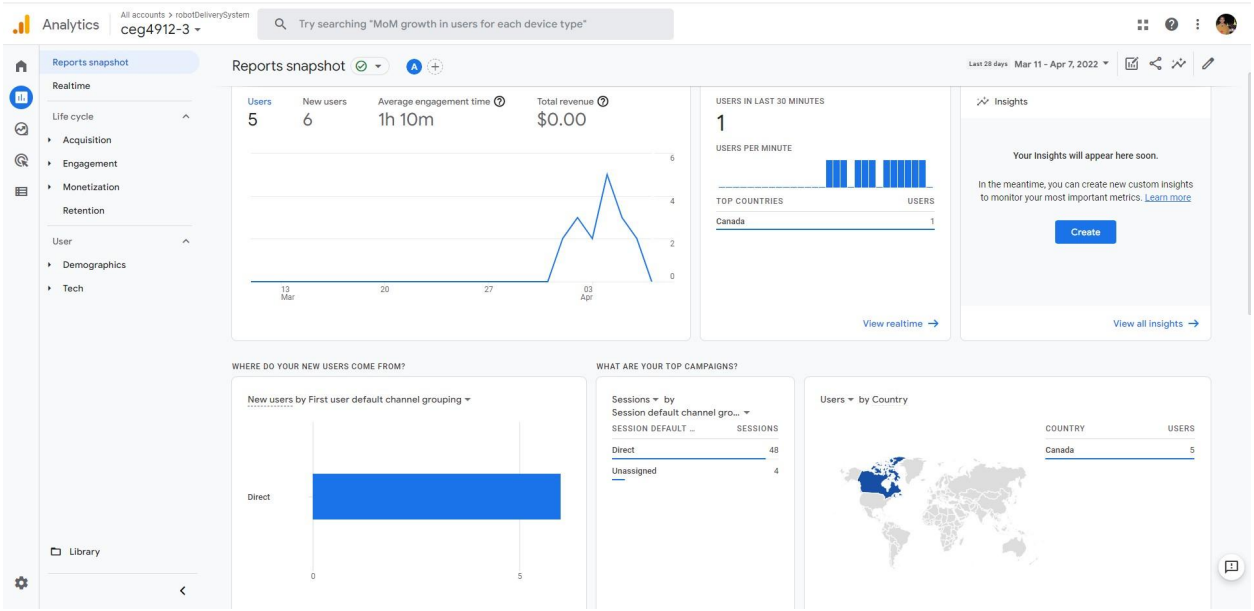

Figure 36: Current View of the Analytics of the application

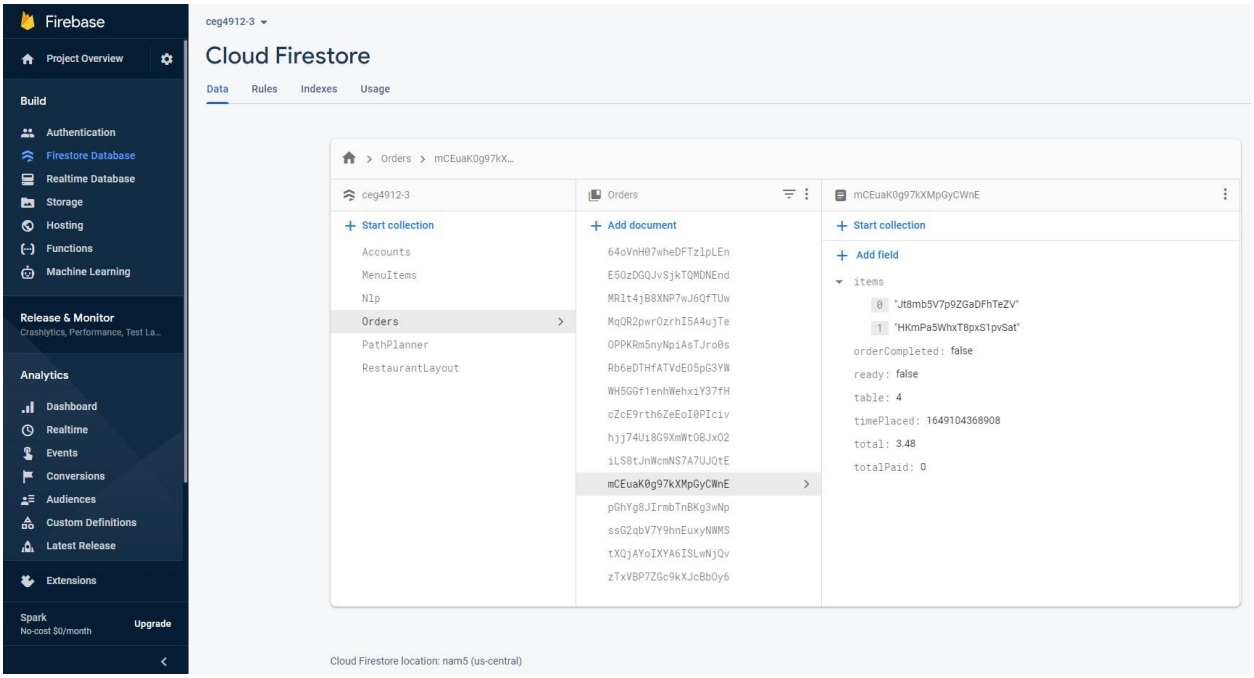

Figure 37: Current View of the Firestore Firebase

3.1.7 Back-End

Since the application will be a web application, it will communicate with a Google Firebase using a backend API. It will use Firebase authentication service to authenticate any users and the Firestore Database to store all additional information such as preferences, past orders and favourites. The database will use collections to separate and organize the different firestore ‘documents’ (where data is stored) and will use rules to protect all user data and restrict access to certain collections from unauthorized users. The following describes the collections that will be stored:

- **Account:** Where additional information for all accounts is saved. Used purely to increase customer experience by keeping track of preferences, contact information, past orders, and favourites as well as for analytics for administrator’s side. This

collection is also restricted by a rule where any user can only see and modify its own data.

- **firsName:** First name of customer
- **lastName:** Last name of customer
- **email:** Email of customer account
- **preferences:** Dietary restrictions saved onto account (e.g., Nut-Free)
- **pastOrders:** Past orders made by customer
- **favourites:** Menu items favorited by customer

- **MenuItem:** Where the information of all the menu items available in the front end is stored. This collection can be viewed by any one but only modified by an admin account.
  - **image:** Image for the current menu Item
  - **name:** Name of the menu item
  - **categories:** Dietary filters which the menu item belongs to (e.g., Vegan, Gluten-Free, etc)
  - **price:** Price of the menu item
  - **type:** Category which the menu item belongs to (eg Appetizer, Burgers, Fish)

- **Nlp:** Where all the user requests made from our Natural Language Processor come. The admin side has an observable on this collection and therefore, every time a customer makes a request through the robot, it appears automatically on the admin dashboard.

    - **Acknowledged:** Flag to indicate whether the request has been acknowledged by an employee or not
    - **Request:** Request from customer
    - **Table:** Indicates table from which the request was made from
- **Order:** All orders made will be saved into the backend after order completion for analytics, as well as for the customer to review on their customer accounts. This collection can be viewed and modified by anyone but only admin accounts can delete documents from it.
    - **items:** Menu items ordered
    - **orderCompleted:** Flag to indicate whether an order has been delivered to the table or not
    - **ready:** Flag to indicate whether the order is ready to bring to table or not
    - **table:** Table number at which the customer sat
    - **timePlaced:** Time at which the order was placed
    - **total:** Total due of the order
    - **totalPaid:** Total amount paid by customer

- **RestaurantLayout:** All tables for the restaurant are stored here. This collection's purpose is to have an up to date status of the availability of the restaurant tables. It is read and updated by the robot and the employee. All changes are reflected in real time on all ends.
    - **available:** Flag to indicate whether a table is available or not
    - **size:** Indicates the number of people that can sit at this table

- **table:** Indicates the number of the table

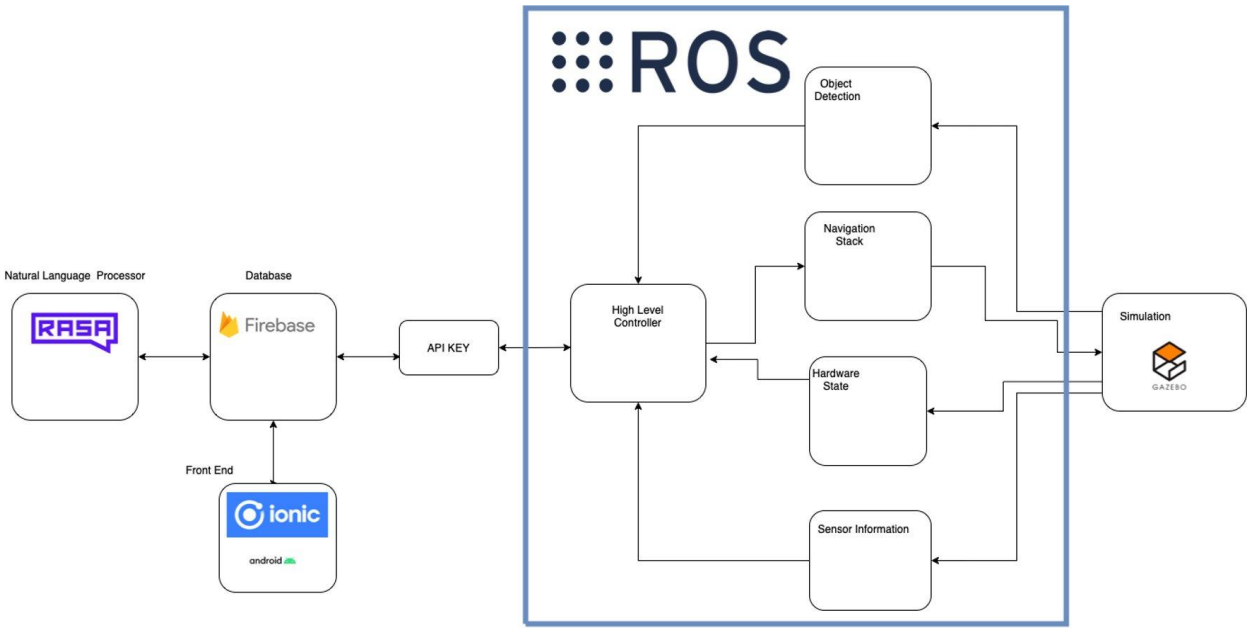

Figure 38: Software Architecture Conceptual Diagram

3.1.5 Low-Level Controller

The low-level integrates into a single entity all the functions used to control or read from different components. Currently, the low-level controller is not entirely completed as we have yet to integrate it with ROS. However, you will find below a list of its current functions that can be called by the high-level controller:

get_battery_level(): This function calls the external function *voltage ()* which returns the instantaneous voltage reading of the Main Batteries. From there, the function uses a custom mathematical formula which we have developed to return the current battery level in the form of a percentage (%).

get_rpm(): This function calls the external function *get_rpm()* inside *motor_encoder.py* to get an instantaneous reading of the current RPM of the servo motors. The value for the reading is returned.

get_speed(): This function calls the external function *get_rpm()* inside *motor_encoder.py* to get an instantaneous reading of the current RPM of the servo motors. It then multiplies this reading by the circumference of the wheel in meters and divides by 60. The result from this mathematical operation is the current speed of the wheel in m/s. This value is returned.

get_sonar_distance(): This function calls the external function *ReadDistance(int)* inside *sonar.py*. The integer argument that must be given when calling this function is simply a minimum distance threshold from the nearest frontal obstacle. The external function that was called will then return to the calling function either *True* or *False* depending on if we

have reached that minimum distance threshold or not. The calling function will return the boolean value.

stop (): This function stops the robot.

move_forward(): This function makes the robot move forward at maximum velocity.

move_backward(): This function makes the robot move backward at maximum velocity.

in_place_right(): This function makes the robot turn to the right in place (along the vertical axis of the robot). This is the fastest way to turn right however it will also cause higher centrifugal forces.

in_place_left(): This function makes the robot turn to the left in place (along the vertical axis of the robot). This is the fastest way to turn left however it will also cause higher centrifugal forces.

sweeping_right(): This function makes the robot turn right along the vertical axis of the left wheel (sweeping motion). This is the slowest way to turn right however it will also reduce the intensity of centrifugal forces. A secondary consequence of using this method to turn is that the robot will move forward by a distance equal to half the wheels trackwidth of the robot.

sweeping_left(): This function makes the robot turn left along the vertical axis of the right wheel (sweeping motion). This is the slowest way to turn left however it will also reduce

the intensity of centrifugal forces. A secondary consequence of using this method to turn is that the robot will move forward by a distance equal to half the wheels trackwidth of the robot.

3.1.6 High-Level Controller

The high-level controller is where all the logic about the behaviour of the robot is. It receives input from the user application, the image processor, the natural language processor, the path planner and the low-level controller and it outputs an action to the robot through the low-level controller. The high-level controller will basically be a while loop constantly requesting input from all the components to choose the best course of action. The requests are made through getter methods implemented in each one of the modules, the decisions are made by if statements inside of the while loop and the actions are performed by setter methods implemented in the low-level controller. The implementation of threads will help to keep a live state of the robot up to date as well as the readings and alerts from all the modules. Especially, from the low-level controller if it encounters an obstacle while moving from point A to B. It is still early on the implementation of the high-level controller module but some of the functions that can be found inside this module are the following:

init_modules_connection(): open connection with the google cloud server to be able to run the path planner, the image recognition module and the natural language processor.

close_modules_connection(): close connection with the google cloud server to save bandwidth when the robot is charging or turned off.

3.2 Hardware Architecture

During the first semester, we decided to start building the hardware part of our robot in advance. This decision was motivated by the desire to test the software for our sensors, motors and camera in the real world and start optimizing our code. Furthermore, this allowed us to observe flaws in our software design in advance and change what was not working. Please note that this design is not final and may change as we continue testing.

The hardware design for our project was done with two goals in mind. The first goal being that the design should be safe to avoid electrical fires or component damage. The second goal being to maximize the battery life of our robot.

To satisfy the need of safety, we have used a 5A fuse that is located between the main batteries and the rest of the circuit. In the event of a short circuit, the fuse will blow, preventing an electrical fire and overheating damage to the circuit elements. Furthermore, we have installed one diode going from the charging source to the main batteries and one diode going from the main batteries to the main circuit to prevent current flowing in the wrong direction. This increases the reliability of our design and protects against user error by protecting our circuit from polarity inversion.

Secondly, since our robot will be used in the catering business, where depending on the time of the day the demand can be very high for a prolonged period of time, we deemed

that it was important for our robot to have a long battery life. Indeed, having a long battery life allows the robot to operate during those prolonged peak hours without needing to stop working to go charge itself. To meet this design criteria, we have decided to use two separate battery modules. The first battery module labeled “Main Batteries” consists of two 6-cell batteries placed in parallel which is used to power the electric servo motors. The second battery module labeled “Secondary Battery” consists of a 10Ah/3.7V battery that is used to power the Raspberry Pi, Arduino, sensors and I/O devices. Not only is using a separate 3.7V battery providing us with more capacity but it is also avoiding the loss of power efficiency that would occur converting the 7.2V battery voltage to a 5V voltage required by our circuit.

The whole power system can be recharged using two Type A USBs which is more convenient than the specialized charger that was provided with the platform. Furthermore, in the future we plan on implementing wireless charging to avoid the need for human interaction in the charging process.

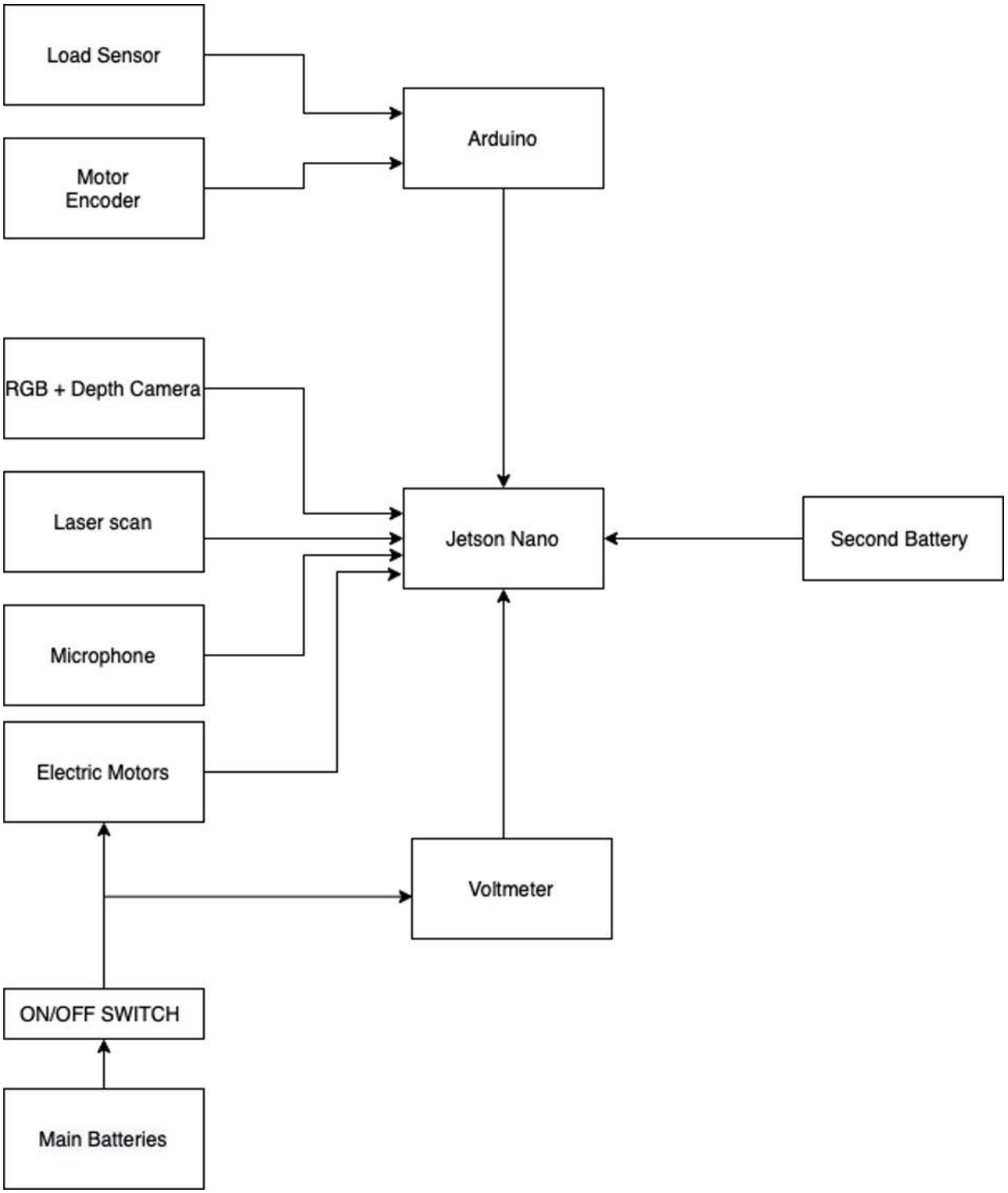

Figure 39: Hardware Architecture Conceptual Diagram

3.3 Frame Design

The frame design consists of the main platform containing the powertrain/electronics and a series of 3D printed parts attached to that platform.

In the figure below, the lower cylinder is the platform provided by the university. We decided to put the majority of our electronics in this section to lower the center of gravity and in this way improve both the stability of the platform and reduce the risk of tipping over.

The next rectangular shape located on top of that cylindrical platform is a 3D printed plate with 4 holes. These holes are used to bolt the remaining frame to the platform using metal bolts.

Above this plate we have attached a series of 3 hollow cylinders. The decision of making them hollow had to do with reduction of material cost, reduction of manufacturing time, reduction of weight, minimizing the height of the center of gravity, increasing the cooling of our electronics and allowing the tablet wires to pass through the structure. Furthermore, a cylindrical structure can support very well forces applied at different angles especially compressive forces (forces applied along the longitudinal axis of the cylinder). Please note that another valid design in terms of structural integrity would have been the use of triangles to form a truss. Although a truss would have been very solid structurally and would have reduced weight drastically over a cylindrical shape, it would have also increased the design complexity and would have been less pleasing aesthetically.

Finally, we have our 3D printed food plate that is attached to the top of the cylinders. The total height of the structure is roughly 0.75m with a maximum radius of 0.27cm.

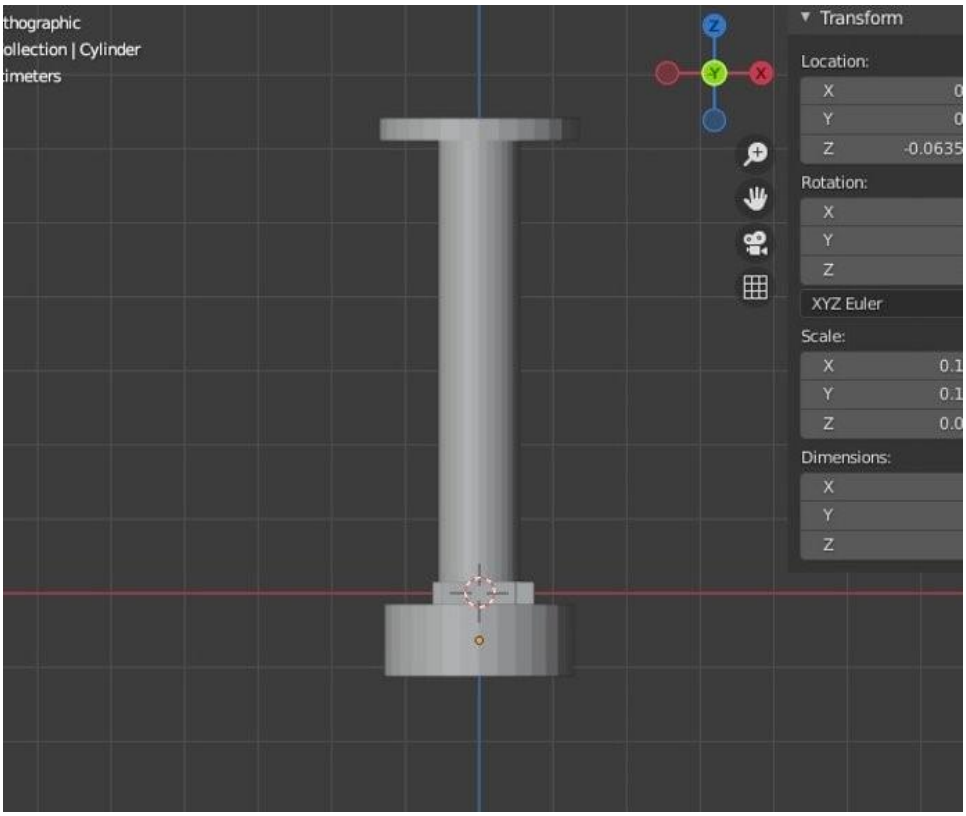

Figure 40: Frame Conceptual Design

4.0 Project Management

4.1 Gantt Chart

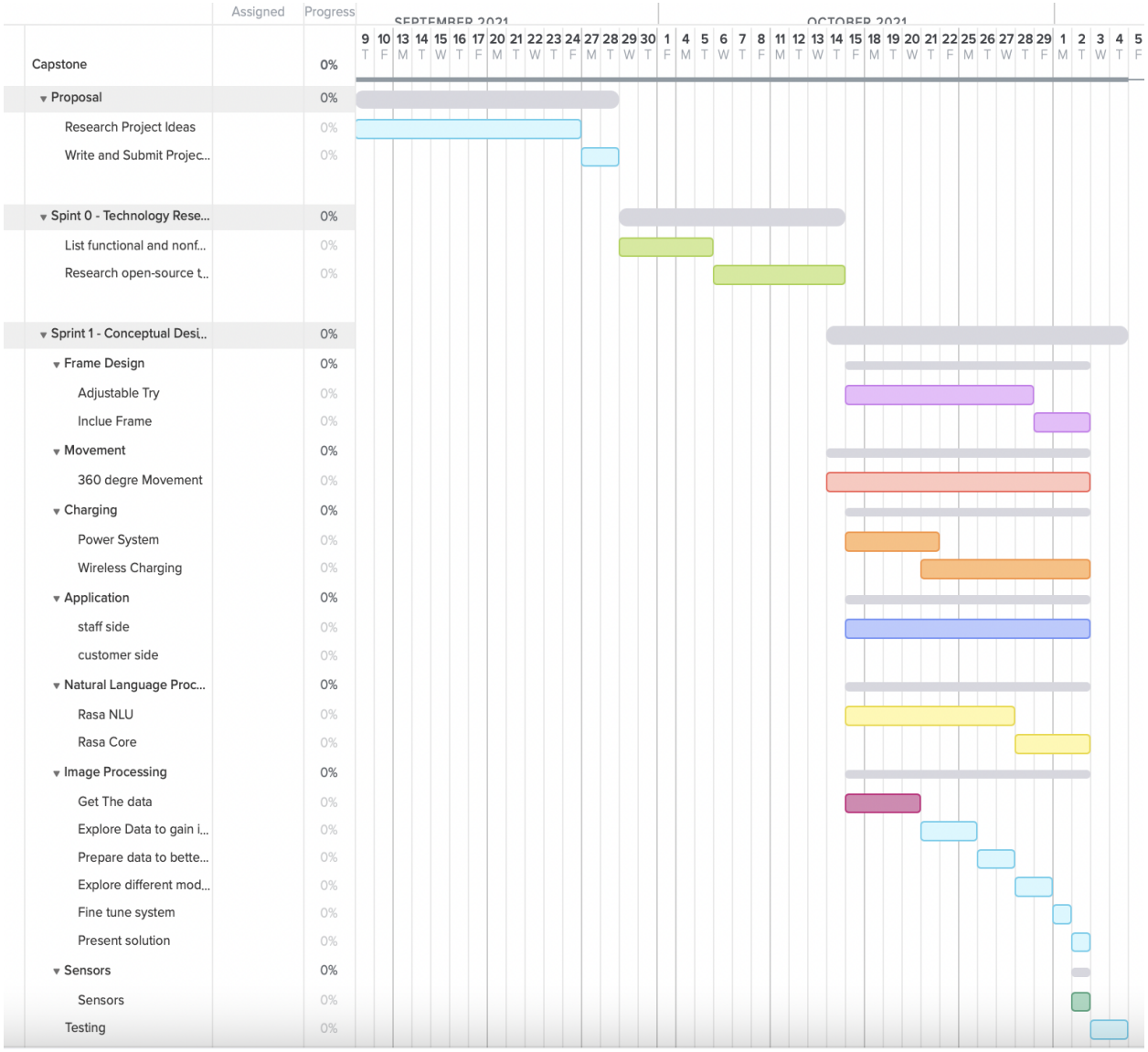

Figure 41: Gantt Chart Fall 2021 (Phase 1 - September 2021 to October 2021)

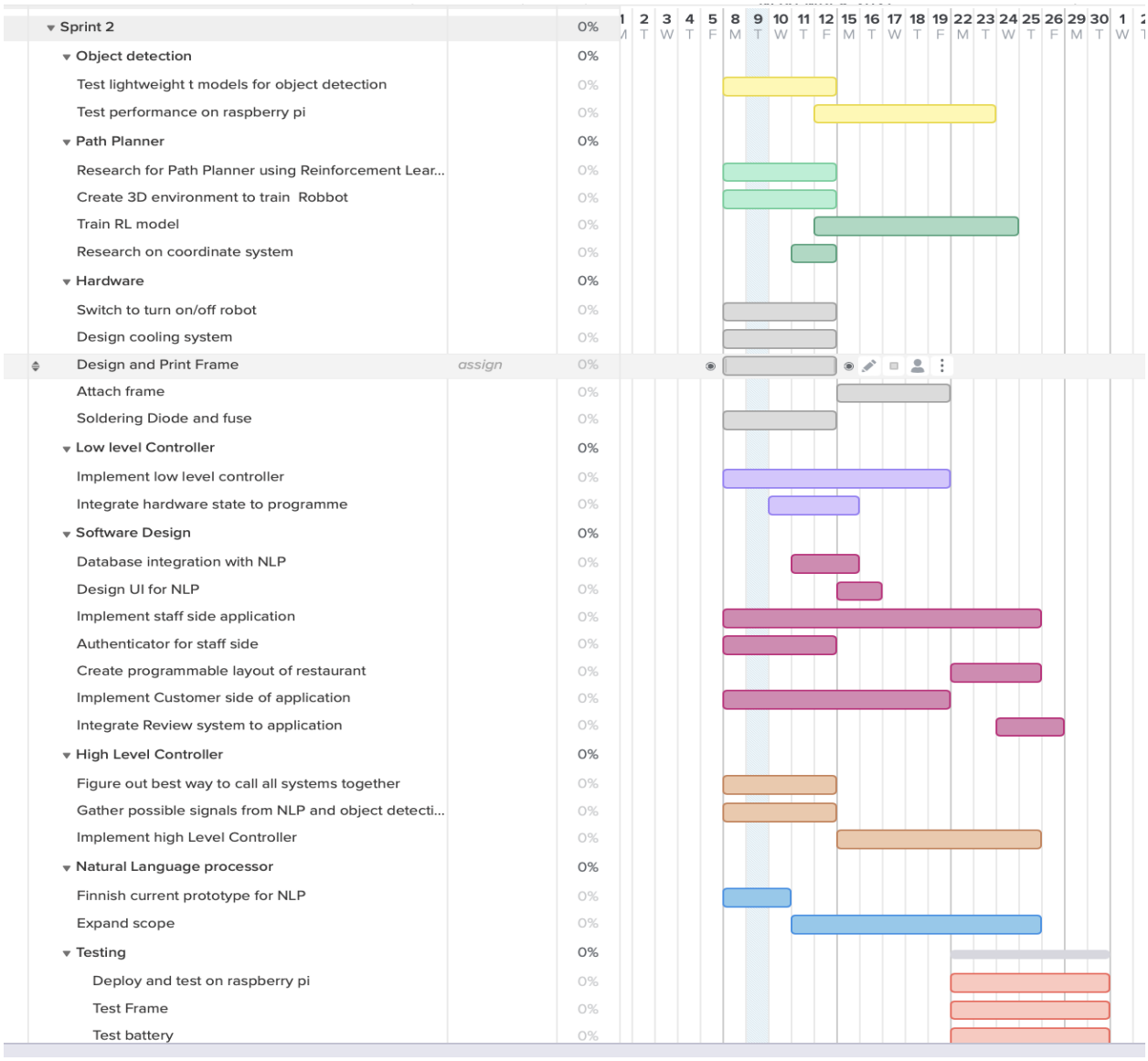

Figure 42: Gantt Chart Fall 2021 (Phase 2 - November 2021 to December 2021)

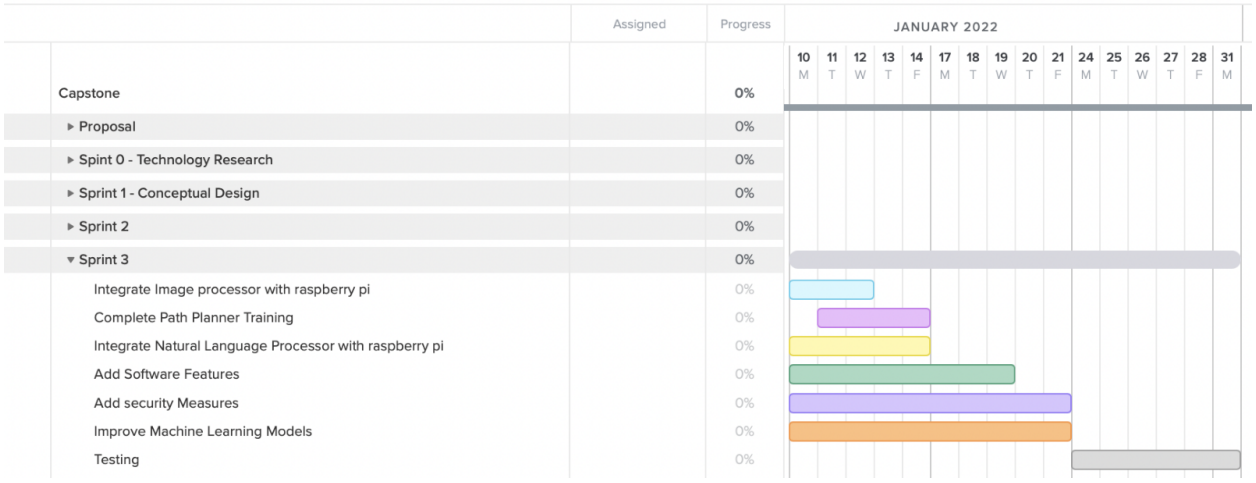

Figure 43: Gantt Chart Winter 2022(Phase 3 - January 2022 to March 2022)

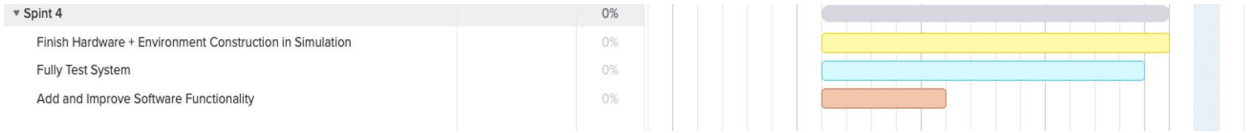

Figure 44: Gantt Chart Winter 2022(Phase 4 - March 2022 to April 2022)

4.2 Estimated Budget

Here is a complete breakdown of the budget needed to build and deploy the food delivery system.

Name	Price (CAD)
ATD Chip	$12.95
Blank PCB Board	$17.99
Analogue Voltmeter	$15.99
ON/OFF switch	$7.99
Jumper wires	$31.98
Fuse assortment	$13.59
Male stacking headers	$12.99
MCP3008-8-Channel 10 bit ADC	$12.78
Arduino 2x	$70
Breadboard	$10
Microphone	$32
Kinect Camera	$20
Google collab Pro + (2 months subscription)	$134.4
The construct sim Learner 2mo subscription	$114.18
Coral accelerator	$59.99 to $210
Jetson Nano Kit B01 Version	$573
Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055	$55
Android tablet	$135
Google cloud platform	$100
Bought	**$408.57**
TOTAL	**$1580.14**

4.3 Risk Management

As the system is deployed, we expect certain risks. The table below shows the risks that can occur, a brief description of them, and possible solutions for these risks.

Risk Item	Description	Resolution Approach	Who	Date
Machine Learning Models.	These risk items can arise: Incompatible data, insufficient data, Overfitting during training, distribution of test set. Hardware capabilities in deployment.	Use of existing data sets, real time tests, use of various performance measure, error analysis.	Amen A, and Harouna S.	As needed.
Hardware	Electrical Fire or Component Damage due to short-circuits or polarity inversion	Use of a 5A fuse to protect against short-circuits. Use of two diodes to protect against polarity inversion.	Marc-Frédérick Tremblay	During hardware design and conception.
Application	Security and Privacy Breaches Slow performances due to connectivity	Use HTTPS when making calls to the database, writing secure rules for application access to the database Limit the number of calls made to the database	Santiago C, Michelle T.	During implementation.

4.4 Contribution List

The following are the contributions since the midterm report:

Amen Abshir

- Implementation of high level controller and database access by the high level controller
- Implementation of the robot's urdf and testing environment.
- Implementation of the path planner

Santiago Cely

- Implementation and testing of signin with external providers, layout of the restaurant, order payment, analytics, user requests and the send order functionality.
- Implementation and testing of Cloud Firestore Rules as well as in app tracking user state to restrict information access from unauthorized users
- Design of the User Interface

Michelle Tang

- Implementation and testing of the admin side of the application: view-current-orders module and edit-menu module. Ensured that both modules were correctly getting and passing updated data to the database.
- Unsuccessful implementation of modal and modal controller for view-current-orders and edit-menu modules.

Harouna Sylla

- Implementation and testing of Natural Language Processor
- Integration of Natural Language Processor in the System

Marc-Frédérick Tremblay

- Hardware design & conception.
- Coding for the sensors.
- Implementation of the low-level controller.
- Implementation of the hard-coded collision avoidance algorithm.
- Frame design & conception.
- Calculations of Center of Gravity (CoG), Static Stability Factor (SSF) and Rollover Risk to optimally distribute the weight in our robot.
- Design of the User Interface

5.0 References

1. Christiansen, B. L. (2021, March 5). *Clear Pros and Cons of an Automated Restaurant*. Hubworks. Retrieved October 23, 2021, from https://zipschedules.com/restaurant-management/automated-restaurant.html
2. Team, D. (2021, September 7). *Benefits of Artificial Intelligence in the Restaurant Industry*. Deputy. Retrieved October 23, 2021, from https://www.deputy.com/blog/benefits-of-artificial-intelligence-in-the-restaurant-industry
3. Justin Lokitz. *The future of work: How humans and machines are evolving to work together* https://www.businessmodelsinc.com/machines/
4. Aurélion Géron *Hands on Machine Learning with Sci-Kit learn, keras and TensorFlow.*
5. Dr. Pierre Payeur, *COMMANDE PAR ORDINATEUR EN ROBOTIQUE Chapitre 3, CEG 4558*
6. Ionicframework. (2021, September 9). *What is hybrid app development?* Ionic Article. Retrieved October 20, 2021, from https://ionic.io/resources/articles/what-is-hybrid-app-development.
7. Google. (n.d.). *Use The Cloud Firestore Rest API | firebase documentation*. Google. Retrieved October 21, 2021, from https://firebase.google.com/docs/firestore/use-rest-api.
8. ROS, https://wiki.ros.org
9. Rasa documentation. https://rasa.com
10. Move base package ROS http://wiki.ros.org/move_base
11. Amcl package ROS http://wiki.ros.org/amcl
12. Gmapping package ROS http://wiki.ros.org/gmapping