JAMZ Business Platform

User Guide

Work by:

Quinn Finnegan
Samual Tan
Younes Sabate
Etinosa Amadasu Clement
Serban Popovici

Table of contents

Introduction **2**

Problem statement **2**

Design criteria **2**

Ideation **4**

Login/Sign-Up **5**

Signup Page 5

Need Help Page 6

Login Page 7

Password Reset 8

First Time Setup 8

Menu Editor **9**

JavaScript - Menu 9

Enter Button 9

Edit Button 10

Remove button 11

Drop-zone 12

Order Manager **12**

Main Dialog 12

Map 14

Profile **14**

Settings **15**

Language translator 15

Template control 16

Page Integration **16**

Sidebar 16

SQL Database 16

Project schedule **17**

Bill of Materials **18**

Future Improvements and Lessons Learnt **18**

Database Integration 18

Menu Page Drop-zone 19

Button Animations 19

Order Shipping 19

Conclusion **19**

Appendix **20**

References 20

Introduction

For our project, we have decided to help JAMZ create a user-interface for restaurant owners. The application allows users to customize and display their menu items, manage incoming and outgoing orders, and connect with JAMZ's automated drone delivery service. To create this interface, we have decided to use programming languages HTML, CSS, Javascript, and PHP. All the code is written using an application called notepad++ which allows for the connection between multiple languages. This was helpful to our project because of the need for both functionality and aesthetics. Javascript was used to give features functionality, while CSS and HTML were used to enhance the aesthetics of our application.

In this report, instructions on how to create each page in our application is briefly explained. This document covers mainly how the key features were made, what programming problems we faced and how we resolved them. In addition, an explanation of how we integrated each subsection is provided. The problem statement, design criteria, ideation stage, bill of materials, and further improvements were also discussed in this document.

The code for our final product can be found on our GitHub repository. The link to this can be found on our makerepo page titled "JAMZ Business Platform A4 by Presidential Constituency" and in the appendix section of this report.

Problem statement

The following is the problem statement for our project, which our team used to generate ideas and also used to guide ourselves towards our final product.

"A simple, user-friendly, and creative user interface is needed for businesses to connect with customers living in rural and suburban areas and deliver their products using drones".

Design criteria

The needs for this user-interface were organized in our design criteria, which explains in detail what the functional and non-functional requirements are, as well as constraints to our design. The condensed version of our design criteria can be found in the table below.

Design Criteria

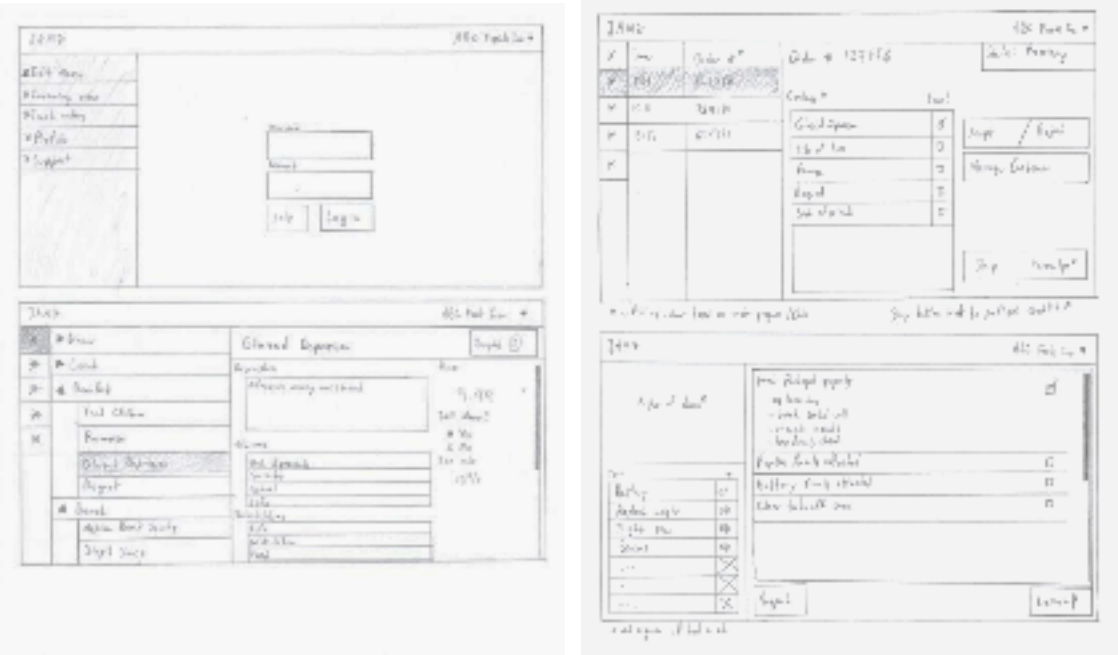
Need	Design Criteria
Functional Requirements	
Login/First-time sign-up page	**Login page** Allow users to log in. Have options for forgotten passwords and possibly two-factor authentication. **Setup page** Available on the first startup, allows businesses to fill out initial information about who they are (address, phone, logo, pictures, name, hours, etc.) as well as username and password.
Edit menu items	Options to add, edit and remove menu items. Alter item descriptions and pictures as well as allow the user to take pictures of items.
Order tracking	Drone telemetry display (battery life, speed, time of flight, time to delivery, GPS coordinates?). Show drone ID with details of order onboard (order number and contents).
Connect with the drone operator	A page that allows the business to communicate with the drone operator through text
Connect with customers	**Incoming order page** View all orders that are in progress, allow businesses to reject orders, and allow businesses to contact customers about orders in progress. **Outgoing order checklist** Provide a checklist to ensure all items are onboard aircraft and also include a preflight checklist for aircraft before drone dispatch.
Non-Functional Requirements	
Customize profile	After the first startup, this allows users to alter their profile but also view reviews and general rating of the business. This page can be used to reply to reviews as well. Allow users to change password and username/email and phone number associated with the account.
Troubleshoot issues	Help/support page
View customer reviews	Customer review/rating page

Different Languages	Settings page to change the language
Constraints	
Match website colours	Make pages and buttons and colour of layout look similar to the colours on JAMZ's website

Ideation

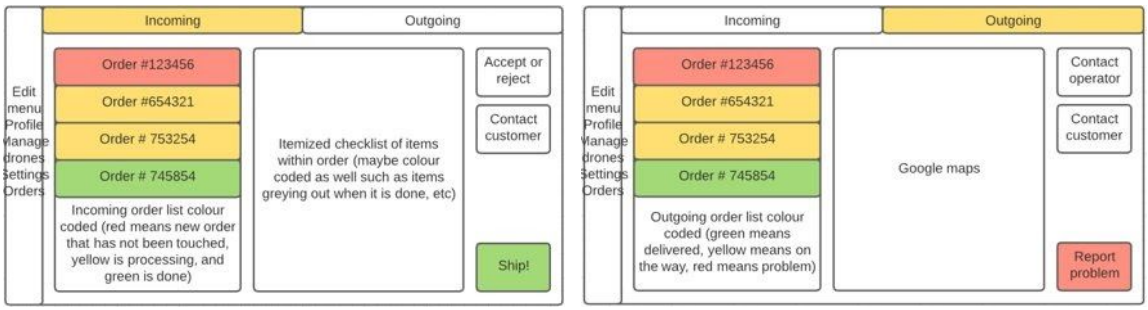
Prior to coding our application, some free-hand sketches were made for each subsection for the interface. Below are some of the sketches of the user-interface design.

Figure 1.1 - Initial sketch ideas



The figures below are refined sketches of the menu editor and the order manager section, which was done on LucidCharts. These sketches helped in the design for the aesthetic and structure of all our pages.

Figure 1.2 - Refined sketches

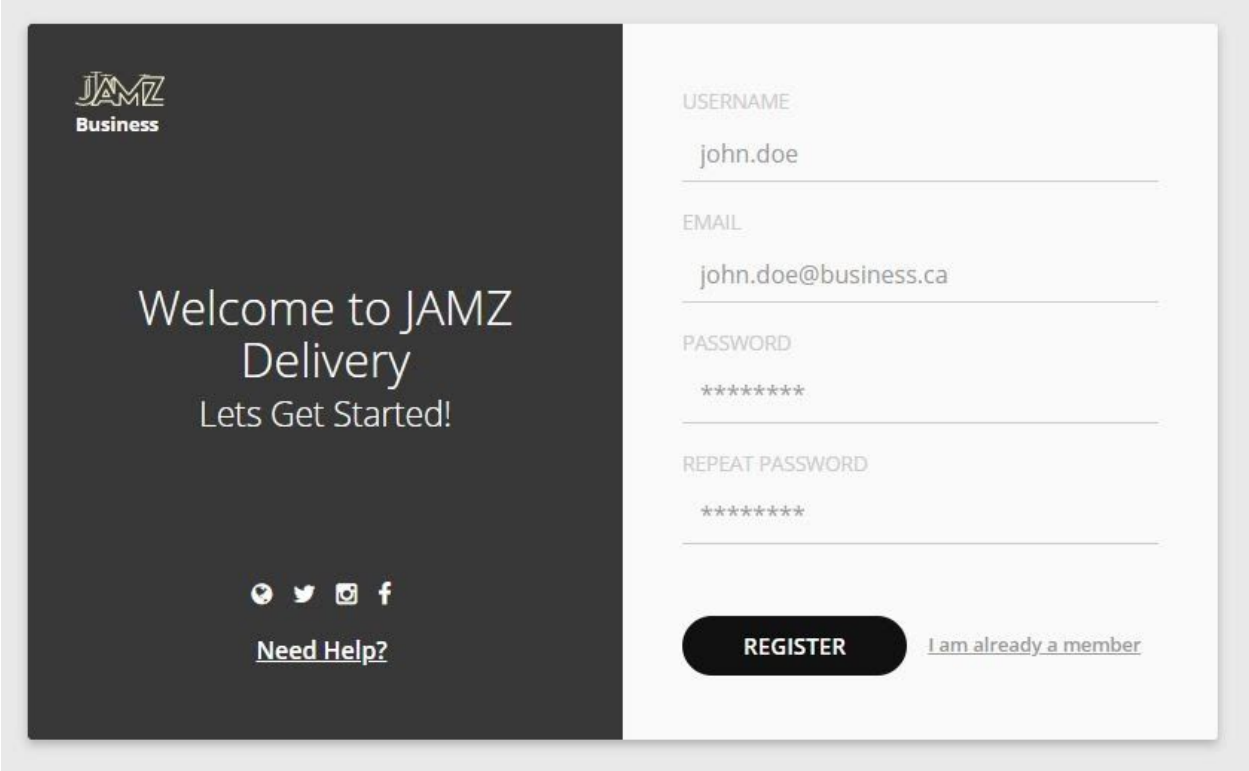


Login/Sign-Up

The sign-up page consists of a form for the user to enter their email and desired username and password. It also contains links to the returning user login page, the “Need Help?” page and JAMZ’s website and social media.

Signup Page

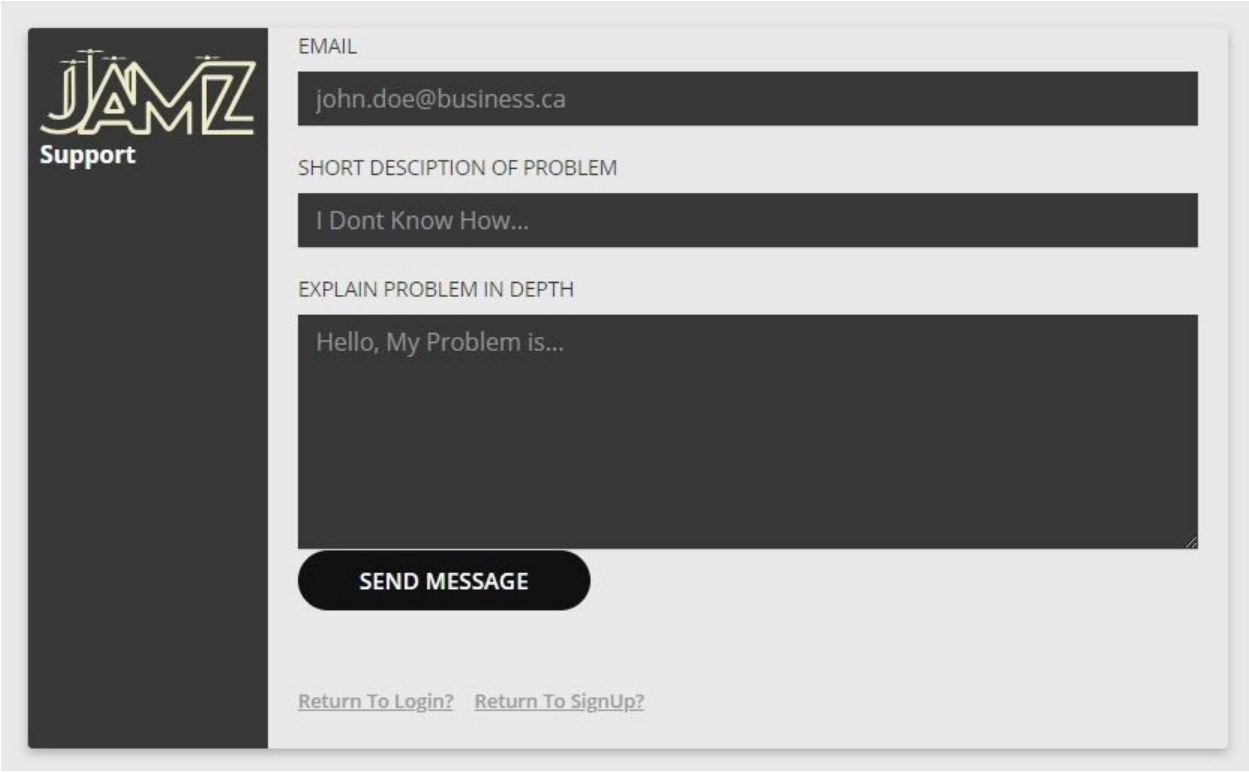
2.1 - First time sign-up



The base of this page is taken from a free signup page template from Alex Devero. This template included all of the CSS and HTML of this page. We then went through and changed all the hyperlinks and thumbnails to their respective JAMZ counterparts and added a link to the “Need Help?” page.

Need Help Page

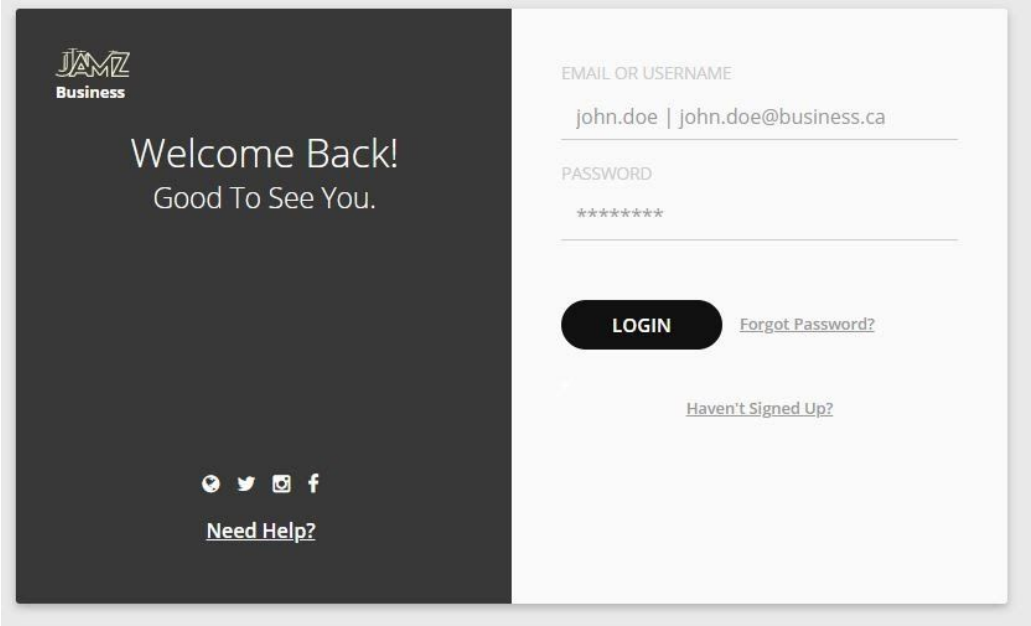
2.2 - Support page



This page allows the user to enter their email and open a service ticket with JAMZ support. The page was made by using a simple form div and 2 input tags with a textarea tag along with a submit button to send the form info. However the action of the form is blank as the actual service ticket function is not implemented yet. The styling is a modified version of the style sheet provided in the aforementioned template.

Login Page

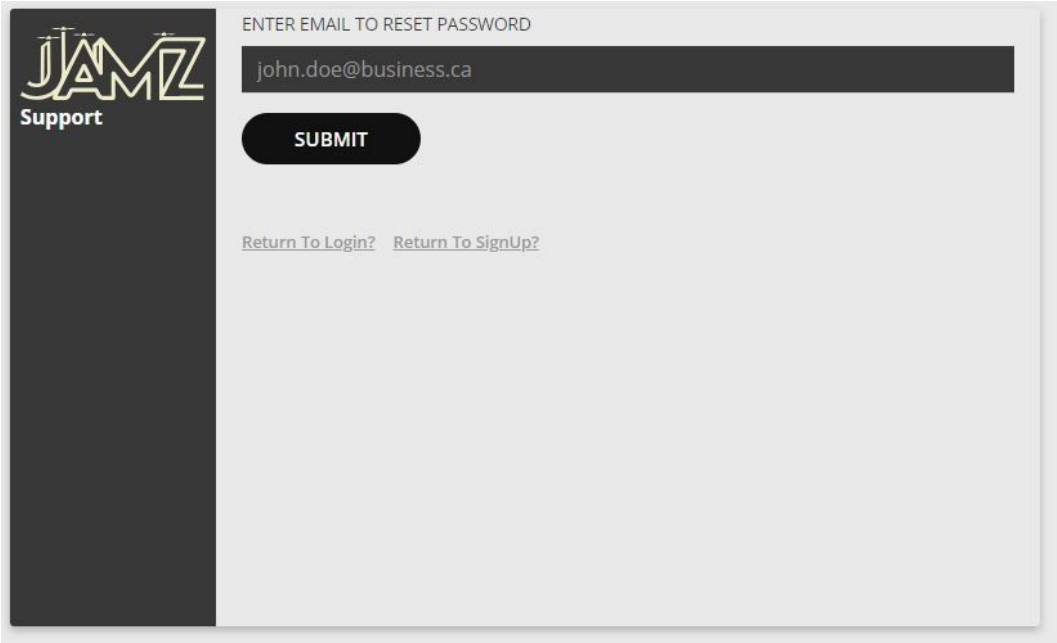
2.3 - Login page



The login page is fundamentally the same as the signup page however on this page the user can log into the app. This page differs from the signup page in that it has the “Forgot Password?” option which links the users to the password reset page below.

Password Reset

Figure 2.4 - Support Page



The password reset page page is based off of the Need Help page's styling and format with the only major difference is there is only the input field to upload a user's email address to which the reset password email will be sent. This page also doesn't have the password reset email feature implemented into the input form.

Figure 2.5 - First time setup

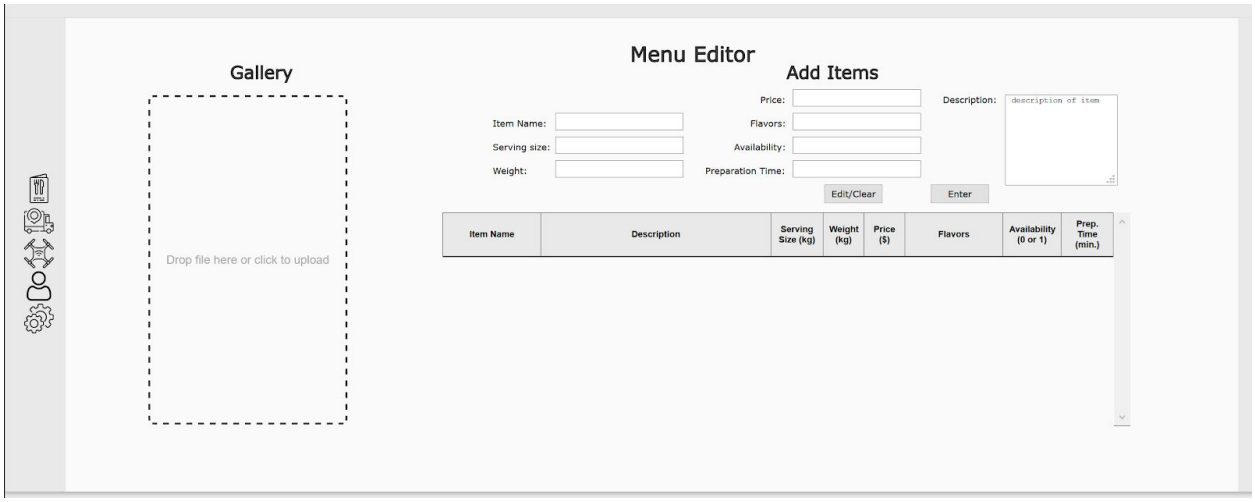
First Time Setup

The first time setup page was made using the same style of a floating box with black banner and the same general style sheet as all other Login/Signup pages. A simple form tag was used to collect the business information. The logo upload section is copied directly from the menu editor page. Finally the hours of operation feature was built using a table and the "time" tag for the time set functionality and a checkbox for the "Closed?" option. Storage of all this information has yet to be implemented into the application.

Menu Editor

The key features of this page are the input fields, the table for menu items and the drag-and-drop feature for pictures. Below is a figure of what the page looks like.

Figure 3.1 - Menu Editor



For the HTML code, we set up a few divs to better organize the structure of the page. We used a div for the buttons, text fields, drop-zone, sidebar and table. All these divs are inside one big div that holds all the contents for this page. Each text, input field, button, and the table itself will all be given an ID so we can give them function and style them on JavaScript and CSS respectively.

JavaScript - Menu

Many of the functions for buttons and other features are written in Javascript, starting with the three main buttons: “Enter”, “edit/clear”, and “remove”. We used the onclick attribute of each element to execute our Javascript functions.

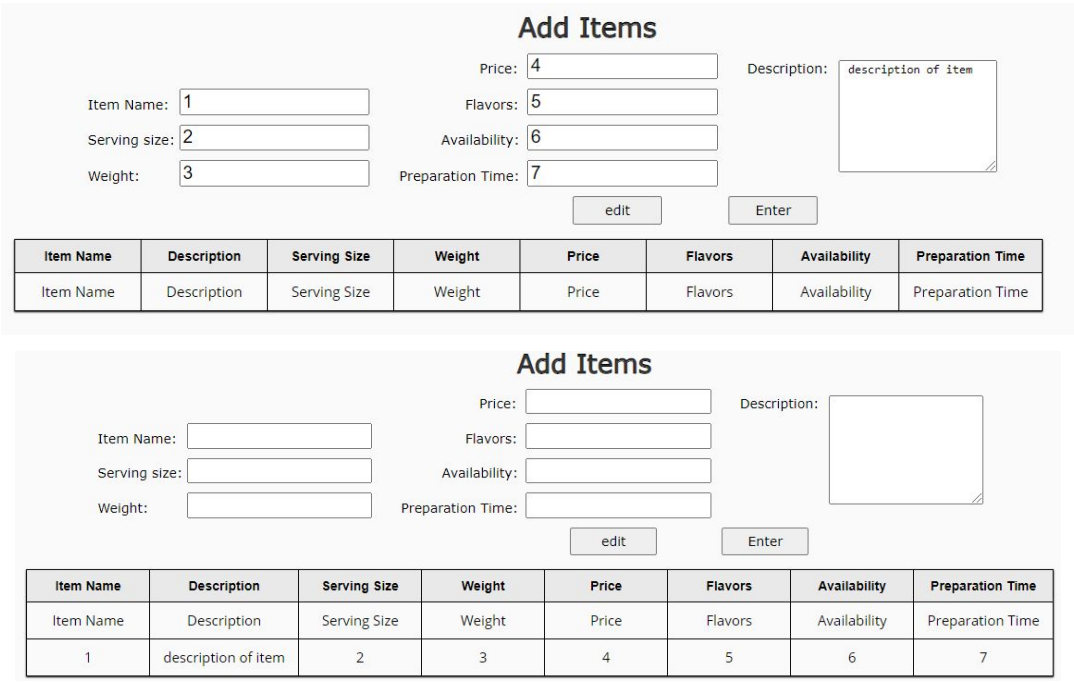
Enter Button

When the user presses the enter button, the program first checks whether any input fields are empty. All fields must have a value before the program adds a new row to the table of items. The function “checkEmptyInput()” helps check this case. Inside the function, we must assign variables to the id of each input field. This can be done using the getElementByID(“[nameoftextfield]”).value. The next step is to check whether that value is empty or not. This can be done using if statements. If the condition is true (empty input), an alert will be sent to the user.

If all the text fields are filled out and the user wishes to add the item to the table, the user simply has to click the enter button. For this to work, a function for when the enter button is pressed is created. In this function, we use an if statement to check whether any text fields are empty, and also if

the user is entering a duplicate row. Duplicate rows are checked by comparing the user's item name input with existing item names within the table. Once we've confirmed that the user input is not a duplicate, the program inserts a new cell in the table that contains the content entered by the user. If implemented correctly, the following figures show what the enter button should do for the user. The code for this feature can be found in functions "checkDuplicate()" and "myEnterButton.onclick = function (element)".

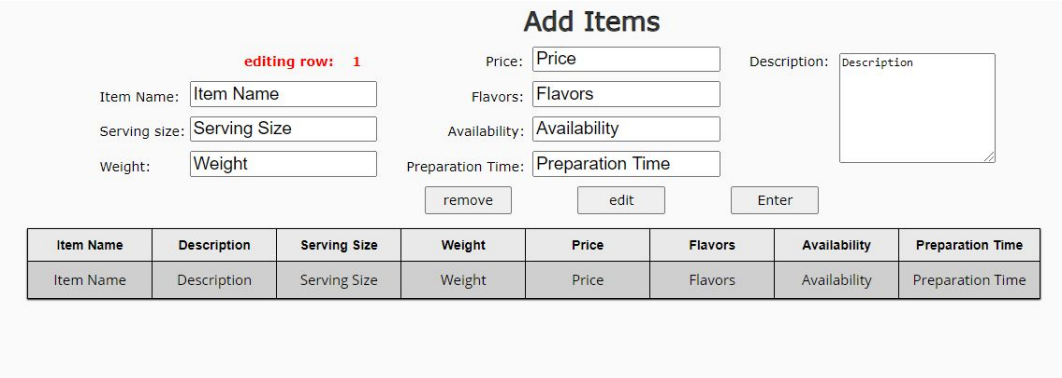
Figure 3.2 - Using the enter button



Edit Button

For the edit button to be effective, the user must select a row in the table to edit. For the program to know which row you want to edit, when the user clicks an item, those contents will appear back onto the input field. Some small red text will appear above the input fields which tells you which row you are editing. Below is a figure of this in action:

Figure 3.3 - Using the edit button



For the data to appear back into the text fields, a function to collect data from the rows must be created. The name of this function is “selectRow(row)”. Inside this function, the program takes the data in each cell and rewrites them back into the text fields.

Once the user clicks on a row and wishes to change the data, the user simply has to overwrite an input in the text field and then click edit. The program takes the newly written data and puts it into the row that was clicked. The code that performs this task can be found in the function “myEditButton.onclick = function (element)”

Remove button

The purpose of this button is to remove the row the user selects. The remove button is initially hidden from the user and only becomes visible when the user selects a row. This was done deliberately to fix a small bug that we encountered earlier. If there isn't any data in the table to begin with, and the user clicks remove, the program removes the header of the table which deletes the table entirely. A quick fix was to only show the remove button when a row is selected. This can be done by taking the “remove button” ID and setting it to “hidden” initially using CSS, and then changing it to “visible” whenever a row is clicked. Below is the code that resolves this issue. This line of code can be found in the “selectRow(row)” function.

FIgure 3.4 - short term solution to bug in remove button

```
187     document.getElementById("remove").style.visibility="visible";
```

Figure 3.5 - Before a row is selected

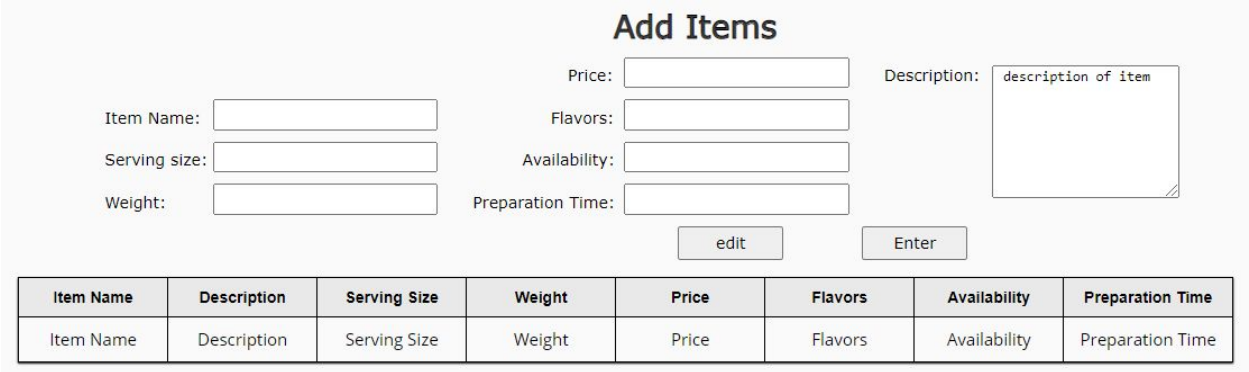
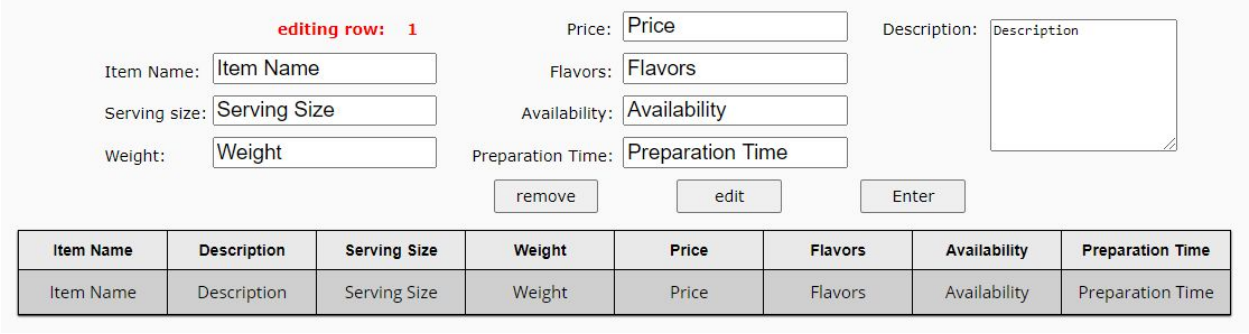
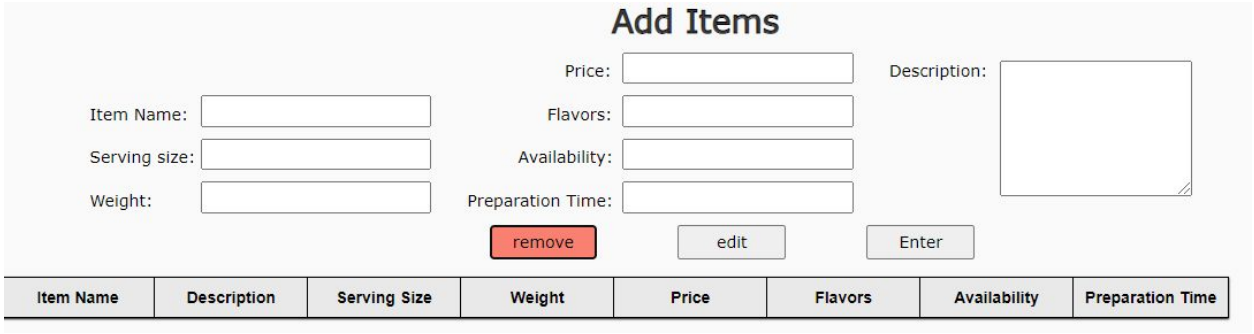


Figure 3.6 - After a row is selected



In the function “myRemoveButton.onclick = function (element)”, the program will remove the selecting row with the line of code “table.deleteRow(rIndex);”. Afterwards, the values inside the text fields are removed by setting their ID value to blank.

Figure 3.7 - After clicking remove



Drop-zone

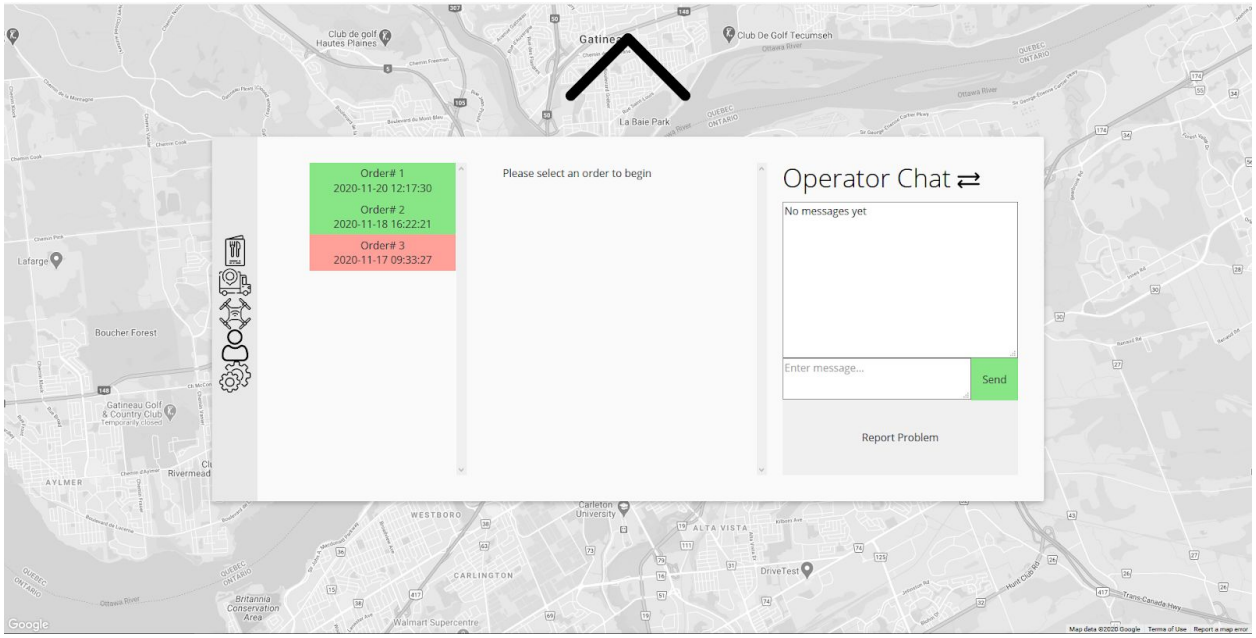
The code for the drop-zone feature was taken from a Youtube tutorial and can be found in the reference section where a link is provided for both the video and the code. Only minor changes in the positioning of the drop-zone were made to the code. The drop-zone color was changed and its area was increased to help remove blank space on the menu page. Such modifications were made in the “.drop-zone” class in the CSS code for the menu.

Order Manager

This part of the application allows the user to view orders and their contents, chat with customers and operators, view all drones, and mark orders as accepted, rejected, or shipped

Main Dialog

Figure 4.1 - Order manager



This page uses a combination of Javascript and PHP to interact with the SQL backend which contains all orders and menu items. It reads rows from the order table by indexing the orderID value in the SQL request and assuming that row exists. Once the orderID exceeds the number of rows in the table, the database will return no data which tells the Javascript that all orders have been read. This breaks sometimes when there is a jump in orderIDs but this should never happen since orders are kept forever in our database model.

The Javascript loads one order at a time and processes that order before going to the next one. Each order is loaded into a local global variable and parsed from there. The content of each order is returned in an array delimited by "/" with sub arrays delimited by ",". For example: "1/1,2,3/NO peanuts, extra fries, NA/2020-11-02 11:17:30/0" is a valid order format. Please refer to the SQL Database section for more details.

The colour of its label in the scrollable order list is set based on its orderStatus variable:

- **0** ⟶ Not yet accepted (shows up as RED)
- **1** ⟶ Accepted but not shipped (shows up as YELLOW)
- **2** ⟶ Shipped (shows up as GREEN)
- **Any other value** ⟶ Error or rejected order (shows up as GREY)

Its contents are then loaded based on a comma delimited list of itemIDs and a separate comma delimited list of item related options which are saved as plain text. The buttons in the bottom right of the main dialog are also adjusted based on the orderStatus variable:

- If orderStatus = 0 ⟶ buttons are accept/reject & problem
- If orderStatus = 1 ⟶ buttons are ship & problem
- Else ⟶ only problem button is shown

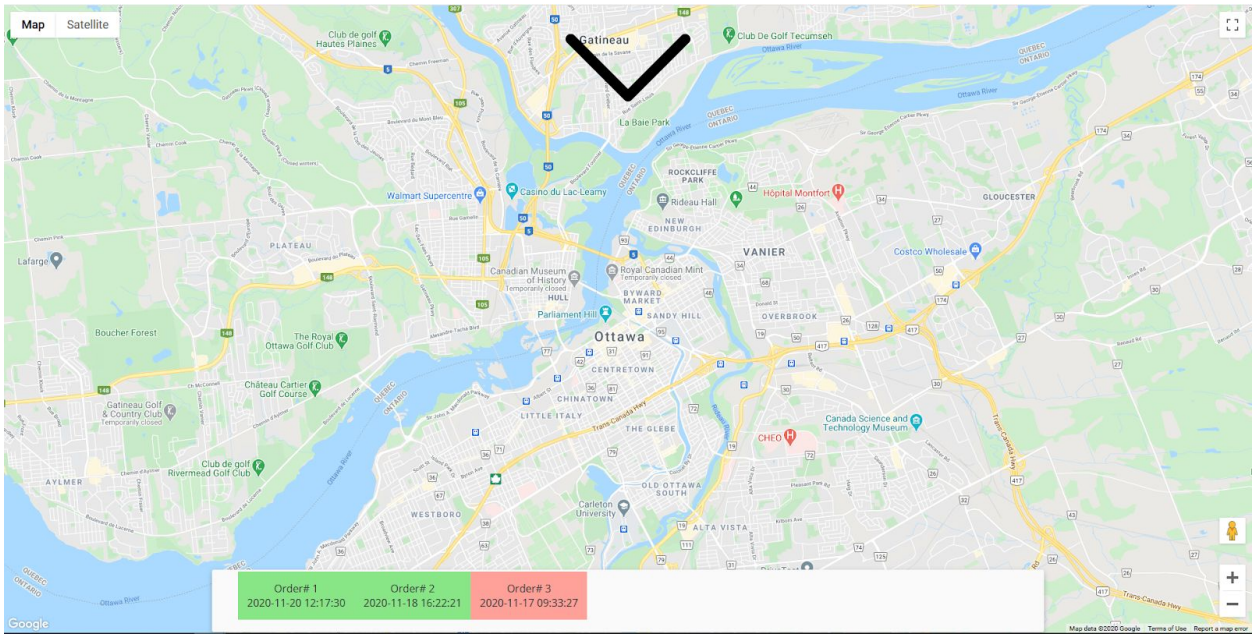
The orders are loaded as <li> elements in a list using Javascript's document.createElement() function. These <li> elements are then appended to two lists: one in the main dialog of the page, and a second in the secondary dialog that only appears when the main dialog is moved out of the way and the map is enabled. The Javascript loads both order lists at the same time using document.getElementsByClass() which generates an array of <ul> elements. It then uses nested loops to scan through the order database once for each dialog, and populates each dialog individually. In doing so, it is possible to vary how list items in each dialog are set up. For example, the "onclick" attribute can vary between the main dialog order list and the map dialog order list.

Once an order is selected, the Javascript uses PHP once again to read items from the menu database based on which itemIDs are saved in the orderContent array. These items are then loaded one by one into <li> elements alongside their relevant options and a checkbox. The checkbox is used to ensure that all items have been checked off before the user presses the ship button. For future integrations, it would be ideal to either set a static list of possible options for items or change the delimiter for the options array since the comma is a poor choice. Our menu table does have a "flav" column which was inserted for this purpose but a text option would also be ideal since some customers could have specific needs that the restaurant may not have thought of.

In terms of the chat, it's only functionality is to append any message the user sends to the existing text inside the big text box. It uses a <button> element to execute the sendMessage() function. In order to switch between chats, there is an icon containing two arrows to the right of the chat title. This icon has an "onclick" attribute which makes it execute the switchChat() function.

Map

Figure 4.2 - Map for drone and order tracking



The map can be activated by either clicking anywhere off the main dialog as well as clicking on the arrow above the main dialog. In order to return to the main dialog, the user must press on the arrow pointing down. The Google Maps API provides a few “stylers” in order to modify the aesthetics of the map. Using these “stylers”, the map has two different states:

- mapHide ⟶ no saturation, decreased gamma
- mapShow ⟶ 100% saturation, 100% gamma

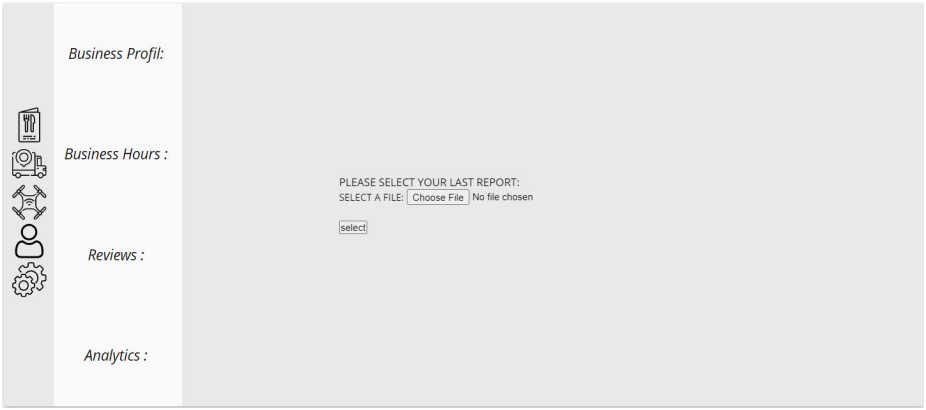
The mapHide state is active when the main dialog is in frame whereas the mapShow state is active when the main dialog has been moved out of the way. Map interaction is also disabled when the main dialog is visible.

The main dialog uses a Javascript based animation to slide up and out of the page while the map dialog uses the same code to slowly rise out of the bottom of the page. The animation is reversed to bring the main dialog back on top of the map and hide the map dialog. In order to synchronise their motion, the position of both dialogs is modified simultaneously since the shift required to hide/show the main dialog is a multiple of the shift required for the map dialog. No code has been added for drone markers on the map but this should not be hard to implement. The dialog at the bottom of the map is meant to allow the user to highlight a drone based on the order it’s transporting. Whenever orders are added to an order list, their “onclick” attribute is modified and that is how the user interacts with the orders.

Profile

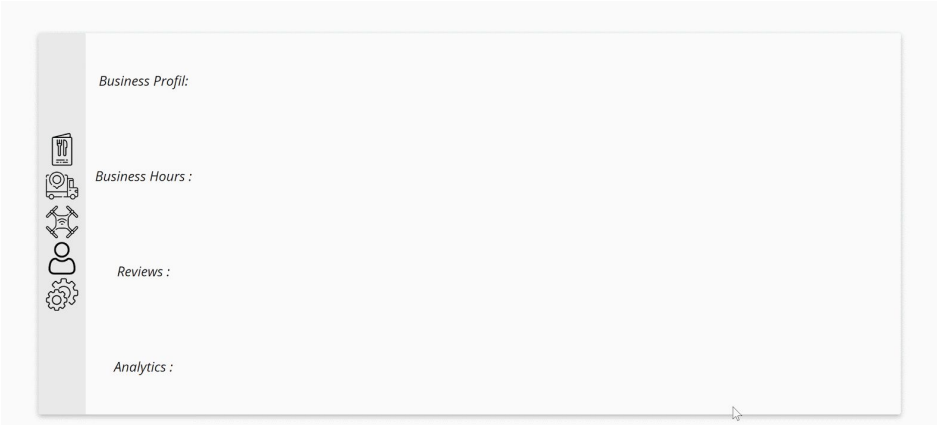
In this section of the application, the user can personalize their profile information, such as phone-number, business address, business email, and hours of operation. In addition, the user can also upload analytical reports. These subsections of the profile page are organized with another sidebar. The figure below is an image of this page.

Figure 5.1 - Profile page



The leftmost sidebar in grey is where the user can jump between other pages. To the right of that is the sidebar specific to the profile page. Each word is clickable and will direct the user to the correct subsection. The subsections are divided using HTML, by giving each section a <section id = " "> tag. For the sliding feature, this is done using the "animation: 2s fadeIn forwards .5s" line of code under "section:target h1" in CSS. A demonstration of this page's functionality is shown in the figure below.

Figure 5.2 - Customizing profile



Settings

Language translator

In the first prototype of the settings page the language translation feature was being powered by google translate...(from https://www.w3schools.com/howto/howto_google_translate.asp) and had the option of all the languages offered by the service, however after much deliberation, it was decided that it would be too much of a hassle to integrate the google translate function to every page along with fitting the function to the aesthetic of the application. Therefore, we decided that it would be in our best interest to just change the language of the text on the page manually and have the function switch out the pages.

Template control

This function would give the user the option to choose between a light mode and a darkmode, this was achieved by writing an id for a button that would change the background colour of the application when the button was clicked. The purpose of this function was to assist users who want a softer display on their screens.

Page Integration

Two key elements are in charge of integrating the pages together: the sidebar and the backend SQL database.

Sidebar

The sidebar allows the user to navigate between pages. It consists of an SVG icon for each page which was obtained from https://www.flaticon.com/. Each icon is within an <li> element of a list with its CSS position attribute set to “relative” which allows us to have them vertically aligned. Furthermore, the icons themselves are <img> objects within <a> objects which enables them to link to other pages. These icons are saved alongside the CSS for the sidebar menu in a folder called “misc” which is a central location for content shared between pages.

SQL Database

The SQL database was primarily integrated to allow the menu and order pages to function in tandem. It currently contains an order table as well as a menu table. Here are the structures of both tables:

Figure 6.1 - Database for orders

Order table structure

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	orderID	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial Fulltext Distinct values
2	orderContent	varchar(1024)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
3	orderOptions	varchar(1024)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
4	orderTime	datetime			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
5	orderStatus	int(11)			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values

1. **orderID**
    a. Automatically incremented integer which is solely relevant to the software, the user never sees or interacts with this value
2. **orderContent**
    a. Comma delimited array of itemIDs which make up an order (example: “1,3,4”)
3. **orderOptions**
    a. Comma delimited array of text strings (one string for each item) (example: “NO peanuts, extra fries, NA”
4. **orderTime**
    a. datetime format which specified when the order was received
5. **orderStatus**
    a. Integer between -1 and 2 unless error
        i. **-1** → order rejected
        ii. **0** → order unacknowledged
        iii. **1** → order accepted
        iv. **2** → order shipped

Figure 6.2 - Database for menu items

Menu table structure

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	itemID	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial Fulltext Distinct values
2	name	varchar(512)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
3	desc	varchar(1024)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
4	serv_size	double			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
5	weight	double			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
6	price	double			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
7	flav	varchar(1024)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
8	avail	int(11)			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
9	prep_time	double			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values

1. **itemID** → Automatically incremented integer which is solely relevant to the software, the user never sees or interacts with this value
2. **name** → 512 character string containing the name of the item
3. **desc** → 1024 character string containing the item description
4. **serv_size** → double variable containing weight of actual food w/o any container in kg
5. **weight** → double variable containing shipping weight of item in kg

6. **price** → double variable containing price in CAD
7. **flav** → comma delimited string containing options available for the item such as varieties of sauce or meat etc
8. **avail** → integer value, either 0 or 1 which determines if item is available for sale or hidden from customers
9. **prep_time** → double value which contains time required for item to be prepared in minutes

The data is collected from SQL tables using PHP to form and send an SQL request. This data is returned as a string with "/" to delimit between table columns and "," to delimit between elements of an array within a column. IMPORTANT: since "/" and "," are used as delimiters, they cannot be part of any data within a database element. It may be best to choose less common delimiters such as "\" and "|" instead.

Project schedule

The table below provides a rough estimate of the time needed to code and test each section of the application.

Time Spent Coding and Testing Each Section

Section	Time (hours)
login/sign-up	25
Menu editor	40
Order manager	40
Profile	30
Settings	25

Bill of Materials

Cost and Link of Each Software Used

Software used	Link	Fee ($CDN)
Notepad++	https://notepad-plus-plus.org/	0
LucidCharts	https://lucid.co/	0
UsbWebServer	https://www.usbwebserver.net/	0
HTML/CSS Template	https://codepen.io/alexdevero/pen/vNRqLL	0

SVG Icons	https://www.flaticon.com/	0 for us but more if the icons are actually being published

Future Improvements and Lessons Learnt

Database Integration

The database integration caused us a couple of issues throughout the project. Firstly, we began by developing our application with dummy data and therefore did not design it well for database integration. A lot of our work between prototype 2 and 3 was rewriting Javascript in order to leverage the functionality of our backend. Furthermore, the database request system currently uses AJAX which is not ideal since this causes a lot of async errors. AJAX stands for Asynchronous Javascript And XML where the asynchronous part is the root of our problems. Our use of AJAX means that the Javascript does not wait for the XML request to complete the data retrieval but instead keeps running. This becomes an issue when loading orders since Javascript tries to read from an empty variable.

Menu Page Drop-zone

Another issue caused by our tardy integration of the database is our picture drop-zone not being able to display individual pictures for every item. This would now be possible since we could include a column in the menu SQL table for picture URLs and download pictures from the user onto the server but we did not have time to build the downloading aspect of this integration. Javascript is executed on the client side of the website and can be used to prepare a picture to be uploaded to our server. We would then need a server side script ready to receive this picture. Finally, a few brief modifications could be done to the menu page so that an item's picture is loaded when its row is selected and some code which allows the user to select which picture corresponds to each item.

Button Animations

Little to no button animations were included in this project and it makes some buttons a little less than obvious. It would be ideal to add a "hover" attribute to the CSS of all our interactable fields since it would help the user know when they have moused over something they can click.

Order Shipping

No pages or popups other than a simple alert were included for shipping orders. This is despite the fact that we have the data available to compute both estimated preparation time and shipping weight which could have enabled us to inform the user as to which JAMZ drone should be used. It would

be ideal to include some kind of page that appears after the ship button is pressed in order to guide the user through the process of shipping the order.

Conclusion

In conclusion Our application is divided into various subsections that are navigated by a sidebar feature, with each subsection carrying out a specific function. The Login and Signup section prompting a new user to create their account and set up their business's basic information. As well as for returning users to login and use the app. In addition, the Menu Editor is a key feature allowing users to create highly customizable menus to give their customers ample choices in products. The main part of the application is the Order Manager. This section is the main interface between the user and the app as they use JAMZ's drone delivery service on a daily basis. It allows for quick and efficient order processing and fulfilling, crucial in the fast paced environment of a restaurant. Other features of the application include the Profile page that allows users to change their business's information and view financial analytics. Finally is the Settings section that allows for the language to be changed to either of Canada's two main languages. As well as change the color of the app from a light color scheme to a dark one. All in all, this application works as a seamless platform for businesses to communicate with their customers and fulfill their orders using JAMZ's innovative drone delivery system.

Appendix

References

https://codepen.io/dcode-software/pen/xxwpLQo	Code for drop-zone in menu
https://www.youtube.com/watch?v=Wtrin7C4b7w	Video for drop-zone in menu
https://www.flaticon.com/	Icons for sidebar
https://codepen.io/alexdevero/pen/vNRqLL	HTML/CSS template
https://notepad-plus-plus.org/	Coding software
https://lucid.co/	Software for sketches
https://www.usbwebserver.net/	Link to server
https://github.com/Serpopovici163/GNG1103Website	Code on Github