National College of Ireland

BSHC

Software Development

2022/2023

Lee Harold

X19429942

X19429942@student.ncirl.ie

“Finale: Reserve/Order App"

Technical Report

Contents

1.0 Introduction

1.1. Background

As I identified a need for an application that has all of the necessary capabilities in one location rather than across multiple sites or external sources, I decided to take on the challenge of developing a reservation, order, and food delivery combination app. I recognized an opportunity to develop an application that would make it simple for people to book reservations, place advance orders, or have food delivered to their home in a convenient and efficient manner with the exponential growth in popularity of food delivery apps. Overall, I viewed the challenge of developing this app as an interesting and satisfying chance to combine the knowledge I gained through my module and independent research into an app that I believe will be helpful to a variety of users in today's busy society.

1.2. Aims

With my app, I want to provide users a quick and easy way to get meals from their preferred restaurants, whether they want to order fast food for collection or delivery or make reservations at sit-down restaurants. From quick and convenient fast food to dining in restaurants, I want to make it simple for users to access a variety of dining options that meet their needs. In the end, I want to design a simple app to book reservations, order meals for pickup, or have food delivered. I created this project with cash still in mind, as many people are not happy moving into a purely contactless or online-payment based society. I wanted to still give people the option to be able to pay with cash when the delivery arrives or when they collect their order.

1.3. Technology

The technologies I will be using in my project are Django, JavaScript and SQLite3. Django is a Python-based framework and came as familiar ground to me due to previous projects and experience with the coding language side. JavaScript is one of the most popular code technologies across the internet, known for its visuals and event-handling traits. I also used jQuery which a JavaScript library for assisting and improving both front and back end functionality, as well as using Ajax. Ajax is a client-side based group of developmental techniques for various web applications for asynchronous performance. It can retrieve data without adjusting the behaviour or visuals of a page. JavaScript and its technologies can assist massively in making a user's experience visually and practically pleasing. Finally, we used SQLite3, which is an external, encrypted database file for small-scale data handling. This was the SQL side of my project. This was very useful as it meant that any additions or removal were done through Django directly and could be checked in the app as admin or in the command line using the sqlite3 file.

1.4. Structure

Firstly the system requirements will be discussed, such as functional requirements and their use cases, with their description, priority and respective main, alternate and exceptional flows. These functional requirements will also include their preconditions, terminations and post conditions. Data, User, Environmental and Usability requirements will also be explained. Design and Architecture of the system will follow, with implementation of technologies after, before moving on to the key Graphical User Interfaces (GUIs) of the project and finally finishing the system section of the report with testing and evaluations.

With the system section concluded, the report will be rounded off with my conclusions around the project, potential future of the project if additional time and resources were available, and finally the references and appendices of the report, including my project proposal and monthly reflections.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

1. Admin should be able to login.
2. Admin should be able to see all users.
3. Admin should be able to create, update and delete users.
4. Businesses should be able to sign-in.
5. Businesses should be able to view orders.
6. Users should be able to see menus and their details.
7. Users should be able to see restaurants and their details.
8. Users should be able to see total cost of order.
9. Users should be able to book reservations.
10. Users should be able to place orders for deliveries.
11. Users should be able to pre-order their food.
12. Users should be able to create accounts.
13. Users should be able to sign-in to their accounts.

Requirement 1 – Admin should be able to login.

2.1.1.1. Description & Priority

Priority 1 – Admin must have control of the site.

2.1.1.2. Use Case

Scope: The scope of this use case is to describe the process for the admin/superuser to sign in.

Description: This use case describes the steps that the admin takes to sign into the admin side of the application

Flow Description: Precondition: Admin account is created but not logged in.

Activation: The admin goes to the admin page.

Main flow:

- The app displays a login page.
- The admin attempts to sign in with the correct credentials.
- The admin gains access to the admin pages and privileges.

Alternate flow:

If credentials are incorrect, page is refreshed and error message is displayed.

Exceptional flow:

If there is a problem signing in and error message will be displayed.

Termination:

The admin is signed in and on the admin page with access to admin functionality.

Post Condition

The system goes into a wait state

Requirement 2– Admin should be able to see all users.

2.1.1.3. Description & Priority

Priority 2 – Admin must be able to see all users.

2.1.1.4. Use Case

Scope: The scope of this use case is to describe the process for the admin to see all users

Description: This use case describes the steps that the admin takes to view all users

Flow Description:

Precondition: Admin is logged in and users table has been created.

Activation: The admin goes to Users table.

Main flow:

- The admin clicks the user's table while on the admin side of the application.
- The page loads and the users are visible.

Alternate flow:

If the admin is signed out during this process they will be redirected to the admin login page

Exceptional flow:

If there is a problem displaying the table and error will be shown.

Termination:

The admin is viewing the users table

Post Condition

The system goes into a wait state

Requirement 3– Admin should be able to create and delete users.

2.1.1.5. Description & Priority

Priority 3 – Admin must be able to create, update and delete users.

2.1.1.6. Use Case

Scope: The scope of this use case is to describe the process for the admin to create, update or delete users.

<u>Description:</u> This use case describes the steps that the admin takes create, save, or update users.

<u>Flow Description:</u>

Precondition: Admin is logged in and users table has been created.

<u>Activation</u>: The admin goes to Users table and clicks a specific user, group of users or the add user option.

<u>Main flow:</u>

Create:

- The admin clicks the add user.
- The admin fills the details.
- The admin clicks save.

Update:

- The admin clicks an existing user.
- The admin changes a detail.
- The admin clicks save.

Delete:

- The admin checks an existing user.
- The admin clicks the dropdown to delete a user.
- The admin clicks go.

<u>Alternate flow:</u>

Create: If the admin does not fill the fields correctly an error will be shown and no user is created.

Update: If the admin does not update a field in the correct format, a user will not be updated and an error message will be shown.

<u>Exceptional flow:</u>

If there is a problem with any of the actions, nothing will be changed.

<u>Termination:</u>

A user is created.

A user’s details are updated.

A user(s) is deleted.

<u>Post Condition</u>

The system goes into a wait state

Requirement 4 – Businesses should be able to sign-in.

2.1.1.7. Description & Priority

Priority 4 – Businesses must be able to sign in.

2.1.1.8. Use Case

Scope: The scope of this use case is to describe the process for a business signing in.

Description: This use case describes the steps that a business takes to sign in.

Flow Description:

Precondition: Business is not logged in and has a valid account already created.

Activation: The business accesses the login page

Main flow:

- The business fill in their details
- The business clicks the sign in button.

Alternate flow:

If credentials are incorrect, page is reloaded.

Exceptional flow:

If there is a problem signing in an error should be shown.

Termination:

The business is signed in.

Post Condition

The system goes into a wait state

Requirement 5 – Businesses should be able to view orders

2.1.1.9. Description & Priority

Priority 5 – Businesses must be able to view their orders (or reservations).

2.1.1.10. Use Case

Scope: The scope of this use case is to describe the process for a business viewing their orders (or reservations).

Description: This use case describes the steps that a business takes to view orders.

Flow Description:

Precondition: Business is logged in and an order has been placed.

Activation: The business accesses the Manage Collections/Deliveries/Bookings in the NavBar

Main flow:

- The business clicks on the page.
- The table is shown.

Alternate flow:

If not the correct business, no table will be shown.

Exceptional flow:

If there is a problem with the table, nothing can be seen.

Termination:

The business is viewing a table of reservations/deliveries/collections.

Post Condition

The system goes into a wait state

Requirement 6 – Users should be able to see menus and their details.

2.1.1.11. Description & Priority

Priority 6 – Users must be able to see the menu and details.

2.1.1.12. Use Case

Scope: The scope of this use case is to describe the process for a user viewing a restaurants menus and details.

Description: This use case describes the steps that a user takes to see menus and their details.

Flow Description:

Precondition: User is logged in and tries to access the services pages.

Activation: The user visits one of the services pages.

Main flow:

- The user clicks a service page.
- The menu loads and is visible with the details of the order, price and different menu items.

Alternate flow:

If user is not signed in, the page will be inaccessible and

Exceptional flow:

If there is a problem with the table, nothing can be seen.

Termination:

The user is viewing the menu and details.

Post Condition

The system goes into a wait state

Requirement 7 – Users should be able to see restaurants and their details.

2.1.1.13. Description & Priority

Priority 7 – Users must be able to see restaurants and their details.

2.1.1.14. Use Case

Scope: The scope of this use case is to describe the process for a user viewing a restaurant and its details.

Description: This use case describes the steps that a user takes to see a restaurant and its details.

Flow Description:

Precondition: User is logged in and tries to access the page before placing an order/booking.

Activation: The user visits one of the services pages without clicking the place order/booking.

Main flow:

- The page loads and the restaurant and details are shown to user.

Alternate flow:

N/A

Exceptional flow:

If there is a problem with the page and error should be shown.

Termination:

The user is viewing the restaurant and details.

Post Condition

The system goes into a wait state

Requirement 8 – Users should be able to see total cost of order

2.1.1.15. Description & Priority

Priority 6 – Users must be able to see total cost of order

2.1.1.16. Use Case

Scope: The scope of this use case is to describe the process for a user seeing the total cost of their order.

Description: This use case describes the steps that a user sees their total cost of an order.

Flow Description:

Precondition: User is logged in and tries to place an order.

Activation: The user visits one of the services pages.

Main flow:

- The page loads and the restaurant and details are shown to user.
- The user chooses their options and adds to their cart/order.
- Total cost is updated.

Alternate flow:

N/A

Exceptional flow:

If there is a problem with JavaScript, the order function will not work correctly.

Termination:

The user can see their total before placing the order.

Post Condition

The system goes into a wait state

2.1.2. Data Requirements

1. User information: The app must keep user data such as names, email addresses, phone numbers, and payment information.
2. Order information: The app will need to keep track of order details, such as the items requested, their quantities, total costs, and delivery addresses.
3. Information on the restaurants: The app will need to keep records of the menus and restaurants that offer delivery services.

2.1.3 User Requirements

1. Usability: The app should be simple to use and allow users to quickly find what they're looking for.
2. Speed: Users may be pressed for time to place an order or make a reservation, so the app needs to respond fast and load.

3. Accuracy: The app needs to accurately display information regarding menus, prices, and availability.
4. Reliability: Users will rely on the app to place orders and make appointments, so it needs to function reliably and consistently.

2.1.4 Environmental Requirements

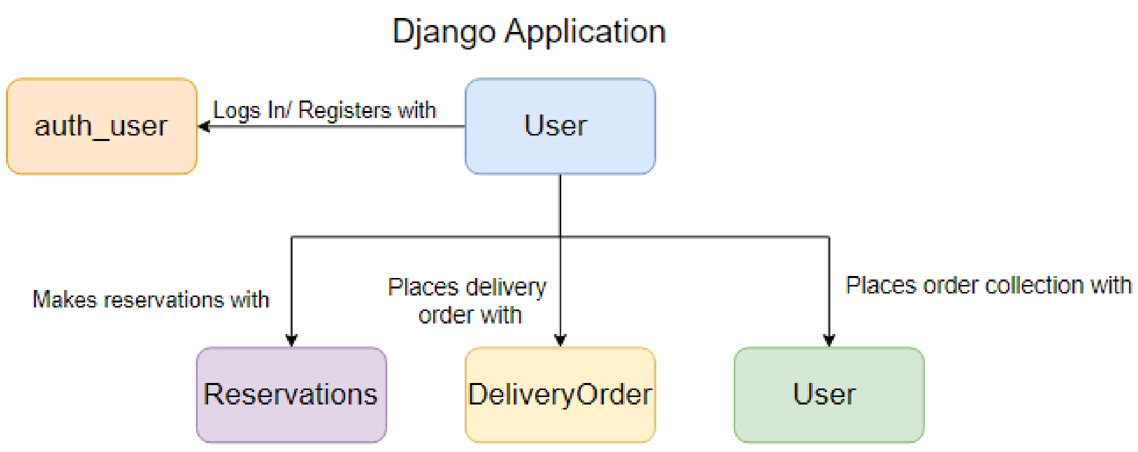
1. Device compatibility: The app should be compatible with a range of devices, including smartphones and tablets and computers.
2. Operating system compatibility: The app should be compatible with the most popular operating systems, such as iOS and Android, in order to reach a broad audience.
3. Performance: There shouldn't be any concerns with the app's performance when handling significant traffic and big amounts of data.

2.1.5 Usability Requirements

1. Navigation that is intuitive: The software should be simple to use, with a logical structure and labelling for buttons and other controls that are straightforward to understand.
2. Design that is responsive: The app must be adaptable to user input and operate without glitches across a variety of platforms.
3. User-friendly interface: The app should have an obvious, visually appealing interface that is simple for people to use.
4. Consistency: The app's appearance and functionality should be consistent to make it simple for consumers to understand and utilise.

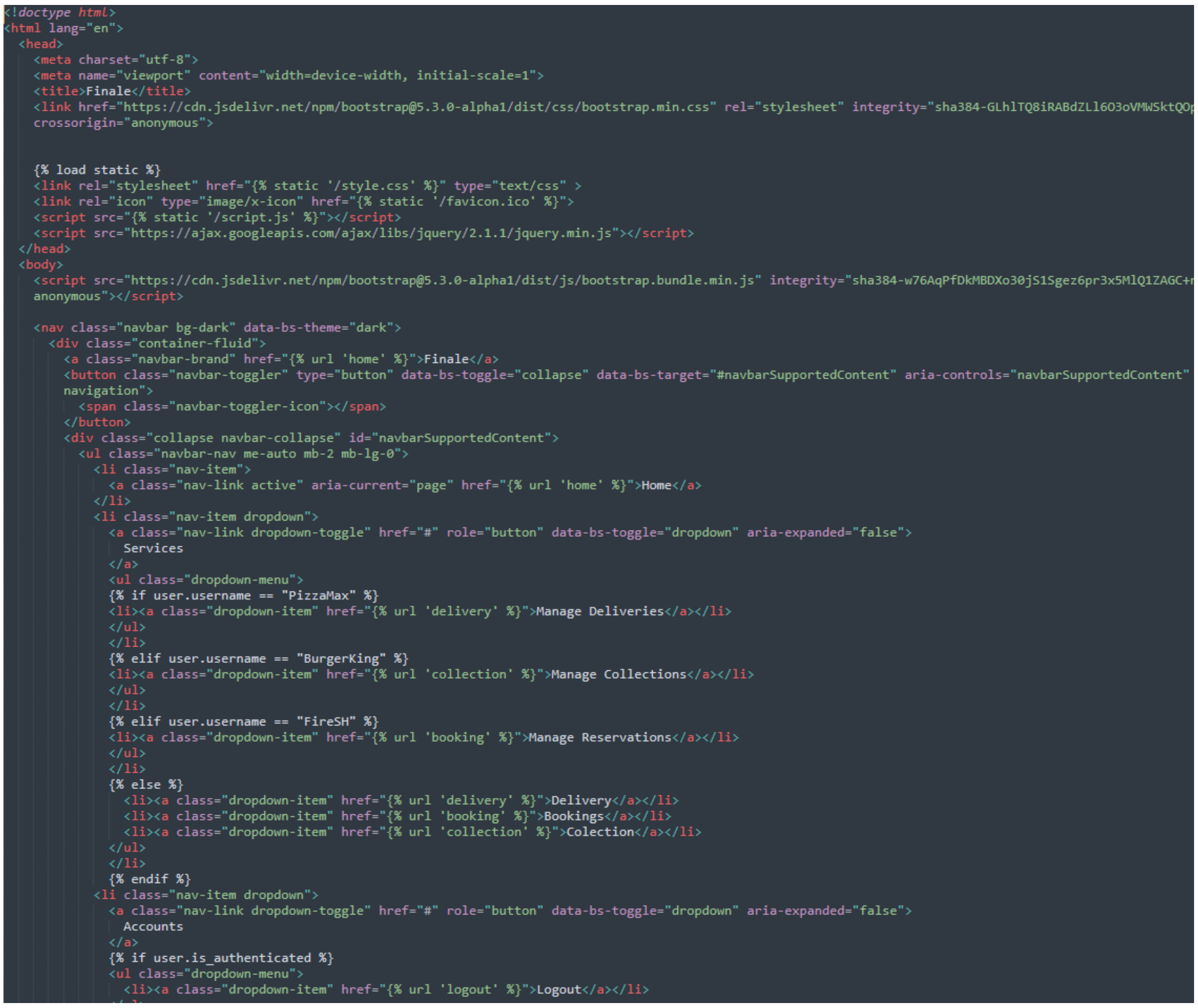
2.2 Design & Architecture

High-Level ERD Diagram with respective forms/table names:



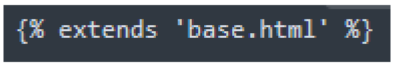
2.3 Implementation

In order to save unnecessary and link externa files efficiently I used a base.html that contains everything that each page needs and simply extends it to every page. This includes the Navbar, if statements for access control, the icon for the site and any necessary external files linked:



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Finale</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp
    crossorigin="anonymous">


    {% load static %}
    <link rel="stylesheet" href="{% static '/style.css' %}" type="text/css" >
    <link rel="icon" type="image/x-icon" href="{% static '/favicon.ico' %}">
    <script src="{% static '/script.js' %}"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
  </head>
  <body>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+n
    anonymous"></script>

    <nav class="navbar bg-dark" data-bs-theme="dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'home' %}">Finale</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
        navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="{% url 'home' %}">Home</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                Services
              </a>
              <ul class="dropdown-menu">
              {% if user.username == "PizzaMax" %}
              <li><a class="dropdown-item" href="{% url 'delivery' %}">Manage Deliveries</a></li>
              </ul>
              </li>
              {% elif user.username == "BurgerKing" %}
              <li><a class="dropdown-item" href="{% url 'collection' %}">Manage Collections</a></li>
              </ul>
              </li>
              {% elif user.username == "FireSH" %}
              <li><a class="dropdown-item" href="{% url 'booking' %}">Manage Reservations</a></li>
              </ul>
              </li>
              {% else %}
                <li><a class="dropdown-item" href="{% url 'delivery' %}">Delivery</a></li>
                <li><a class="dropdown-item" href="{% url 'booking' %}">Bookings</a></li>
                <li><a class="dropdown-item" href="{% url 'collection' %}">Colection</a></li>
              </ul>
              </li>
              {% endif %}
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                Accounts
              </a>
              {% if user.is_authenticated %}
              <ul class="dropdown-menu">
                <li><a class="dropdown-item" href="{% url 'logout' %}">Logout</a></li>
```

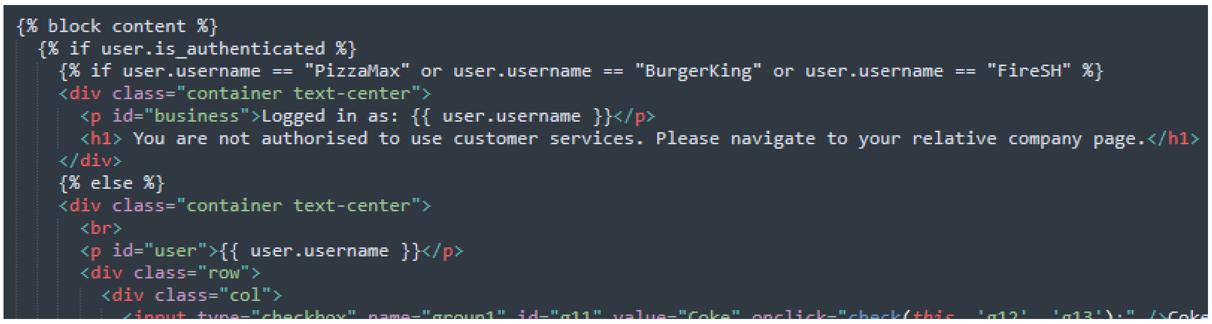
All of that code is simply imported and reused through an extends command:



```
{% extends 'base.html' %}
```

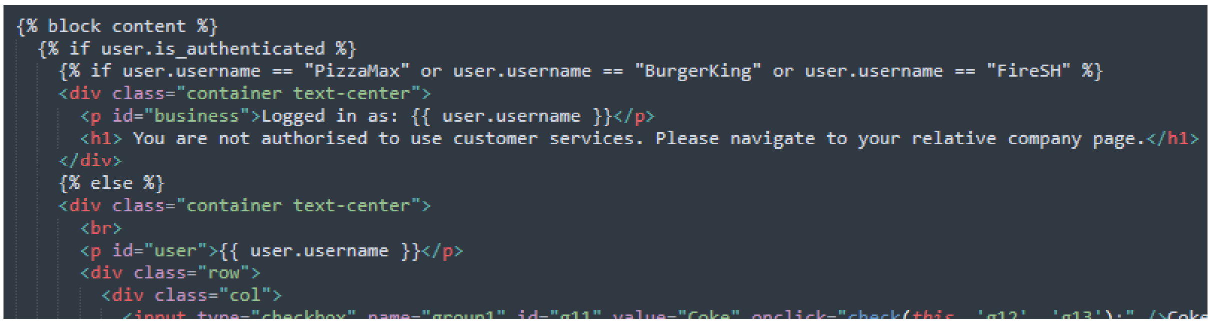
Any page with that present will inherit every external file, if statement, bootstrap CSS and Navbar seamlessly without having to link or re-type everything across each file. The main body of each page is wrapped around by a {% block content %} and {% endblock %}. Anything outside of those block statements will be generated from the extended base.html.

Access Control was done through generic and specified if statements. For example, I used Django's built in login check to check if the user was signed in, then also checked the username, if it was a business it was locked out of the services pages:

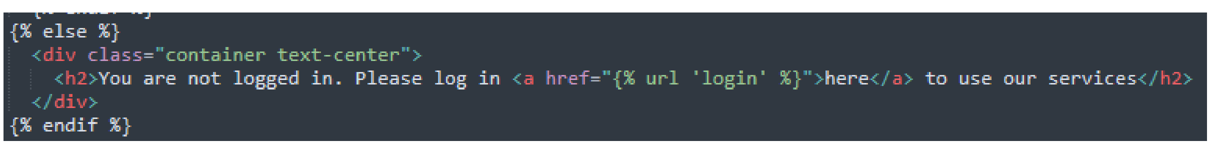


```
{% block content %}
  {% if user.is_authenticated %}
    {% if user.username == "PizzaMax" or user.username == "BurgerKing" or user.username == "FireSH" %}
    <div class="container text-center">
      <p id="business">Logged in as: {{ user.username }}</p>
      <h1> You are not authorised to use customer services. Please navigate to your relative company page.</h1>
    </div>
    {% else %}
    <div class="container text-center">
      <br>
      <p id="user">{{ user.username }}</p>
      <div class="row">
        <div class="col">
          <input type="checkbox" name="group1" id="g11" value="Coke" onclick="check(this, 'g12', 'g13');" />Coke
```

If logged in as a business, access message is displayed, if not logged in, separate access message is displayed, and if logged in and not a business, regular access is granted:



```
{% block content %}
  {% if user.is_authenticated %}
    {% if user.username == "PizzaMax" or user.username == "BurgerKing" or user.username == "FireSH" %}
    <div class="container text-center">
      <p id="business">Logged in as: {{ user.username }}</p>
      <h1> You are not authorised to use customer services. Please navigate to your relative company page.</h1>
    </div>
    {% else %}
    <div class="container text-center">
      <br>
      <p id="user">{{ user.username }}</p>
      <div class="row">
        <div class="col">
          <input type="checkbox" name="group1" id="g11" value="Coke" onclick="check(this, 'g12', 'g13');" />Coke
```

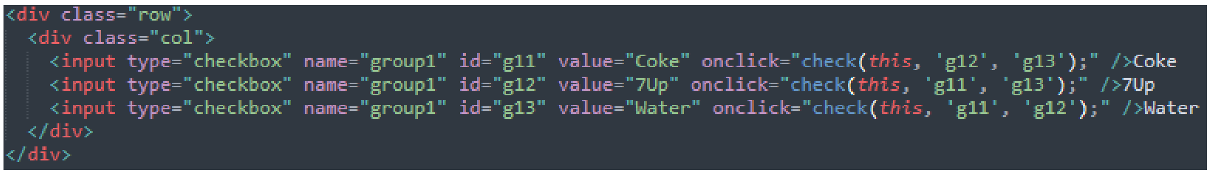


```
  {% endif %}
{% else %}
  <div class="container text-center">
    <h2>You are not logged in. Please log in <a href="{% url 'login' %}">here</a> to use our services</h2>
  </div>
{% endif %}
```

To get the order and price from the customer I used JavaScript and jQuery for responsive interaction for menu items and their options. I did this through button groups that uncheck themselves if a button is pressed or an item added to the order.

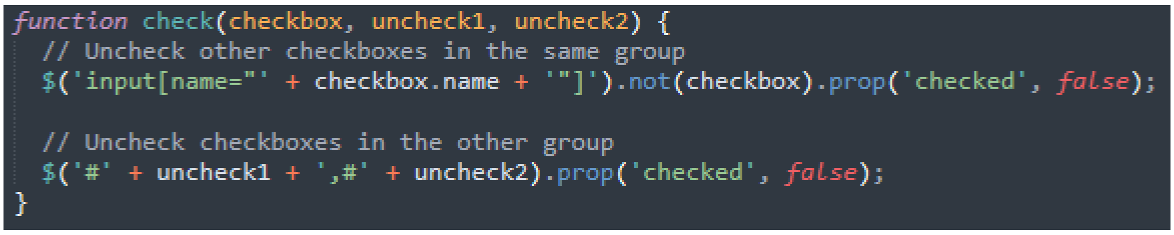
Both delivery and collection use the same functions and buttons.

The checkboxes:



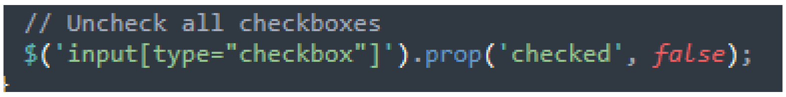
```
<div class="row">
  <div class="col">
    <input type="checkbox" name="group1" id="g11" value="Coke" onclick="check(this, 'g12', 'g13');" />Coke
    <input type="checkbox" name="group1" id="g12" value="7Up" onclick="check(this, 'g11', 'g13');" />7Up
    <input type="checkbox" name="group1" id="g13" value="Water" onclick="check(this, 'g11', 'g12');" />Water
  </div>
</div>
```

The one-at-a-time selection:



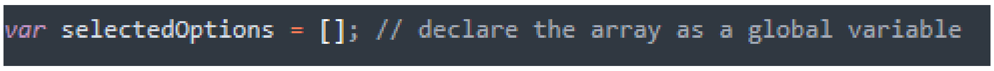
```
function check(checkbox, uncheck1, uncheck2) {
  // Uncheck other checkboxes in the same group
  $('input[name="' + checkbox.name + '"]').not(checkbox).prop('checked', false);

  // Uncheck checkboxes in the other group
  $('#' + uncheck1 + ',#' + uncheck2).prop('checked', false);
}
```

Uncheck when a button is clicked:

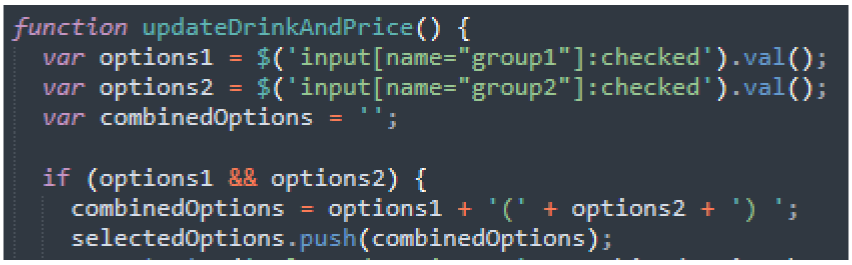


```
  // Uncheck all checkboxes
  $('input[type="checkbox"]').prop('checked', false);
}
```

For getting the order and price from the checkboxes we create an empty array and send the item values as strings. So Coke is an item, 330ml is an option; Coke(330ml) will be added as the item to the order. The .push pushes the items to the selectedOptions array is named combinedOptions:

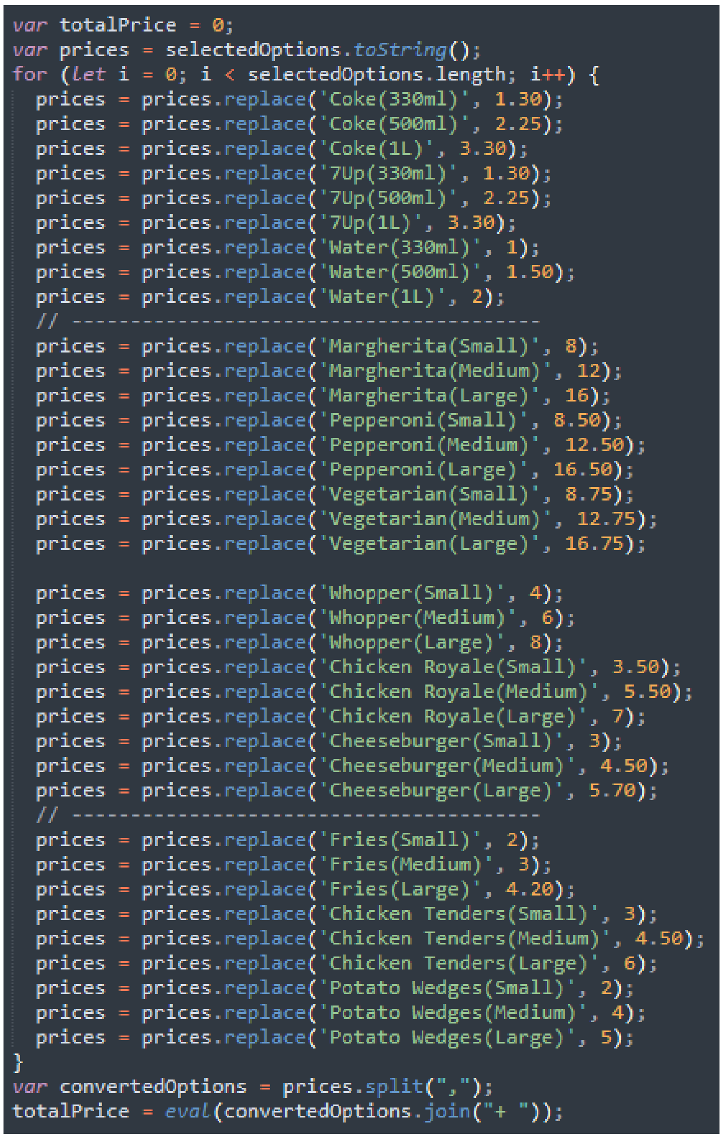


```
var selectedOptions = []; // declare the array as a global variable
```



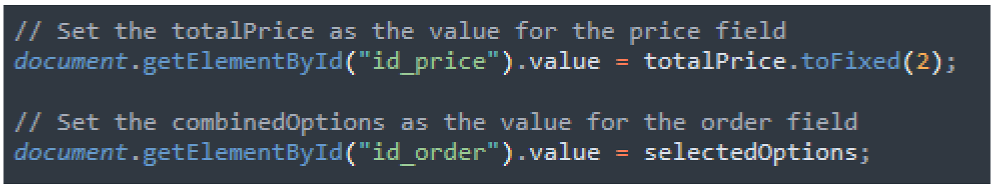
```
function updateDrinkAndPrice() {
  var options1 = $('input[name="group1"]:checked').val();
  var options2 = $('input[name="group2"]:checked').val();
  var combinedOptions = '';

  if (options1 && options2) {
    combinedOptions = options1 + '(' + options2 + ') ';
    selectedOptions.push(combinedOptions);
```

To get the price I convert the selectedOptions array into a string, replace the items with their price, convert back into an array, joining the items with a + instead of , and run the array like a sum, one price + the next and so on until all items are calculated:



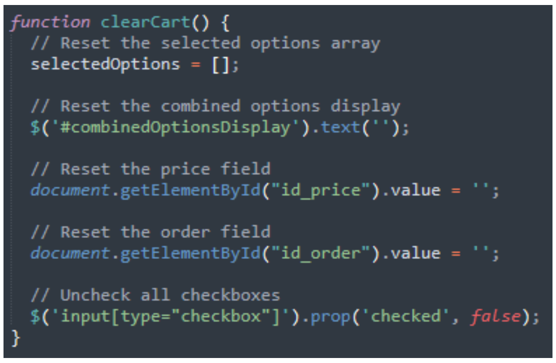
```
var totalPrice = 0;
var prices = selectedOptions.toString();
for (let i = 0; i < selectedOptions.length; i++) {
  prices = prices.replace('Coke(330ml)', 1.30);
  prices = prices.replace('Coke(500ml)', 2.25);
  prices = prices.replace('Coke(1L)', 3.30);
  prices = prices.replace('7Up(330ml)', 1.30);
  prices = prices.replace('7Up(500ml)', 2.25);
  prices = prices.replace('7Up(1L)', 3.30);
  prices = prices.replace('Water(330ml)', 1);
  prices = prices.replace('Water(500ml)', 1.50);
  prices = prices.replace('Water(1L)', 2);
  // ----------------------------------------
  prices = prices.replace('Margherita(Small)', 8);
  prices = prices.replace('Margherita(Medium)', 12);
  prices = prices.replace('Margherita(Large)', 16);
  prices = prices.replace('Pepperoni(Small)', 8.50);
  prices = prices.replace('Pepperoni(Medium)', 12.50);
  prices = prices.replace('Pepperoni(Large)', 16.50);
  prices = prices.replace('Vegetarian(Small)', 8.75);
  prices = prices.replace('Vegetarian(Medium)', 12.75);
  prices = prices.replace('Vegetarian(Large)', 16.75);

  prices = prices.replace('Whopper(Small)', 4);
  prices = prices.replace('Whopper(Medium)', 6);
  prices = prices.replace('Whopper(Large)', 8);
  prices = prices.replace('Chicken Royale(Small)', 3.50);
  prices = prices.replace('Chicken Royale(Medium)', 5.50);
  prices = prices.replace('Chicken Royale(Large)', 7);
  prices = prices.replace('Cheeseburger(Small)', 3);
  prices = prices.replace('Cheeseburger(Medium)', 4.50);
  prices = prices.replace('Cheeseburger(Large)', 5.70);
  // ----------------------------------------
  prices = prices.replace('Fries(Small)', 2);
  prices = prices.replace('Fries(Medium)', 3);
  prices = prices.replace('Fries(Large)', 4.20);
  prices = prices.replace('Chicken Tenders(Small)', 3);
  prices = prices.replace('Chicken Tenders(Medium)', 4.50);
  prices = prices.replace('Chicken Tenders(Large)', 6);
  prices = prices.replace('Potato Wedges(Small)', 2);
  prices = prices.replace('Potato Wedges(Medium)', 4);
  prices = prices.replace('Potato Wedges(Large)', 5);
}
var convertedOptions = prices.split(",");
totalPrice = eval(convertedOptions.join("+ "));
```

For seeing the price and order of an order and displaying to the user I used JavaScript the directly target the form fields and hard-set the value after the functionality of the button takes place. I target the form fields by using getElementByID of the fields and hard-setting the value to the results of the function



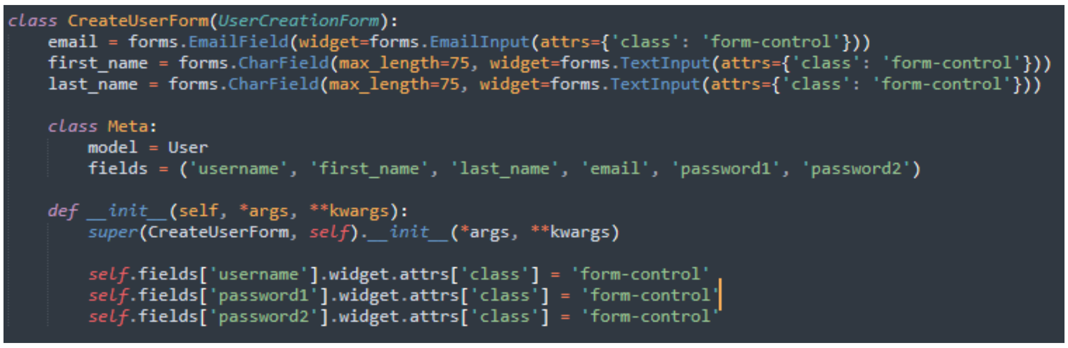
```
// Set the totalPrice as the value for the price field
document.getElementById("id_price").value = totalPrice.toFixed(2);

// Set the combinedOptions as the value for the order field
document.getElementById("id_order").value = selectedOptions;
```

User can empty their cart at any time through the clearCart function. It resets the array and variables, the fields and unchecks any checked boxes:



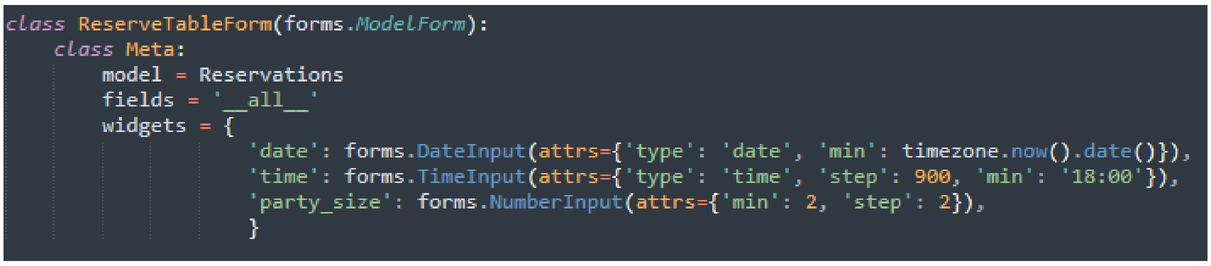
```
function clearCart() {
  // Reset the selected options array
  selectedOptions = [];

  // Reset the combined options display
  $('#combinedOptionsDisplay').text('');

  // Reset the price field
  document.getElementById("id_price").value = '';

  // Reset the order field
  document.getElementById("id_order").value = '';

  // Uncheck all checkboxes
  $('input[type="checkbox"]').prop('checked', false);
}
```

To limit forms and only show what's needed you can specify the fields shown, any arguments needed, special attributes for a field and add extra functionality to a form As can be seen with CreateUserForm I made sure first and last name had a set max-length and that that fields are specified as email format:



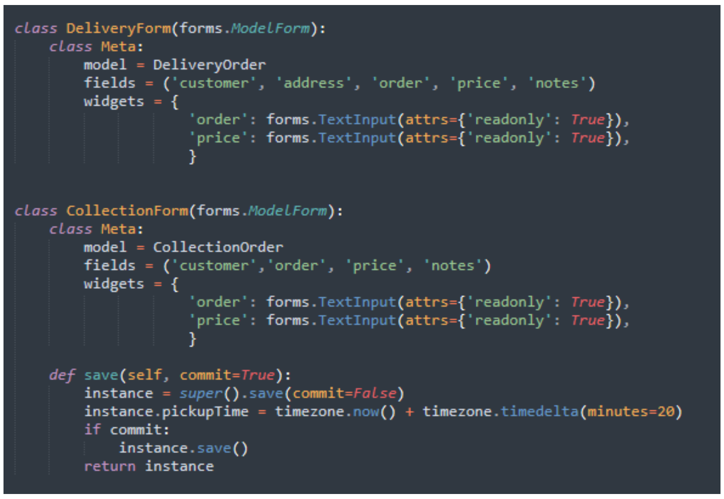
```
class CreateUserForm(UserCreationForm):
    email = forms.EmailField(widget=forms.EmailInput(attrs={'class': 'form-control'}))
    first_name = forms.CharField(max_length=75, widget=forms.TextInput(attrs={'class': 'form-control'}))
    last_name = forms.CharField(max_length=75, widget=forms.TextInput(attrs={'class': 'form-control'}))

    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', 'email', 'password1', 'password2')

    def __init__(self, *args, **kwargs):
        super(CreateUserForm, self).__init__(*args, **kwargs)

        self.fields['username'].widget.attrs['class'] = 'form-control'
        self.fields['password1'].widget.attrs['class'] = 'form-control'
        self.fields['password2'].widget.attrs['class'] = 'form-control'
```

In ReserveTableForm I showed all fields but specified attributes in the widget section, such as forcing date and number fields, setting minimum values of time and even the increments that can be booked (every 15 minutes):



```
class ReserveTableForm(forms.ModelForm):
    class Meta:
        model = Reservations
        fields = '__all__'
        widgets = {
            'date': forms.DateInput(attrs={'type': 'date', 'min': timezone.now().date()}),
            'time': forms.TimeInput(attrs={'type': 'time', 'step': 900, 'min': '18:00'}),
            'party_size': forms.NumberInput(attrs={'min': 2, 'step': 2}),
            }
```

DeliveryForm and CollectionForm are almost identical apart from a slight change. Both have fields specified as read-only(where order and price are previously explained to be hardcoded) but collection has a special argument that if the form is valid the pickupTime field is set to be 20 minutes from when the form is submitted:

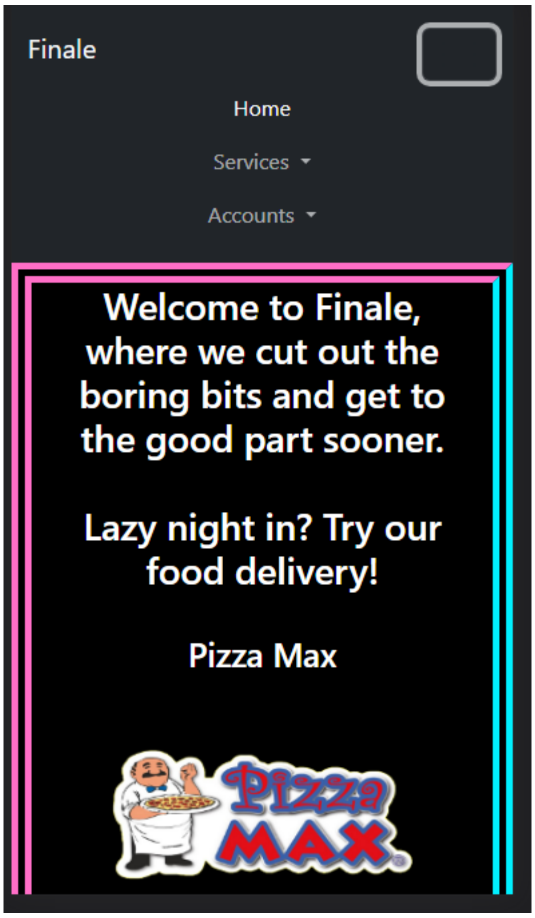


```
class DeliveryForm(forms.ModelForm):
    class Meta:
        model = DeliveryOrder
        fields = ('customer', 'address', 'order', 'price', 'notes')
        widgets = {
                    'order': forms.TextInput(attrs={'readonly': True}),
                    'price': forms.TextInput(attrs={'readonly': True}),
                    }


class CollectionForm(forms.ModelForm):
    class Meta:
        model = CollectionOrder
        fields = ('customer','order', 'price', 'notes')
        widgets = {
                    'order': forms.TextInput(attrs={'readonly': True}),
                    'price': forms.TextInput(attrs={'readonly': True}),
                    }

    def save(self, commit=True):
        instance = super().save(commit=False)
        instance.pickupTime = timezone.now() + timezone.timedelta(minutes=20)
        if commit:
            instance.save()
        return instance
```

2.4 Graphical User Interface (GUI)

Some screenshots from the GUI:

Home Page and Navbar:



Finale

Home

Services

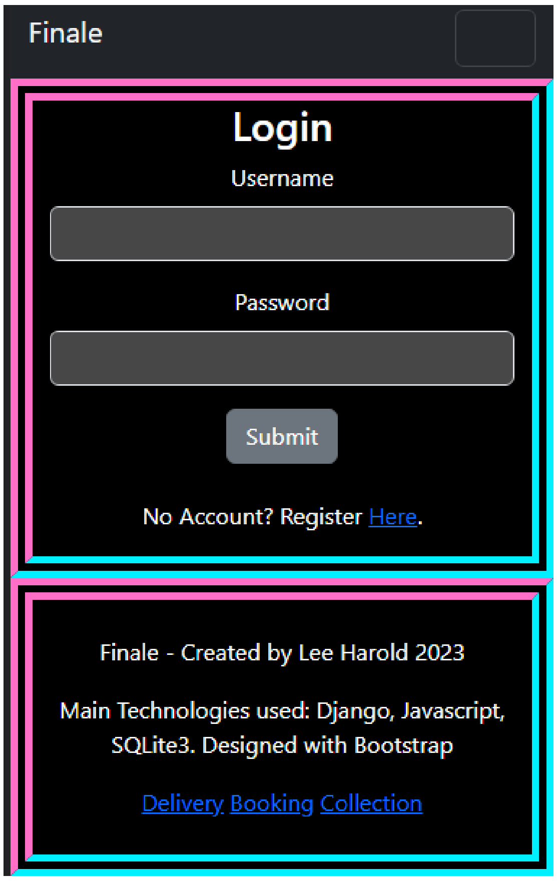
Accounts

Welcome to Finale, where we cut out the boring bits and get to the good part sooner.

Lazy night in? Try our food delivery!

Pizza Max

Login Page:


Finale

Login

Username

Password

Submit

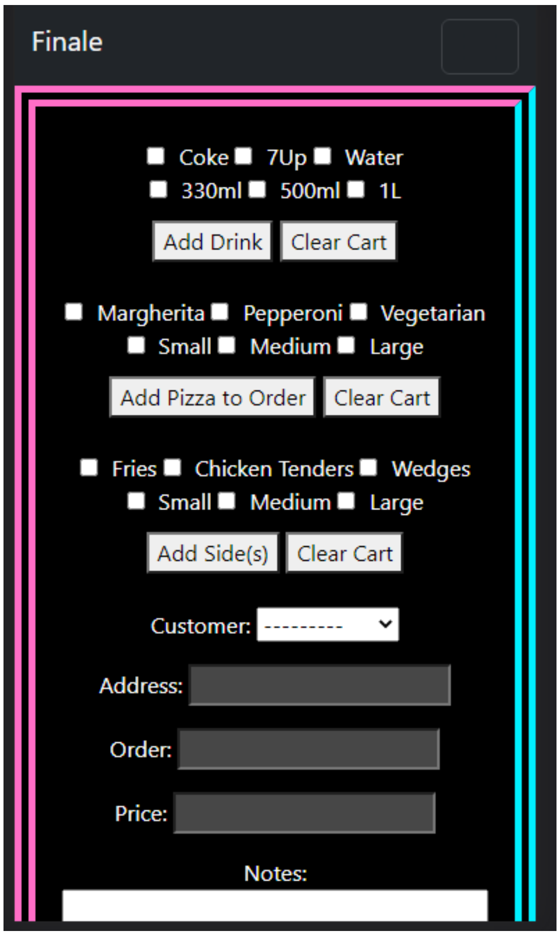
No Account? Register Here.

Finale - Created by Lee Harold 2023

Main Technologies used: Django, Javascript, SQLite3. Designed with Bootstrap

Delivery Booking Collection

DeliveryAdd(placing a delivery order):


Finale

☐ Coke ☐ 7Up ☐ Water
☐ 330ml ☐ 500ml ☐ 1L

Add Drink | Clear Cart

☐ Margherita ☐ Pepperoni ☐ Vegetarian
☐ Small ☐ Medium ☐ Large

Add Pizza to Order | Clear Cart

☐ Fries ☐ Chicken Tenders ☐ Wedges
☐ Small ☐ Medium ☐ Large

Add Side(s) | Clear Cart

Customer: ---------

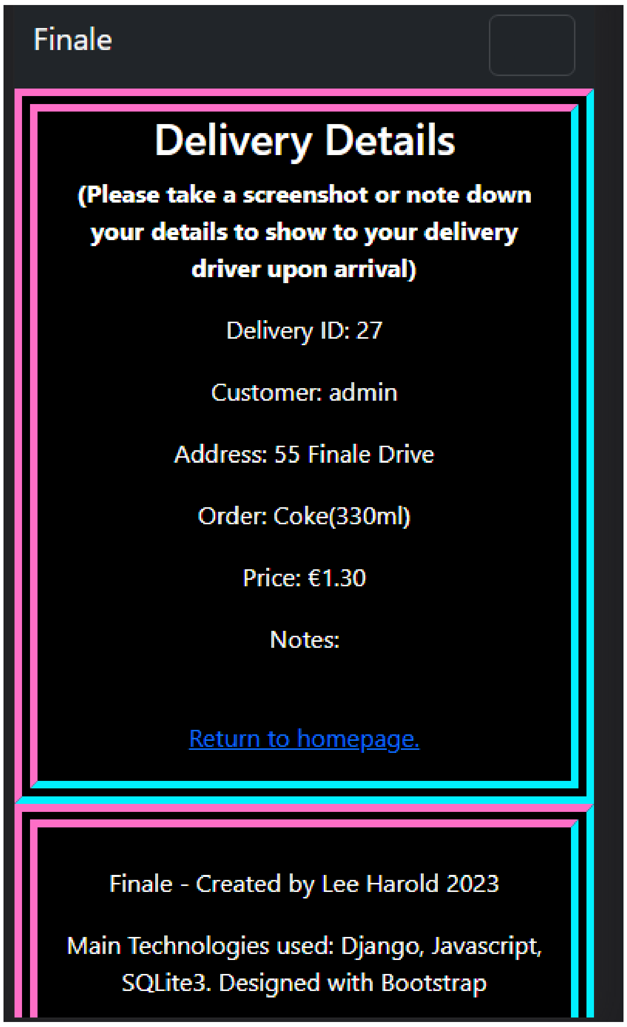
Address:

Order:

Price:

Notes:

Details Page after placing an order:


Finale

Delivery Details

(Please take a screenshot or note down your details to show to your delivery driver upon arrival)

Delivery ID: 27

Customer: admin

Address: 55 Finale Drive

Order: Coke(330ml)

Price: €1.30

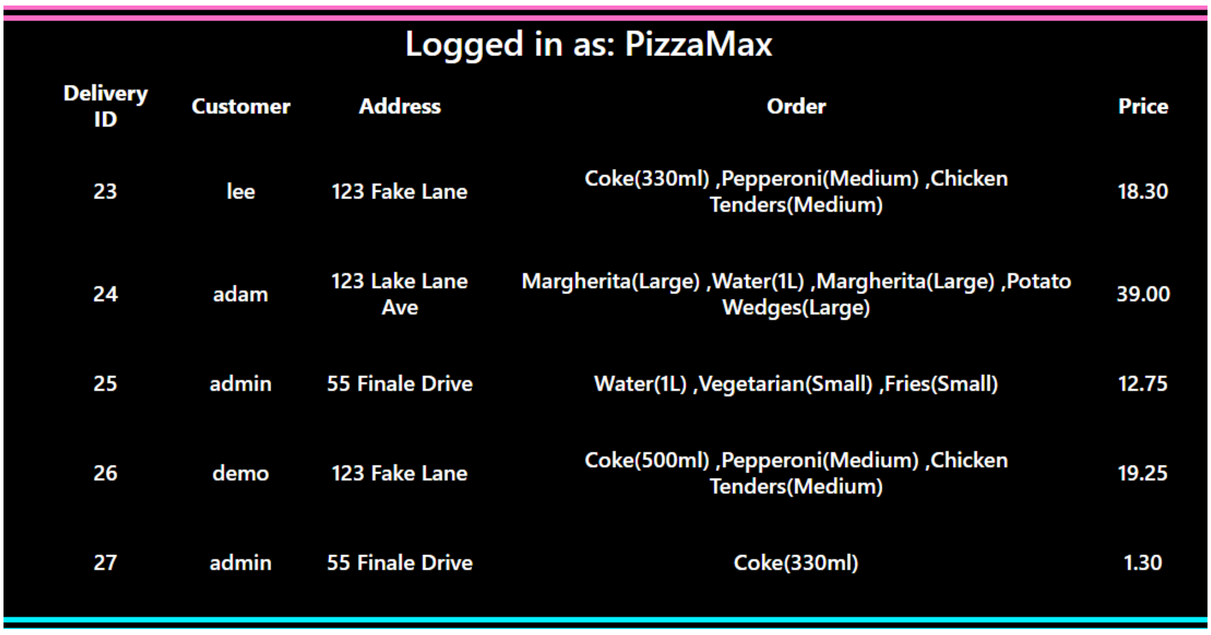
Notes:

Return to homepage.

Finale - Created by Lee Harold 2023

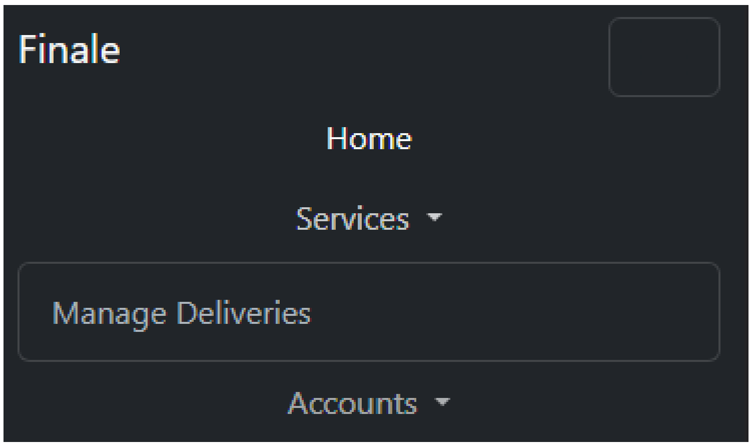
Main Technologies used: Django, Javascript, SQLite3. Designed with Bootstrap

Table of orders logged in as a business:


Logged in as: PizzaMax

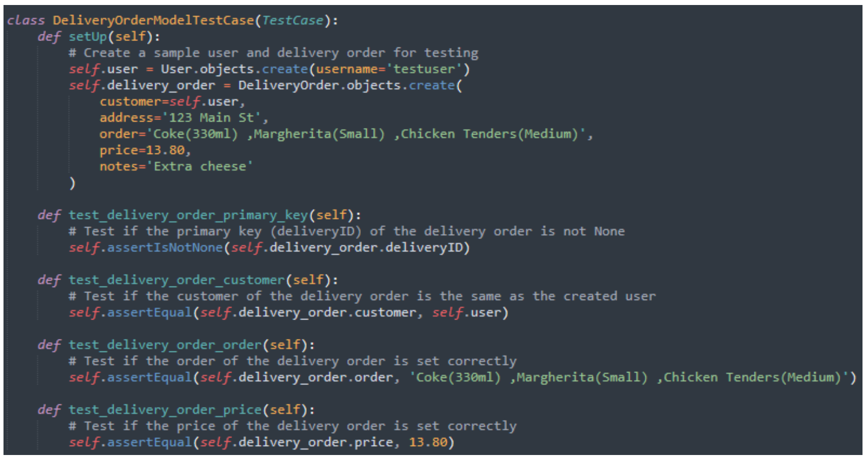
Delivery ID	Customer	Address	Order	Price
23	lee	123 Fake Lane	Coke(330ml) ,Pepperoni(Medium) ,Chicken Tenders(Medium)	18.30
24	adam	123 Lake Lane Ave	Margherita(Large) ,Water(1L) ,Margherita(Large) ,Potato Wedges(Large)	39.00
25	admin	55 Finale Drive	Water(1L) ,Vegetarian(Small) ,Fries(Small)	12.75
26	demo	123 Fake Lane	Coke(500ml) ,Pepperoni(Medium) ,Chicken Tenders(Medium)	19.25
27	admin	55 Finale Drive	Coke(330ml)	1.30

Special Navbar when logged in as a business:

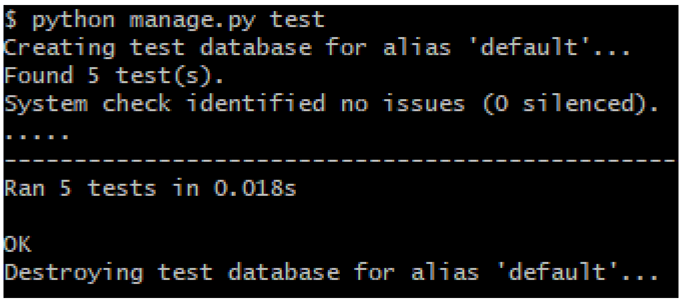


2.5 Testing

Testing with tests.py file:

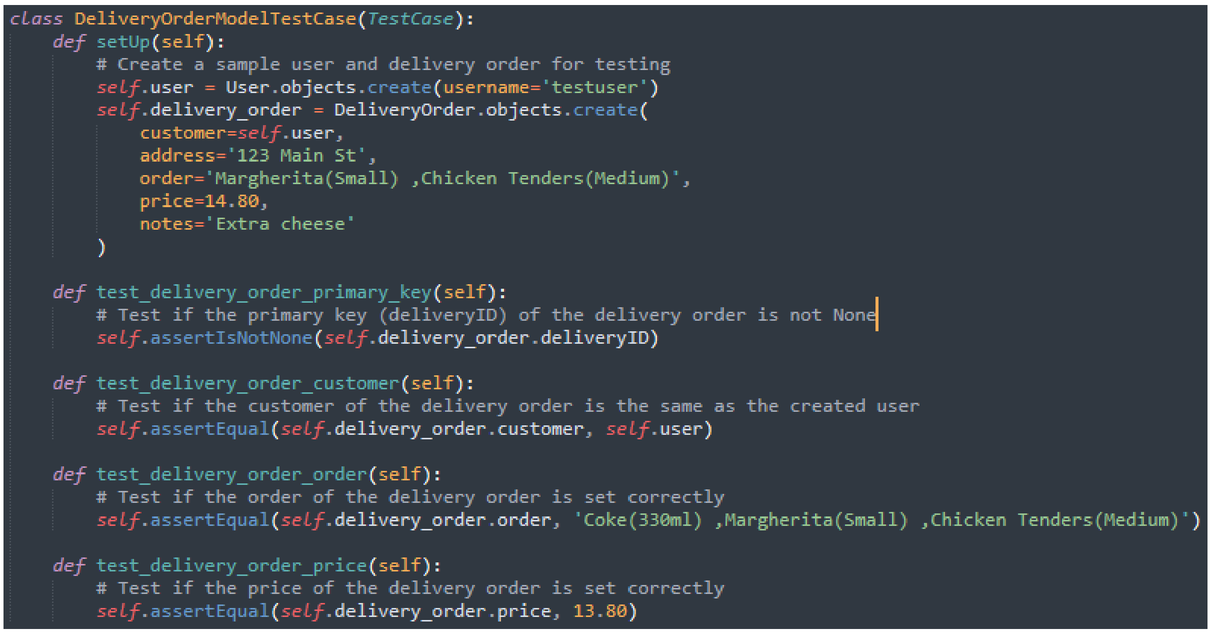


```python
class DeliveryOrderModelTestCase(TestCase):
    def setUp(self):
        # Create a sample user and delivery order for testing
        self.user = User.objects.create(username='testuser')
        self.delivery_order = DeliveryOrder.objects.create(
            customer=self.user,
            address='123 Main St',
            order='Coke(330ml) ,Margherita(Small) ,Chicken Tenders(Medium)',
            price=13.80,
            notes='Extra cheese'
        )

    def test_delivery_order_primary_key(self):
        # Test if the primary key (deliveryID) of the delivery order is not None
        self.assertIsNotNone(self.delivery_order.deliveryID)

    def test_delivery_order_customer(self):
        # Test if the customer of the delivery order is the same as the created user
        self.assertEqual(self.delivery_order.customer, self.user)

    def test_delivery_order_order(self):
        # Test if the order of the delivery order is set correctly
        self.assertEqual(self.delivery_order.order, 'Coke(330ml) ,Margherita(Small) ,Chicken Tenders(Medium)')

    def test_delivery_order_price(self):
        # Test if the price of the delivery order is set correctly
        self.assertEqual(self.delivery_order.price, 13.80)
```

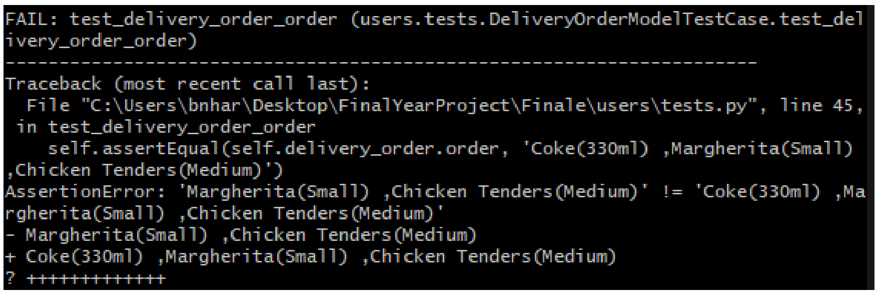


```
$ python manage.py test
Creating test database for alias 'default'...
Found 5 test(s).
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 0.018s

OK
Destroying test database for alias 'default'...
```

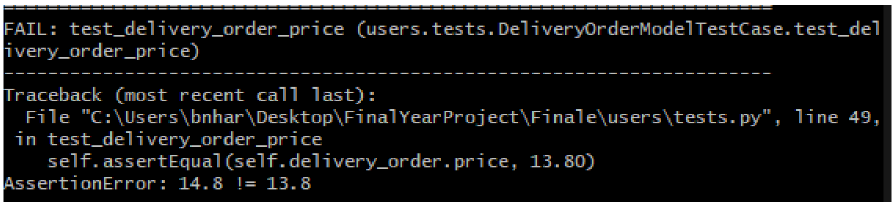
Forcing an error through incorrect testing:



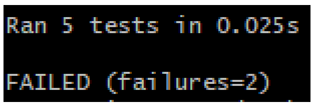
```
class DeliveryOrderModelTestCase(TestCase):
    def setUp(self):
        # Create a sample user and delivery order for testing
        self.user = User.objects.create(username='testuser')
        self.delivery_order = DeliveryOrder.objects.create(
            customer=self.user,
            address='123 Main St',
            order='Margherita(Small) ,Chicken Tenders(Medium)',
            price=14.80,
            notes='Extra cheese'
        )

    def test_delivery_order_primary_key(self):
        # Test if the primary key (deliveryID) of the delivery order is not None
        self.assertIsNotNone(self.delivery_order.deliveryID)

    def test_delivery_order_customer(self):
        # Test if the customer of the delivery order is the same as the created user
        self.assertEqual(self.delivery_order.customer, self.user)

    def test_delivery_order_order(self):
        # Test if the order of the delivery order is set correctly
        self.assertEqual(self.delivery_order.order, 'Coke(330ml) ,Margherita(Small) ,Chicken Tenders(Medium)')

    def test_delivery_order_price(self):
        # Test if the price of the delivery order is set correctly
        self.assertEqual(self.delivery_order.price, 13.80)
```



```
FAIL: test_delivery_order_order (users.tests.DeliveryOrderModelTestCase.test_del
ivery_order_order)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\bnhar\Desktop\FinalYearProject\Finale\users\tests.py", line 45,
 in test_delivery_order_order
    self.assertEqual(self.delivery_order.order, 'Coke(330ml) ,Margherita(Small)
,Chicken Tenders(Medium)')
AssertionError: 'Margherita(Small) ,Chicken Tenders(Medium)' != 'Coke(330ml) ,Ma
rgherita(Small) ,Chicken Tenders(Medium)'
- Margherita(Small) ,Chicken Tenders(Medium)
+ Coke(330ml) ,Margherita(Small) ,Chicken Tenders(Medium)
? +++++++++++++
```

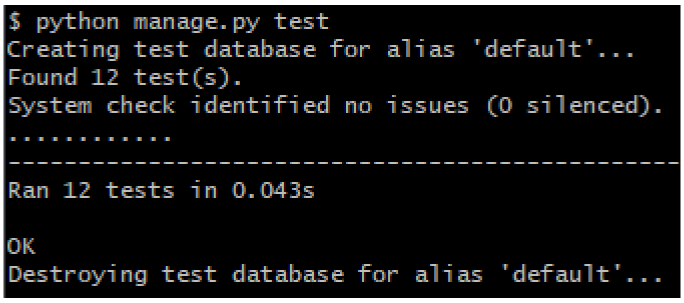


```
FAIL: test_delivery_order_price (users.tests.DeliveryOrderModelTestCase.test_del
ivery_order_price)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\bnhar\Desktop\FinalYearProject\Finale\users\tests.py", line 49,
 in test_delivery_order_price
    self.assertEqual(self.delivery_order.price, 13.80)
AssertionError: 14.8 != 13.8
```



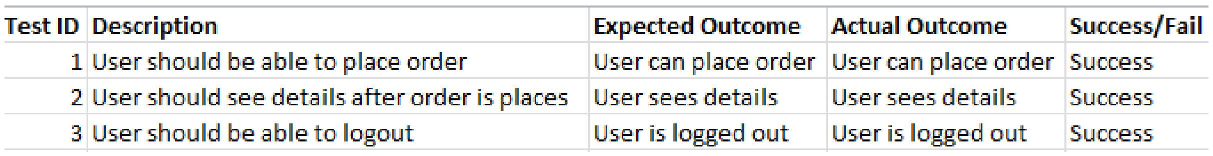
```
Ran 5 tests in 0.025s

FAILED (failures=2)
```

Full successful test file ran:



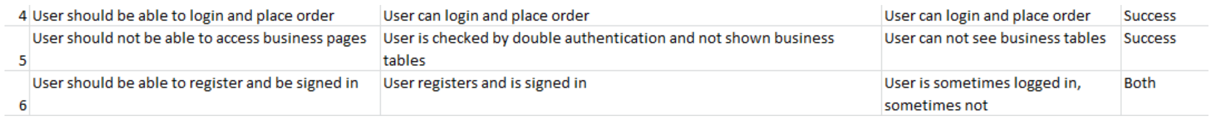
```
$ python manage.py test
Creating test database for alias 'default'...
Found 12 test(s).
System check identified no issues (0 silenced).
............
----------------------------------------------------------------------
Ran 12 tests in 0.043s

OK
Destroying test database for alias 'default'...
```

Example of some manual tests ran on individual functions:



Test ID	Description	Expected Outcome	Actual Outcome	Success/Fail
1	User should be able to place order	User can place order	User can place order	Success
2	User should see details after order is places	User sees details	User sees details	Success
3	User should be able to logout	User is logged out	User is logged out	Success

Example of manual tests ran on a string of functions together:



4	User should be able to login and place order	User can login and place order	User can login and place order	Success
5	User should not be able to access business pages	User is checked by double authentication and not shown business tables	User can not see business tables	Success
6	User should be able to register and be signed in	User registers and is signed in	User is sometimes logged in, sometimes not	Both

2.6 Evaluation

My overall evaluation of the technologies and projects used are that they are very effective and useful for the scale being use, and could be easily scalable if needed. There is more efficient ways to perform the JavaScript functionality performed but the way I coded it also makes it easier to read. With more time and experience the amount of code could be cut down and functionality stays the same or is improved.

3 Conclusions

Initial thoughts for the limitations and disadvantages of the project stem from my own ability and the timeframe and restrictions around the development cycle due to my attention also being needed for other modules and their assessments throughout the year. Outside of a college setting I could prioritise and streamline the development of the project with effective time and resource management. The advantages and strengths I want to bring to the project are to be convenient and user-friendly, while also being effective and useful for the applications goals. Ordering food should be convenient for any level of user.

4 Further Development or Research

With additional time and resources, I would like to improve and scale the application for more widespread and industry-standard usage. I would attempt to allow for exponential growth by adopting an external database (separate from Django's built-in data handling) that accommodates large amounts of data in order to handle high ceiling of users and orders within the application.

5 Appendices

This section should contain information that is supplementary to the main body of the report.

5.1 Project Proposal

National College of Ireland

Project Proposal

Reserve / Order App

20/10/2022

BSHC

Software Development

2022/2023

Lee Harold

X19429942

x19429942@student.ncirl.ie

Contents

1.0 Objectives

This project sets out to create a web application that allows a user to make reservations, order courses in advance, order food to be ready for collection or order food for delivery with participating restaurants, takeaways and fast-food establishments. A customer will need to be verified somehow to prevent theft or wrong orders so a form of verification needs to be used when the customer arrives or the employee who is delivering food arrives at the location of delivery.

2.0 Background

I chose to undertake this project as the current market for these types of applications are scarce. It can be a hassle needing multiple apps for similar services, so I want to incorporate multiple food-industry services into one. This will allow the user to stay contained within the app while still placing orders, deliveries and reservations. As the food industry becomes more and more disconnected with the social aspect I felt it better to lean into the change than to oppose it, which was a big motivation for the project idea.

3.0 State of the Art

Most, if not all reservation apps lack an interconnected system for reserving and/or ordering in one app without external links or apps/websites. For example, a popular app on iPhone called ‘OpenTable’ holds similar ideals of reservations and ordering, but to order food you are funnelled externally to companies such as Deliveroo or JustEat. I want to bring the services together and accessible through only itself without external sources or links. This makes it stand out from similar products as it will be an all-in-one app for reserving and ordering.

4.0 Technical Approach

I will attempt to take the foundations and services provided by more popular apps and combine them into one web application for more streamlined accessibility. I will need to find a way for the web application to take orders and store them and add functionality of cancelling or changing orders (before a certain amount of time has passed). I’m not sure about methods of verification as they would need to be used both with deliveries, collections and reservations. I’m trying to make the process of orders, deliveries and reservations as simple as possible.

I will be taking an Agile approach to the development of the project, probably one or two week increments of rough task deadlines. I have a very basic plan for now before I start breaking down intricate details and milestones. I will be breaking down requirements into priority, so back-end, front-end, security (and authentication), then front-end. Once the main bulk of project work is underway I will have a more clear understanding of setting milestones and future tasks.

I will be starting by researching the possible programming languages I can use and making sure that they are viable alongside any future other software or technologies introduced to the project further along in the life cycle. This is the most important part of the project outside of the application itself as if it is done incorrectly it could derail deadlines and set me back weeks if my software and technologies do not synergise in the ways I need.

After the research stage the first priority is implementing C.R.U.D functionality for the orders, reservations and deliveries as it is the most integral part of the project. Once this is achieved different milestones can/will start leaning into or sharing similar tasks. These will have different levels of priority and time spent in order to manage deadlines and workload efficiently.

5.0 Technical Details

My initial thoughts are to use Django for my goals, but I will need to complete more detailed research to make sure they are useable down the line as I combine my different web application functionalities. Django is a Python-based framework for use on websites and web applications. I’ve decided to use Django as I am very familiar with Python and I have previous experience.

6.0 Special Resources Required

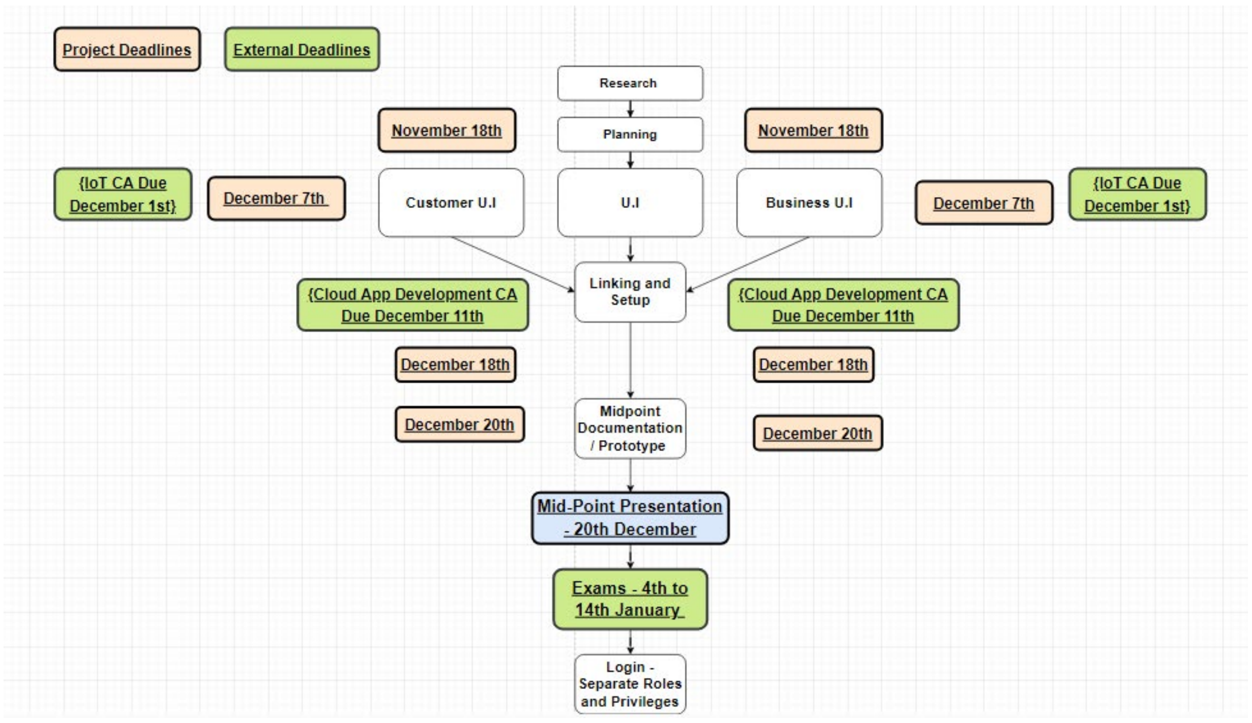
N/A

7.0 Project Plan

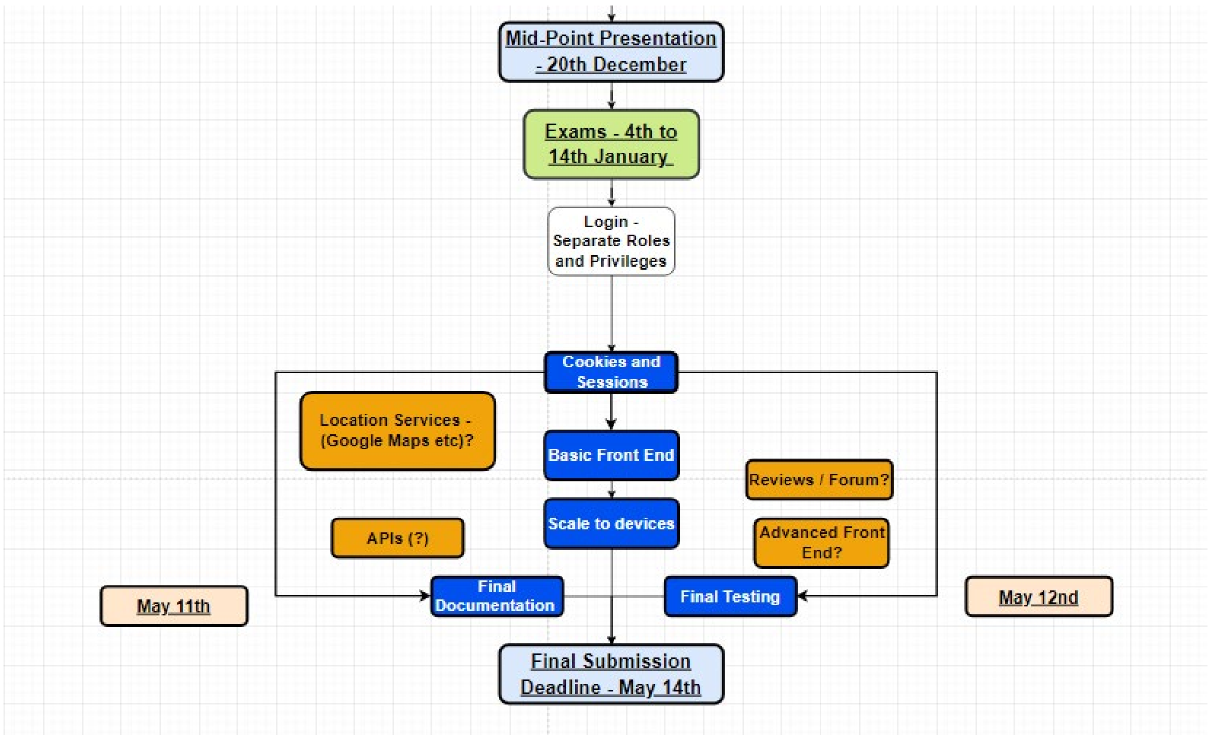
In the first few weeks I plan to complete my research to choose my resources (software, languages, technologies etc.) and get the project off of the ground with basic point-to-point C.R.U.D for orders on the customer before moving onto showing orders on both customer and business side. Front-end is low priority as it is not necessary so early on unless it makes testing easier.

A working prototype is needed for the midpoint which is the 20th of December so tasks and deadlines will be more fluid around that time in order to meet requirements and deadlines. From January until May the main focus will be shifted to security, authentication, testing, and front-end visuals. Below is a rough idea of the order of tasks based on priority and stages of development in stages. Stage 1 is set and planned. Stage 2 will be decided post-semester one exams as I have a better feel for timing with Semester 2 module deadlines:

Stage 1: Beginning to Midpoint


Project Deadlines
External Deadlines
Research
Planning
November 18th
November 18th
{IoT CA Due December 1st}
December 7th
Customer U.I
U.I
Business U.I
December 7th
{IoT CA Due December 1st}
{Cloud App Development CA Due December 11th
Linking and Setup
{Cloud App Development CA Due December 11th
December 18th
December 18th
December 20th
Midpoint Documentation / Prototype
December 20th
Mid-Point Presentation - 20th December
Exams - 4th to 14th January
Login - Separate Roles and Privileges

<u>Stage 2: Midpoint to Deadline (To be Refined Later)</u>

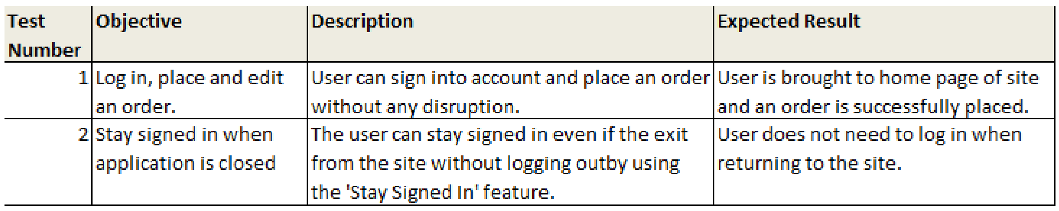

Mid-Point Presentation - 20th December
Exams - 4th to 14th January
Login - Separate Roles and Privileges
Cookies and Sessions
Location Services - (Google Maps etc)?
Basic Front End
Reviews / Forum?
Scale to devices
APIs (?)
Advanced Front End?
May 11th
Final Documentation
Final Testing
May 12nd
Final Submission Deadline - May 14th

8.0 Testing

Integration testing will be straightforward with core functions possibly targeted for use. They can be broken down and combined in different ways to be more simplified or efficient as needed. For example:

Test Number	Objective	Description	Expected Result
1	Gain access to site through login	Login using account details.	User is brought to home page of site and is signed in.
2	Place delivery order.	Choose an item from the menu, confirm order, address and payment details.	Order is placed and visible on both customer and restaurant side
3	Edit order details.	Before a certain amount of time as passed, an order can be edited. Edit an order.	Order details are updated and are reflected to both customer and

As you can see the functionality can be broken to short and concise tasks but can be evolved and adapted to be more intricate and involved such as a line of testing for a chain of functions, such as going from the login page to editing an order in one session.



Test Number	Objective	Description	Expected Result
1	Log in, place and edit an order.	User can sign into account and place an order without any disruption.	User is brought to home page of site and an order is successfully placed.
2	Stay signed in when application is closed	The user can stay signed in even if the exit from the site without logging outby using the 'Stay Signed In' feature.	User does not need to log in when returning to the site.

System tests will be used later when the software is further along and developed with most if not all of its functionality. This will allow full end-to-end testing of the application/site as a whole.

I will be testing the order functions on the business side with fake, pre-made account details and orders to avoid any ethical issues.

I will research the use of automated testing to check functionality sequences as previously discussed e.g. Log in, place, edit in one sequence of actions. This can assist in spreading out time and resources by not using manual testing for the entire testing process.

6.2 Reflective Journals

October:

Supervision & Reflection Template

Student Name	Lee Harold
Student Number	x19429942
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: October

What? My project idea was approved with amendments and my research increased.
So What? Research for the project became focused as the idea was approved. This helps planning going forward. I still

have decisions to make towards the project.
Now What? I can research my resources and begin planning for their uses.

Student Signature	Lee Harold

November:

Supervision & Reflection Template

Student Name	Lee Harold
Student Number	X19429942
Course	BSHCSD4
Supervisor	Frances Sheridan

Month:

What? I chose the majority of my technologies that will be used in the project. I began initial steps with connecting my database to my I.D.E and began basic C.R.U.D implementation with a very simple front-end.	
So What? The project got off of the ground and allows me to develop and build around the initial starting point. Connecting the database with my I.D.E was a great success after some issues. I needed to roll back to an earlier version of my database due to different authentication issues with the I.D.E extension. For now I have no issues in my development but I am sure I will encounter many during the life cycle of the project.	
Now What? I will try to stick to my amended timeline of tasks and deadlines as close as possible and only adjust when absolutely necessary. This will allow sufficient time for any blocks or obstacles I encounter as well allow me to spread my time and efforts efficiently throughout the following months while other modules also receive necessary time and effort.	
Student Signature	Lee Harold

December – January

December:

Student Name	Lee Harold
Student Number	X19429942
Course	BSHCSD4
Supervisor	Frances Sheridan

Month:

What?   I worked on my midpoint presentation, prototype and demo ahead of the deadline and reflected on the progress made, as well as any positives and negatives in the development cycle so far
So What?   I was able to judge and review the achieved and unachieved milestones of the project while planning on steps going forward alongside adjustments to the plan and approaches.
Now What?   I can take these resulting conclusions regarding my progress and apply practical solutions in order to address them.

Student Signature	Lee Harold

January

Student Name	Lee Harold
Student Number	X19429942
Course	BSHCSD4
Supervisor	Frances Sheridan

Month:

What? I researched some tutorials and ideas related to my project going forward.
So What? I became more familiar with some techniques and looked into some ideas that I could incorporate later.
Now What? I'll outline some possible uses for these and where and when I can/should integrate them.

Student Signature	Lee Harold