Evaluation

Effectiveness of Python and tools & techniques used

A scripting language is a programming language designed for integrating and communicating with other programming languages. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++. For this project I used HTML and CSS with Python. Scripting languages are high level, interpreted, dynamic and have simple syntax.

A high level language is a programming language with strong abstraction from the details of the computer, python matches this description so it qualifies as a high level language. An interpreter translates and executes a program line by line, without requiring them previously to have been compiled into a machine learning program, python uses an interpreter to execute it's code. Dynamic programming language, in computer science, is a class of high-level programming languages which, at runtime, execute many common programming behaviours that static programming languages perform during compilation, python operates like this meaning it is a dynamic language. Python has simple syntax which is easily readable by humans. All of this together allows me to evaluate python as a string scripting language.

Python is an OOP language, however it does allow you to use the functional paradigm or even the imperative paradigm. Python is firstly an Object Oriented Programming language so it is easier to use it as such, but it still works very well when using the functional paradigm.

I used Django for my project. Django was very good and efficient to use. For some parts I used functional programming, this was for my view files for each of my apps. This worked well, as it made it easy to render the views as calling the functions was simple. For my models and other complex operations such as the popularity operation, I used OOP. This worked very well, as it made my software easier to maintain, more reusable and as whole made my development faster. In the URL files of each app I used the imperative paradigm. This is because I only had to have an array called urlpatterns that would match a URL with the appropriate view function. Since this was such a simple problem, using the imperative paradigm worked well here. Django also handled all of my SQL statements for me, which sped up development time. I used different paradigms at appropriate times for the challenge I needed to solve, and it worked.

Another library that I used was redis. Redis allowed me to create a recommendation and was simple to use once installed, but was quite difficult to install which I have documented in the development testing chapter. I originally used WeasyPrint to create PDFs, but it was horrendously difficult to install so I switched to xhtml2pdf, which worked brilliantly.

I use Visual Studio Code for my IDE. This IDE was very good to use, as it not only supported Python, but also HTML & CSS which I needed as I was using Django. The ability to use all three of these languages in one IDE at the same time was extremely valuable and very beneficial. Some other benefits about Visual Studio Code are: code completion, line numbering, syntax highlighting and the debugger. I also found being able to use the command line while in the IDE very helpful as well.

Another benefit of Visual studio Code was being able to customise my experience depending on my needs with the use of extensions, which I used numerously to help my development of this solution, such as Django, Python, Terminal and vscode – icons.

Compare & Contrast System

The two commercial systems I investigated were Gloria Food and TastyIgniter.Tastyigniter has more functionality for the manager as there are more graphs and high level data processing. However, my system has similar functionality to Gloria Food. My code has some advanced functionality such a s a recommendation engine and a fully functioning coupon system. Both of these systems offer some form of a coupon system but not a recommendation engine, making my system more functional in this regard.

In terms of reliability, my system would probably fall short of both as the code is not optimised for efficiency, so it may run slower in parts than Gloria Food or TastyIgniter, for example my code does not send batch emails and whenever I need to do so, my code loops through the email addresses and sends an individual email to each one.

I believe that my user interface is better than Gloria Food's and TastyIgniter's as it is simpler and more intuitive, making it easier to use for a new, inexperienced user.

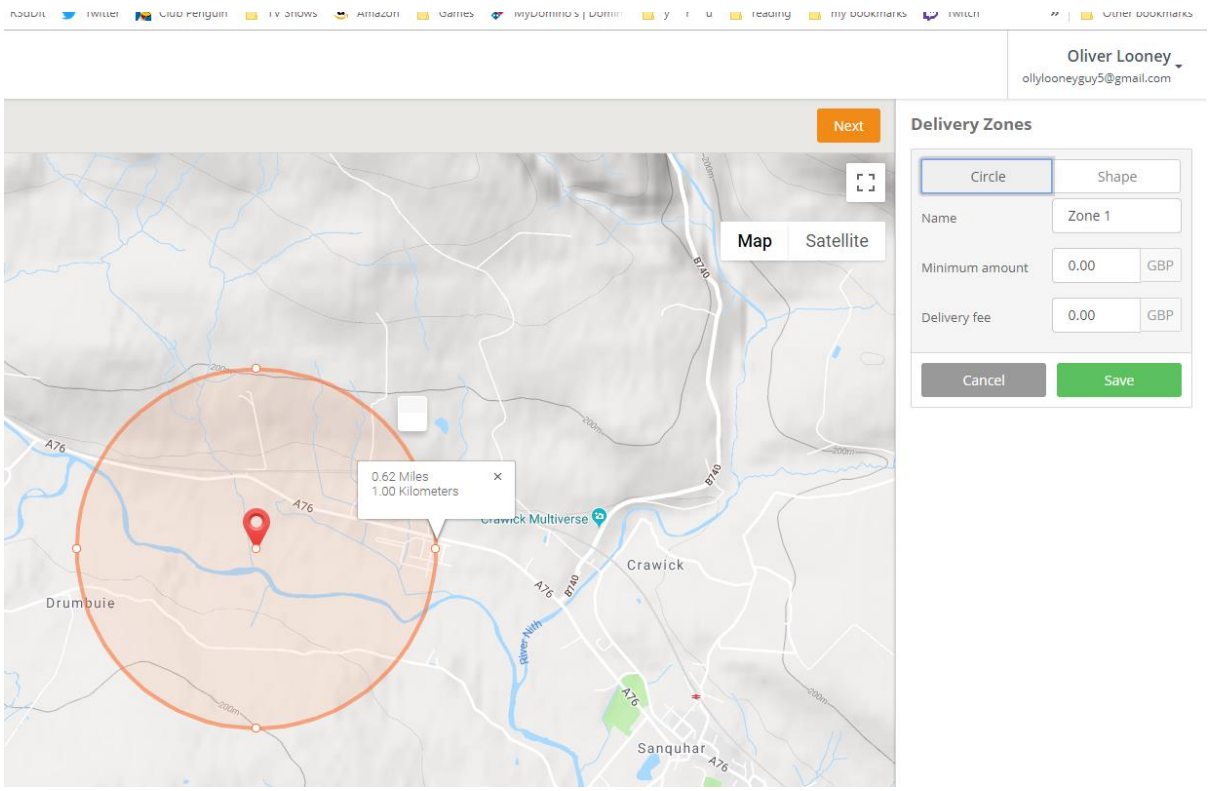
Both Gloria Food and TastyIgniter can export their data to CSV as well as PDF, while my system can only create PDFs, so my system has less data portability than the two commercially available systems that I investigated.

Functionality I said I would use from these systems in Investigation:

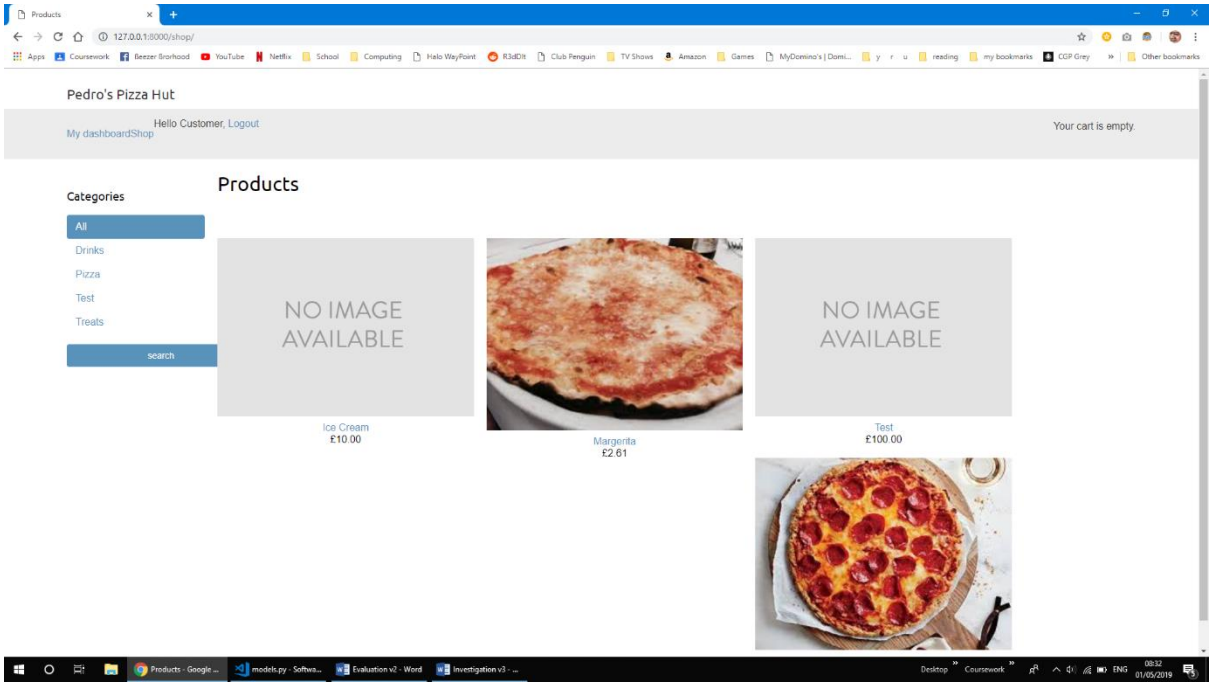
1. Delivery zones
2. Have many products in a group to easily organise the menu
3. Preview
4. Allow products & categories to have images stored
5. Products can only be ordered on certain days if the manager wishes
6. Custom background picture
7. Opening & closing times
8. Allowing the admin to set up options for pre ordering
9. PDF file output
10. Table that ranks the products for the manager
11. Table that ranks the products for the customer
12. Heatmap
13. Dashboard
14. Search
15. View information on the shop

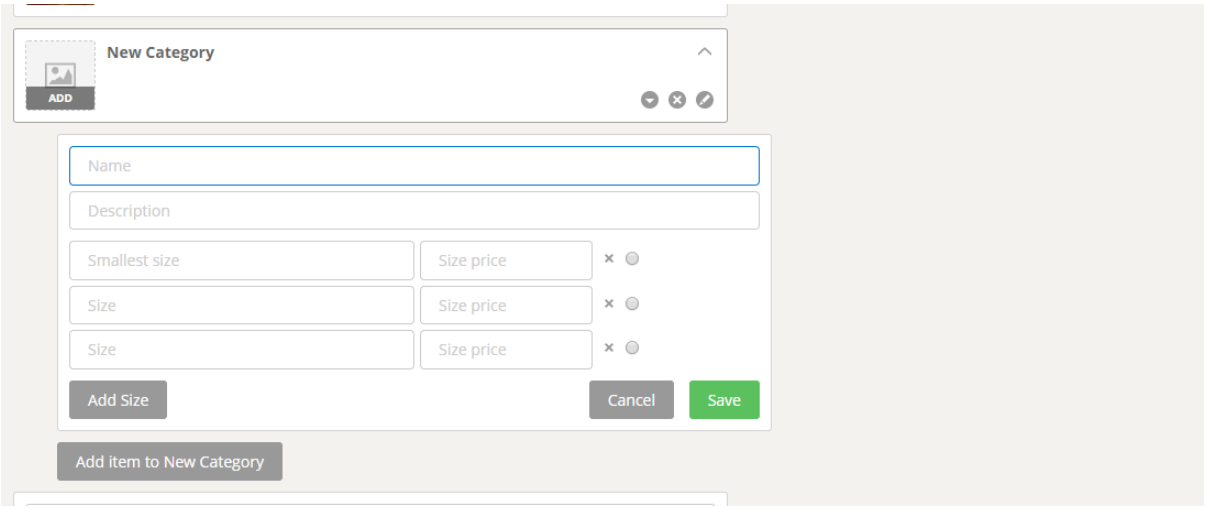
Comparing my system to these features:

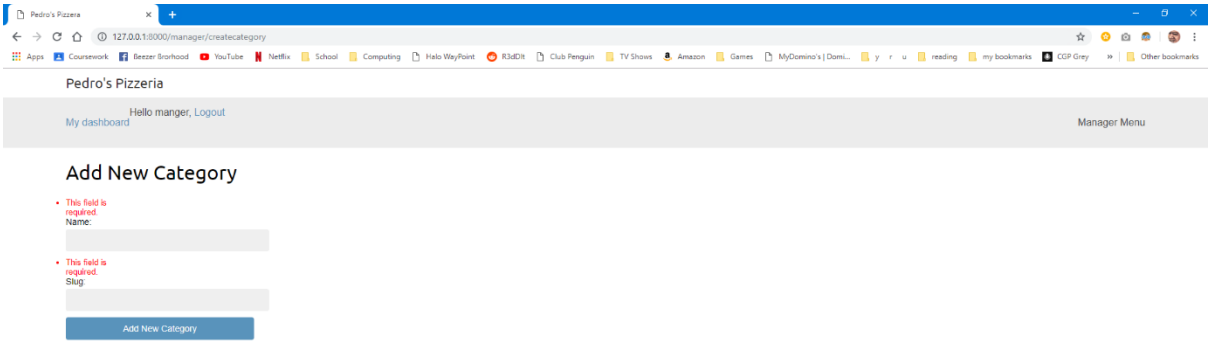
1. My system did not have delivery zones, as I did not have enough time to implement this feature. The reason why it was such a substantial problem was that this required me to use the google maps API to get this to work. I attempted to do set it up but I had to use a real, commercial account for it and as I was running out of time, I had to leave it. Due to these reasons, this my system has less functionality than the commercially available system in this regard, and this is a shortcoming. Delivery zone in a commercial system:



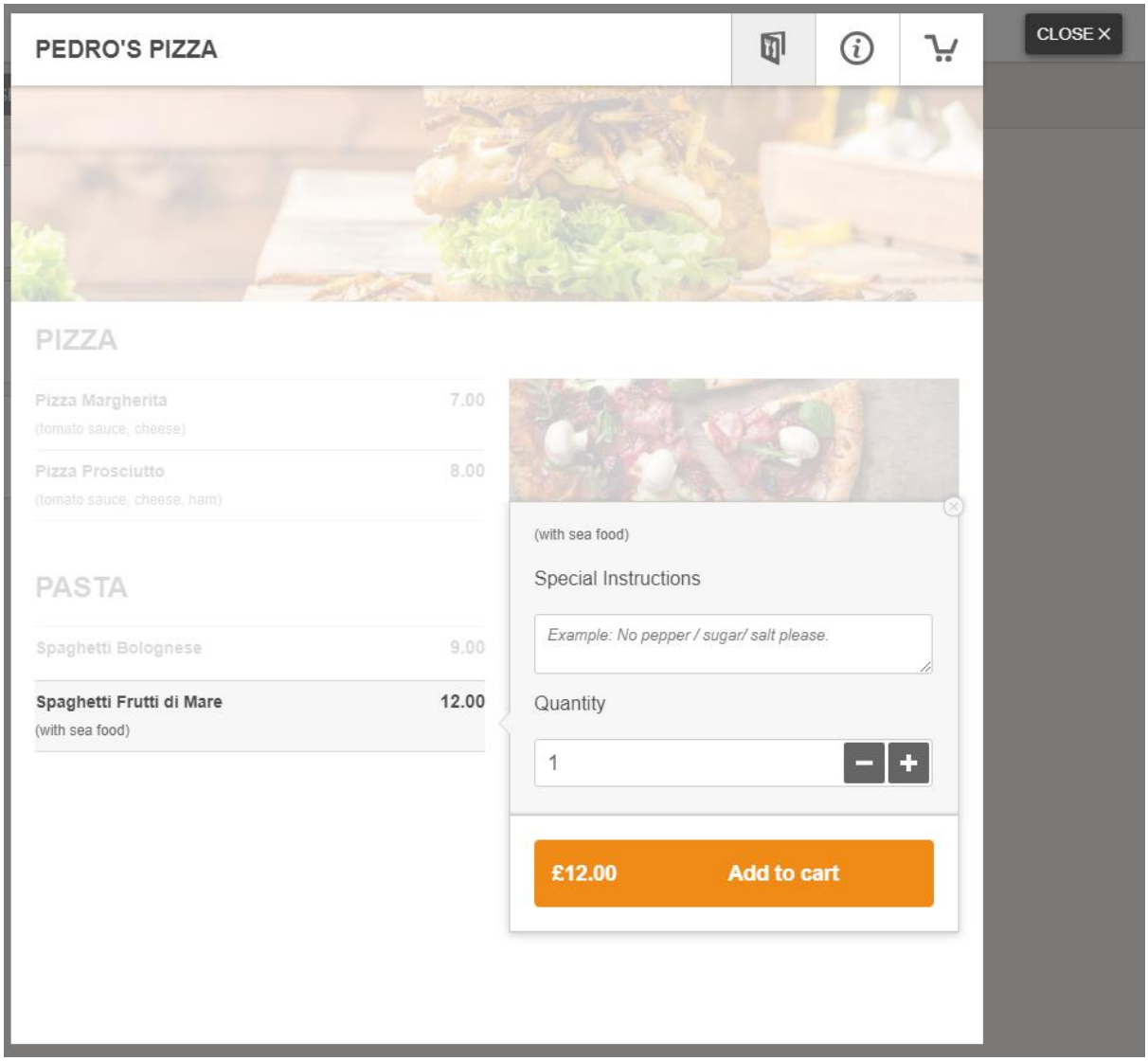
2. I did implement this, it is a good feature because it makes ordering from my website a much easier & streamline process, if the user knows what category they would like. Since I achieved this good feature my system has similar functionality to the commercially available the system that I investigated. I implemented this by creating a category entity that has a one category to many products relationship in my database structure.
   Below you can see a comparison of a commercial system adding a new category to my system adding a new category.



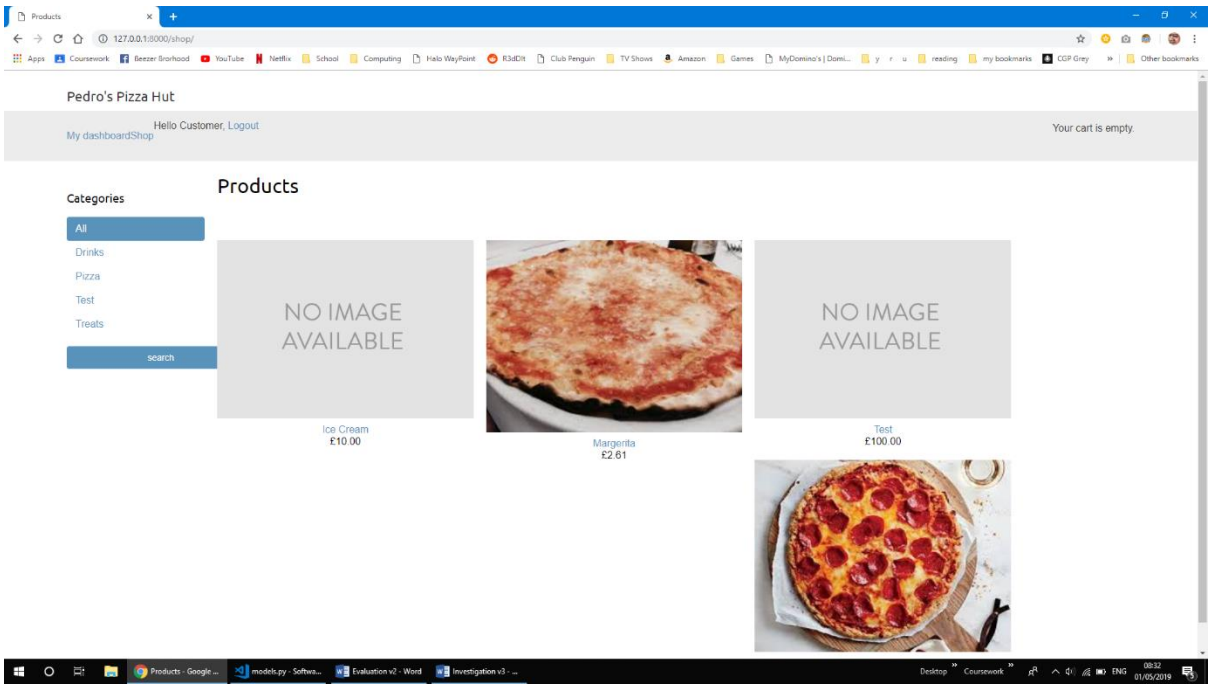




3. I did not manage to implement a menu preview option for the manager, as they edited the menu on the admin version of the website. This was a shortcoming as it means that once the manager an edit to the menu, or the products or categories on the website, they would have to log in on a customer's account to view the changes, which is tedious and time consuming. This would have required me to create new html template files and new view functions in the manager's Django app in order to implement a menu preview.
   For this reason, the lack of implementation of this feature made it a shortcoming, and less reusable than a commercially available system. Below is a commercial system's menu preview



4. I partially succeeded on this objective. The manager can store a single image with every single product, which is displayed on the product detail page and in the shop. To save an image to a product, the manager does not save the actual image to the database, as that would be too impractical, but the manager saves the file location of the image. Then, whenever the image is required, my code retrieves the file location of the image form the products table, and then retrieves the image. If there is no image a default image is retrieved instead, as a backup in case the manager has not saved an image. I did not implement this for the category as I did a not have a dedicated view category page so I thought it would be unnecessary. This succeeded so I can compare my system with the commercially available system that I investigated and say that on this objective, my system was just as functional, and because of that, I can justify that this was a good feature. Below you can see my system displaying the product's image, if there is one to do so.

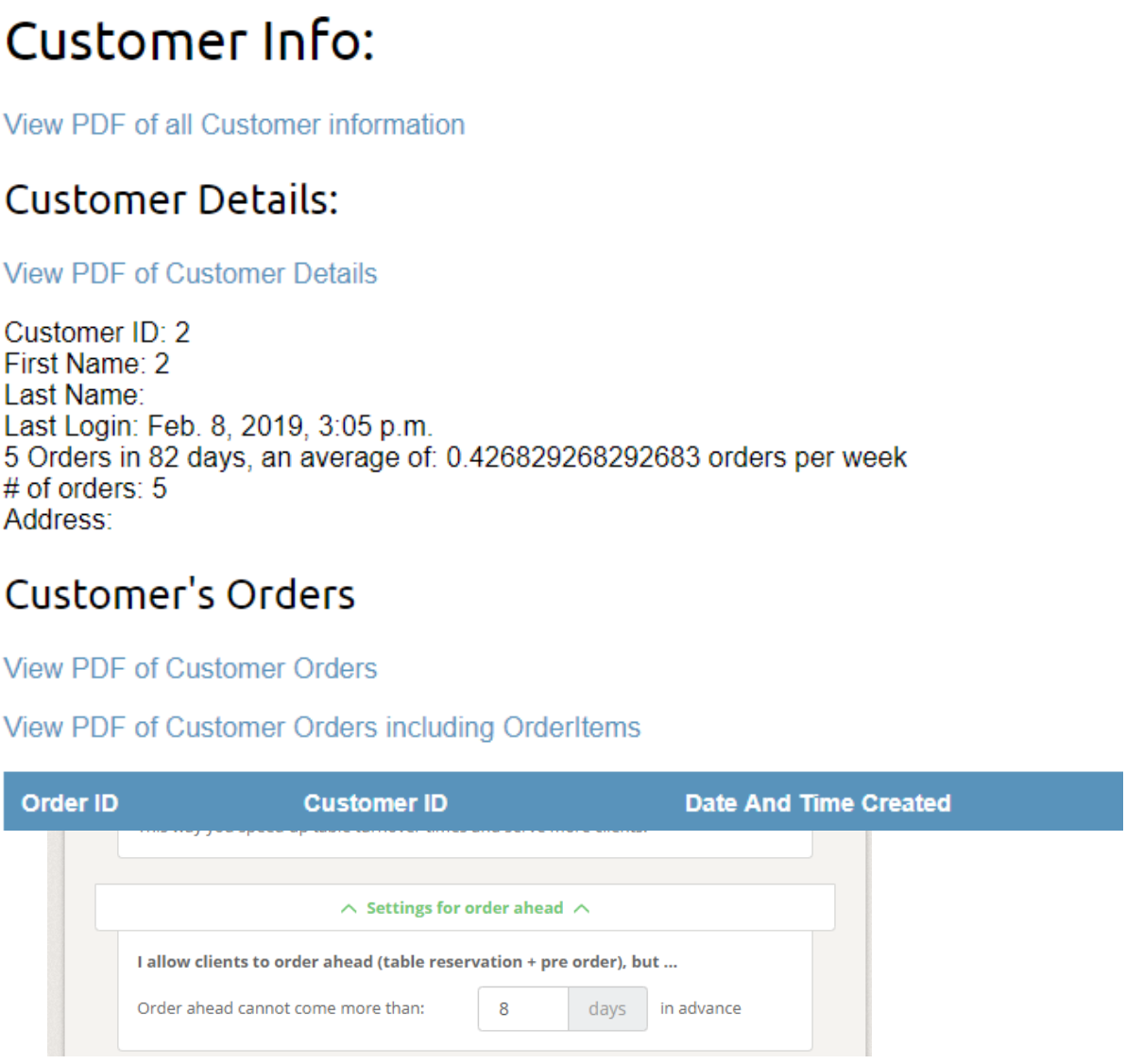


5. I was unsuccessful in implementing this feature. I decided after the prototype that I would not have enough time to implement any of the time functionality. This was a quite serious shortcoming of my system. The other commercial systems that I investigated, had this feature, so on this objective those systems are more functional.
   If I were to implement this feature, I would use the Arrow python third party library to do so. I used this library in the account.model.py file. From arrow's website it is: a Python library that offers a sensible, human-friendly approach to creating, manipulating, formatting and converting dates, times, and timestamps. It implements and updates the datetime type, plugging gaps in functionality, and provides an intelligent module API that supports many common creation scenarios. Simply put, it helps you work with dates and times with fewer imports and a lot less code.
   A lot of the new algorithms that I would have to code in order to implement this feature, would probably be mostly validation functions, for example validating that the date entered is within a realistic time range to avoid errors, I would need to access the current time of the user's system for this algorithm.
   Another algorithm that I would have to be code would be an algorithm that would take in all of the times of an event and complete some high level data processing on the data. This would include, the average, the inter quartile range and a prediction for each time which would help the manager set the target times for different events.
   If I were to implement this I would probably use a serial text file to store the different attributes that would be stored about each event and their times. I think that I would do this because the data would be accessed so infrequently, by probably only the manager so the slower and less efficient access of a serial compared to a random(direct access) file would no matter that much. To access the data I would probably use a linear search algorithm for access a particular event.

6. I just completely left out a custom background picture, as it adds no functionality. I can justify that this is not really a shortcoming, as it is such a small & trivially feature. To implement this feature I would need to delve deeper in the CSS code, and add in this feature in one of the base CSS files that are imported for all of the other html template's CSS code.
7. I also left out this feature as it would simply block the user from placing orders whenever the restaurant was closed. Since my code will never go live, this would have just proved a nuisance anyway while I coded and tested my system. However for a real life situation with would be a serious shortcoming, as it would mean that the shop could get orders, and thus angry customer's even when the restaurant is closed. Sine the commercially available system has this feature it is better in this regard. Below is how a commercially available system that I investigated implement this feature.

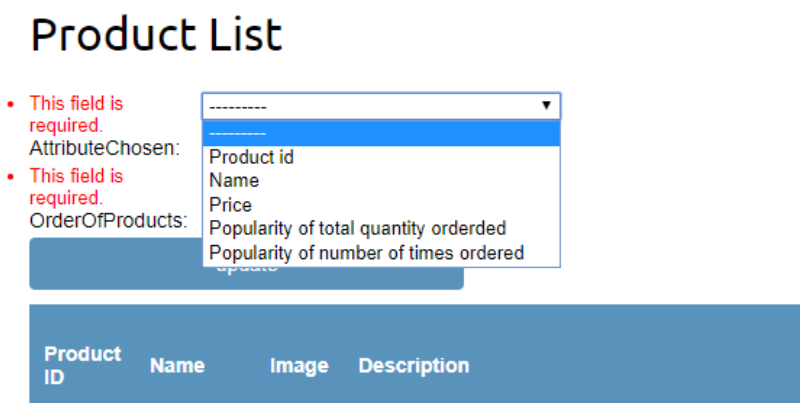
8. This was the same problem and explanation as number 7

9. I successfully implemented PDF output in my system very well. I had many many options for different PDF outputs for the customer, through the receipt, and the all of the staff, which ranged from receipts & order tickets to data extraction from the system. I believe I did better than the commercially available systems I investigated for this objective as I have many more options. Since I made my PDF outputs so easy to use and with a lot of different combinations, I can justify that this was a very strong feature. Below I will show a sample of the different PDF outputs that I achieved. In order to implement this, I had to use a render to pdf function. This

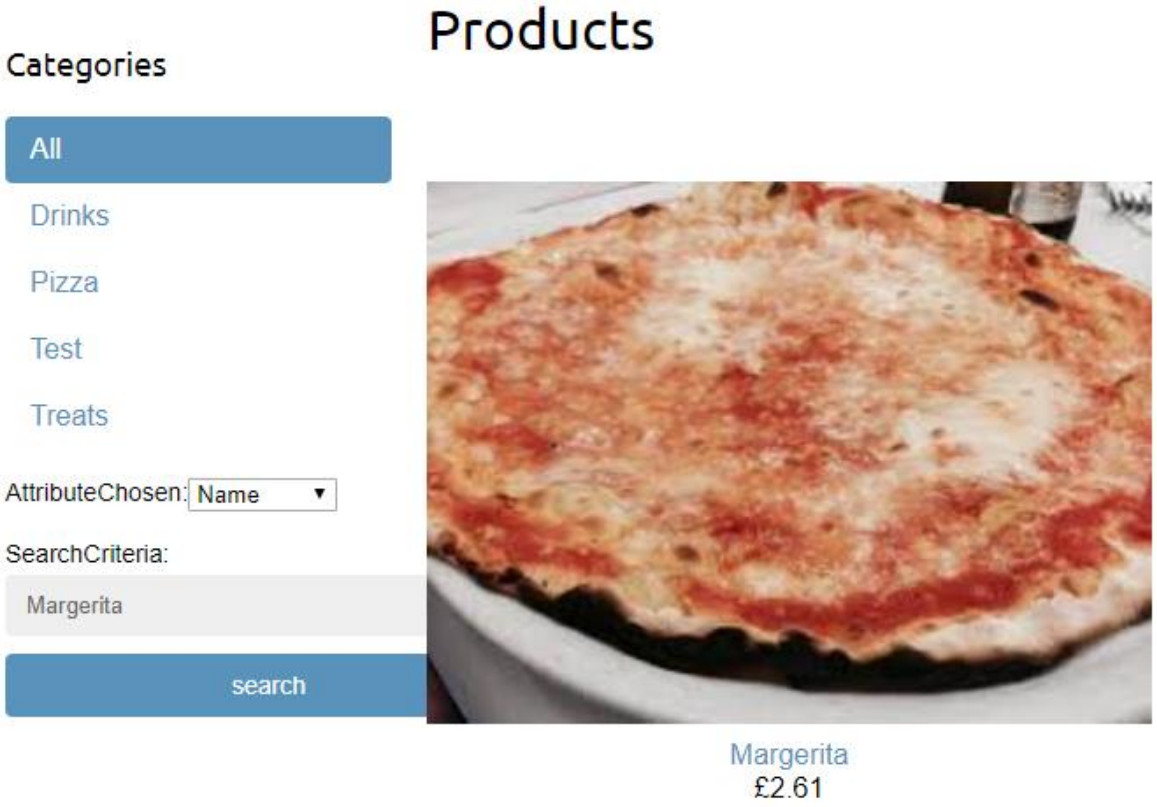


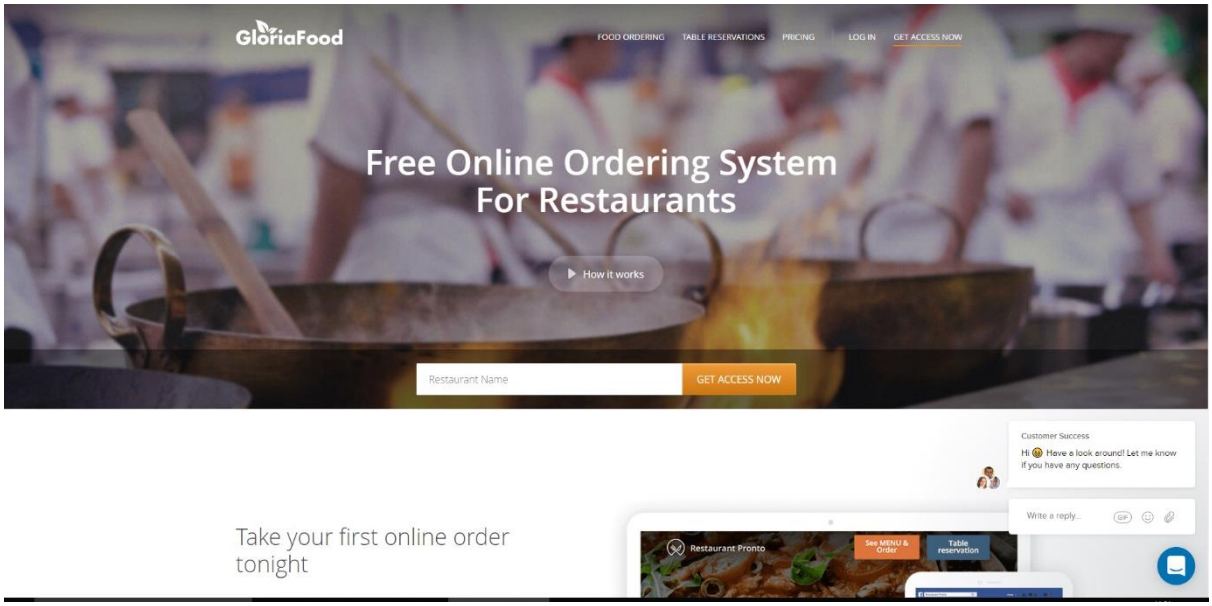
function imported the required third part libraries such as pisa and xhtml2pdf. I passed in the data and html templates that would be used to design and structure the pdf file, into the function to make my solution more efficient, and more reusable which is one of the core principles of OOP.

10. I successfully implemented this feature very well. The products can be ranked by many different attributes and variables, which are shown below:
    This feature was quite challenging for me to implement in my system as I had to calculate and

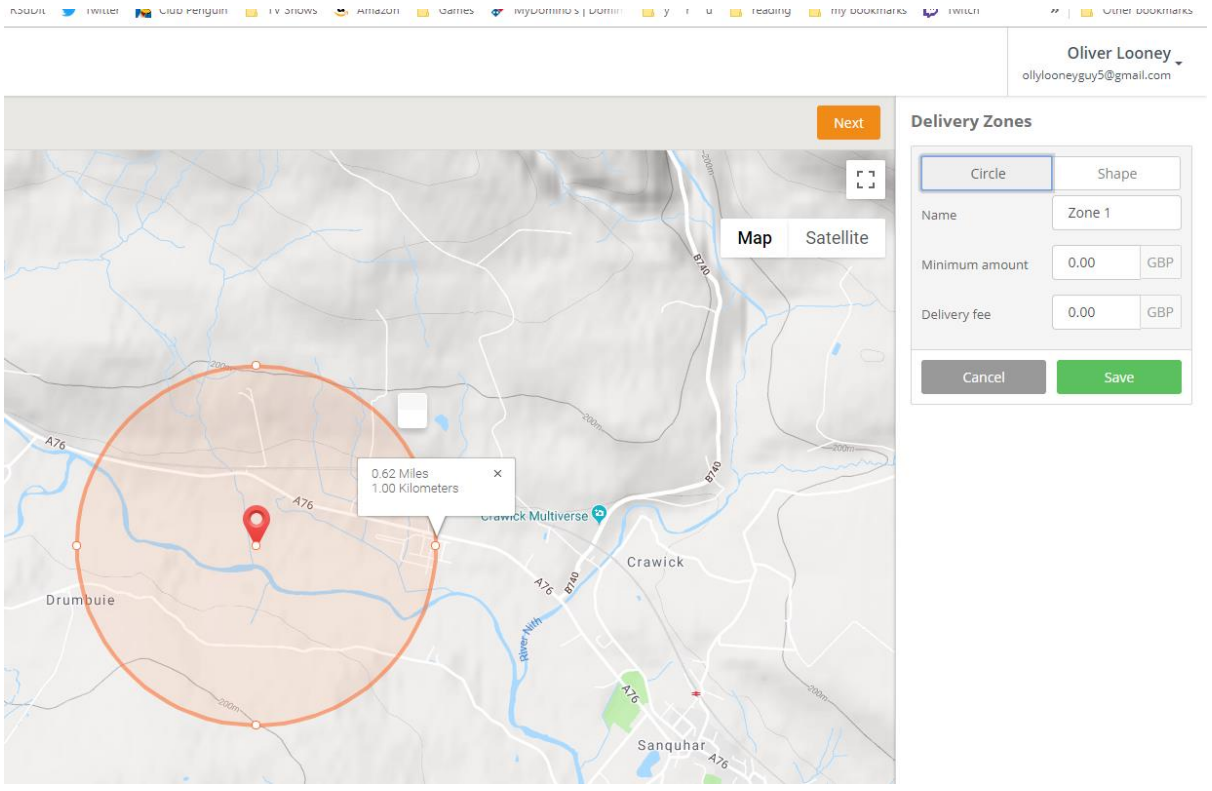


rank the products by their popularity, which I have explained the complexity and how I did that in the development testing chapter. To summaries it, I had to create two custom classes, one that used a recursive bubble sort algorithm and another which is a child class of the sorting class, which finds the popularity. It uses inheritance to call of the sorting methods of the parent class. This was a very high level and advanced solution to the problem, so I can justify that this was a very good feature and better than the commercially available systems that I investigated.





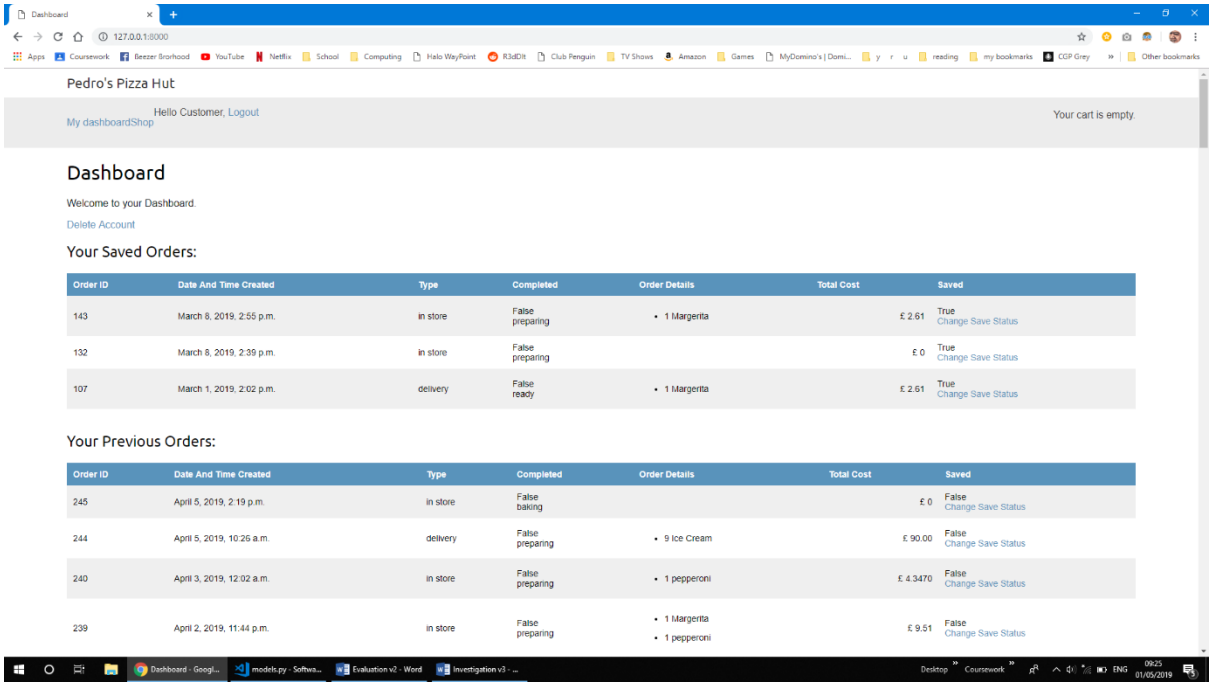
11. I did not implement this feature as I deemed it was unnecessary. However I did implement a product search for the customer to use & interact with. This is shown below at the end of this section. One way I could improve this was to change the search so that you would not have to have the exact spelling, or an auto complete as you enter in the search terms. This solution however, works perfectly and so I can justify it as a good feature. Above is a search from a commercially available system that I investigated.
12. I could not implement this heat map graph as it was too advanced for me in this time. In order to implement this I would have to import some high level python third party library that would generate this graph. This would've been an extremely helpful and useful feature for the manager, so the fact that I have failed to implement this is a big shortcoming. Below is an



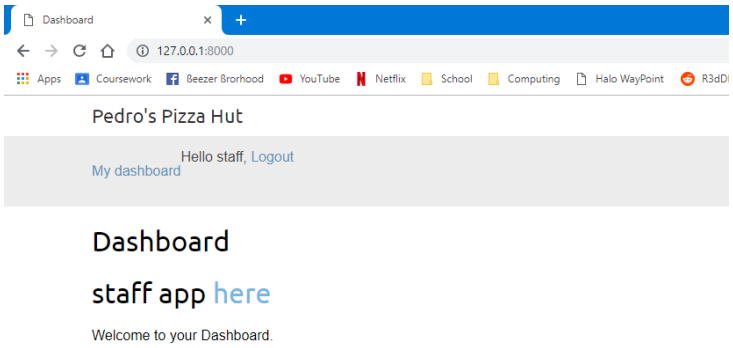
example of a commercially available system that I investigated that had successfully implemented this feature:

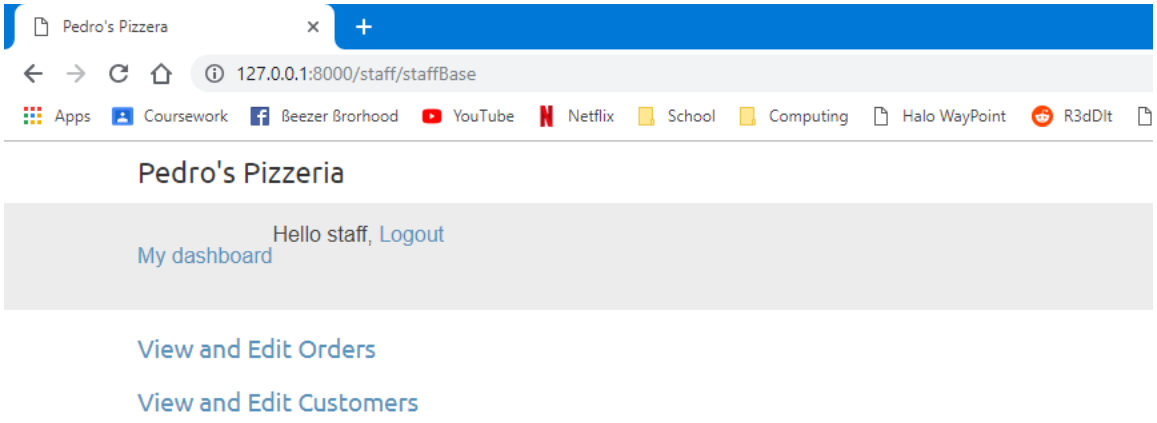
13. I successfully implanted a dashboard for every single different level of access of my system. I shall show each of these different dashboards of my system below. This feature makes navigating my system very easy, and second nature to new nature. Thanks to this, I can justify that this is a good feature. However, it is not as good as other commercially systems that dashboards. This is because the commercially available systems have more data on their dashboards.
    Each dashboard is rendered by a view function in the views.py file of every Django app that has a dashboard. Each view function renders a html template file to create the dashboard. My dashboards:

Customer:

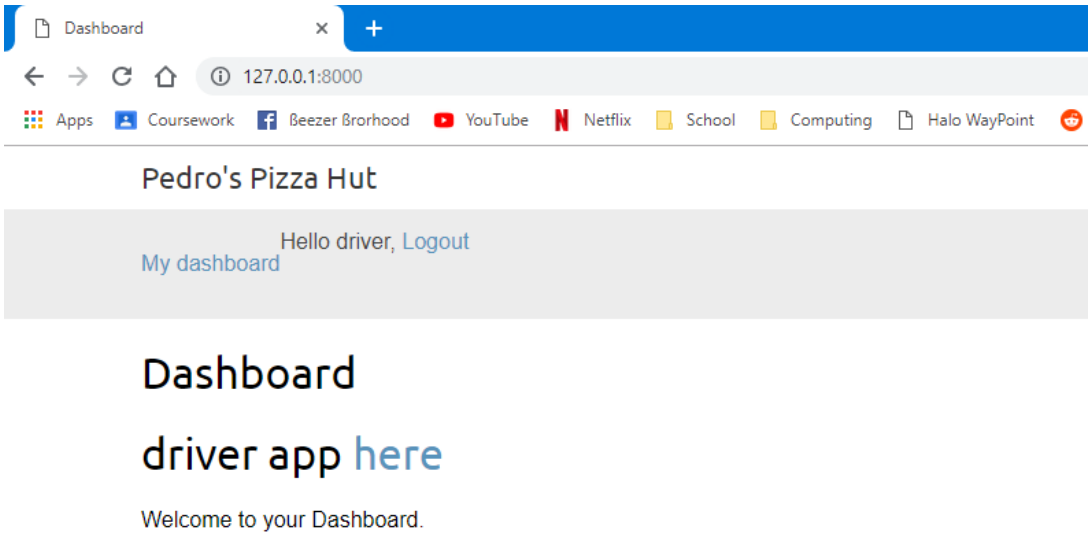


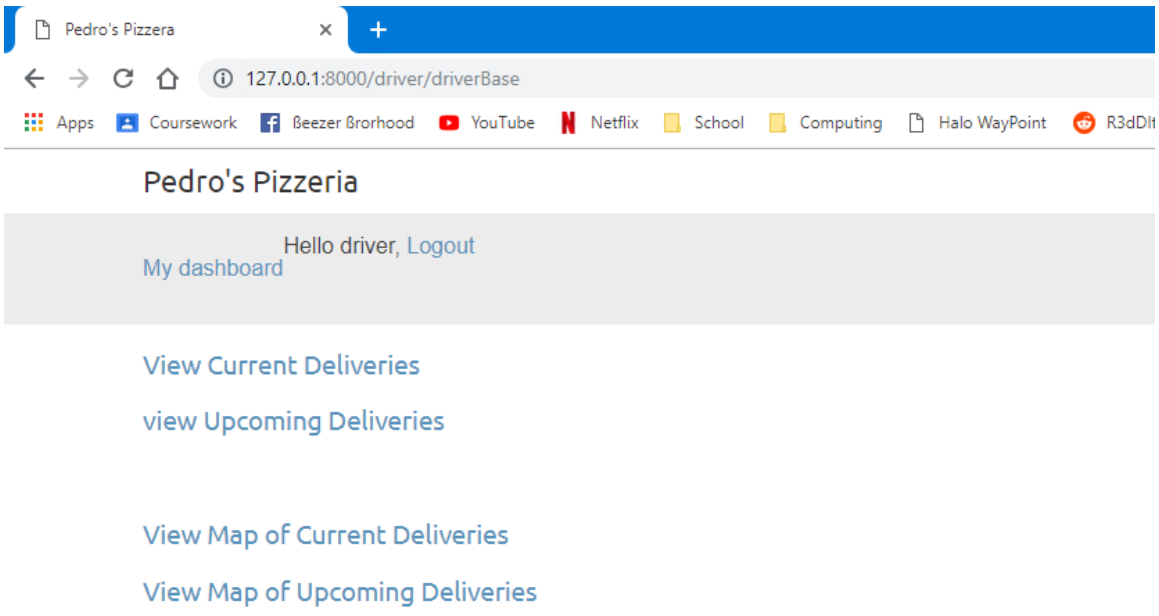
Staff:



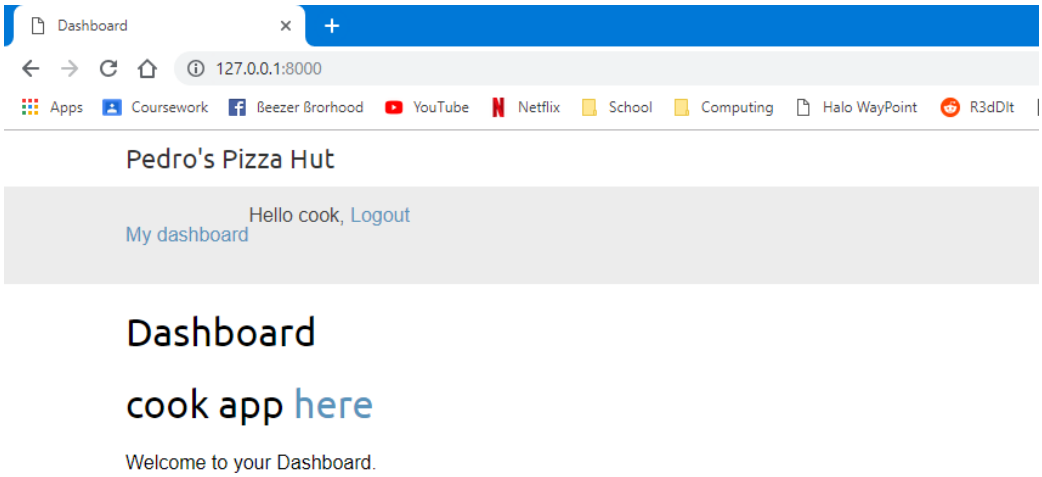

Pedro's Pizzera
127.0.0.1:8000/staff/staffBase
Apps
Coursework
Beezer Brorhood
YouTube
Netflix
School
Computing
Halo WayPoint
R3dDlt
Pedro's Pizzeria
Hello staff, Logout
My dashboard
View and Edit Orders
View and Edit Customers

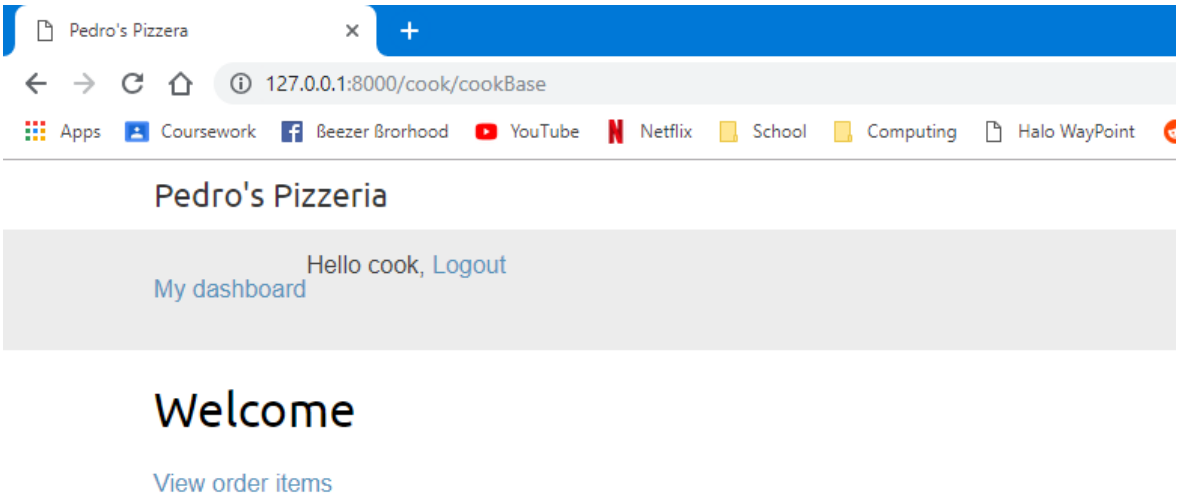
Driver:


Dashboard
127.0.0.1:8000
Apps
Coursework
Beezer Brorhood
YouTube
Netflix
School
Computing
Halo WayPoint
Pedro's Pizza Hut
Hello driver, Logout
My dashboard
Dashboard
driver app here
Welcome to your Dashboard.

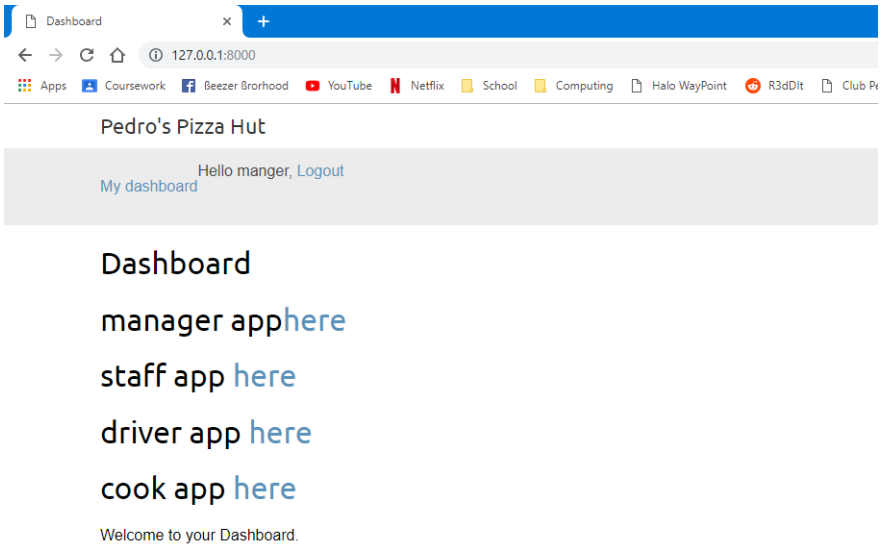

Pedro's Pizzera
127.0.0.1:8000/driver/driverBase
Apps
Coursework
Beezer Brorhood
YouTube
Netflix
School
Computing
Halo WayPoint
R3dDlt
Pedro's Pizzeria
Hello driver, Logout
My dashboard
View Current Deliveries
view Upcoming Deliveries
View Map of Current Deliveries
View Map of Upcoming Deliveries

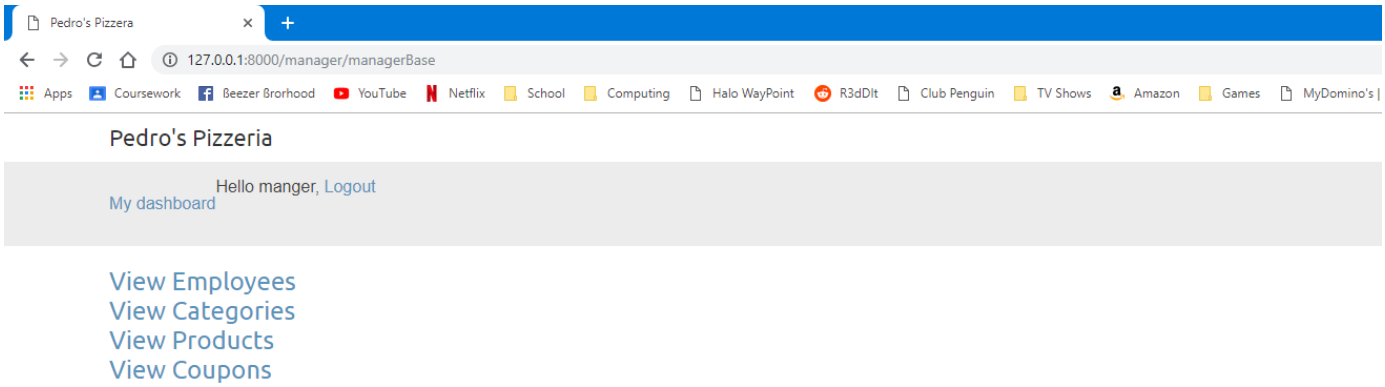
Cook:


Dashboard
127.0.0.1:8000
Apps
Coursework
Beezer Brorhood
YouTube
Netflix
School
Computing
Halo WayPoint
R3dDIt
Pedro's Pizza Hut
Hello cook, Logout
My dashboard
Dashboard
cook app here
Welcome to your Dashboard.

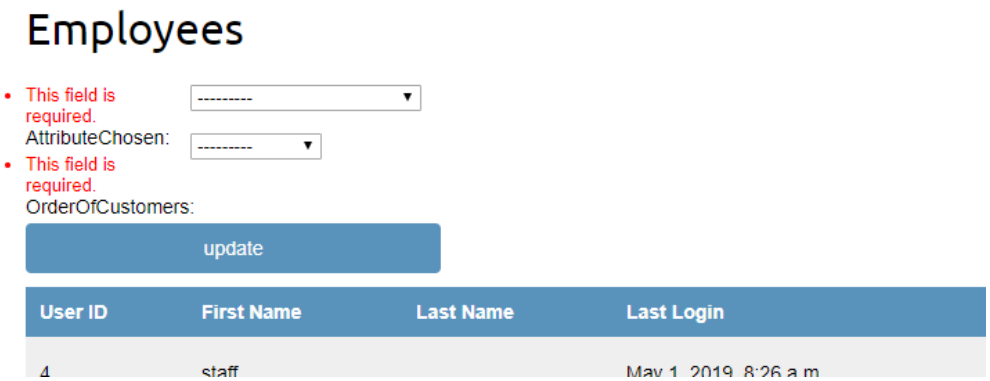

Pedro's Pizzera
127.0.0.1:8000/cook/cookBase
Apps
Coursework
Beezer Brorhood
YouTube
Netflix
School
Computing
Halo WayPoint
Pedro's Pizzeria
Hello cook, Logout
My dashboard
Welcome
View order items

Manager:





14. I successfully implemented this feature multiple times throughout my system. I had completed this with more options, such as ranking, than the commercially available systems. Due to this, I

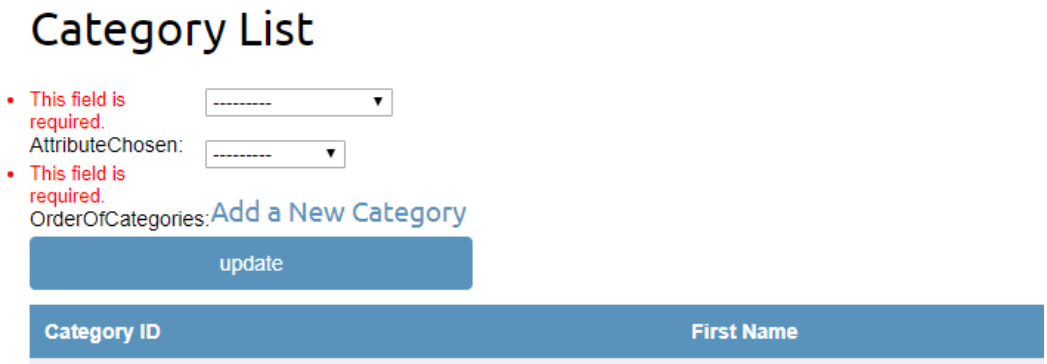


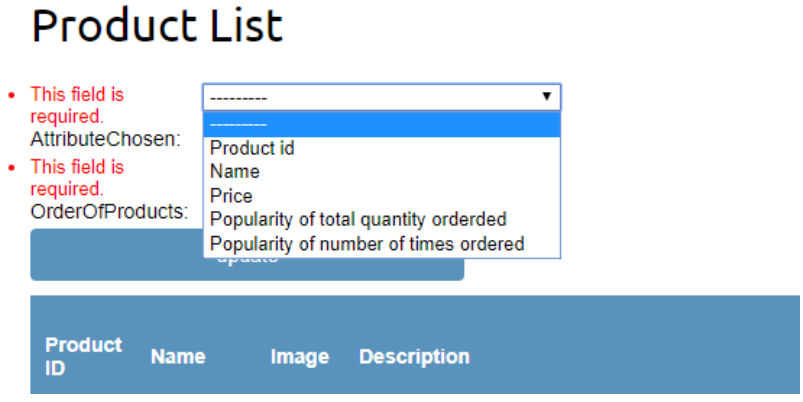
can justify that this was a good feature and that it performed better in my system than on the commercially available systems.

This was just a small sample of implementations of search in my system. I have the search functions in the views.py file of each Django app that requires a search. I implemented this search by simply using a .get operation on the object/class that I am trying to search for, below is an example of how I did this:

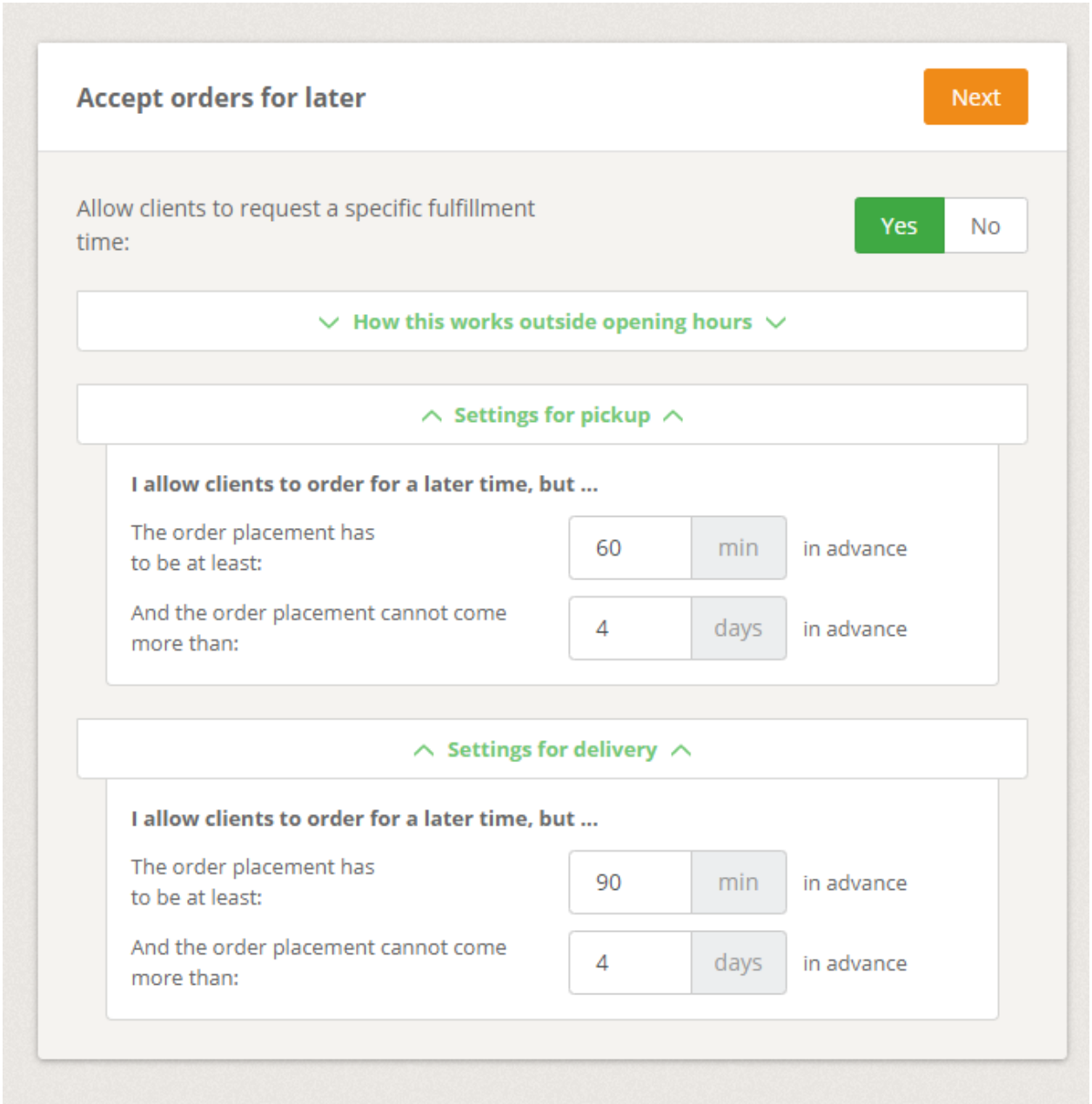
```
if cd["attributeChosen"] == '0':
                                products = Product.objects.get(available =
True, id = cd['searchCriteria'])
                            elif cd["attributeChosen"] == '1':
                                products = Product.objects.get(available =
True, name = cd['searchCriteria'])
```

However, for some searches/ rankings of certain tables I required a more complex search than simply retrieving a specific record form a table in my database. For these searches/rankings I had to create my own sorting class and popularity class myself. This required me to create a complex recursive bubble sort in the methods of this class. I also had to use inheritance to be able to use these sorting methods in the popularity class. This was very complicated and high level, and was explained in more detail is the development testing chapter.





15. I did not implement this objective at all. This was trivial feature so I just left it out. This was a shortcoming but not too major as it does not affect the basic functionality of the site. Below is a third party implementation of this feature.



Good features, Shortcomings & Improvements

For this section, I am going to use my user requirements from the investigation chapter to show the good features and shortcomings.

Customer

Number	Objective/ Success Criteria	Priority	Good feature or shortcoming
1.	Must be able to create, edit and delete their account	High	Good feature
2.	Must be able to save their previous orders so that they can quickly re order them in the future	High	Good feature
3.	Must be able to view the menu	High	Good feature
4.	Must be able to edit the quantities of each item before the order is saved	High	Good feature
5.	If an item is not available, the customer must not be able to order it	High	Good feature
6.	Must be told if an item is not able to be ordered and why	Low	Good feature
7.	Must be able to quickly login using a barcode printed by the store	Medium	Shortcoming
8.	Must have the option of adding extra items such as dips while in the checkout	Medium	Shortcoming
9.	They should by recommended other products	High	Good feature
10.	Must be able to order online, not just in the store	High	Good feature
11.	Must be able to see the status of their order as it is being prepared	High	Good feature
12.	Must be able to track their delivery driver	Medium	Shortcoming
13.	Must have an accurate estimation of how long it will take to order and deliver	Medium	Shortcoming
14.	If the restaurant is closed, the customers should not be able to order and should instead be given the option of pre-ordering for when the restaurant is open again	Medium	Shortcoming
15.	The customer should have the option of viewing all of their previous orders	High	Good feature
16.	The customer should be able to see a leader board of the most popular items in each category	Low	Shortcoming
17.	The customer should be able to easily see the opening, delivery, collection and closing hours of the restaurant	Medium	Shortcoming
18.	Must be able to easily access any discount offers from the restaurant, e.g.: receive email or SMS offers	High	Good feature
19.	Must be notified on progress on their order and when it is ready	High	Shortcoming
20.	If it is a collection they should receive a notification to collect their order	High	Shortcoming
21.	If it is a delivery they should receive a notification that it is now being delivered and then they should be able to track their driver	High	Shortcoming

22.	Must be able to view all the products at once or view them by category	High	Good feature
23.	Must be able to search for a specific product	High	Good feature
24.	Must be able to access a product detail page for each product	High	Good feature
25.	Must be able to add multiple products of varying quantities to the cart	High	Good feature
26.	Must have a cart	High	Good feature
27.	Must be recommended other products on the product detail page that other customer's have bought with the product currently being viewed	High	Good feature
28.	Must be recommended other products on the cart detail page based off what products the customer is ordering	High	Good feature
29.	Must be able to remove items from the cart	High	Good feature
30.	Must be able to edit the quantities of items in the cart	High	Good feature
31.	Must be able to go to the product detail page of a product from the cart	Medium	Good feature
32.	Must be able to apply a coupon code in the cart	High	Good feature
33.	Must be able to continue shopping after looking at the cart	High	Good feature
34.	Must be able to go to the checkout after looking at the cart	High	Good feature
35.	As all times the customer must have easy access to the dashboard, shop and cart	High	Good feature
36.	Must be able to view a summary of their order in the checkout	Low	Good feature
37.	Must be able to chose the type of order	High	Good feature
38.	Must be able to create orders	High	Good feature
39.	Must be taken to an order confirmation page after creating the order	High	Good feature
40.	Must see relative details at the confirmation page	High	Good feature
41.	Must be able to view a receipt in a pdf file	High	Good feature
42.	Must receive an email confirming their order	High	Good feature
43.	Must be able to login & logout	High	Good feature
44.	Must be able to view their details	Medium	Shortcoming
45.	Must be able to reset their password	Medium	Good feature

Counter Staff

Number	Objective/ Success Criteria	Priority	Good feature or shortcoming
46.	They must need permission from an admin account to activate their account	High	Good feature
47.	They should be able to create, edit and delete customer accounts for people over the phone	Medium	Shortcoming
48.	They should be able to create, edit and delete customer accounts for people over the phone	Medium	Shortcoming
49.	They should also be able to view the customer's saved orders	Medium	Shortcoming

50.	They should be able to see the status of the customer's order	High	Good feature
51.	They should be notified on its progress and when it is ready	High	Shortcoming
52.	They should be able to see how long each item takes and how long the customer's order should take	Medium	Shortcoming
53.	Any orders that they take should be associated with their account	High	Shortcoming
54.	They should be able to view their shift times	Low	Shortcoming
55.	They should receive an SMS message or email reminding them about their shift times the day before	Low	Shortcoming
56.	Should be able to view all of the orders, live order and previous orders	High	Good feature
57.	Should be able to view a pdf receipt of each order	High	Good feature
58.	Must be able to update the status of each order	High	Good feature
59.	Must be able to delete an order	High	Good feature
60.	Must be able to go to an order detail page for each order	High	Good feature
61.	Must be able to search & rank all orders	High	Good feature
62.	Must be able to view all the order items in an order	High	Good feature
63.	Must be able to update the status of each order item	High	Good feature
64.	Must be able to remove an order item	High	Good feature
65.	Must be able to view a pdf ticket for each order item	High	Good feature
66.	Must be able to relevant customer information in the order detail page	High	Good feature
67.	Must be able to view all customers	High	Good feature
68.	Must be able to search & rank customers	High	Good feature
69.	Must be able to delete customers	High	Good feature
70.	Must be able to go to a customer detail page for each customer	High	Good feature
71.	Must be able to see all of the customer's information on this page and all of their orders	High	Good feature
72.	All of the data must be able to be viewed as a pdf file or csv file	High	Good feature
73.	Some high level data processing on the customer must be shown here	High	Good feature
74.	Form the customer detail page the staff should be able to go to that order detail page	High	Good feature
75.	Must be able to view their details	Medium	Shortcoming
76.	Must be able to reset their password	Medium	Good feature

Kitchen Staff

Number	Objective/Success Criteria	Priority	Good feature or shortcoming
77.	They must need permission from an admin account to activate their account	High	Good feature

78.	They must be able to view how much of each ingredient that they have left	Medium	
79.	They must be able to update the status for the order part that they are responsible for. They should not be able to affect the status of the whole order, as the whole order is only ready when all of the order parts are	High	Good feature
80.	They should be able to view & print out the stickers for the boxes. They should not be able to access the data, the system should access it and create the sticker. The sticker should include: OrderID, Box number/ number of boxes, type(in store, delivery or collection), relevant timestamps, CustomerName, subtotal and TotalCost. The stickers will be pdfs	High	Good feature
81.	They should be able to change the quantities of each ingredient. This is to keep the system accurate. Whenever an order is placed the system will automatically subtract the quantity of ingredient used but, if there is an accident, the kitchen staff may need to update it	High	Shortcoming
82.	When the kitchen staff finish an order, they must update the system which will notify the counter staff, driver or customer	High	Shortcoming
83.	The system should analyse all of the previous orders and the time and day, and from that be able to roughly predict to the kitchen staff how much of each item they will have to make. This allows the system to be more accurate when tracking how much of each ingredient the restaurant needs for the next day or week	Medium	Shortcoming
84.	They should be able to view their shift times	Medium	Shortcoming
85.	They should receive an SMS message or email reminding them about their shift times the day before	Medium	Shortcoming
86.	Must be able to view their details	Medium	
87.	Must be able to reset their password	Medium	Good feature
88.	Must be able to view all order items	High	Good feature
89.	Must be able to search and rank these order items	High	Good feature

Driver

Number	Objective/Success Criteria	Priority	Good feature or shortcoming
90.	They must need permission from an admin account to activate their account	High	Good feature
91.	They must receive a notification whenever an order is ready for them to deliver	Medium	Shortcoming
92.	They should be able to view all of the contact information on the customer whom they are delivering to. They should only have access while they are delivering	High	Good feature

93.	They should be able to view the entire order and all of the order part, to make sure that they have everything. They should only have access while they are delivering	High	Good feature
94.	When the driver begins the delivery they should put in the time into the system and again when they finish, so that the delivery times can be tracked. If the driver starts driving without inputting the time, thy should get a notification to do so. If they do not then an error message should be saved instead	Medium	Shortcoming
95.	The driver should see a map on the system with an overlay of the customer's addresses	Medium	Shortcoming
96.	They should be able to view their shift times	Low	Shortcoming
97.	They should receive an SMS message or email reminding them about their shift times the day before	Low	Shortcoming
98.	Must be able to view their details	Medium	
99.	Must be able to reset their password	Medium	Good feature
100.	Must be able to view a pdf file of the order receipt	High	Good feature

Manager

Number	Objective/Success Criteria	Priority	Good feature or shortcoming
101.	The manager must be able to enable or disable accounts	High	Good feature
102.	Whenever a member of staff is creating an account they must have their account enabled by the manager before they can do anything, while the customer's account should be automatically enabled	High	Shortcoming
103.	The manager must be able to edit the opening, delivery, pick up and closing times for each day	Low	Shortcoming
104.	The manager should be able to hide items on the menu from the customer if they do not want that item to be ordered but do not want the hassle of deleting the record temporarily	Medium	Good feature
105.	The manager should have the option of hiding how long it takes to prepare items or how long it takes to deliver, from the customer. However, the staff should still see it so that they can improve to the target	Low	Shortcoming
106.	The manager should set the target times for preparing food in the menu table and delivery times in the delivery zone table. Not everything in the menu table will be visible by the customer	Medium	Shortcoming
107.	The admin should be able to create delivery zones that have different delivery prices, e.g.: within x miles of the restaurant is costs £y	Medium	Shortcoming
108.	The admin should also be able to view a heatmap of the local area, that is overlaid with customers' addresses. This should give the admin the option to show the intensity of the heatmap based off just number of customers, number of orders or the total cost of the orders. The admin should	Medium	Shortcoming

	also be able to view just collections, deliveries, instore, all or any combination they wish to view		
109.	The admin should also be able to see how long it takes to prepare each item on average and how long it takes it deliver on average	Medium	Shortcoming
110.	The admin should be able to set the target times for each delivery zone	Low	Shortcoming
111.	The admin should be able to set the target times for preparing each item	Low	Shortcoming
112.	The admin should also be able to create a custom SMS or email message that would be sent out to any or a combination customers, drivers, kitchen staff or counter staff. The admin should also have the option of sending out SMS messages or emails to people of just a specific attribute, e.g: customers over a certain age or drivers who are working at a certain time	Medium	Shortcoming
113.	The admin should be able to save next weeks shifts in the system	Low	Shortcoming
114.	The admin should receive a report at the end of every day and week showing the total sales instore, delivery and collection for that week. They should also see how much revenue they generated and how many new customers joined	Medium	Shortcoming
115.	For each graph or report that the admin can create/view, they should be able to save it as a PDF, and then use the system to email it out to any members of staff or customers that they may want to	High	Shortcoming
116.	The manager must be able to view all the employees	High	Good feature
117.	The manager must be able to delete employees in this page	High	Good feature
118.	The manager must be able to change access levels in this page for each employee	High	Good feature
119.	The manager must be able to search/rank the employees in the page	High	Good feature
120.	The manager must be able to view all the categories in a view categories page	High	Good feature
121.	The manager must be able to delete categories in this page	High	Good feature
122.	The manager must be able to search/rank categories in this page	High	Good feature
123.	The manager must be able to add a new category in this page	High	Good feature
124.	The manager must be able to view all products in a view products page	High	Good feature
125.	The manager must be able to hide products in this page	High	Good feature
126.	The manager must be able to delete products in this page	High	Good feature
127.	The manager must be able to search/ rank products in this page	High	Good feature

128.	The manager must be able to add a new product in this page	High	Good feature
129.	The manager must be able to view all the coupons in a view coupons page	High	Good feature
130.	The manager must be able to turn on/off the coupons in this page	High	Good feature
131.	The manager must be able to delete coupons in this page	High	Good feature
132.	The manager must be able to search/ rank coupons in this page	High	Good feature
133.	The manager must be able to add a new coupon in this page	High	Good feature
134.	The manager must also have access to all of the other staff's apps	High	Good feature

Improvements:

Implement a whole Django app that is just dedicated to keeping track of how long different operations take. This app would include tracking how long order items take, the whole order and the deliveries. This app would be responsible for using libraries such as chart.js to create graphed output and other high level data processing with the data collected for the manager to view in the manager app.

Another improvement would be to replace and view functions that are repetitive and could be combined with an abstract class such as the one I created for sorting objects by popularity. This would optimise my code, make it more reusable, easier to understand and maintain.

Another improvement would be to get batch emails to work. A problem I oversee is that the server may be held up by this slowing down the system as a whole, as the hardware will be a limiting facter, i.e: the hardware used will dictate how fast Django can send emails. A way around this would be to use the libraries RabbitMQ and Celery, which manage the server side of issues like this by dealing with this on their own server, allowing the system to run as normal.

Personal Strengths & Weaknesses

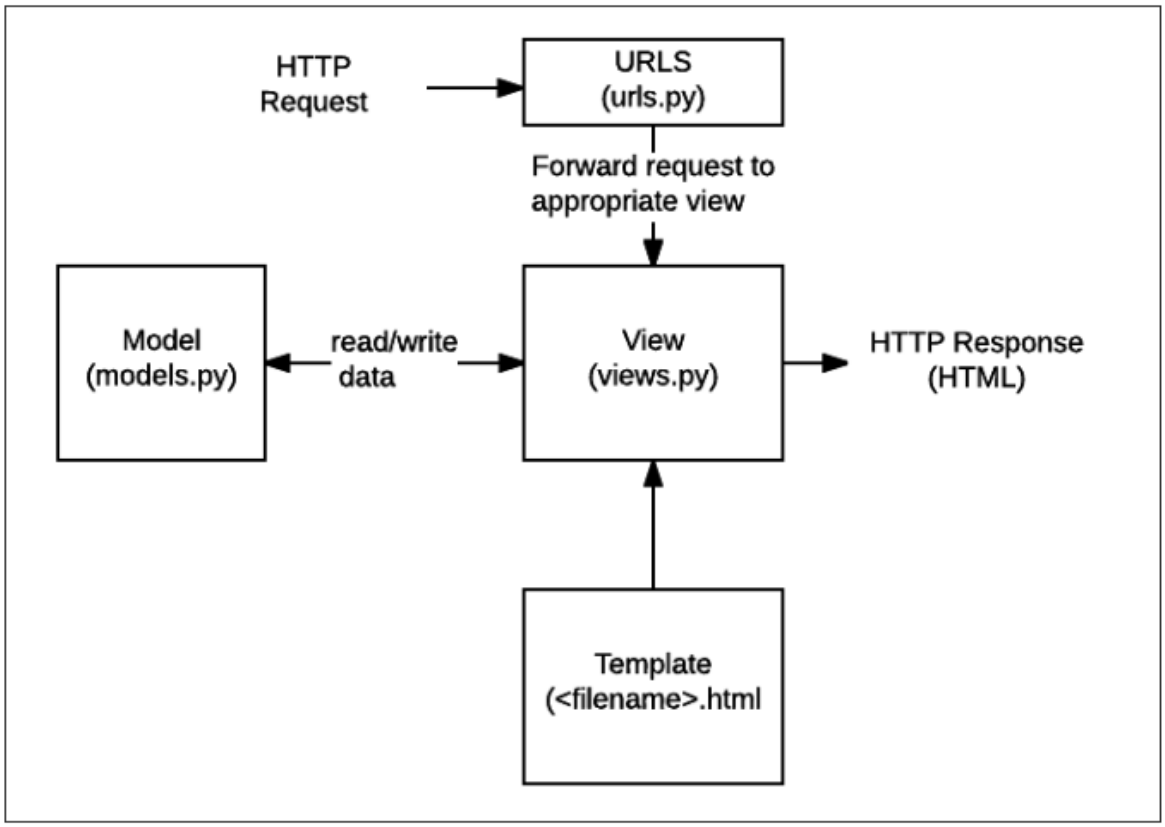
I encountered a lot of problems and learnt a lot of new skills while doing this coursework. When I began I did not know how to use OOP, but now I have lots of experience in this paradigm, I fully see and understand it's benefits and why it is used over functional, and I agree. I also learnt about the value of virtual environments after I had to deal with a lot of problems with different code on my machine requiring different versions of third party libraries. At its core, the main purpose of Python **virtual environments** is to create an isolated **environment** for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has. I started this coursework eight months ago, and now I have finished it. I have learnt a lot and gained a lot of valuable skills, especially in object-oriented programming and building Django apps. The main problems that I encountered as I did this was that pretty much every time, I went to do something, it was my first timer doing it, whether it was a new object, function or write up. This coursework had a very steep learning curve, which by the end I successfully overcame.

Once I had learnt how to do something however, for example making a Django app or generating a PDF, it was very straightforward to so that operation or task in the future.

I spent a long time working on my database structure and designing it, before I implemented it in third normal form. This led me to have very little redundant data and no many to many relationships, which in turn led to my database having strong data integrity, data consistency and referential integrity.

Throughout this coursework, I used a wide range of data structures including string variables, integers, Boolean variables, arrays, tuples and dictionaries to pass object(s) through to the html template files, to render the pages.

I also had very little experience with web based applications, but after using Django, a python web framework, I now know how to create websites using python. I have a deeper understanding of how a website sowrks after using Django. I now understand the relationships between html template files, the



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in an URL, and pass these to a view function as data.
- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

urls.py file, the views.py file and the models of a Django app, and how they come and work together to give the user a functioning website.

The main problems I encountered were first learning how to use OOP and Django. For example I struggled with using classes and objects efficiently and effectively at the start, as I was used to functional programming. Also, at the start is struggled with understand how Django worked, but now I fully understand it.

Another set of problems I encountered were third party libraries. I had a lot of trouble with using some high level libraries such as Redis & WeasyPrint, which I have already explained earlier and in development testing.

Some of my weaknesses were time management. I mismanaged my time early on but thankfully I worked very hard at the end to rectify this. I have learnt my lesson about being disciplined and working consistently on a project over a long period of time. My skills as a developer have greatly increased over these eight months of coursework, far beyond my expectations.

What would I do differently?

Analysis:

For my fact finding I did an investigation and discussion. Two questions are asked at this stage: How does the current system work? And what is required of a new system? I obtained answers to these questions through Interviews, Questionnaires, examining documentation and observing stakeholders. To improve I would just do more detail, and increase the number of stakeholder's I interviewed and observed.

Design:

In the design chapter I designed the data models, algorithms, modularity of code, UML diagrams, plans for classes, functions and interfaces. Now that I have more knowledge and skills, I would be able to more accurately design the data models, algorithms and classes. Also, I would go more in depth when designing the mock ups for the user interfaces, as that would've been more helpful. I would also have more UML & DFD diagrams as they are extremely helpful for understanding what the system is going to do. I would also plan more modularity of my code so I would know before I start coding hwo much code I will be able to reuse.

Prototype:

I would add multiple cycles or iterations to the prototype instead of just doing it once. I would like to use AGILE or RAD development, and at the end of each cycle or iteration bring the stakeholders in to have a look and get some feedback, so that I know what I am doing is what they want.

Software Development:

This could be streamlined by developing individual apps completely on their own and then integrating them together near the end. I developed a user interfaces using view functions that render a html template page. I did not need to build generic SQL, as Django handled that for me. Creating and importing scripts was fine, but I did have a problem with importing. I had to work around a circular import error, which I explained in the development testing. With Django, the database was built once I created the user models, which I had designed in my design chapter.

I could have refactored my code with some of my view functions, into abstract classes. Below are some examples of what I think could have been refactored.

```
#these functions change the access level by changing th bool value in the db
def changeAccessStaff(request, employee_id):
    employee = MyUser.objects.get(id = employee_id)
    if employee.is_staff == True:
        employee.is_staff = False
    else:
        employee.is_staff = True

    employee.save()
    return viewEmployees(request)

def changeAvailablity(request, product_id):
    product = Product.objects.get(id = product_id)

    if product.available == True:
        product.available = False
    else:
        product.available = True

    product.save()
    return viewproducts(request)

def changeAccessCook(request, employee_id):
    employee = MyUser.objects.get(id = employee_id)
    if employee.is_cook == True:
        employee.is_cook = False
    else:
        employee.is_cook = True

    employee.save()
    return viewEmployees(request)

def changeAccessDriver(request, employee_id):
    employee = MyUser.objects.get(id = employee_id)
    if employee.is_driver == True:
        employee.is_driver = False
    else:
        employee.is_driver= True

    employee.save()
    return viewEmployees(request)
```

I think the above functions could have been refactored into an abstract class, as they are quite repetitive.

These pdf generator view functions could have been refactored into an abstract class as well. This is because there is very little difference in them, and they are all doing the same abstract operation

```
#these last functions each generate a seperate pdf file from using a html
template
def generate_pdf_recipt(request, order_id, *args, **kwargs,):
    order = Order.objects.get(id = order_id)
    customer = MyUser.objects.get(id = order.customer_id)

    pdf = render_to_pdf('orders/pdf/pdf_reciept.html',
                        {'order': order,
                        'customer': customer})
    return HttpResponse(pdf, content_type='application/pdf')

def generate_pdf_ticket(request, item_id, *args, **kwargs,):
    item = OrderItem.objects.get(id = item_id)

    pdf = render_to_pdf('staff/pdfs/item_ticket.html',
                        {'item': item})
    return HttpResponse(pdf, content_type='application/pdf')

def generate_pdf_customerOrdersInfo(request, customer_id, *args, **kwargs,):
    customer = MyUser.objects.get(id = customer_id)
    orders = Order.objects.filter(customer_id = customer.id)

    pdf = render_to_pdf('staff/pdfs/customer_orders_pdf.html',
                        {'customer': customer,
                        'orders': orders})
    return HttpResponse(pdf, content_type='application/pdf')

def generate_pdf_customerDetails(request, customer_id, *args, **kwargs,):
    customer = MyUser.objects.get(id = customer_id)

    pdf = render_to_pdf('staff/pdfs/customerDetailspdf.html',
                        {'customer': customer,})
    return HttpResponse(pdf, content_type='application/pdf')

def generate_pdf_cusOrdersAndOrderItems(request, customer_id, *args,
**kwargs,):
    customer = MyUser.objects.get(id = customer_id)
    orders = Order.objects.filter(customer_id = customer.id)

    pdf = render_to_pdf('staff/pdfs/cusOrdersAndOrderItemsPDF.html',
                        {'customer': customer,
                        'orders': orders})
    return HttpResponse(pdf, content_type='application/pdf')

def generate_pdf_allCusInfo(request, customer_id, *args, **kwargs,):
```

```
    customer = MyUser.objects.get(id = customer_id)
    orders = Order.objects.filter(customer_id = customer.id)

    pdf = render_to_pdf('staff/pdfs/allcusInfoPDF.html',
                        {'customer': customer,
                        'orders': orders})
    return HttpResponse(pdf, content_type='application/pdf')
```

Testing:

My test data was very effective, where I could, I used valid, erroneous, null and extreme data to fully test my system. My approach to testing was user testing, I tested my user requirements and then for reach user requirement, where possible, I broke it down into more tests to make sure I fully tested my system. I caught quite a few failed tests, so my data was very effective and my program was very robust in catching errors.

I could have improved my testing by using Django automated tests. This is where you write a test function and then you can call the tests file, which runs all the tests, every time you make a change to your code, making sure that you catch and find bugs as soon as you create them, if you do. If I were to do this again, I would write these Django tests as I programmed my solution so I could make use of this.