

Post - Prototype

## Part One - Feedback

I choose to ask two of my classmates to complete my questionnaire as they are aware of the methods that I am using and are designing similar systems so will be able to give good feedback on what is wrong and give good options on what can be done through their experience with their own solutions.

## Part Two – Describe Feedback

Questions:

- 1) Are there any redundancies or problems in the data structures?
- 2) Is there any data I have neglected to store that should be stored?
- 3) Is there enough validation?
- 4) What do you think of the UI?
- 5) What do you think of the security?
- 6) What do you think of the inputs & outputs?
- 7) Are there any other features that I should implement that could improve my system?

Answers:

- 1) Student A:  
Yes, you should have the personal information in a different table from the access related attributes, username & password.  
Student B:  
I think agree that there should be a separate access table.
  - I agree with this feedback, this was also my original plan from my design chapter, but I took a shortcut just for the prototype
- 2) Student A:  
Yes, you need to be able to order different sizes for most products  
Student B:  
Yes, you should also store the user's gender so that you can make the automated emails more personal for example
  - I agree with both of their feedback. I shall have to change two of my data structures to implement this. This will be shown below.
- 3) Student A & B:  
No, the only validation is the built in Django validation.
  - I agree with this, but I did not implement validation as it was only a prototype. My validation algorithms are in my design chapter
- 4) Student A:  
I think it is good but a bit clumsy when it comes to the dashboard. The dashboard should have more features.  
Student B:  
It is mostly good however you should make it easier to navigate between sections. I think you should let the user set their own custom colours

- I agree with student B about the navigation, I shall remedy this by adding a simple paragraph element to each html template file. To further improve upon this I shall make these links to the previous pages that you visited. I shall not be letting the users set their own custom colours as this is not necessary.
- I also agree with student A about the dashboard. The dashboard is like this just because it is the prototype. I have planned to have features here in the investigation & design chapters.

5) Student A & B:

The security is very good, as Django handles the password encryption. However, there is no way to gain access to an account when you forget your password. Also, currently there is a bug where any customer can make an order for any other customer.

- I agree with this feedback. I did not have account recovery in the prototype as it was not necessary, but it will be in the final solution. The bug was caused by a problem I had with Django forms.

6) Student A:

I think they are good enough for the prototype.

Student B:

I think they could be improved but this would just be the layout of them

- I agree with their feedback but it is mostly the design of the inputs and outputs that they have, not any algorithmic problems so I shall just leave that for the final solution.

7) Student A:

You should be able to track your order on a simple, basic IOS or Android app. Also You should do away with some of the more high level advanced features, as you are running out of time, for example: the heatmap and closing times just seem unrealistic at this point.

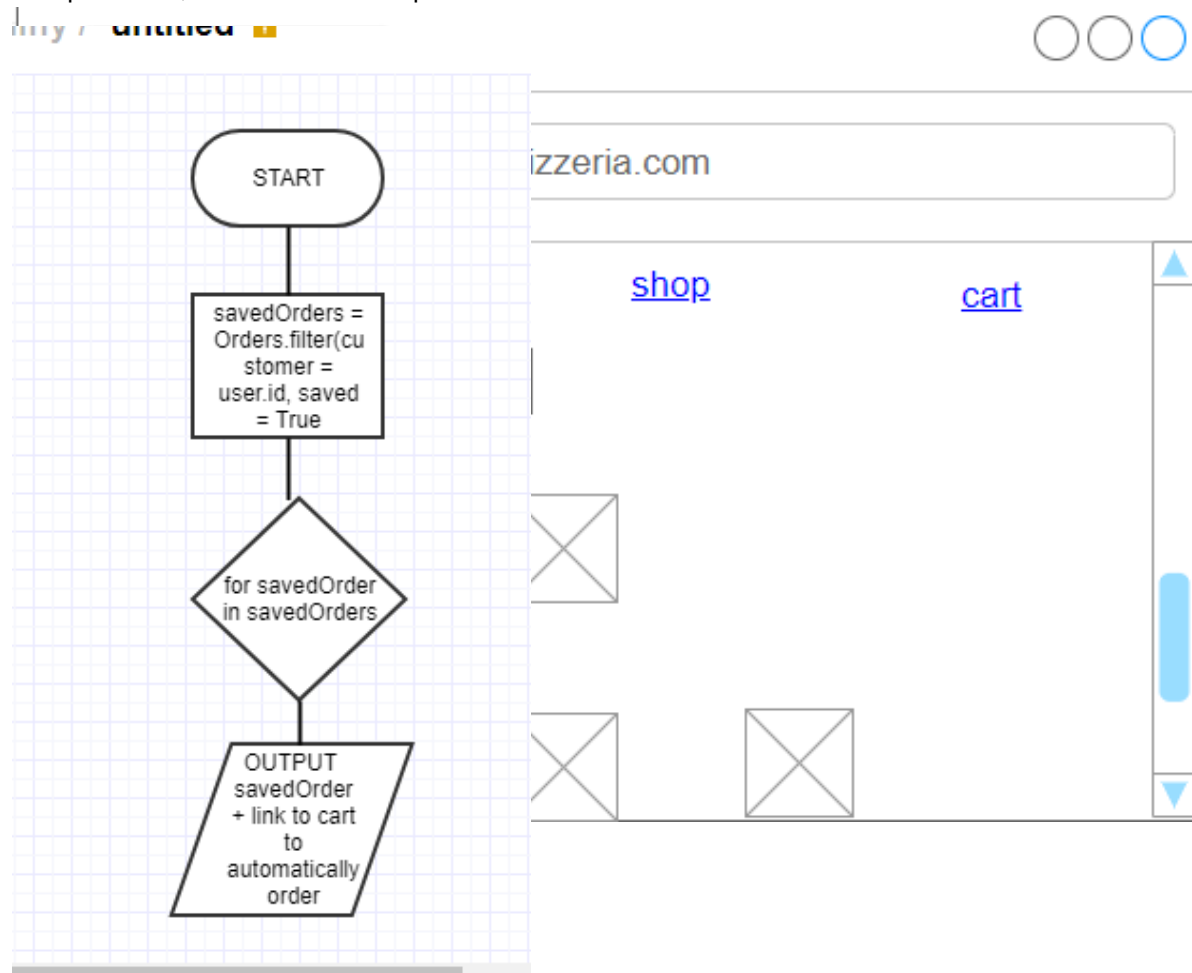
Student B:

I think you could improve your system by adding in a coupon system into the system. The manager should be able to ass a new coupon, and the customer's should be notified that a new coupon has been added. This coupon system should be integrated with the system seamlessly, so that it is not awkward for the customer to enter and apply a coupon.

- I agree with Student B's feedback, and as such I shall fully design it later on in this chapter, and then implement it in my code.
- I do not agree with Student A's first feedback as it is far beyond the scope of this project, and I do not have time for it. It would also require a new programming framework, so it is simply unrealistic.
- I agree with the second part of Student A's feedback as I do now realise some parts that I have planned will have to be left out, but this will be addressed in more detail in the testing and evaluation chapters.

## Changes

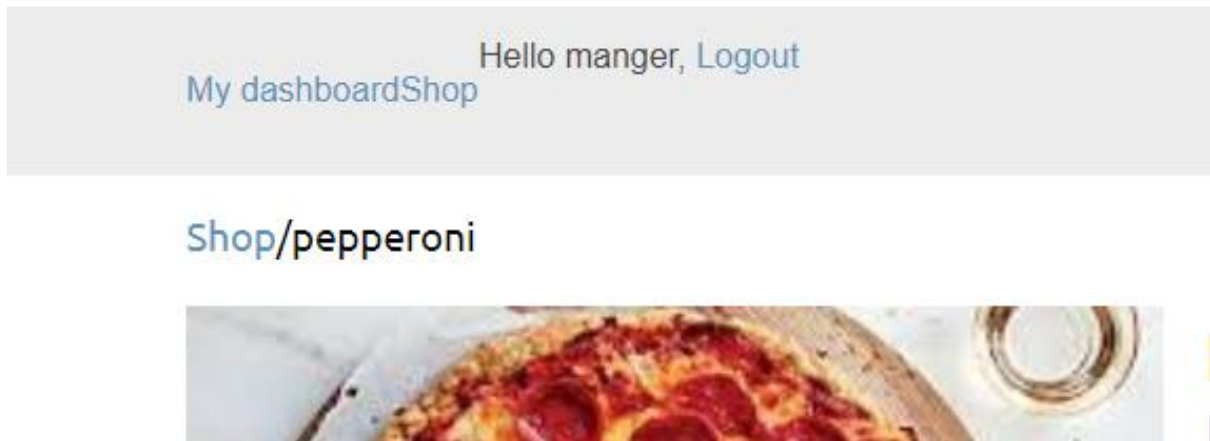
For question 4, I did a new mock up for the customer's dashboard.



For the second part of the feedback of question 3, I decided to go back and quickly add it in. Here is the code I added to the product html template file, line 9:

Which added this:

```
1 {% extends "shop/base.html" %}
2 {% load static %}
3
4 {% block title %}
5     {{ product.name }}
6 {% endblock %}
7
8 {% block content %}
9     <h3><a href = "/shop/">Shop</a>{{ product.name }}</h3>
10     <div class="product-detail">
11         
13         <h1>{{ product.name }}</h1>
14         <h2><a href="{% if product.category.get_absolute_url %}">{{ product.category }}</a></h2>
15         <p class="price">{{ product.price }}</p>
16         <form action="{% url "cart:cart_add" product.id %}" method="post">
```



This was simple to add and makes the site much easier to navigate.

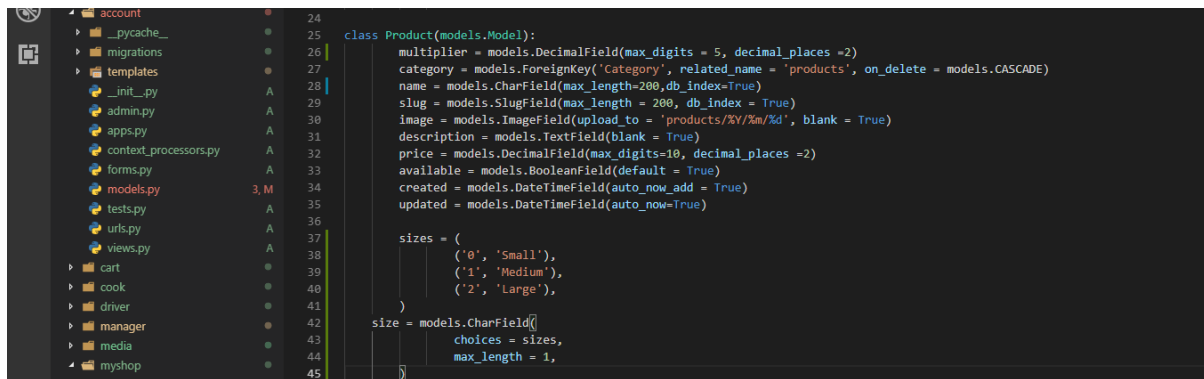
For question 2's feedback I had to change some of my data structures. I had to add a new Boolean attribute to the user table called gender. Here is the code for that, line 8:

```

File Edit Selection View Go Debug Terminal Help models.py - prototype -
EXPLORER
  OPEN EDITORS
    models.py account 3, M
  PROTOTYPE
    .vs
    .vscode
    account
    __pycache__
    migrations
    templates
    __init__.py
models.py x
1 from django.db import models
2 from django.contrib.auth.models import AbstractUser
3 import datetime
4 from django.urls import reverse
5
6
7 class MyUser(AbstractUser):
8     gender = models.BooleanField(default = True)
9     password = models.CharField(max_length=50)
10    username = models.CharField(max_length=50, unique = True)
11    is_superuser = models.BooleanField(default = False)
12    is_staff = models.BooleanField(default = False)

```

I shall also have to add a two new attributes to the product table. These will be size, which will be a charfield with the options chosen from a look up table. And a multiplier value to change the price depending on the size. Here is the code to add these attributes, line 16 & lines 37-45:



```
24
25
26 class Product(models.Model):
27     multiplier = models.DecimalField(max_digits=5, decimal_places=2)
28     category = models.ForeignKey('Category', related_name='products', on_delete=models.CASCADE)
29     name = models.CharField(max_length=200, db_index=True)
30     slug = models.SlugField(max_length=200, db_index=True)
31     image = models.ImageField(upload_to='products/%Y/%m/%d', blank=True)
32     description = models.TextField(blank=True)
33     price = models.DecimalField(max_digits=10, decimal_places=2)
34     available = models.BooleanField(default=True)
35     created = models.DateTimeField(auto_now_add=True)
36     updated = models.DateTimeField(auto_now=True)
37
38     sizes = (
39         ('0', 'Small'),
40         ('1', 'Medium'),
41         ('2', 'Large'),
42     )
43     size = models.CharField(
44         choices=sizes,
45         max_length=1,
```

To implement this further I will have to add in more algorithms, so I shall leave it to the final solution. An example of some of the new pseudocode required:

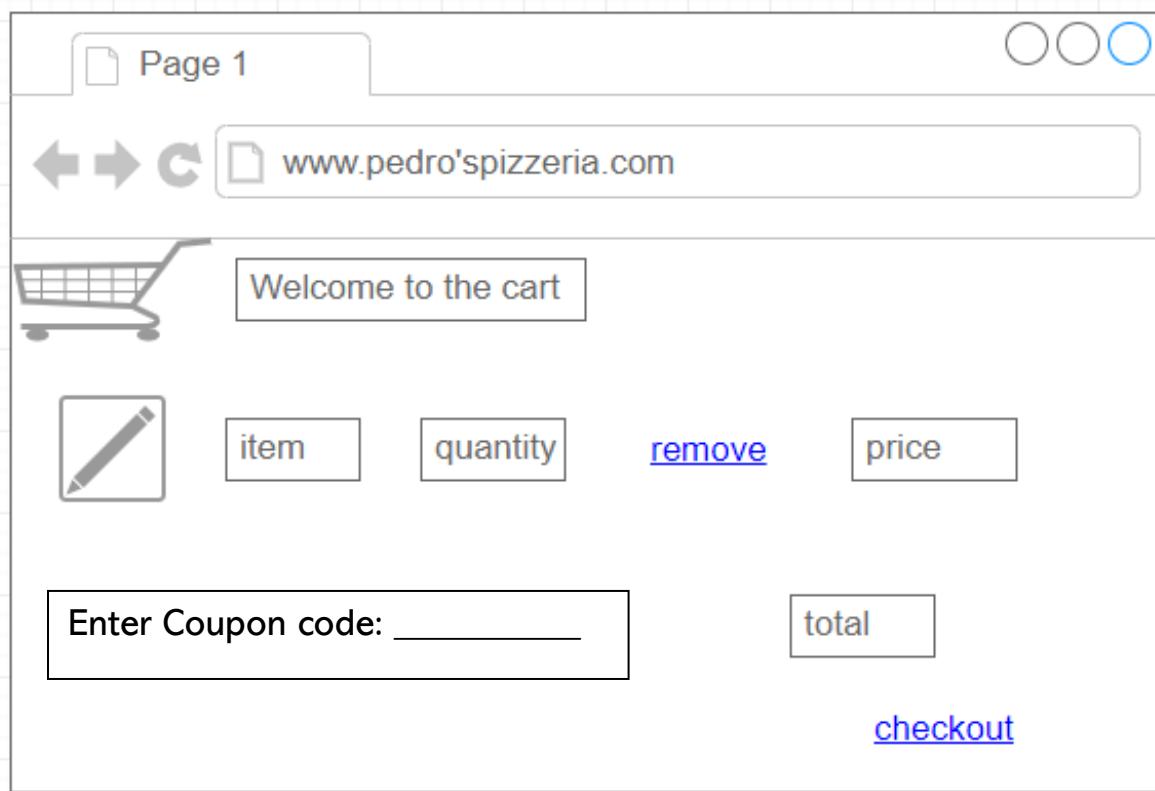
```
newPrice = self.price * (self.size[0] * self.multiplier)
```

For question 4, the feedback reported a bug with my code is that the customer chooses themselves from a look up table. This was a shortcut for my prototype that I will not have in the final solution. I shall solve it by changing how I import different models so that I can import the user model into the orders file.

## Question 7:

### Changes to input & outputs

Here is the redesigned cart section:



This redesigned cart section allows the customer to easily and seamlessly apply a coupon to their basket, just before they checkout.

Here is the design of the new section where the manager will view coupons:

Page 1

www.pedro'spizzeria.com

**Sort Coupons** **Add a new Coupon**

#	Task	Start	Effort	20/10/2014							27/10/2014					
				M	T	W	T	F	S	S	M	T	W	T	F	
1	Task 1	20/10/2014 8:00 AM	40h													
2	Task 2	20/10/2014 8:00 AM	40h													
3	Task 3	20/10/2014 8:00 AM	40h													
4	Task 4	20/10/2014 8:00 AM	40h													

Here is the design of the new section where the manager will add new coupons:

Page 1

www.pedro'spizzeria.com

[Select start date of coupon](#) [Select end date of coupon](#)

October 2014

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

October 2014

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

Discount

**Add a new Coupon**



I will need a new entity for the coupons. I do not intend to change any of the other data structures other than add in the coupon Id in the order table as a foreign key. The new Order data dictionary will be shown below also. This will require a new data dictionary, which is below:

ORDER								
Related to Table:					Customer, OrderPart			
Foreign Keys:					Customer_id			
General table description:								
Field Name	R	I	Data Type	Length	Input Validation	Default	Description	Typical Data
Order_id	Y	Y	integer	2 bytes	It is auto incremented, therefore no validation is required	none	Primary key, uniquely identifies each order	3
Customer_id	Y	N	integer	2 bytes	Presence check Type check Check if the relationship can exist	none	Foreign key, identifies the order's customer	3
Coupon_id	N	N	integer	2 bytes	Presence check Type check Check if the relationship can exist	none	Foreign key, identifies the order's coupon, if any	7
orderType	Y	N	string	1 byte	Look Up Check Presence Check	's' In store	A single character code that saves whether the order is in store, delivery or collection	('s', 'd', 'c')
orderStatus	Y	N	String	1 byte	Look Up Check	'p' preparing	A single character code that stores what stage of the process the order is on	('p', 'b', 'r')
created	Y	N	Date time	10 bytes	Format Check Presence Check Range Check	Current date time	Date and time of order creation	01/05/2019 01:24
updated	Y	N	Date time	10 bytes	Format Check	Current date time	Date of time of when the	01/05/2019 01:24

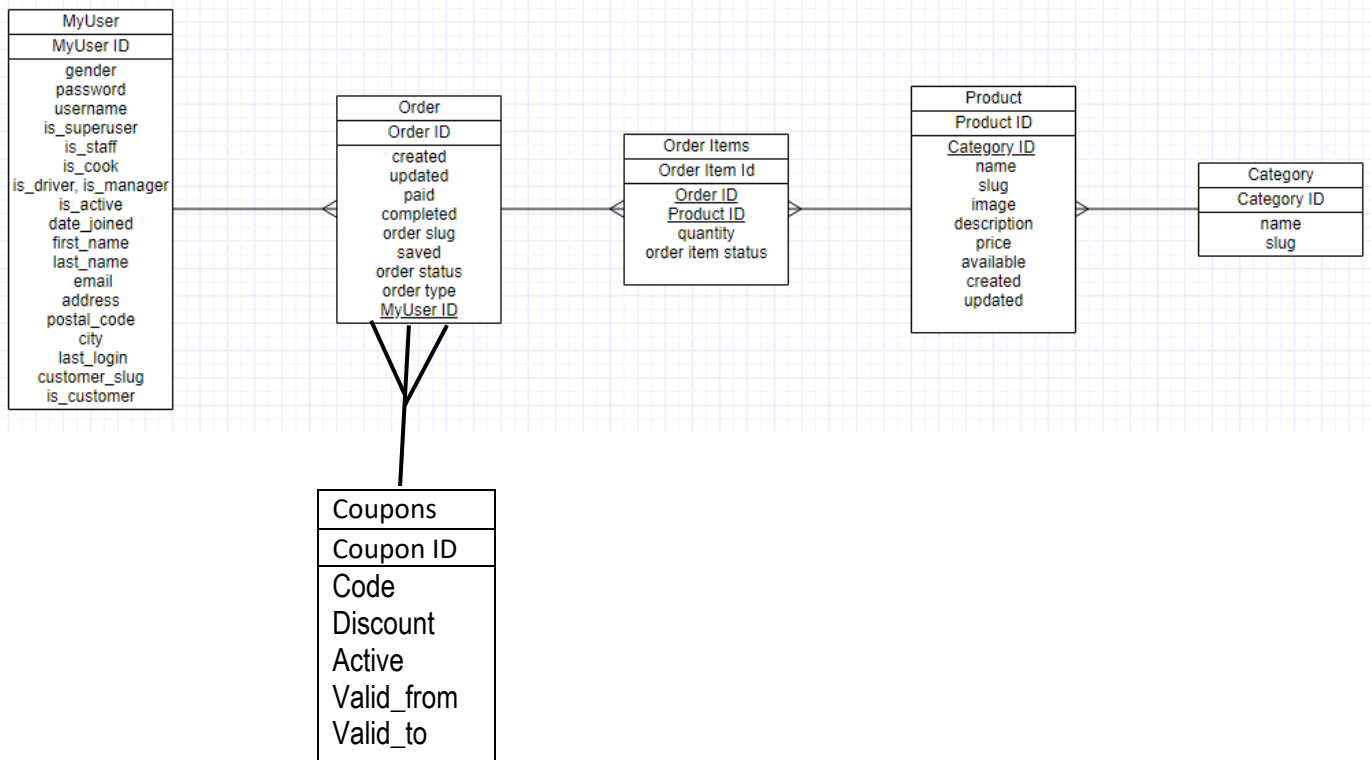
					Presence Check Range Check		order was last updated	
completed	Y	N	Boolean	1 byte	Presence Check	0	Boolean flag to determine if it has been completed	0
order slug	Y	N	String	200 bytes	Presence Check	Order ID	Used in the url of order detail page	3
saved	Y	N	boolean	1 byte	Presence Check	0	Boolean flag to determine if it has been saved	0

### *Coupon:*

This table is required because the system must allow the employees to enter the relevant and necessary information on coupons. This information must be kept secure and must only be used for the reasons stated during collection of data.

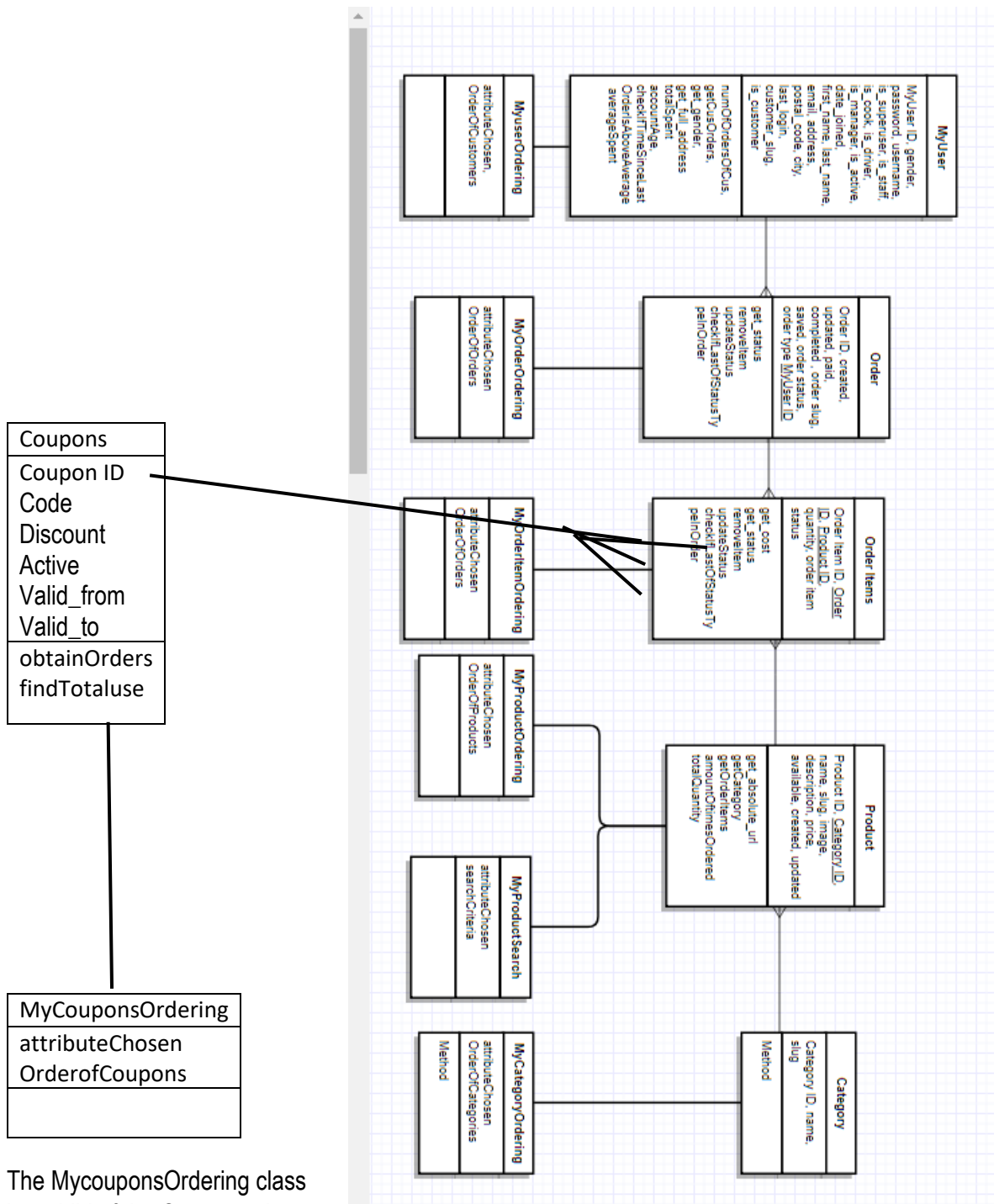
COUPONS								
Related to Table:								
Foreign Keys:								
General table description:								
Field Name	R	I	Data Type	Length	Input Validation	Default	Description	Typical Data
Coupon_id	Y	Y	Integer	2 bytes	It is auto incremented, therefore no validation is required	none	Primary key, uniquely identifies each Coupon	3
Code	Y	N	Integer	2 bytes	Presence check Type check Check if the relationship can exist	None	Foreign key, identifies the product's category	3
discount	Y	N	String	200	Length Check Presence Check	None	Name of product	Fanta
active	Y	N	boolean	1 byte	Presence Check	1	Determines if the coupon is	1 or 0

							active or not	
Valid_from	Y	N	Datetime	1 byte	Format Check	None	Date and time of when the validity of the coupon window is open	01/05/2019 01:24
Valid_to	Y	N	DateTime	1 byte	Format Check	None	Date and time of when the validity of the coupon window is closed	01/05/2019 01:24



My updated Third Normal form Entity Relationship Diagram is shown above.

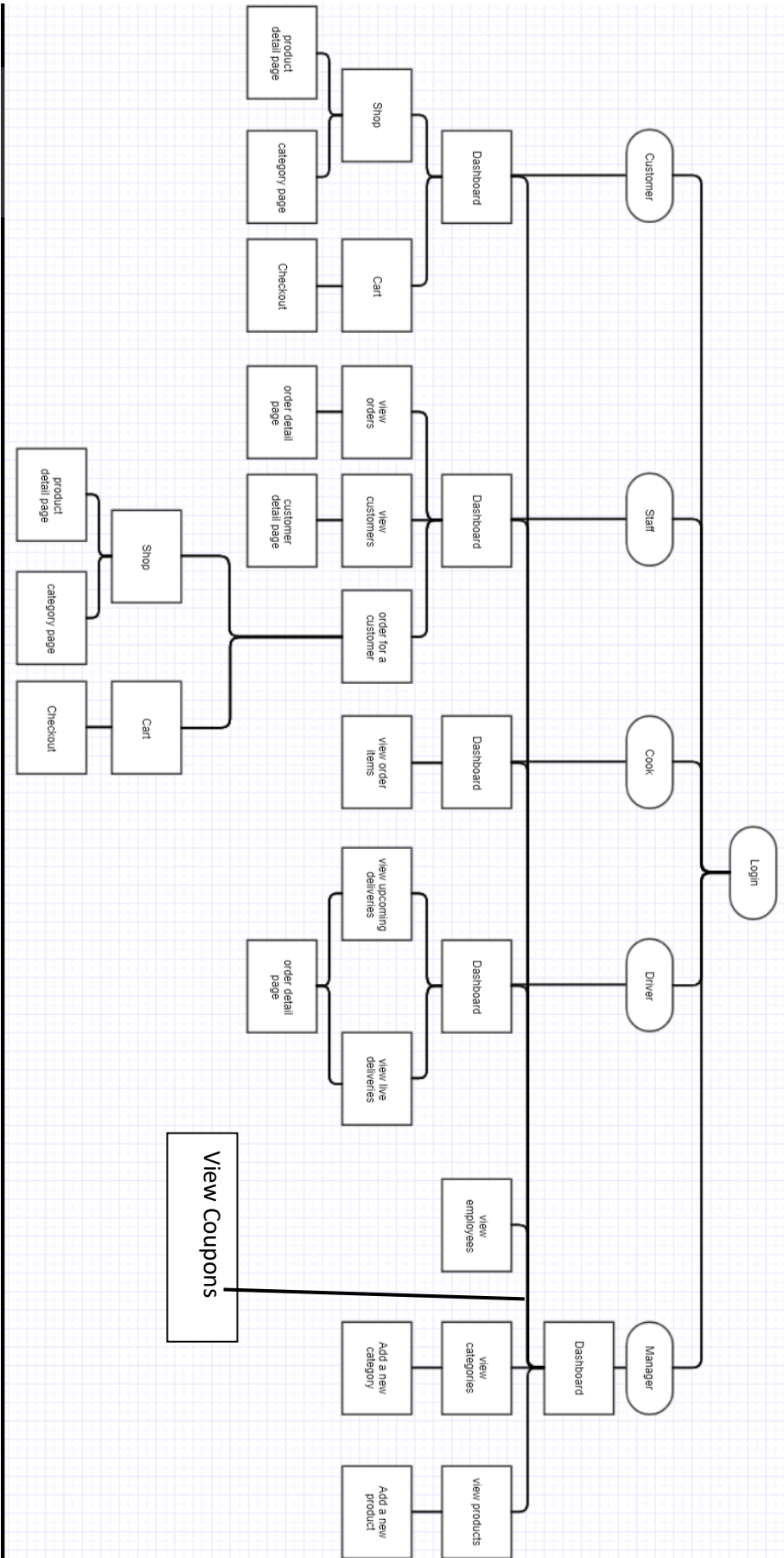
My updated Class Diagram is shown below.



The MycouponsOrdering class is a child of the Coupons class.

It is used to create the Django form that orders the coupons table for the manager to view. This page has been designed, and is above.

My updated functional decomposition diagram is shown below.



To fully implement this new Coupons app, I will have to change some of my algorithms. These will be

updated below:

New algorithm for adding an order:

```
if cart.coupon:
    order = Order.objects.create(customer_id = request.user.id,
                                  orderType = cd['orderType'],
                                  coupon_id = cart.coupon.id,
                                  discount = cart.coupon.discount)
else:
    order = Order.objects.create(customer_id = request.user.id,
                                  orderType = cd['orderType'],)
order.orderslug = order.id
order.save()

for item in cart:
    OrderItem.objects.create(order = order,
                              product = item['product'],
                              price = item['price'],
                              quantity = item['quantity'])

order.orderslug = order.id
order.save()
```

I have now fully designed this new feature that was recommended for me to add to my system in the feedback.