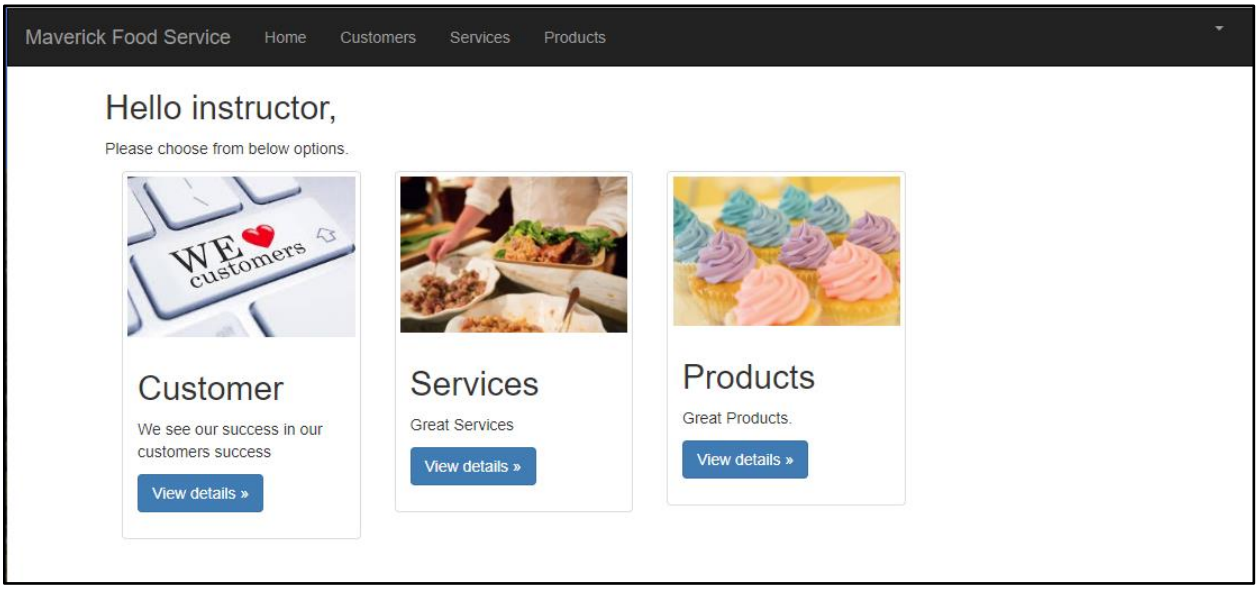
Maverick Food Services (MFS):
MFS is a fictitious organization which provides food services in a campus setting. The catering manager for the food services organization needs to keep track of customers in order to grow their profits by marketing to these customers and their colleagues. Some type of a database is needed to track customers and the products and services they acquire over time. This type of system is often called a customer information database or a customer relationship management database or CRM for short. In this assignment, you will build a CRM for the Maverick Food services organization. The application shell is provided which initiates the application but the remaining functionality must be implemented and deployed to PythonAnywhere.
Sections 1-8 take you through developing part of the application. After completing Sections 1 – 8, complete sections 9 and 10 to add functionality to the application.

Section 1 - Tour of the application you will be developing in Assignment 2

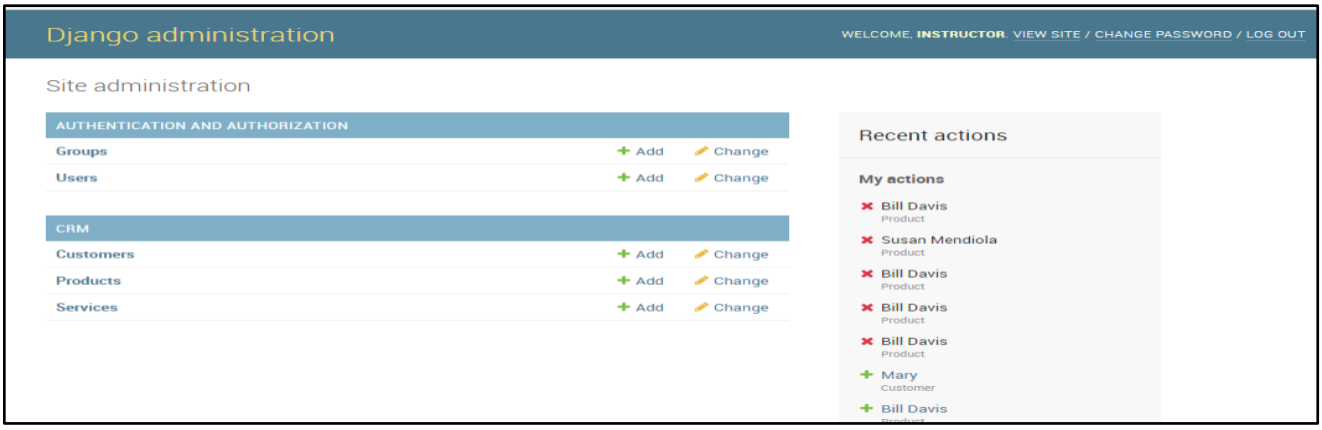
Sample Landing Page for the Maverick Food Service CRM



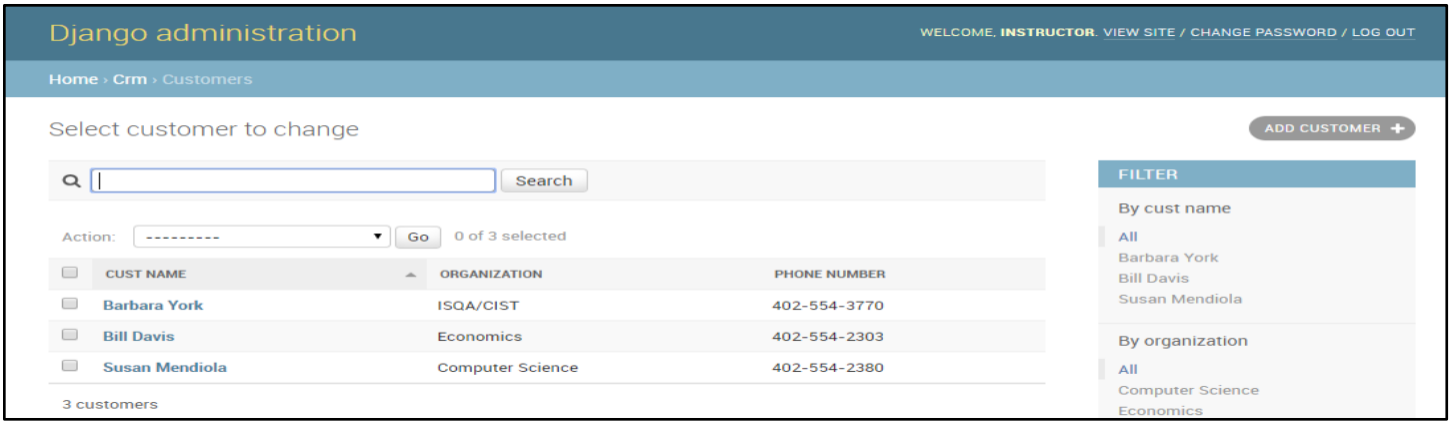
There are two roles in this sample application:

- The administrators
  - have access to the information for all customers, services and products.
  - can add, update and delete customers, services and products
  - have access to the Admin section of Django to set up new employee accounts.
- Employees can:
  - Add, update, and delete Customers
  - Add, update, and delete Products and Services for each customer

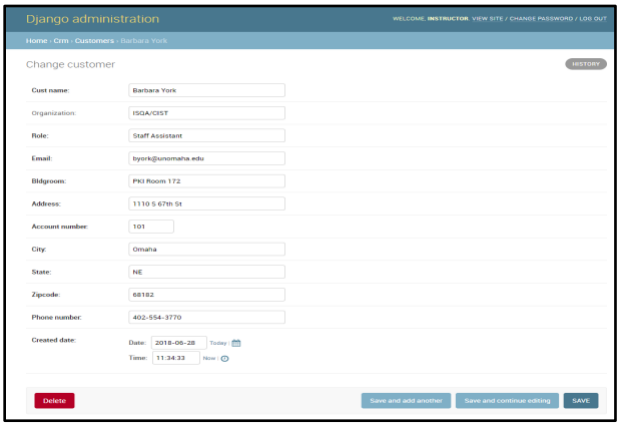
Below you see the Admin panel of the application available to administrators. As you learned in the blog assignment, you add “/admin” to the URL to access it.



The Blog assignment showed the Groups and Users in the admin panel. In this application, a given customer can acquire many services and products. Below is a view of the customers page showing the ability to add, update, and delete customers:

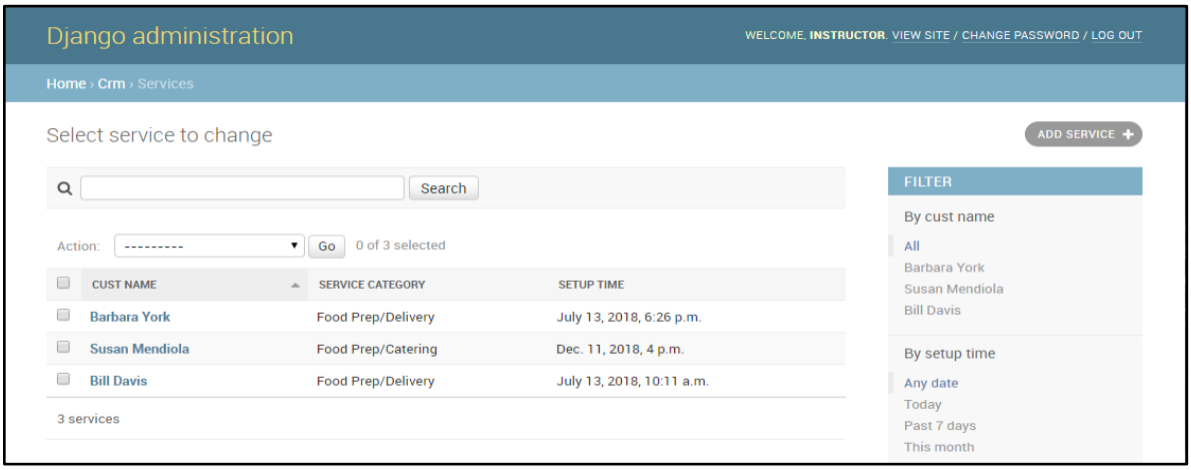


Here is an edit and update of one of the records:

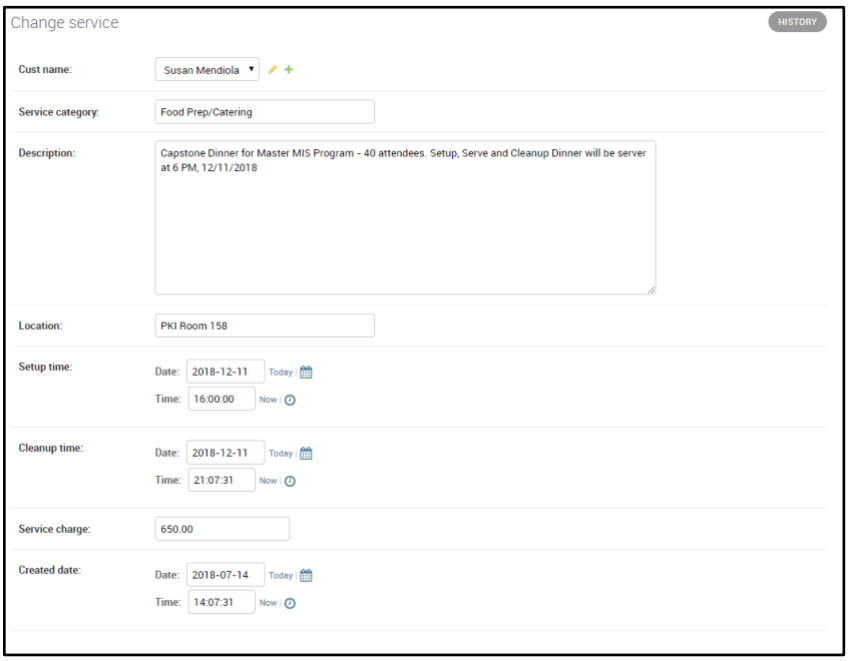


Note the history. This points out the logging capabilities provided by Django.

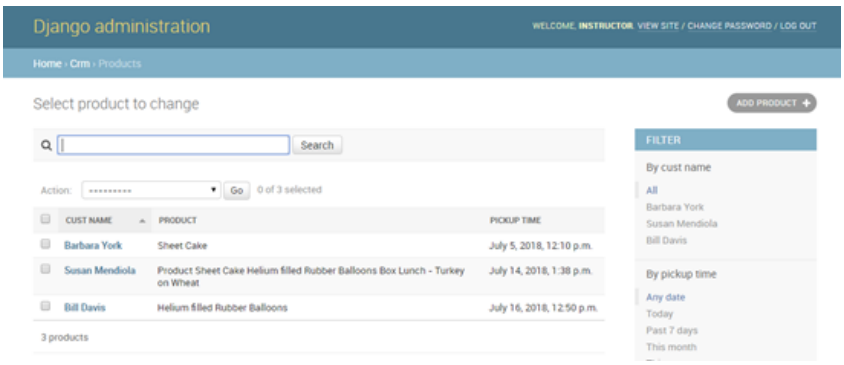
Below you find a list of the service transaction in the Service Table.



As with the customers, you can add, update and delete services for each customer.



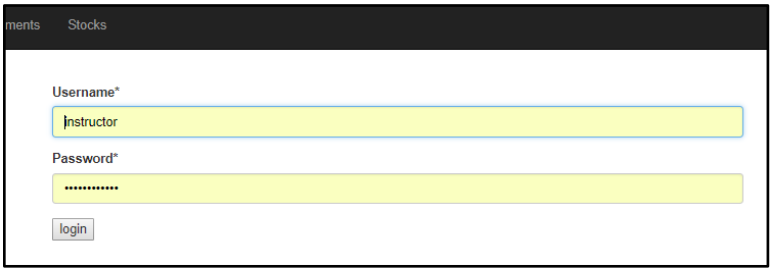
Next we have the page listing of the products purchased by the Maverick Food Service customers.



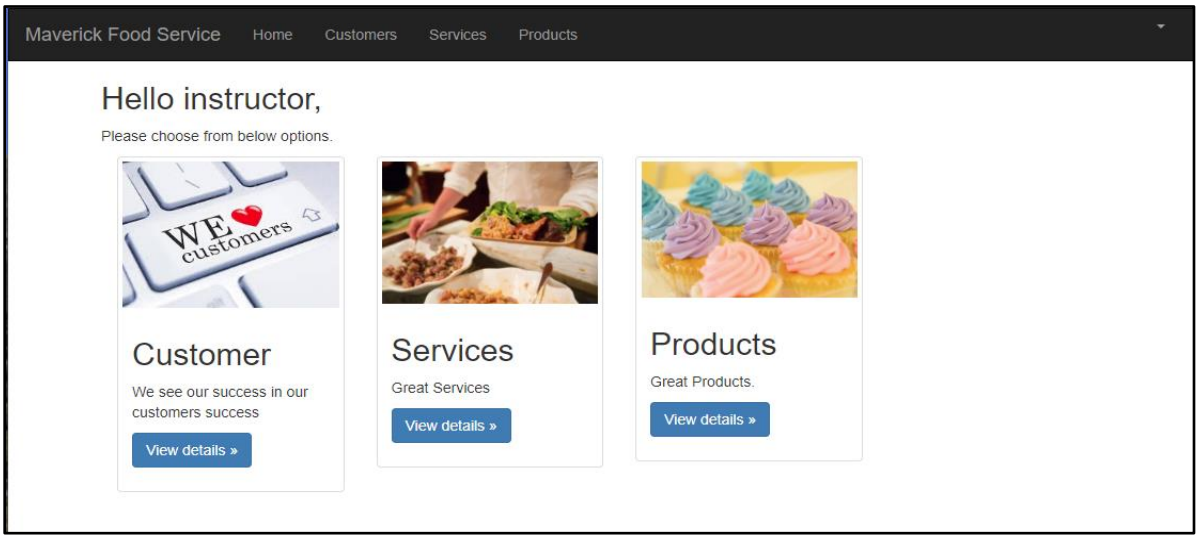
As with Services you can click on a Product and edit or delete the products.

The screens shown below show what may be accessed by any employee of Maverick Food Service. In this scenario, we will assume that employees are provided their ID and PW by the administrator.

To sign in, the employee would click the login button after entering their ID and PW:

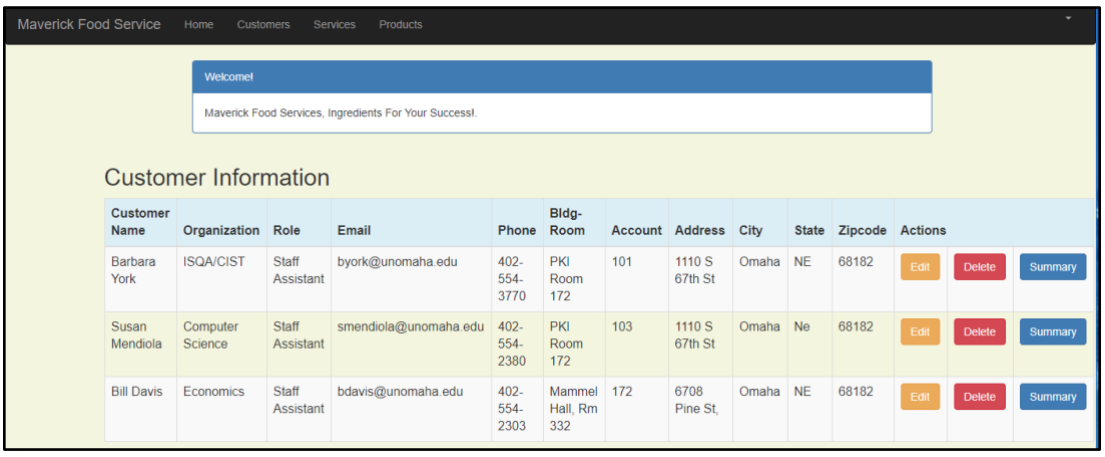


After successfully signing in using the ID and PW provided by the administrator, the employee will see the landing page below:



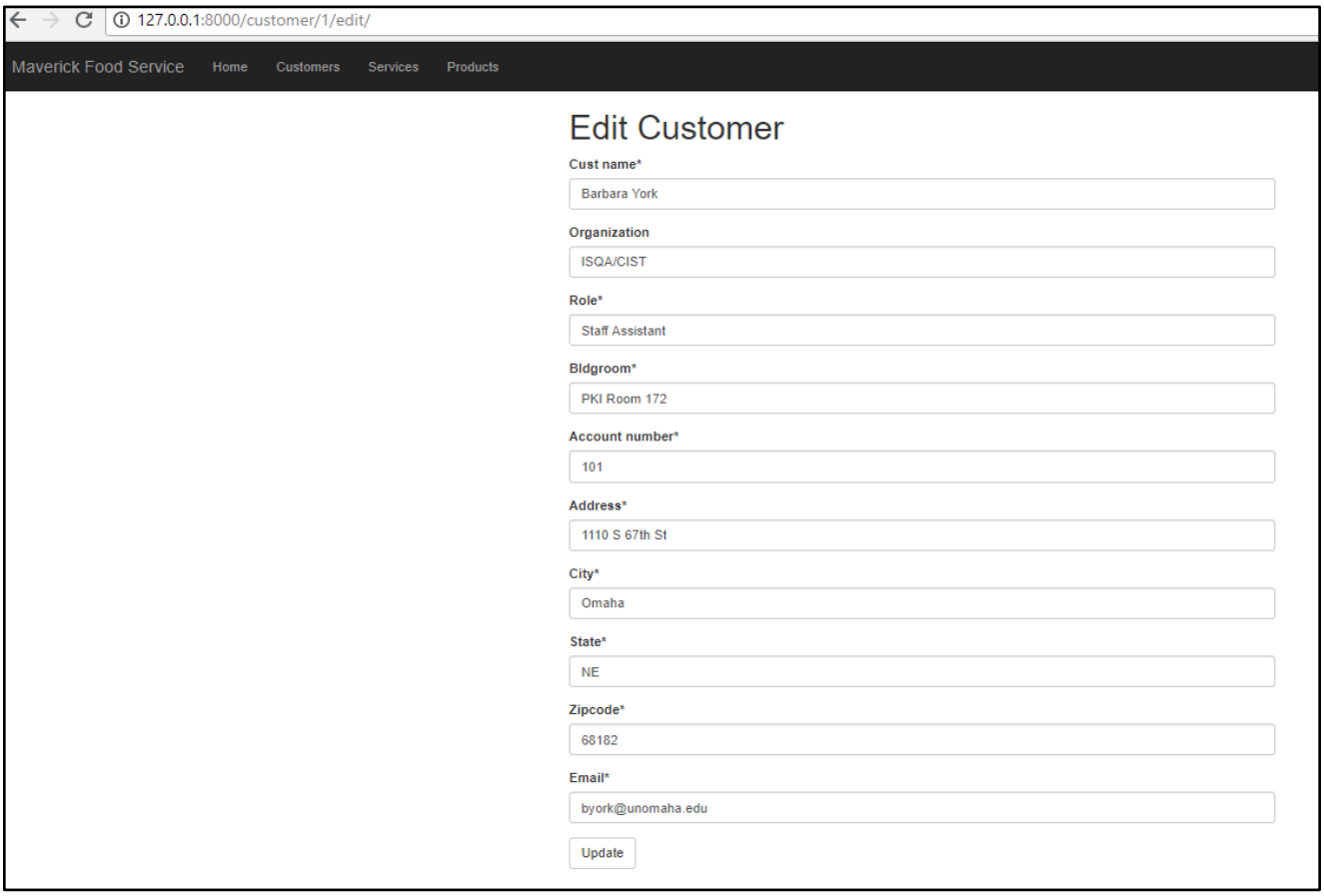
No other features such as “forgot your password” are in the assignment. This feature can be added to earn extra credit.
Employees can view a list of customers and update customer information with a page similar to the one shown below.

Customer Screen



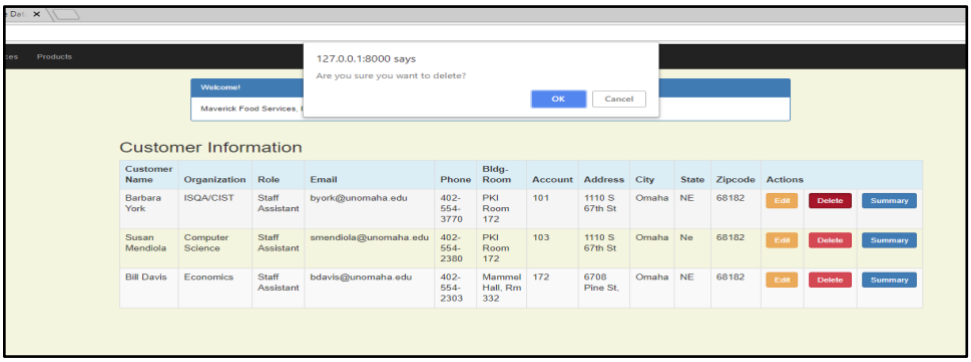
Customer Name	Organization	Role	Email	Phone	Bldg-Room	Account	Address	City	State	Zipcode	Actions		
Barbara York	ISQA/CIST	Staff Assistant	byork@unomaha.edu	402-554-3770	PKI Room 172	101	1110 S 67th St	Omaha	NE	68182	Edit	Delete	Summary
Susan Mendiola	Computer Science	Staff Assistant	smendiola@unomaha.edu	402-554-2380	PKI Room 172	103	1110 S 67th St	Omaha	Ne	68182	Edit	Delete	Summary
Bill Davis	Economics	Staff Assistant	bdavis@unomaha.edu	402-554-2303	Mammel Hall, Rm 332	172	6708 Pine St,	Omaha	NE	68182	Edit	Delete	Summary

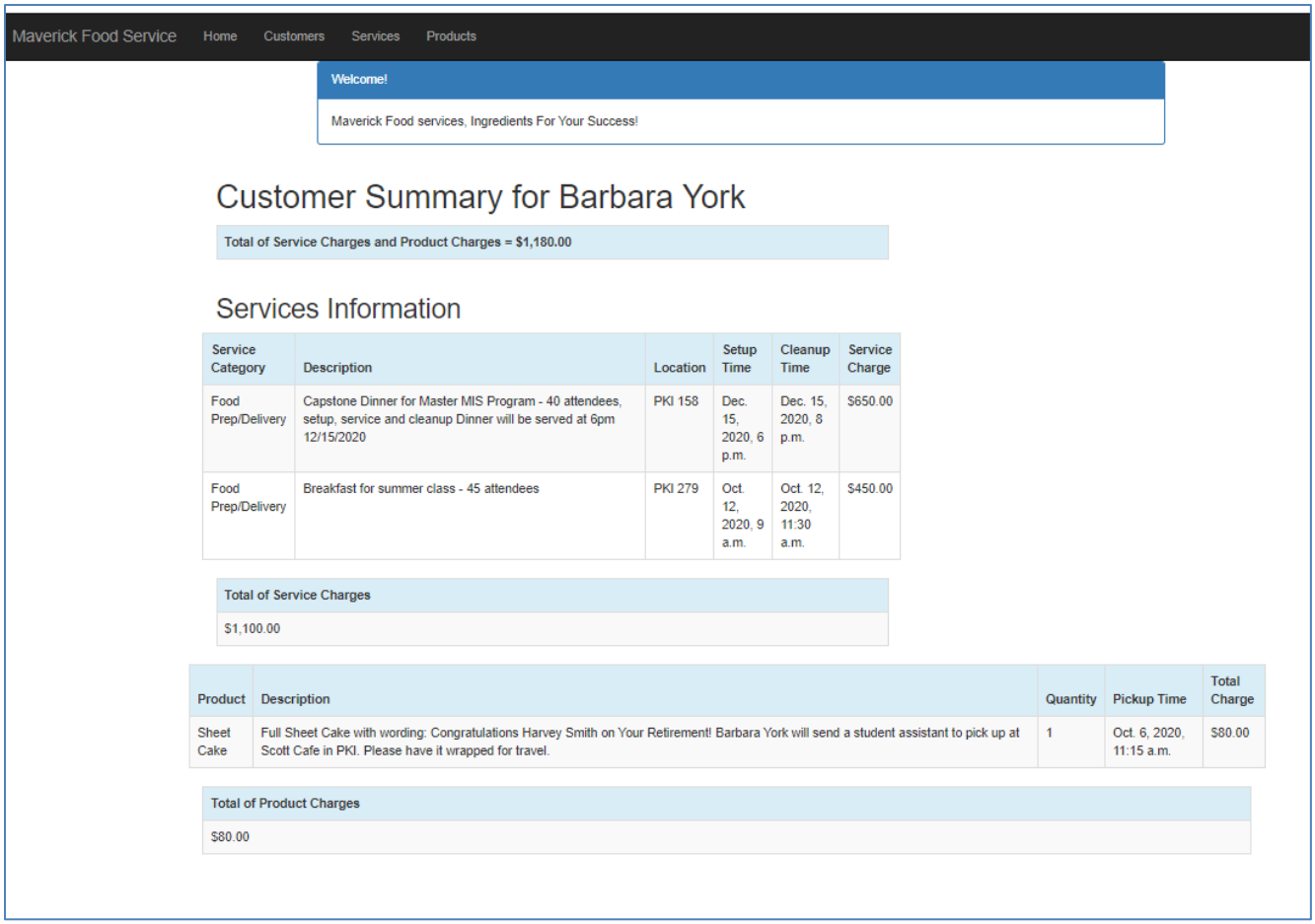
Customer Edit - Employees can edit customer information with the screen shown below:



The Customer delete button brings up a warning. **NOTE**: **Since we are using the CASCADE option when setting up the model in Django, deleting a customer will also delete all information in services and products associated with that customer.**

Customer delete:



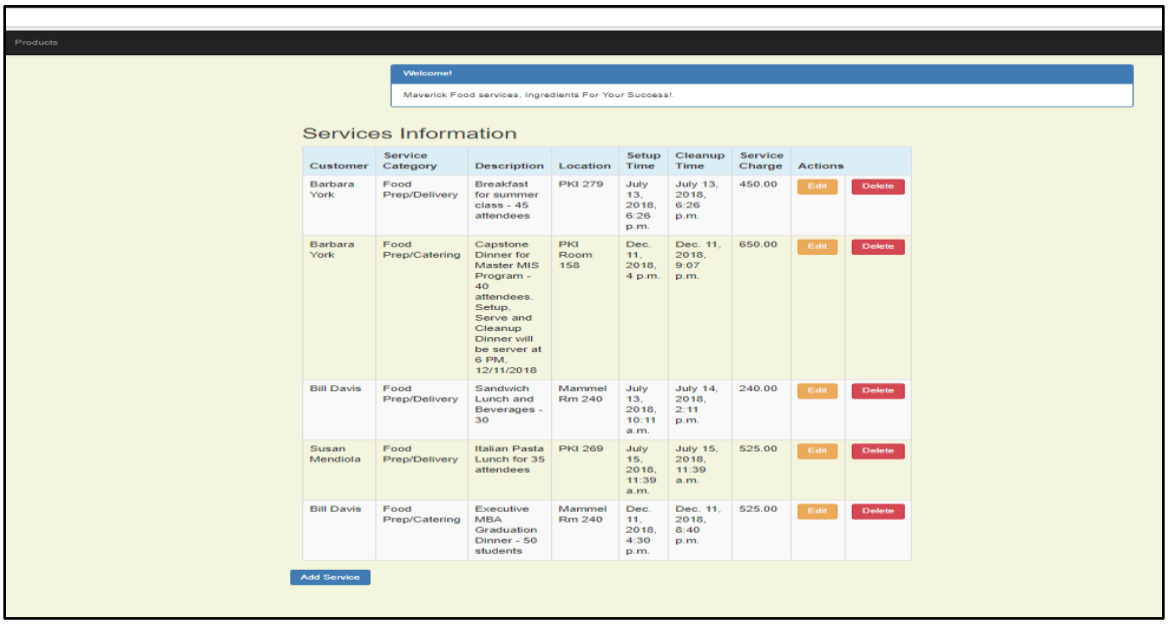


Report - The summary function is a report of all services and products acquired by a customer. This is discussed at the end of the tutorial.

According to the report above, Barbara York has ordered $1,180 services and products from MFS for her department.

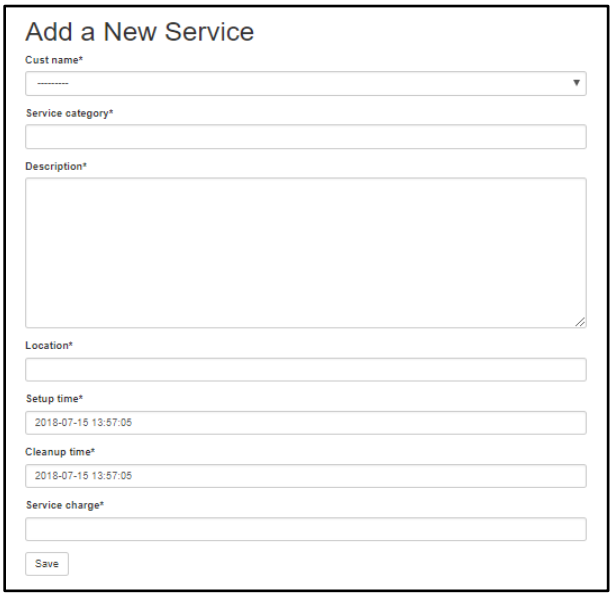
Service List:

The service list screen allows employees of MFS (Maverick Food Service) to add services provided to customers, including those working for the university and external parties, requiring food services for events they are hosting in a UNO building. This page allows for adding a new service provided, editing the information, and deleting a service provided. Sample customer, services, and products shown.

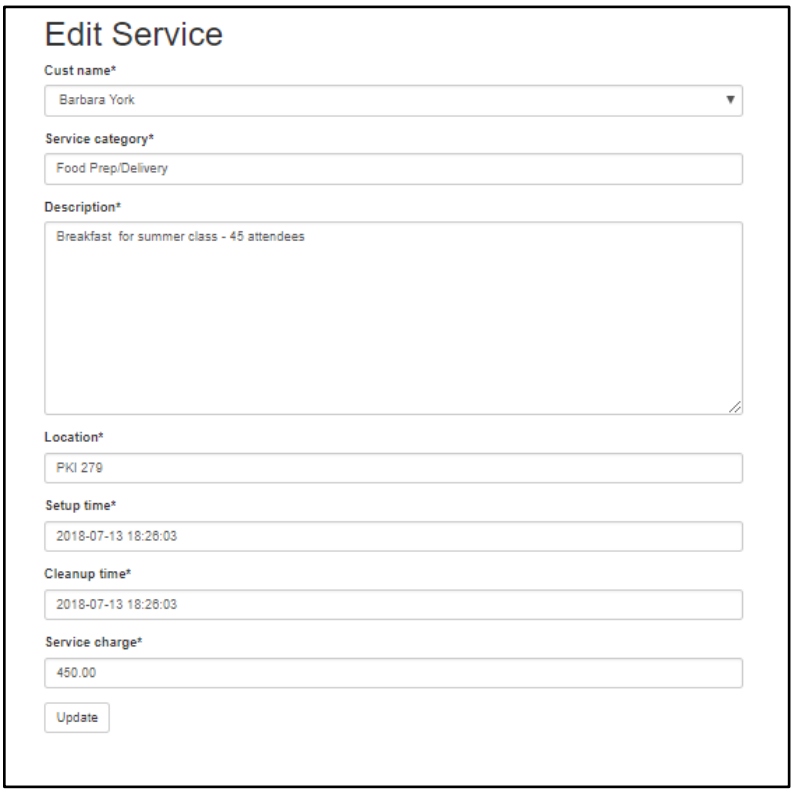


Customer	Service Category	Description	Location	Setup Time	Cleanup Time	Service Charge	Actions	
Barbara York	Food Prep/Delivery	Breakfast for summer class - 45 attendees	PKI 279	July 13, 2018, 6:26 p.m.	July 13, 2018, 6:26 p.m.	450.00	Edit	Delete
Barbara York	Food Prep/Catering	Capstone Dinner for Master MIS Program - 40 attendees. Setup, Serve and Cleanup Dinner will be server at 6 PM, 12/11/2018	PKI Room 158	Dec. 11, 2018, 4 p.m.	Dec. 11, 2018, 9:07 p.m.	650.00	Edit	Delete
Bill Davis	Food Prep/Delivery	Sandwich Lunch and Beverages - 30	Mammel Rm 240	July 13, 2018, 10:11 a.m.	July 14, 2018, 2:11 p.m.	240.00	Edit	Delete
Susan Mendiola	Food Prep/Delivery	Italian Pasta Lunch for 35 attendees	PKI 269	July 15, 2018, 11:39 a.m.	July 15, 2018, 11:39 a.m.	525.00	Edit	Delete
Bill Davis	Food Prep/Catering	Executive MBA Graduation Dinner - 50 students	Mammel Rm 240	Dec. 11, 2018, 4:30 p.m.	Dec. 11, 2018, 8:40 p.m.	525.00	Edit	Delete

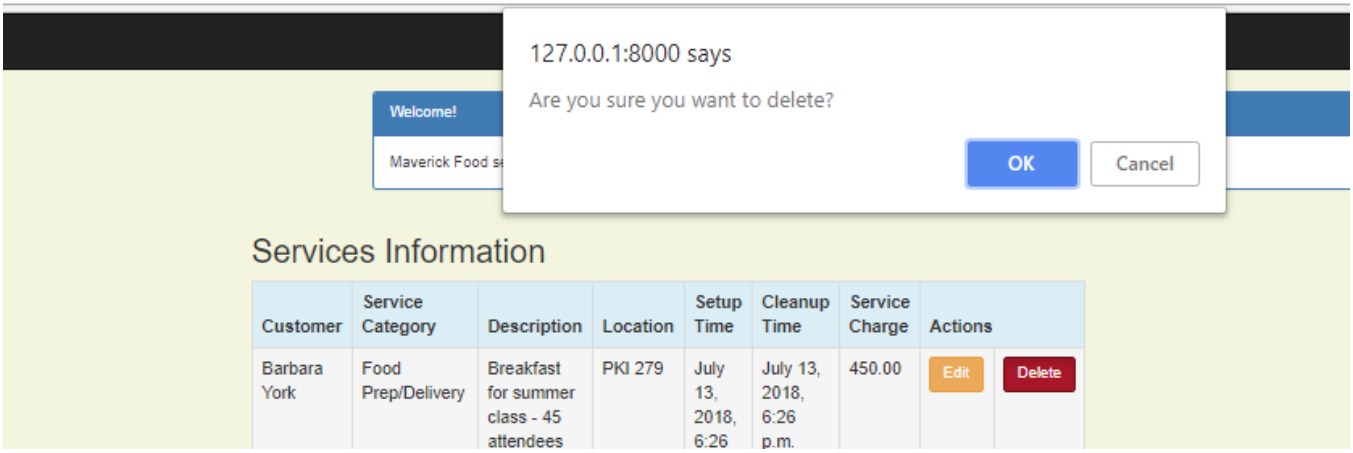
Clicking ‘Add Service’, brings up the following screen:



Editing an existing record in a similar manner.



Deleting a service presents a warning, but only a single service record is deleted. The customer is not deleted.



Products:

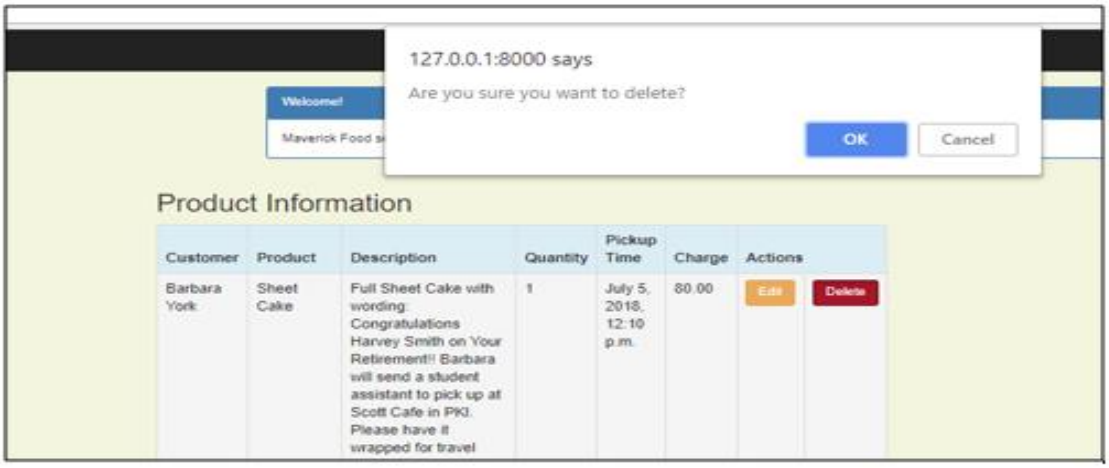
All employees may add, update and delete products sold to customers using screens shown below:

Product List

Add a new product

Edit a Product

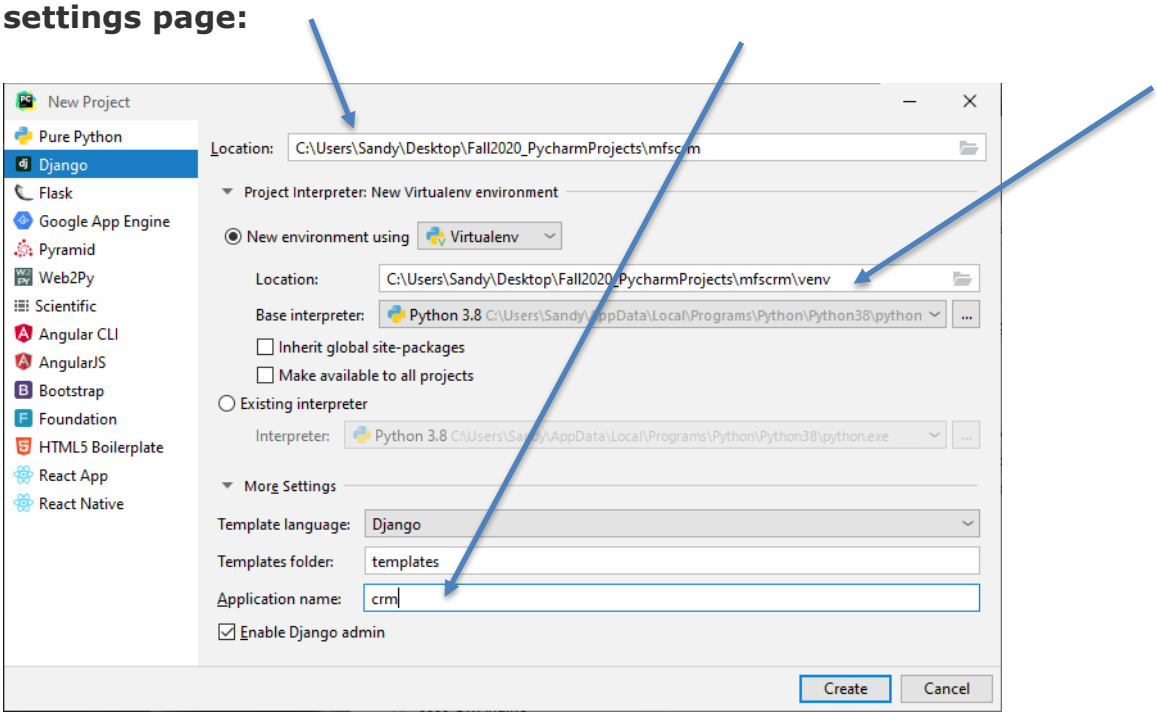
Delete product works same as delete service



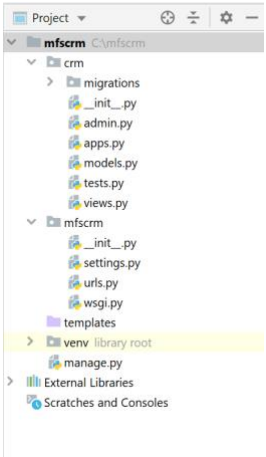
Section 2 - Directions for Setting up the Maverick Food Service Project and Installing Django

(Note that all these directions assume you are using python 3.8.x and Django 3.x)

Start PyCharm and select **'Create New Project'**. In the New Project window, name the project **'mfscrm'** and set the app name to **'crm'.**
Ensure you specify the Virtual Environment and the Python Interpreter. Your directory structure will look different from mine. The screenshot below shows the settings page:



After the creation of the project completes, your project files should look like the following:

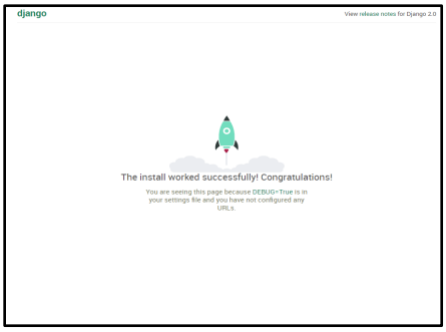


Click the **'settings.py'** file in the mfscrm/mfscrm/ directory/folder to view the settings for the project and add the following line of code to this file just after the comments at the top of the file – this should be near line 12 in the file:

```
import os
```

Verify the install and creation of the ‘crm’ application worked by opening the Terminal window in Pycharm and entering:
python manage.py runserver

In an Internet browser window go to http://127.0.0.1:8000/
You should see this:



Next, edit the ‘settings.py’ file and change the TIME_ZONE to ‘America/Chicago’.

If you see the following error in **PyCharm -**

TypeError: argument of type 'WindowsPath' is not iterable

Make the following 3 changes to the ‘settings.py’ file:
REPLACE 3 lines - be sure to keep the original indention:

```
1. REPLACE:
BASE_DIR = Path(__file__).resolve().parent.parent

WITH
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


2. REPLACE: - under TEMPLATES:
'DIRS': [BASE_DIR / 'templates']
WITH
'DIRS': [os.path.join(BASE_DIR, 'templates')]

3. REPLACE: - under DATABASES:
'NAME': BASE_DIR / 'db.sqlite3',
WITH
'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
```

Section 3 - Populating a Database with Base Django tables and a Customer Table

Step 2 – Exit the application if it is running in the terminal window.
Next, create the Django project tables by running our first migration of this project. Run the following command in the Terminal window:

python manage.py migrate

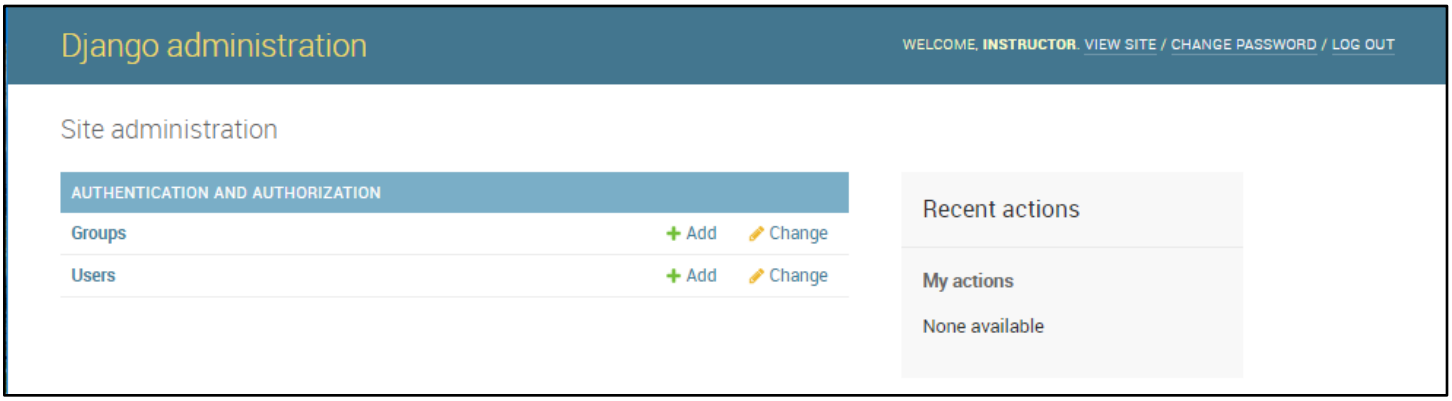
This will create the default tables for any Django project/application. You should see the following displayed in the Terminal window:

Operations to perform:
Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
Applying contenttypes.0001_initial... OK
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK

Now create a superuser for the application using the following command in the Terminal window:
python manage.py createsuperuser

Create a superuser account of your choice.

Now start the server again with the command: **python manage.py runserver.**
Go to: http://127.0.0.1:8000/admin and sign in with your new superuser credentials. You should see the admin panel you add admin to our URL like this:



If the above admin page does not load after entering your superuser's username and password, return to http://127.0.0.1:8000/admin

Check out the capabilities in this admin panel to create new users and even groups with different permissions. Add a new user through this panel. You will notice users default to 'active user', not a superuser. unless you check the box to change them to a superuser. You may also set up groups of users with similar permissions from the admin interface.

Return to Pycharm and exit the application in the terminal window by entering Ctrl+C.

Next: We will create a customer table for the Maverick Food Service CRM project. This will allow us to easily manage contact information of the customers. Add the following information about our customer table into the **models.py** file in the **crm** app. Edit the **Models.py** file in the crm > migrations directory/folder:

Models.py

```
from django.utils import timezone
from django.db import models


# Create your models here.
class Customer(models.Model):
    cust_name = models.CharField(max_length=50)
    organization = models.CharField(max_length=100, blank=True)
    role = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    bldgroom = models.CharField(max_length=100)
    address = models.CharField(max_length=200)
    account_number = models.IntegerField(blank=False, null=FaIse)
    city = models.CharField(max_length=50)
    state = models.CharField(max_length=50)
    zipcode = models.CharField(max_length=10)
    phone_number = models.CharField(max_length=50)
    created_date = models.DateTimeField(
        default=timezone.now)
    updated_date = models.DateTimeField(auto_now_add=True)

    def created(self):
        self.created_date = timezone.now()
        self.save()

    def updated(self):
        self.updated_date = timezone.now()
        self.save()

    def __str__(self):
        return str(self.cust_name)
```

To quickly sum up what we just added, the first line above tells Django to also import its feature timezone. If you look at the code, you will see we added *timezone.now(),* so without importing this utility in Django, we cannot use these features. "timezone.now" is a feature Django offers to stamp the current date and time of when the data is added or updated in the database.

Now that we have a database table for our customers, we will add it to the *admin.py* file so that we can add our customer numbers, address, city, state, etc.

Open the your *admin.py* file in the crm directory/folder. You should see the following (greyed out):

```
from django.contrib import admin
```

To continue on from here, replace it with the code shown below:

admin.py

```
from django.contrib import admin

from .models import Customer

class CustomerList(admin.ModelAdmin):
    list_display = ( 'cust_name', 'organization', 'phone_number' )
    list_filter = ( 'cust_name', 'organization')
    search_fields = ('cust_name', )
    ordering = ['cust_name']


admin.site.register(Customer)
```

Next, create the first table beyond the standard Django Table for our application by making and then running a migration. The migration files can be seen in the migration folder. They are very valuable when we want to deploy our application.

We need create the migrations from the data model defined for the customer. Do this by issuing the following command in the Terminal window in the **mfscrm** directory/folder:

python manage.py makemigrations

The results should be:

```
Migrations for 'crm':
  crm\migrations\0001_initial.py
    -  Create model Customer
```

Now, enter this command

python manage.py migrate

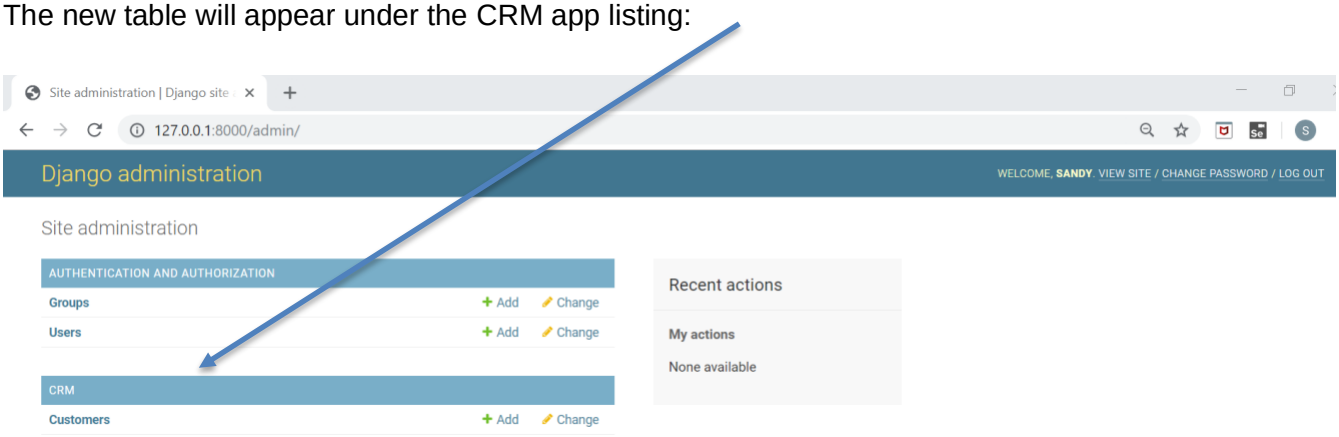
You should receive this message:

```
Operations to perform:
  Apply all migrations: admin, auto, contenttypes, crm, sessions
Running migrations:
  Applying crm.0001_initial… OK
```

View the table in the admin panel of the app. To do this startup the server with

python manage.py runserver

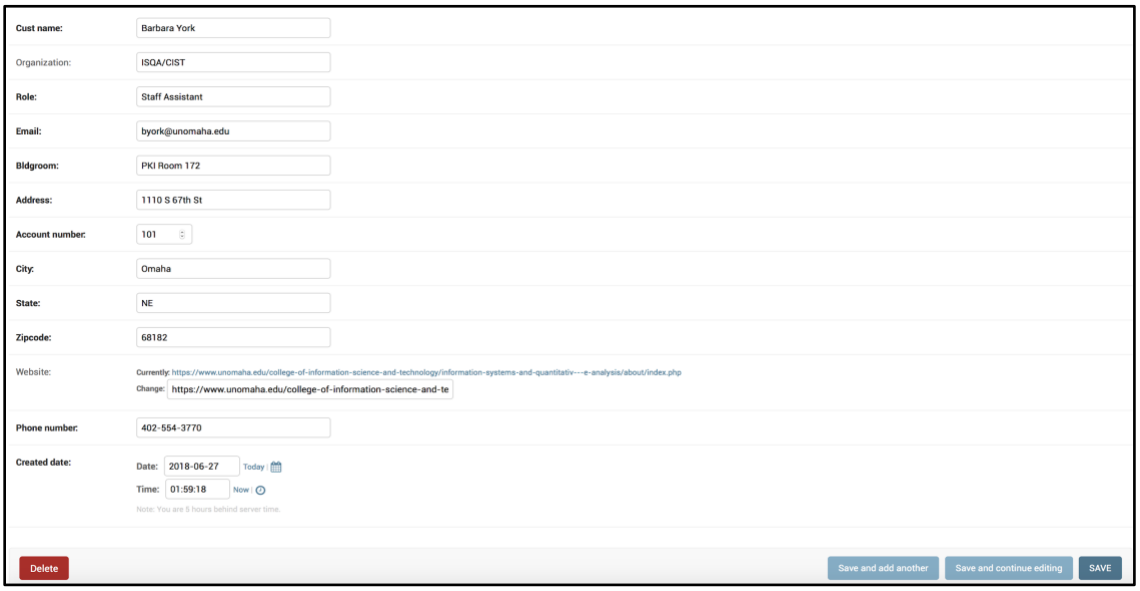
In your Internet browser, go to: http://127.0.0.1:8000/admin/
The new table will appear under the CRM app listing:



Add some customers to the application by clicking the "+ Add" button for the Customers.
Please add the following 3 customers shown below to your database.

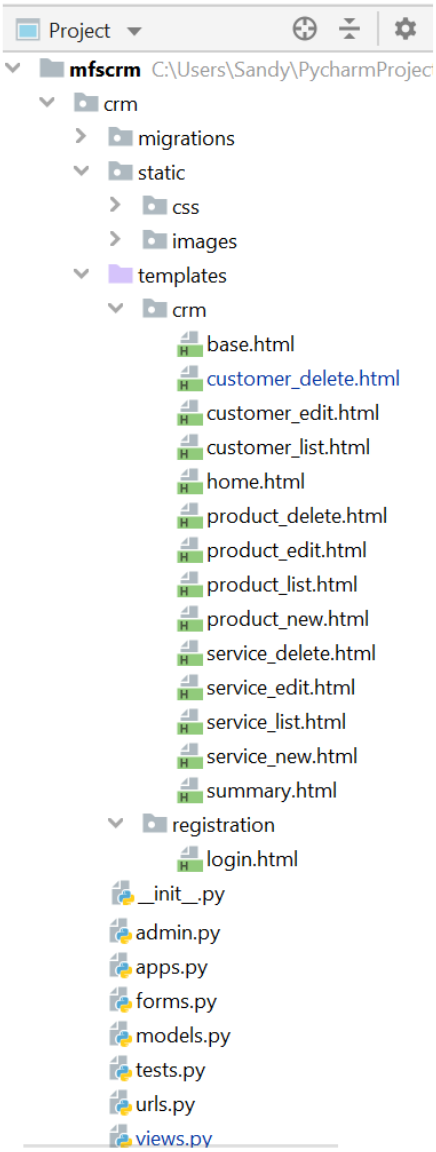
Cust_name	Organization	Role	Email	BldgRoom	Address	Account _No	City	State	Zipcode	Phone Number
Barbara York	ISQA/CIST	Staff Assistant	byork@unomaha.edu	PKI Room 172	1110 S 67th St	101	Omaha	NE	68182	402-554-3770
Susan Mendiola	Computer Science	Staff Assistant	smendiola@unomaha.edu	PKI Room 172	1110 S 67th St	103	Omaha	NE	68182	402-554-2380
Bill Davis	Economics	Staff Assistant	bdavis@unomaha.edu	Mammel Hall, Rm 332	6708 Pine St,	172	Omaha	NE	68182	402-554-2303

Example 1:



View the tables in the database using the DB Browser for SQLite. Click here to download: http://sqlitebrowser.org/. Once the SQLite Browsers is installed you can open the file called **db.sqlite3** in the project directory. Check out the system tables and the new customers table.

We will be adding a number of pages in this tutorial. The screenshot below shows a layout of the pages in this application when you have completed all the remaining steps in the tutorial and their locations:
Preview of what you will create in this tutorial:



Section 4 - Creating a Web Page to Display and Update the Customer List.

While the admin panel of Django does provide the ability to add and edit the customer information, you do not want to provide superuser access for all users. In order to provide access to the food service employees we need to create some HTML templates which allow these users to login and manage the customer information without being given superuser access.

Django offers a few different ways to “view” your web apps. Some developers create function-based and class-based generic view. Function-based views trigger a Python function for a web request, and returns a web response.

For our customer's page, all we want to do is display all of our customers and all of their contact information, just like it was entered through in the admin side. We will be adding two buttons: one for updating their info and one to delete them from our system.

For that, we will create a Django form. Create a new python file in the **crm** app called **forms.py**. Right-click 'crm' in the Project browser pane, and select 'New' -> Python File and name it forms.py.
Copy and paste the code shown below to the forms.py file:

forms.py

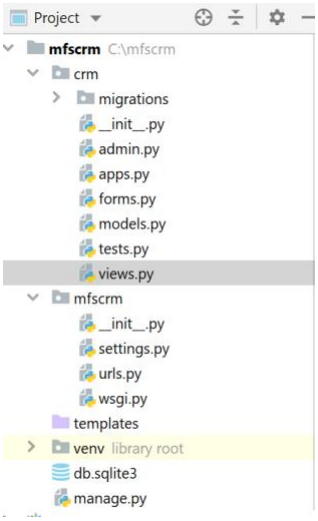
```
from django import forms
from .models import Customer

class CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = ('cust_name', 'organization', 'role', 'bldgroom', 'account_number', 'address',
                  'city', 'state', 'zipcode', 'email','phone_number')
```

Since Django is a model-view-template framework, we now must describe what we want to display in a template in the **views.py** file. Double click to open the views.py file in **crm** app folder, and replace the code with the following:

views.py

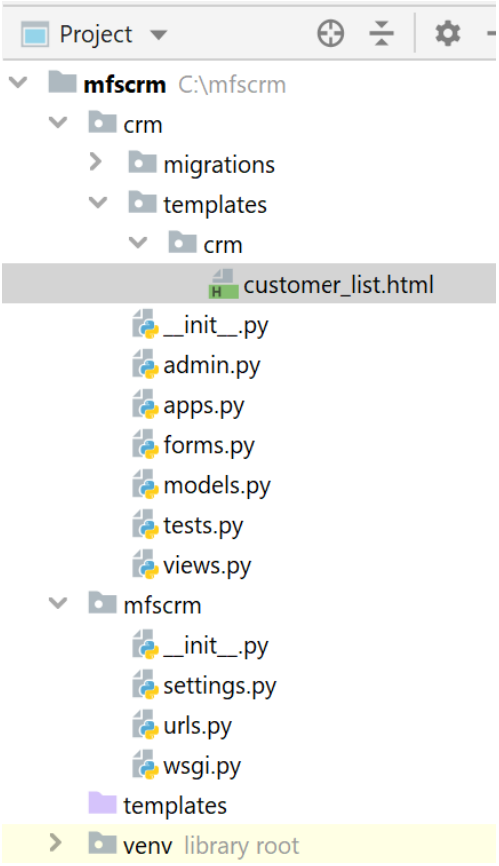
```
from django.shortcuts import render
from .models import *
from .forms import *


def customer_list(request):
    customer = Customer.objects.filter(created_date__lte=timezone.now())
    return render(request, 'crm/customer_list.html',
                  {'customers': customer})
```

Notice the last line above indicates where Django needs to look for a template, to display our URL for customer list.



Next, create a template directory inside the ‘crm’ app folder. Create it by right-clicking on the crm directory in the project browswer pane in PyCharm, and selecting New, and then Directory. Name it templates.

Next we need another directory inside of the templates directory. Create this directory and name it ‘crm’. Create a new HTML file inside the **templates/crm** directory and call it **customer_list.html** as shown below.



Use copy-paste to replace the contents of the customer_list.html file, with the following html:

The customer_list.html code continues onto the next 2 pages – be sure to copy all the code.

customer_list.html

```
<!DOCTYPE html>
<html lang="en">
{% block content %}

  <html>
  <head>
    <meta charset="UTF-8">
    <title>Maverick Food Services</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">

  </head>
  <body>
  <style>
    body {
      background-color: beige;
    }
  </style>
```

```
<div class="container">
   <div class="row">
      <div class="col-md-10 col-md-offset-1">
         <div class="panel panel-primary">
            <div class="panel-heading">Welcome!</div>
            <div class="panel-body">
               Maverick Food Services, Ingredients For Your Success!.
            </div>
         </div>
      </div>
   </div>
</div>
<div class="row">
   <h2 style="padding-left: 15Px">Customer Information</h2>
</div>
<div>
   <table class="table table-striped table-bordered table-hover">
      <thead>
      <tr class="bg-info">
         <th>Customer Name</th>
         <th>Organization</th>
         <th>Role</th>
         <th>Email</th>
         <th>Phone</th>
         <th>Bldg-Room</th>
         <th>Account</th>
         <th>Address</th>
         <th>City</th>
         <th>State</th>
         <th>Zip Code</th>

         <th colspan="3">Actions</th>
      </tr>
      </thead>
      <tbody>
      {% for customer in customers %}
         <tr>
            <td>{{ customer.cust_name }}</td>
            <td>{{ customer.organization }}</td>
            <td>{{ customer.role }}</td>
            <td>{{ customer.email }}</td>
            <td>{{ customer.phone_number }}</td>
            <td>{{ customer.bldgroom }}</td>
            <td>{{ customer.account_number }}</td>
            <td>{{ customer.address }}</td>
            <td>{{ customer.city }}</td>
            <td>{{ customer.state }}</td>
            <td>{{ customer.zipcode }}</td>
            <td><a href="" class="btn btn-warning">Edit</a>

            <td><a href=""
                  onclick="return confirm('Are you sure you want to delete?')"
                  class="btn btn-danger">Delete</a>
            </td>
```

```
            <td><a href=""
                class="btn btn-primary">Summary</a>

        </tr>
      {% endfor %}
      </tbody>
    </table>

  </div>
  </body>
  </html>
{% endblock %}
</html>
```

Now we need to set up our url path for this web page. We have an overall urls.py at the project level, but we also generally create one for each of the apps in our project. In the **mfscrm/crm** folder, create a Python file called urls.py and add the following to this file:

crm\urls.py

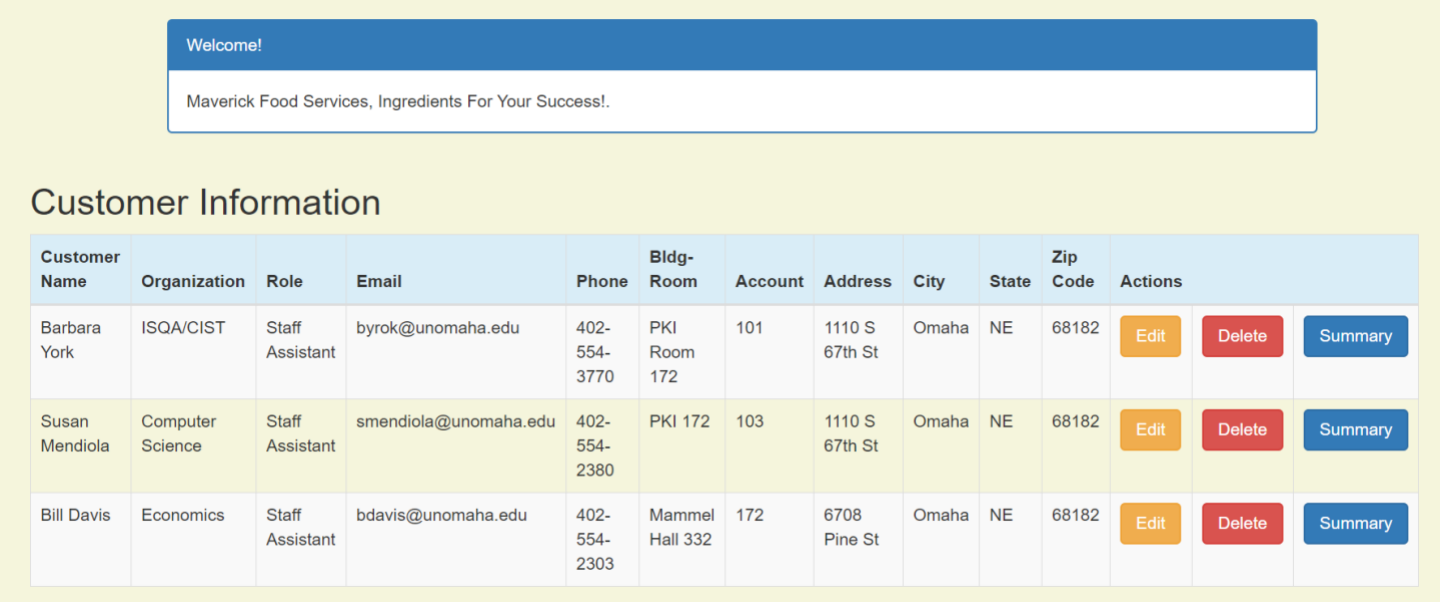
```
from django.conf.urls import url
from . import views
from django.urls import path

app_name = 'crm'
urlpatterns = [
    path('', views.customer_list, name='customer_list'),
    path('customer_list', views.customer_list, name='customer_list'),
]
```

In order for the application to use this urls.py file, we need to tell the **project-level urls.py** file about our crm urls.py. We do this by updating the **mfscrm/urls.py** as shown below. Be sure to add 'include' to the import statement on the 2nd line of the file. Note that the path('',include('crm.urls')), uses **2 single quotes** not a double-quote at the beginning.

mfscrm/urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
  path('admin/', admin.site.urls),
  path('', include('crm.urls')),
]
```

At this point you should be able to take an early peek at the customer list view. Run the server with
python manage.py runserver
In your browser, type in http://127.0.0.1:8000/customer_list
It should look like the screenshot below, assuming you added the 3 customers as instructed earlier:



Section 5 - Creating the ability to update information in the list.

Now we will create the ability to edit a record using the edit view. The process is similar to the above process except we will introduce the ability to edit a record in a template.

Default forms work in all environments. You do not need to install anything.

Next, create a **customer_edit.html** file in the **crm/templates/crm** directory. Right-click on templates/crm, choose “New” and choose “HTML File”. Name it **customer_edit** and click OK.

Our customer_edit file is going to serve as a form so we will use the “form action” html that will do the heavy lifting for our edits. Replace the code provided by the system by copy and pasting the following code into the customer_edit.html file:

customer_edit.html

```
<!DOCTYPE html>
<html lang="en">

{% block content %}
    <h1>Edit Customer</h1>
    <form method="POST" class="customer-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Update</button>
    </form>
{% endblock %}
</html>
```

3. Now that we have an html file for editing or updating customer information, let's configure the views.

Edit the crm/v*iews.py* file in crm, go ahead and ADD the following code:
views.py

Add at the top of views.py the 'get_object_or_404':
```
from django.shortcuts import render, get_object_or_404
```

THEN add the following at the bottom of the file:
```
def customer_edit(request, pk):
  customer = get_object_or_404(Customer, pk=pk)
  if request.method == "POST":
     # update
     form = CustomerForm(request.POST, instance=customer)
     if form.is_valid():
        customer = form.save(commit=False)
        customer.updated_date = timezone.now()
        customer.save()
        customer = Customer.objects.filter(created_date__lte=timezone.now())
        return render(request, 'crm/customer_list.html',
                      {'customers': customer})
  else:
      # edit
     form = CustomerForm(instance=customer)
  return render(request, 'crm/customer_edit.html', {'form': form})
```

4. Finally, we need to update the crm/urls.py with the highlighted code below:
crm/urls.py

```
from django.conf.urls import url
from . import views
from django.urls import path

app_name = 'crm'
urlpatterns = [
    path('', views.customer_list, name='customer_list'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
]
```

5. Next: update the **crm/templates/customer_list.html**.

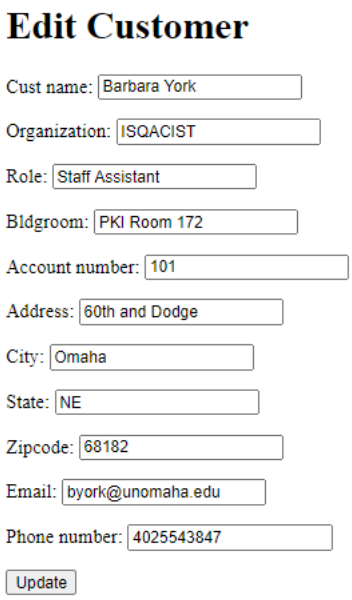
REPLACE this line in the middle of the file:
```
  <td><a href="" class="btn btn-warning">Edit</a>
```

WITH this line:
```
<td><a href="{% url 'crm:customer_edit' pk=customer.pk %}" class="btn btn-warning">Edit</a>
```

Notice how our url for customer_edit page differs from customer_list url? This is because customer list has all of our customers listed, but when we're making changes to our customer table, we are only making the change for one customer record at a time. This is why the primary key (pk) that Django automatically creates when we

enter information in our database is going to be used now. Thus we are using *.pk* inside the url path and customer.pk when declaring the HREF/URL path.

You should now be able to edit a record.
Run the server and click the 'Edit' button for a customer and make an edit to the address of one of your customers. Ensure it updates back in the list display after clicking update. The display of the customer edit form should look like the following:



Section 6 - Creating the ability to delete information in the list.

You will follow the same steps you used in activating the Edit button in activating the Delete button.
1. Start by creating a ***customer_delete.html*** file in **crm/templates/crm**. Inside of the file, add this code:

customer_delete.html

```
<!DOCTYPE html>
<html lang="en">
{% block body_block %}
    <h1>Delete {{ customer.name }}?</h1>

    <form method="post">
        {% csrf_token %}
        <input type="submit" class="btn btn-danger" value="Delete">
        <a href="{% url 'crm:home' pk=customer.cust_number %} ">Cancel</a>

    </form>

{% endblock %}

</html>
```

And **change** the **crm/templates/crm/customer_list.html** as follows:

REPLACE:

```
<td><a href=""
    onclick="return confirm('Are you sure you want to delete?')"
    class="btn btn-danger">Delete</a>
</td>
```

WITH:

```
<td><a href="{% url 'crm:customer_delete' pk=customer.pk %}"
    onclick="return confirm('Are you sure you want to delete?')"
    class="btn btn-danger">Delete</a>
</td>
```

2. Next we will update **crm/views.py** in 2 ways.

crm/views.py

Add this at the top of views.py after the other import statements:

```
from django.shortcuts import redirect
```

Add the following to the bottom of the views.py file:

```
def customer_delete(request, pk):
  customer = get_object_or_404(Customer, pk=pk)
  customer.delete()
  return redirect('crm:customer_list')
```

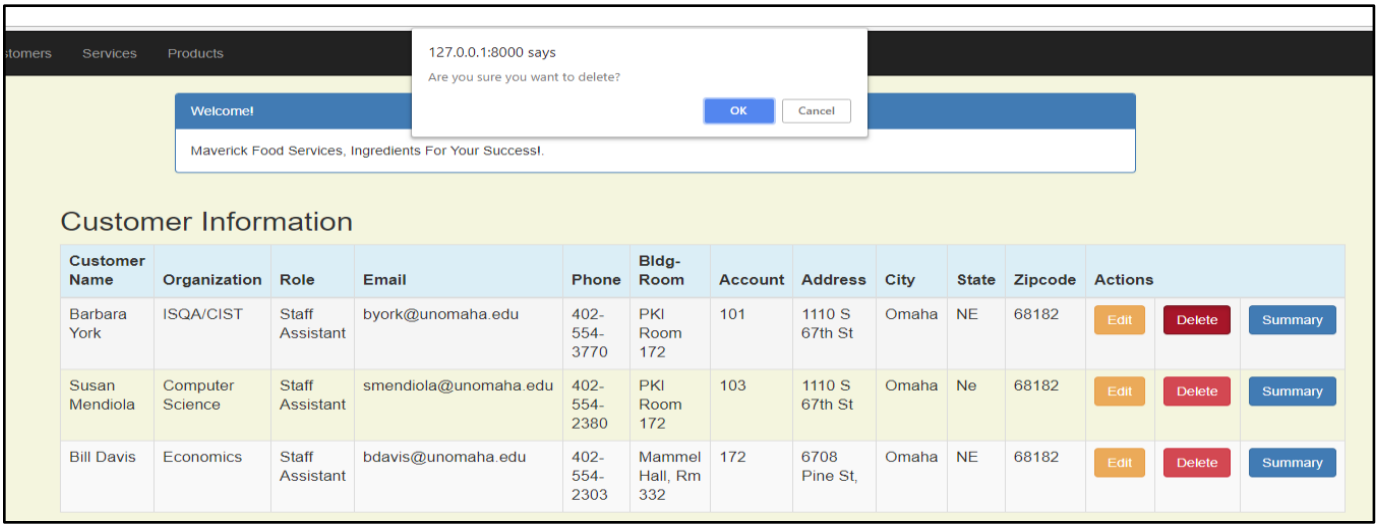
3. We then add the following to the **crm/urls.py** file

```
path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
```

The complete **crm/urls.py** file should now look like the following:

```
from django.conf.urls import url
from . import views
from django.urls import path

app_name = 'crm'
urlpatterns = [
    path('', views.customer_list, name='customer_list'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
]
```

Now, when you run the server and attempt to delete a customer record you should receive this:
Click OK if you would like to delete the customer.



Section 7 - Adding Services and Product Data Models

We have now shown how to list customer information, edit the information, and delete a customer record, all outside of the admin panel.

Now we need to allow our staff to add, update and delete services and products they sell to customers.

To begin we need to start by adding to the model. Add the highlighted code shown below to the **crm/models.py**: Note that this code continues onto the next page – be sure to copy-and-paste all the new code to add to the crm/models.py file:

```
from django.db import models
from django.utils import timezone


# Create your models here.
class Customer(models.Model):
    cust_name = models.CharField(max_length=50)
    organization = models.CharField(max_length=100, blank=True)
    role = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    bldgroom = models.CharField(max_length=100)
    address = models.CharField(max_length=200)
    account_number = models.IntegerField(blank=False, null=False)
    city = models.CharField(max_length=50)
    state = models.CharField(max_length=50)
    zipcode = models.CharField(max_length=10)
    phone_number = models.CharField(max_length=50)
    created_date = models.DateTimeField(
        default=timezone.now)
    updated_date = models.DateTimeField(auto_now_add=True)

    def created(self):
        self.created_date = timezone.now()
        self.save()

    def updated(self):
        self.updated_date = timezone.now()
        self.save()

    def __str__(self):
        return str(self.cust_name)


class Service(models.Model):
    cust_name = models.ForeignKey(Customer, on_delete=models.CASCADE, related_name='services')
    service_category = models.CharField(max_length=100)
    description = models.TextField()
    location = models.CharField(max_length=200)
    setup_time = models.DateTimeField(
        default=timezone.now)
    cleanup_time = models.DateTimeField(
        default=timezone.now)
    service_charge = models.DecimalField(max_digits=10, decimal_places=2)
    created_date = models.DateTimeField(
        default=timezone.now)
    updated_date = models.DateTimeField(auto_now_add=True)

    def created(self):
        self.created_date = timezone.now()
        self.save()
```

```
    def updated(self):
        self.updated_date = timezone.now()
        self.save()

    def __str__(self):
        return str(self.cust_name)

class Product(models.Model):
    cust_name = models.ForeignKey(Customer, on_delete=models.CASCADE, related_name='products')
    product = models.CharField(max_length=100)
    p_description = models.TextField()
    quantity = models.IntegerField()
    pickup_time = models.DateTimeField(
        default=timezone.now)
    charge = models.DecimalField(max_digits=10, decimal_places=2)
    created_date = models.DateTimeField(
        default=timezone.now)
    updated_date = models.DateTimeField(auto_now_add=True)

    def created(self):
        self.created_date = timezone.now()
        self.save()

    def updated(self):
        self.updated_date = timezone.now()
        self.save()

    def __str__(self):
        return str(self.cust_name)
```

After you have added these new tables to the model, you won't see any changes in your application until you make and migrate the changes.

Exit the application if it is currently running.

Create and run migrations by entering:
python manage.py makemigrations
Followed by:
python manage.py migrate

Next, to see the new tables in the admin panel, you will need to add the highlighted code to the admin.py. Be sure you add the imports of the Service and Product at the top of the file.

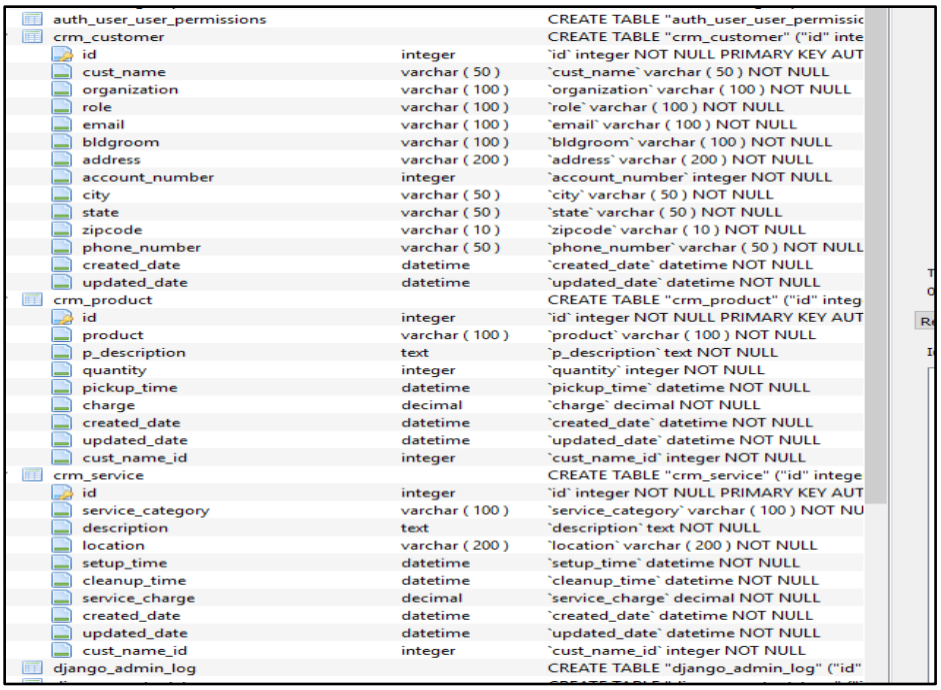
crm/admin.py

```
from django.contrib import admin

from .models import Customer, Service, Product


class CustomerList(admin.ModelAdmin):
    list_display = ( 'cust_name', 'organization', 'phone_number' )
    list_filter = ( 'cust_name', 'organization')
    search_fields = ('cust_name', )
    ordering = ['cust_name']


class ServiceList(admin.ModelAdmin):
    list_display = ( 'cust_name', 'service_category', 'setup_time')
    list_filter = ( 'cust_name', 'setup_time')
    search_fields = ('cust_name', )
    ordering = ['cust_name']


class ProductList(admin.ModelAdmin):
    list_display = ( 'cust_name', 'product', 'pickup_time')
    list_filter = ( 'cust_name', 'pickup_time')
    search_fields = ('cust_name', )
    ordering = ['cust_name']


admin.site.register(Customer, CustomerList)
admin.site.register(Service, ServiceList)
admin.site.register(Product, ProductList)
```

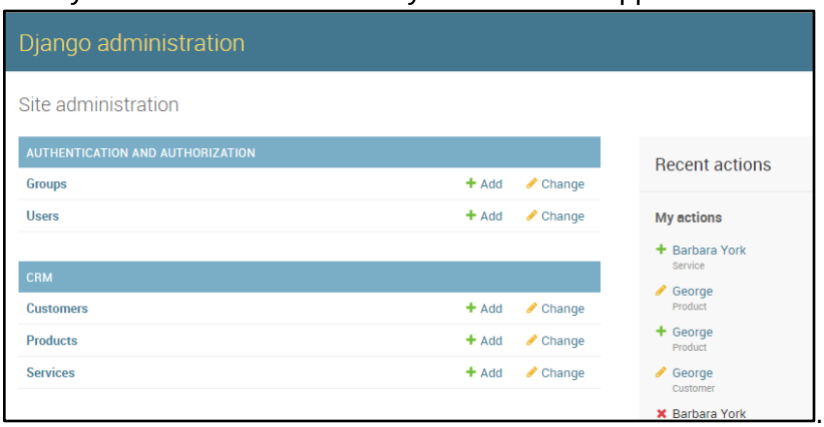
Let's take a minute and understand the relationship between tables.

As you can imagine, customer and services and customer and products are one to many relationships – each customer may have many services, and each customer may have many products. If you downloaded and opened the DB Browser for SQLite (mentioned earlier) from: https://sqlitebrowser.org/ and now open the **mfscrm/db.sqlite3** file, you will see the tables below. Each table created in Django comes with an auto-incrementing 'id' by default. You can see this in the illustration captured using DB Browser for SQLite below.

You will see the ID field in customer, service and product table. It is a NOT NULL, AUTO-INCREMENTING, INTEGER, PRIMARY KEY field for all tables. There is a customer_id field in both the product and service tables which is a foreign key referencing the ID field in the customer table.



Now you should see this when you restart the application and look at the admin panel:



Go ahead and add data now to the Services and Products tables. For starters we just need to have 1 service and 1 product for each of the 3 customers. Add values of your choice.

Section 8 - Adding a Home Page and Base Template

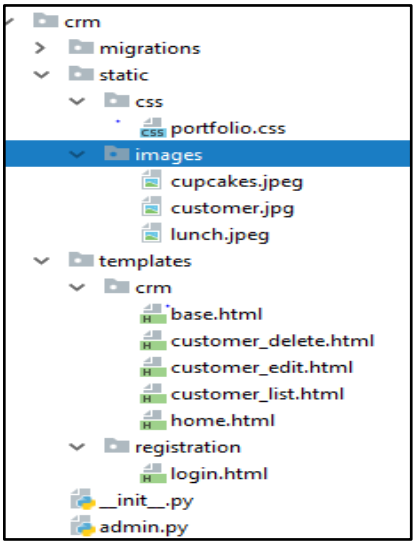
You should begin to see a pattern as you begin to build the **CRUD (create, read, update and delete)** pages for services and products. This tutorial shows you how to add a new service and list services.

You will then need to add the delete and edit functionality for services and then continue to add the product add, edit and delete functionality **on your own**.

Before continuing with the CRUD **(create, read, update and delete)** functionality, we will need to create a navigation page and add some simple security to these pages outside of the admin site.

First: Extract the static files (pictures, CSS. etc…) for the application provided for you in the zip file downloaded for this assignment on Canvas.

Unzip the files and review the code. You will see a folder named 'crm-files' with 2 sub-folders – 'registration' and 'static'
1. Move (or copy) the '**static'** folder into the **crm** directory in your project.
2. Move (or copy) the '**registration'** folder into the **crm/templates** folder.
Here is what your directory structure should look like when you have placed all the files in the appropriate places:



This adds login and logout page capability for a person that is not an administrator.

You can use exercises in your Django text to add security capability to this for extra credit.

Next, add the **base.html** file and the **home.html** files to the **templates/crm** directory. These files will give our site a home page. They will reference the jpg image files that are provided in the static files directory.

Replace the HTML code provided with the following in the **base.html** file to provide a common look and feel for all pages in this application. NOTE THIS CODE CONTINUES ON THE NEXT 3 PAGES. BE SURE TO COPY AND PASTE ALL THE CODE:
crm/templates/crm/base.html

```
<!DOCTYPE html>
{% load static %}

<html lang="en">
<head>
    <title>Maverick Food Services</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style>
```

```
        /* Remove the navbar's default margin-bottom and rounded borders */
        .navbar {
            margin-bottom: 0;
            border-radius: 0;
        }

        /* Set height of the grid so .sidenav can be 100% (adjust as needed) */
        .row.content {
            height: 450px
        }

        /* Set gray background color and 100% height */
        .sidenav {
            padding-top: 20px;
            background-color: #f1f1f1;
            height: 100%;
        }

        /* Set black background color, white text and some padding */
        footer {
            background-color: #555;
            color: white;
            padding: 15px;
        }

        /* On small screens, set height to 'auto' for sidenav and grid */
        @media screen and (max-width: 767px) {
            .sidenav {
                height: auto;
                padding: 15px;
            }

            .row.content {
                height: auto;
            }
        }
    </style>
</head>
<body id="app-layout">
<nav class="navbar navbar-inverse">
    <div class="container-fluid">
        <div class="navbar-header">

            <!-- Collapsed Hamburger -->
            <button type="button" class="navbar-toggle" data-toggle="collapse"
                    data-target="#myNavbar">
                <span class="sr-only">Toggle Navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>

            <!-- Branding Image -->
            <a class="navbar-brand" href="/">
                Maverick Food Service
            </a>
```

```
        </div>
        <div class="collapse navbar-collapse" id="myNavbar">
            <ul class="nav navbar-nav">
                <li><a href="{% url 'crm:home' %}">Home</a></li>
                <li><a href="{% url 'crm:customer_list' %}">Customers</a></li>
               <li><a href="{% url 'crm:customer_list' %}">Services</a></li>
                <li><a href="{% url 'crm:customer_list' %}">Products</a></li>

            </ul>
            <ul class="nav navbar-nav navbar-right">
                {% if user.is_authenticated %}
                    <li class="dropdown">
                        <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false">
                            <span class="caret"></span>
                        </a>
                        <ul class="dropdown-menu" role="menu">
                            <li><a href="{% url 'logout' %}"><i class="fa fa-btn fa-sign-
out"></i>Logout</a></li>
                        </ul>
                    </li>
                {% else %}
                    <li><a href="{% url 'login' %}"><span class="glyphicon glyphicon-log-
in"></span> Login</a></li>
                {% endif %}
            </ul>
        </div>
    </div>
</nav>
<div class="content container">
    <div class="row">
        <div class="col-md-8">
            {% block content %}
                <div class="links">
                    <!-- Example row of columns -->
                    <div class="row">
                        <div class="col-md-3">
                            <div class="thumbnail">
                                <img src="{% static "images/customer.jpg" %}"
alt="customer"/>
                                <div class="caption">
                                    <h2>Customer</h2>
                                    <p>Delighting customers and giving them a great dining
experience is our main goal. </p>
                                    {% if user.is_authenticated %}
                                        <p><a class="btn btn-default" href="{% url
'crm:customer_list' %}"
                                              role="button">View
                                           details &raquo;</a></p>
                                    {% endif %}
                                </div>
                            </div>
                        </div>
                        <div class="col-md-3">
                            <div class="thumbnail">
                                <img src="{% static "images/cupcakes.jpeg" %}"
```

```
alt="cupcakes"/>
                                <div class="caption">
                                    <h2>Products</h2>
                                    <p>Our food and services are second to none.</p>
                                    <p><a class="btn btn-default" href="{% url
'crm:customer_list' %}"
                                                role="button">View
                                            details &raquo;</a></p>
                                </div>
                            </div>
                        </div>
                        <div class="col-md-3">
                            <div class="thumbnail">
                                <img src="{% static "images/lunch.jpeg" %}" alt="lunch"/>
                                <div class="caption">
                                    <h2>Services</h2>
                                    <p>High quality products at affordable prices.</p>
                                    <p><a class="btn btn-default" href="{% url
'crm:customer_list' %}" role="button">View
                                            details &raquo;</a></p>
                                </div>
                            </div>
                        </div>

                    </div>
                </div>
            {% endblock %}
        </div>
    </div>
</body>
</html>
```

IMPORTANT NOTE: You should note that the above page references the "**crm:customer_list**" where you would expect it to see **service** or **product**. This is because we have not completed these pages and the base would generate an error at this point if we tried to access non-existent pages. **You will need to go back and change this when you add the additional functionality required in the assignment.**

Next, create the **crm/templates/crm/home.html** file and replace the provided code with the shown below. Note it also is references the "**crm:customer_list**" when it should be referencing services or products. This will be updated in a future step of the tutorial. NOTE- THIS CODE CONTINUES TO THE NEXT PAGE

crm/templates/crm/home.html

```
<!DOCTYPE html>
<html lang="en">
{% extends 'crm/base.html' %}
{% load static %}
{% block content %}

  <body>
  {% if user.is_authenticated %}
    <div class="page-container">

      <h2 class="top-menu">Hello {{ user.username }},</h2>
      <p>Please choose from below options.</p>

    </div>
  {% endif %}
  <div class="content container">
    <div class="row">
      <div class="col-md-12">
        <div class="links">
          <!-- Example row of columns -->
          <div class="row">
            <div class="col-md-3">
              <div class="thumbnail">
                <img src="{% static "images/customer.jpg" %}" alt="customer"/>
                <div class="caption">
                  <h2>Customer</h2>
                  <p>We see our success in our customers success </p>
                  <p><a class="btn btn-primary" href="{% url 'crm:customer_list' %}"
                      role="button">View
                    details &raquo;</a></p>
                </div>
              </div>
            </div>
            <div class="col-md-3">
              <div class="thumbnail">
                <img src="{% static "images/cupcakes.jpeg" %}" alt="cupcakes"/>
                <div class="caption">
                  <h2>Services</h2>
                  <p>Great Services</p>
                  <p><a class="btn btn-primary" href="{% url 'crm:customer_list' %}"
                      role="button">View
                    details &raquo;</a></p>
                </div>
              </div>
            </div>
            <div class="col-md-3">
              <div class="thumbnail">
                <img src="{% static "images/lunch.jpeg" %}" alt="lunch"/>
                <div class="caption">
```

```
                        <h2>Products</h2>
                        <p>Great Products.</p>
                        <p><a class="btn btn-primary" href="{% url 'crm:customer_list' %}"
                            role="button">View
                            details &raquo;</a></p>
                    </div>
                </div>
            </div>

        </div>
    </div>
</div>
</body>

{% endblock %}

</html>
```

Next, add the highlighted code shown below to the **crm/views.py**. There is the updated views.py. NOTE THIS CODE CONTINUES ON THE NEXT PAGE:

```
from django.contrib.auth.decorators import login_required
from django.shortcuts import render
from .models import *
from .forms import *
from django.shortcuts import render, get_object_or_404
from django.shortcuts import redirect



now = timezone.now()
def home(request):
   return render(request, 'crm/home.html',
                 {'crm': home})

@login_required
def customer_list(request):
    customer = Customer.objects.filter(created_date__lte=timezone.now())
    return render(request, 'crm/customer_list.html',
                  {'customers': customer})

@login_required
def customer_edit(request, pk):
   customer = get_object_or_404(Customer, pk=pk)
   if request.method == "POST":
       # update
       form = CustomerForm(request.POST, instance=customer)
       if form.is_valid():
           customer = form.save(commit=False)
```

```
            customer.updated_date = timezone.now()
            customer.save()
            customer = Customer.objects.filter(created_date__lte=timezone.now())
            return render(request, 'crm/customer_list.html',
                          {'customers': customer})
    else:
        # edit
        form = CustomerForm(instance=customer)
    return render(request, 'crm/customer_edit.html', {'form': form})

@login_required
def customer_delete(request, pk):
    customer = get_object_or_404(Customer, pk=pk)
    customer.delete()
    return redirect('crm:customer_list')
```

Next, update the URLS.
Replace the contents of the **crm/urls.py** with the code shown here:
crm/urls.py

```
from django.conf.urls import url
from . import views
from django.urls import path, re_path

app_name = 'crm'
urlpatterns = [
    path('', views.home, name='home'),
    re_path(r'^home/$', views.home, name='home'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
]
```

Replace the contents of the **mfscrm\urls.py** with the code shown here:
mfscrm\urls.py

```
from django.conf.urls import url, include
from django.contrib import admin
from django.urls import path


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('crm.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Update each of the template files to insert the reference to the **base.html** file.
Insert:
{% **extends 'crm/base.html'** %}
just after the `<html lang="en">` line (line 2) of each of the following files in the **crm/templates/crm** folder:

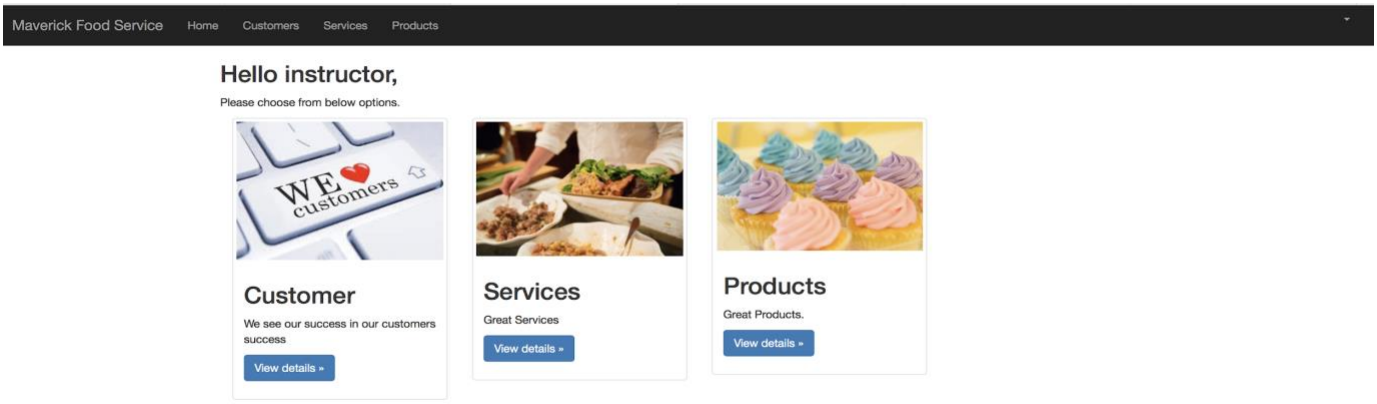
customer_edit.html
customer_list.html

Next add this line in the **mfscrm/settings.py** at the end or bottom of the file:

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```

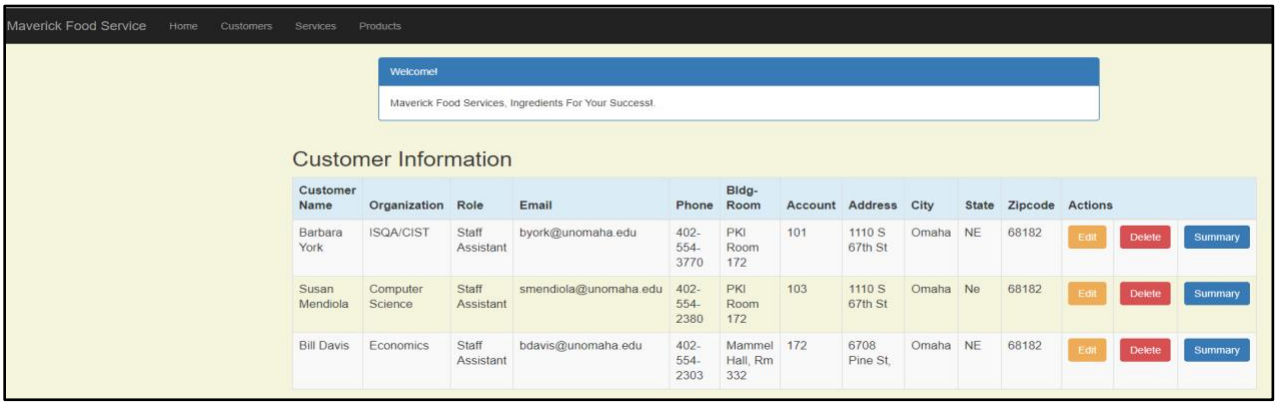
Now it is time to check out what this looks like. Issue this command again in the Terminal window:
python manage.py.runserver.

You should now see the homepage when you at the http://127.0.0.1:8000/ or the http://127.0.0.1:8000/home/. It should look like this. A sample style and jpeg files are used, but you are welcome to change these to other images if desired. Just be professional!

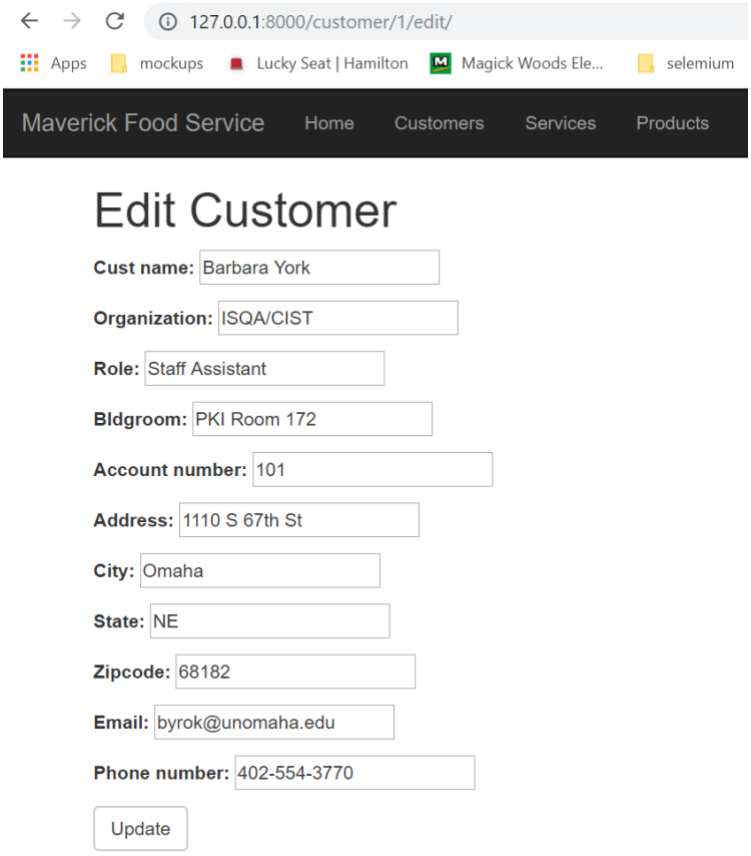


You can now the navigation provided by **base.html** at the top of the page. Currently, when you click on **services** and **products** you will find this redirects the link to the customer list to avoid an error.
When you add the code to work with services and products, **you will need to replace the links.**

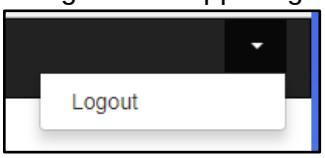
When you click on View details for customers or the Customers link in the navigation bar, you should see something like this:



When you click Edit for a Customer, you should see something like this:



Now let's check out the basic security we have added for this application. Logout by clicking on the small triangle in the upper right hand corner of the navigation bar.



You should arrive back at the home page. If you attempt to access any of the data without logging in, you will be prompted for a password.
There is much more you can do to extend the features of the built-in security model of Django. This includes adding a “forget your password”, sign up a new user, and a change your password features. You are welcome to add these features for extra credit. Your textbook gives great examples for these features.

9. Adding the ability to add, edit and delete Services for a customer outside the admin panel

If you examine the model fields for the Services entries, and review how we created views, templates and URL entries for the Customer, you should have a good idea of how to add the services and product features.

This tutorial go through adding the list of services and adding a new service.
For the remainder of the assignment, you will need to follow the pattern and create the ability to edit and delete a service for a customer outside the admin panel.

Let's begin by adding the service list template. Create a **service_list.html** file in the **crm/templates/crm** directory. Replace the provided code with the following code – NOTE THIS CODE CONTINUES TO THE NEXT PAGE:

crm/templates/crm/service_list.html

```
<!DOCTYPE html>
{% extends 'crm/base.html' %}
{% block content %}
  <head>
    <meta charset="UTF-8">
    <title>Maverick Food service</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">

  </head>
  <body>
  <style>
    body {
      background-color: beige;
    }
  </style>
  <div class="container">
    <div class="row">
      <div class="col-md-10 col-md-offset-1">
        <div class="panel panel-primary">
          <div class="panel-heading">Welcome!</div>
          <div class="panel-body">
            Maverick Food services, Ingredients For Your Success!.
          </div>
        </div>
      </div>
    </div>
  </div>
  <div class="row">
    <h2 style="padding-left: 15Px">Services Information</h2>
  </div>
  <div>
    <table class="table table-striped table-bordered table-hover">
      <thead>
      <tr class="bg-info">
        <th>Customer</th>
        <th>Service Category</th>
        <th>Description</th>
        <th>Location</th>
        <th>Setup Time</th>
        <th>Cleanup Time</th>
        <th>Service Charge</th>
        <th colspan="3">Actions</th>
      </tr>
      </thead>
      <tbody>
      {% for service in services %}
        <tr>
          <td>{{ service.cust_name }}</td>
          <td>{{ service.service_category }}</td>
```

```
            <td>{{ service.description }}</td>
            <td>{{ service.location }}</td>
            <td>{{ service.setup_time }}</td>
            <td>{{ service.cleanup_time }}</td>
            <td>{{ service.service_charge }}</td>
            <td><a href=""
                class="btn btn-warning">Edit</a></td>
            <td><a href=""
                class="btn btn-danger">Delete</a>
            </td>
          </tr>
        {% endfor %}
        </tbody>
      </table>

   </div>


   </body>
{% endblock %}
```

Next, **add** the following to the bottom of the **crm/views.py** file after the code to define customer_delete:

crm/views.py

```
@login_required
def service_list(request):
  services = Service.objects.filter(created_date__lte=timezone.now())
  return render(request, 'crm/service_list.html', {'services': services})
```

Add the highlighted line to the **crm/urls.py – be sure to include the comma at the end**.

crm/urls.py

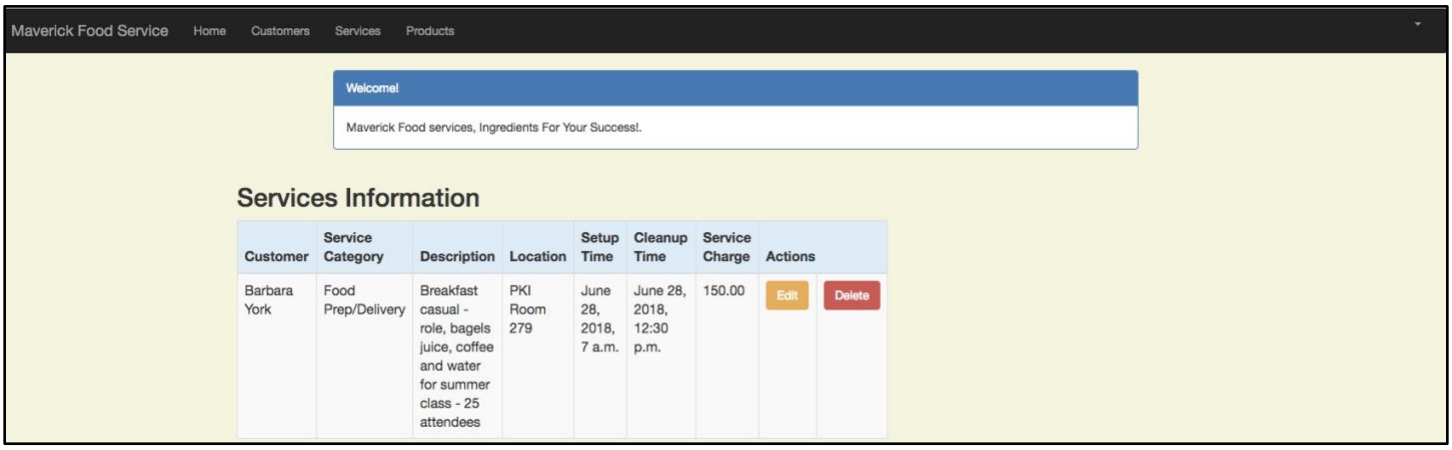
```
from django.conf.urls import url
from . import views
from django.urls import path, re_path

app_name = 'crm'
urlpatterns = [
    path('', views.home, name='home'),
    re_path(r'^home/$', views.home, name='home'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
    path('service_list', views.service_list, name='service_list'),
]
```

In the earlier steps of this tutorial, we put the ‘customer_list.html’ in place of the not-yet-created service_list.html page in the home.html and base.html files. This allowed the application to run without any errors due to the missing service_list.html file. Edit the **home.html** and **base.html** and change the **customer_list.html** to the **service_list.html** in the **3 places** that apply to Services. You will need to do this a total of 3 times between the two pages.

Now it's time to test out our changes. Again start of web app server with the command:
python manage.py runserver\admin

- **Add a service for one of your customers**

Then return to the Home screen and view the Services page.
You should see something similar to the following – A service was added to Barbara York in this example:



The Edit and Delete buttons do not function yet, but you will fix these in a similar manner as you did with the customer page above.

Next we will activate the 'Add Service' Button.

Step 1 - To do this we will need to add the **service_new.html** template file the **crm/templates/crm** directory. Replace the provided code with the code shown below:
crm/templates/crm/service_new.html

```
<!DOCTYPE html>
<html lang="en">
{% extends 'crm/base.html' %}
{% block content %}
   <h1>Add a New Service</h1>
   <form method="POST" class="service-form">{% csrf_token %}
        {{ form.as_p }}
      <button type="submit" class="save btn btn-default">Save</button>
   </form>

{% endblock %}

</html>
```

Step 2 - We need a form to add or edit service records, so we need to add the following to the **forms.py** file. Edit **crm/forms.py** and add the highlighted code:

crm/forms.py

```
from django import forms
from .models import Customer, Service

class CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = ('cust_name', 'organization', 'role', 'bldgroom', 'account_number',
'address',
                  'city', 'state', 'zipcode', 'email','phone_number')


class ServiceForm(forms.ModelForm):
   class Meta:
       model = Service
       fields = ('cust_name', 'service_category', 'description', 'location', 'setup_time',
'cleanup_time', 'service_charge' )
```

Step 3 - We need to update the **crm/views.py** to include the service_new. Add the code shown below to the bottom of the file – just below the definition of service_list:

crm/views.py

```
@login_required
def service_new(request):
  if request.method == "POST":
     form = ServiceForm(request.POST)
     if form.is_valid():
        service = form.save(commit=False)
        service.created_date = timezone.now()
        service.save()
        services = Service.objects.filter(created_date__lte=timezone.now())
        return render(request, 'crm/service_list.html',
                  {'services': services})
  else:
     form = ServiceForm()
     # print("Else")
  return render(request, 'crm/service_new.html', {'form': form})
```

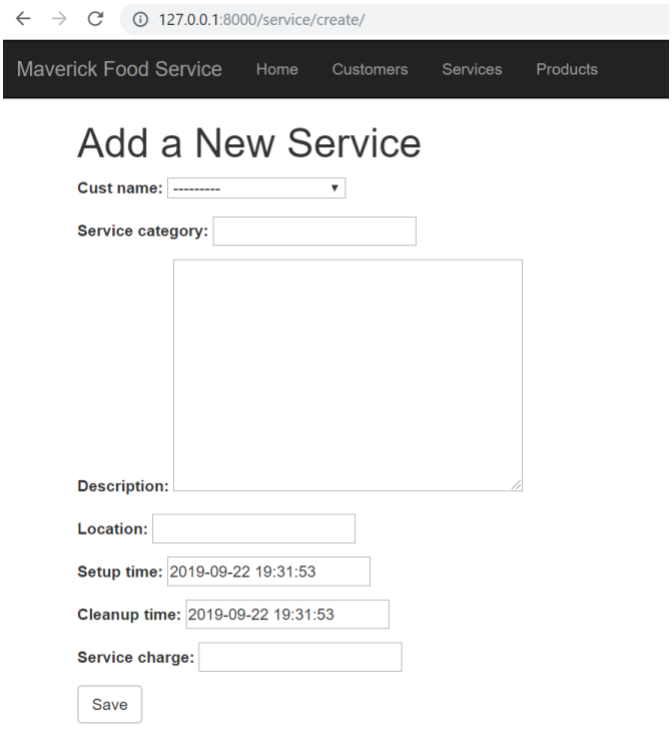
Step 4 – Next, we need to add the highlighted line to the **crm/urls.py** file:

```
from django.conf.urls import url
from . import views
from django.urls import path, re_path

app_name = 'crm'
urlpatterns = [
    path('', views.home, name='home'),
    re_path(r'^home/$', views.home, name='home'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
    path('service_list', views.service_list, name='service_list'),
    path('service/create/', views.service_new, name='service_new'),
]
```

Step 5 – In order to prevent an error initially, the URL for the Add Service button was not added to service_list.html. We now need to add it. Add the URL as shown here to the existing div class for 'Add Service' in the **crm/templates/crm/service_list.html** :
Add the code shown below near the bottom of the **service_list.html**, on the line immediately before the </body> html tag.

```
<div class="row">
   <a href="{% url 'crm:service_new' %}" class="row"><span
          class="btn btn-primary">Add Service</span></a>
</div>
```

You should now be able to add a new service to your current customers by clicking the 'Add Service' button when viewing the list of services. When you press this Add Service button you should see:



Next, add the functionality to edit a Service.

Step 1 - To add the functionality to edit a service, add the **service_edit.html** template page to the **crm/templates/crm** folder and replace the provide code with the code shown below:

crm/templates/crm/service_edit.html

```
<!DOCTYPE html>
<html lang="en">
{% extends 'crm/base.html' %}
{% block content %}
    <h1>Edit Service</h1>
    <form method="POST" class="service-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Update</button>
    </form>
{% endblock %}

</html>
```

Step 2 - Since we previously added the form for adding or editing services, we need to nothing more with forms.py.

Step 3 – Update the **crm/views.py** by adding the following code to the bottom of the existing file after the definition of the service_new:

crm/views.py

```
@login_required
def service_edit(request, pk):
    service = get_object_or_404(Service, pk=pk)
    if request.method == "POST":
        form = ServiceForm(request.POST, instance=service)
        if form.is_valid():
            service = form.save()
            # service.customer = service.id
            service.updated_date = timezone.now()
            service.save()
            services = Service.objects.filter(created_date__lte=timezone.now())
            return render(request, 'crm/service_list.html', {'services': services})
    else:
        # print("else")
        form = ServiceForm(instance=service)
    return render(request, 'crm/service_edit.html', {'form': form})
```

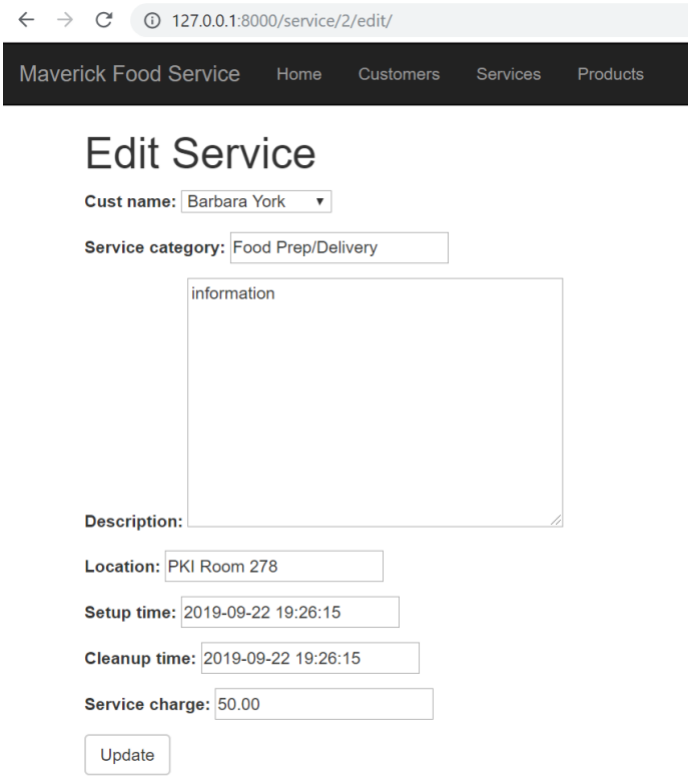
Step 4 - Next, we need to add the highlighted line to the **crm/urls.py** file

```
from django.conf.urls import url
from . import views
from django.urls import path, re_path

app_name = 'crm'
urlpatterns = [
    path('', views.home, name='home'),
    re_path(r'^home/$', views.home, name='home'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
    path('service_list', views.service_list, name='service_list'),
    path('service/create/', views.service_new, name='service_new'),
    path('service/<int:pk>/edit/', views.service_edit, name='service_edit'),
 ]
```

Step 5 - Don't forget, we allowed the buttons to show up on the initial service_list.html page by using a blank URL. Now we must fill it in with the correct URL as shown below (this should be at or near line 55 in the file).

```
<td><a href="{% url 'crm:service_edit' pk=service.pk %}"
                    class="btn btn-warning">Edit</a></td>
```

You should now be able to edit an existing service for your current customers by clicking the 'Edit' button when viewing the list of Services. When you press this Edit Service button you should see something similar to the following:



Great work!! You have now learned how to create the essential elements of a typical CRUD **(create, read, update and delete)** Django application.

TO DO: Section 9. Now it is your turn! Follow the pattern demonstrated in the steps above.

Complete the following on your project:

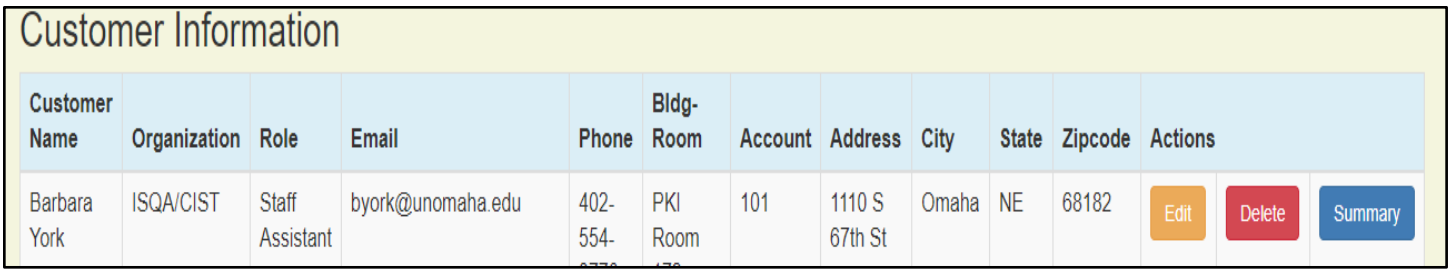
1. A food service manager should be able to **delete services** of their customers.
2. A food service manager should be able to **list all the products** sold to their customers
3. A food service manager should be able to **add a new product** sold to their customers
4. A food service manager should be able to **edit a product** sold to their customers
5. A food service manager should be able to **delete a product** sold to their customers

TO DO: Section 10. Adding the Summary Page to Our Project

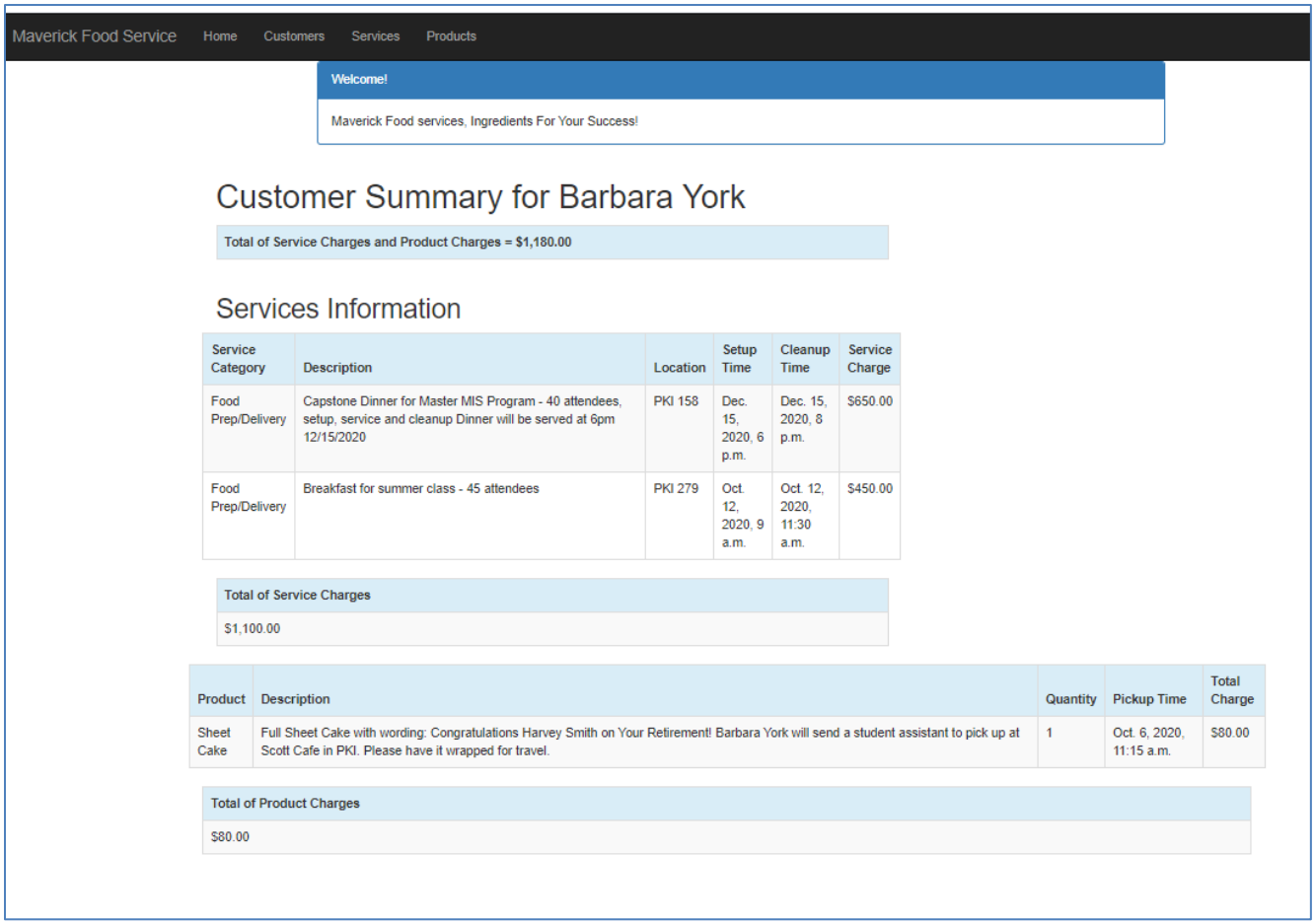
A food service manager should be able to **create a summary report** of all services and products sold to a customer.

One of the major goals of any customer relationship management system to clearly show what customers bought from us on a regular basis and who are our top clients. If you want to answer these questions, you can easily extract the data using Python and save it to a csv file for analysis with tools like Excel. However, you also want to see the recent purchases of a given customer without extracting data and analyzing offline.

Utilizing the steps on the following pages, we have created a simple report which summarizes the recent purchases of both services and products. **Your next task is to create that view**. Below is a snapshot of the view we are looking for when the 'Summary' button on the 'Customer Information' page is clicked:



Once you click the summary button shown above generate the following report:



Step 1 - To start, we now need to install some additional packages. Be sure you have terminated the application, if it was running, and that your virtual machine is running (i.e, ensure you see (venv) in front of your prompt in the Terminal window). Once you have confirmed this, then you will add the Django Mathfilters. This component allows you to display and do simple math in a template. To install this package, enter the following command in the Terminal window:

pip install django-mathfilters

You should see something like this:

```
(venv) C:\Users\Sandy\Desktop\Fall2020_PycharmProjects\mfscrm>pip install django-mathfilters
Collecting django-mathfilters
  Using cached https://files.pythonhosted.org/packages/8a/c6/107083a63a564664830e352af330563763654972d27d56e42d9b6e3c744f/django_mathfilters-1.0.0-py3-none-any.whl
Installing collected packages: django-mathfilters
Successfully installed django-mathfilters-1.0.0
```

Next – add django-mathfilters to the **requirements.txt** file.

Next add **'mathfilters'** and **'django.contrib.humanize',** to the INSTALLED_APPS in **mfscrm/settings.py** as shown below – be sure to include the quotes and commas exactly as shown:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crm.apps.CrmConfig',
    'mathfilters',
    'django.contrib.humanize',
]
```

Notice You need to add the **django.contrib.humanize**, but you do not need to pip install it.
More information on how to use Django mathfilters can be found at the link below:
https://pypi.org/project/django-mathfilters/

Django.contrib.humanize is actually a part of Django and documentation can be found at:
https://docs.djangoproject.com/en/2.0/ref/contrib/humanize/
It allows you to format numbers with placement of commas etc.

Step 2. – Next, use the two calculated values in the views and a summary.html template. In the **crm/views.py** add the following:

```
from django.db.models import Sum
from _decimal import Decimal
```

Then add the following to the bottom of **crm/views.py** to define our summary:

```
@login_required
def summary(request, pk):
    customer = get_object_or_404(Customer, pk=pk)
    customers = Customer.objects.filter(created_date__lte=timezone.now())
    services = Service.objects.filter(cust_name=pk)
    products = Product.objects.filter(cust_name=pk)
    sum_service_charge = \
        Service.objects.filter(cust_name=pk).aggregate(Sum('service_charge'))
    sum_product_charge = \
        Product.objects.filter(cust_name=pk).aggregate(Sum('charge'))

    # if no product or service records exist for the customer,
    # change the 'None' returned by the query to 0.00
    sum = sum_product_charge.get("charge__sum")
    if sum== None:
        sum_product_charge = {'charge__sum' : Decimal('0')}
    sum = sum_service_charge.get("service_charge__sum")
    if sum== None:
        sum_service_charge = {'service_charge__sum' : Decimal('0')}

    return render(request, 'crm/summary.html', {'customer': customer,
                                'products': products,
                                'services': services,
                                'sum_service_charge': sum_service_charge,
                                'sum_product_charge': sum_product_charge,})
```

The above Python function, summary, makes all elements of the Customer, Service and Product objects available. These values are then summarized using the Django **aggregation function** called **Sum**. Now we are ready to develop the **summary.html** template to display the summary.

Step 4 - The summary page is an HTML template that is *started* for you below. Add a new HTML file to the **crm/templates/crm** folder and name it **summary.html.** You will need to create the remaining parts of the summary page for this assignment.
Below is the *starter* **crm/templates/crm/summary.html** file. Replace the provided code for summary.html with the code shown below – NOTE: This code continues to the next page.

```
{% extends 'crm/base.html' %}
{% block content %}
{% load mathfilters %}
{% load humanize %}

<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            <div class="panel panel-primary">
                <div class="panel-heading">Welcome!</div>
                <div class="panel-body">
                    Maverick Food services, Ingredients For Your Success!
                </div>
            </div>
        </div>
    </div>
</div>
<div class="row">
  <h1 style="padding-left: 15Px">Customer Summary for {{ customer.cust_name }}</h1>

</div>

    <div class="row">
        <h2 style="padding-left: 15Px">Product Information</h2>
    </div>
    <div class="row">
        <table class="table table-striped table-bordered table-hover">
            <thead>
            <tr class="bg-info">
                <th>Product</th>
                <th>Description</th>
                <th>Quantity</th>
                <th>Pickup Time</th>
                <th>Total Charge</th>

            </tr>
            </thead>
            <tbody>
            {% for product in products %}
                <tr>
                    <td>{{ product.product }}</td>
                    <td>{{ product.p_description }}</td>
                    <td>{{ product.quantity|intcomma }}</td>
                    <td>{{ product.pickup_time }}</td>
```

```
                    <td>{{ product.charge|intcomma }}</td>
                </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
        <table class="table table-striped table-bordered table-hover">
        <thead>
            <tr class="bg-info">
                <th>Total of Product Charges</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>{{ sum_product_charge.charge__sum|intcomma }}</td>

            </tr>
        </tbody>
    </table>
</div>

{% endblock %}
```

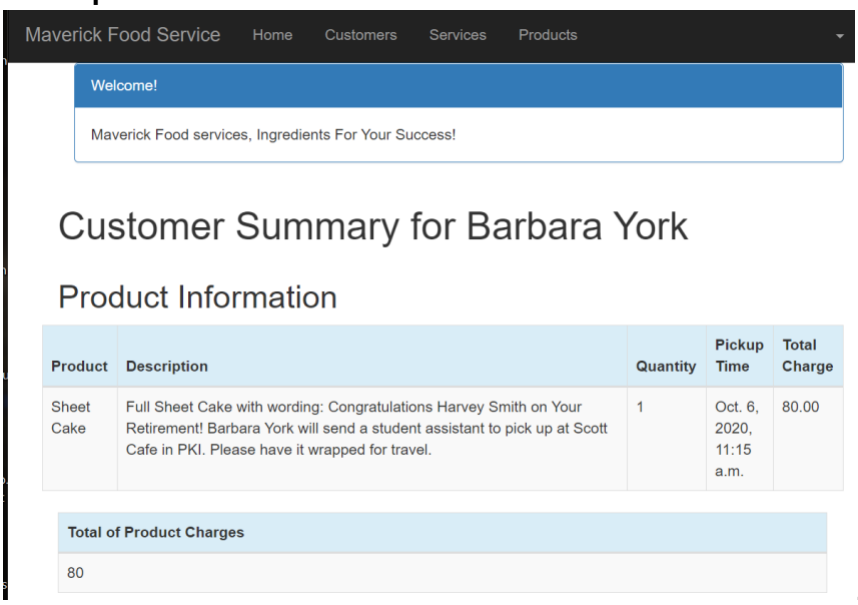
Step 5 – To allow the summary to be visible, add the highlighted line to the **crm\urls.py**

```
from django.conf.urls import url
from . import views
from django.urls import path, re_path

app_name = 'crm'
urlpatterns = [
    path('', views.home, name='home'),
    re_path(r'^home/$', views.home, name='home'),
    path('customer_list', views.customer_list, name='customer_list'),
    path('customer/<int:pk>/edit/', views.customer_edit, name='customer_edit'),
    path('customer/<int:pk>/delete/', views.customer_delete, name='customer_delete'),
    path('service_list', views.service_list, name='service_list'),
    path('service/create/', views.service_new, name='service_new'),
    path('service/<int:pk>/edit/', views.service_edit, name='service_edit'),
    path('customer/<int:pk>/summary/', views.summary, name='summary'),
]
```

Step 6 - The summary.html also needs a button to activate it. Edit the **crm/templates/crm/customer_list.html** and replace the empty href in the Summary button object with the highlighted URL shown below (at or near line 77 in the file):

```
<td><a href="{% url 'crm:summary' pk=customer.pk %}"
                    class="btn btn-primary">Summary</a>
```

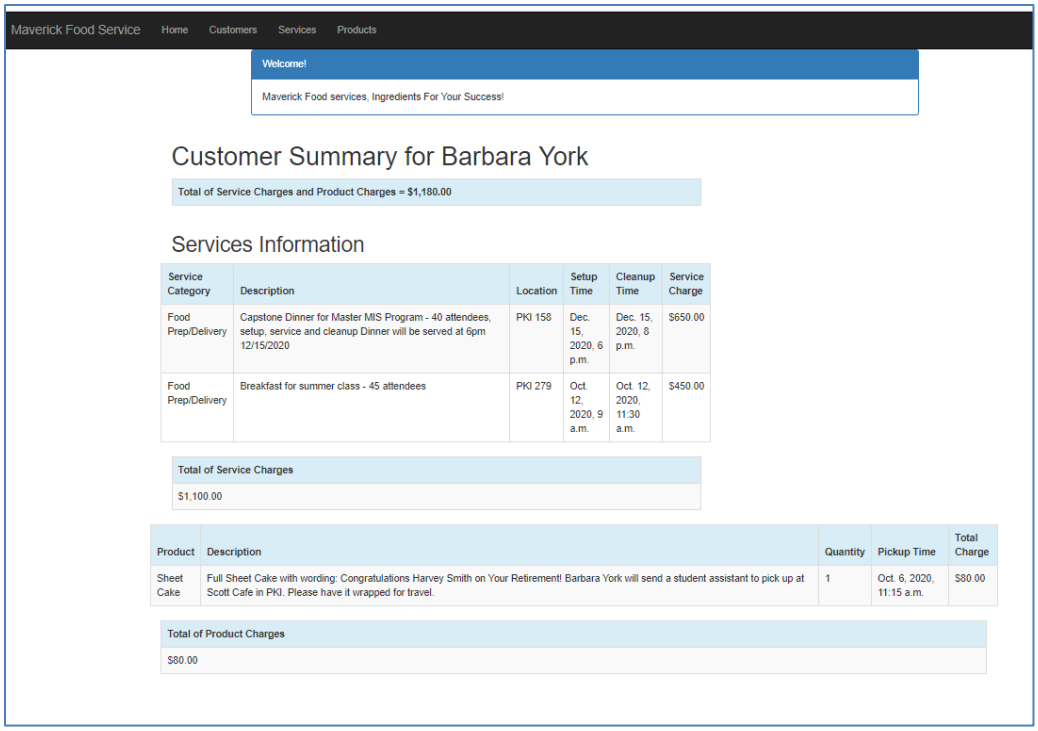
Return to the application, view the Customer List, and click the Summary button for one of your customers to view the partial customer summary.
A sample screen is shown below.



REMINDER: This is only a partial version of the customer summary

Using this starter, shell you need to develop the complete summary similar to the sample shown below. It does NOT need to look exactly like the one below. There are many ways to summarize a summary. Check out the two tools you will now use to do some of the math and formatting
https://docs.djangoproject.com/en/2.0/ref/contrib/humanize/
https://pypi.python.org/pypi/django-mathfilters

Now it's your turn. Complete the code in the summary.html template so that it works as shown below.



Step 7 - Once you have completed all the items listed above and tested to ensure everything is working on your own local machine:

1. Create a 'requirements.txt' to include all the packages and software required for your application. The requirements.txt file for this project should include the most recent version of Django:

   ```
   django==3.1.1
   django-mathfilters
   ```

   Note: Every project deployed to PythonAnywhere must have a requirements.txt file containing a list of the required packages for the deployed application.

2. Create a '.gitignore' file. It should contain:

   *.pyc
   db.sqlite3
   venv
   __pycache__
   local_settings.py
   .idea

3. Push your code to the GitHub repo provided for Assignment 2 posted with this Assignment on Canvas. Publish your project to GitHub.

4. Create a new repository for deploying to PythonAnywhere. Deploy your app to PythonAnywhere. You will need to use the --nuke option when you create your app in order to replace the App you created for Assignment 1 in this class. Provide a link to your deployed app on the Canvas Assignment submission link.

Step 8 - Congratulate yourself for learning how to develop, test and deploy a multi-table Django project!!!

Be sure to return to the Assignment sheet to complete the additional requirements for this assignment.