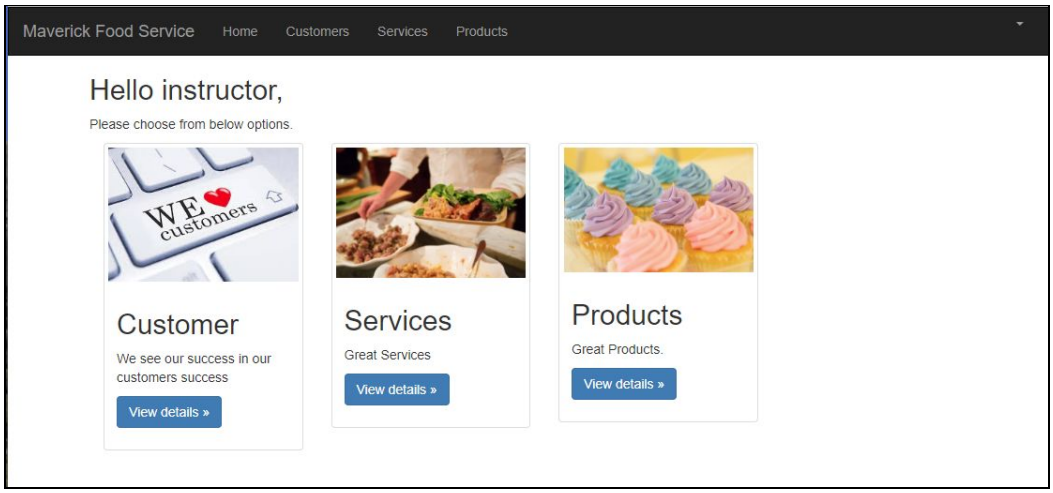
Maverick Food Services Customer Relationship Tutorial Using Python Django 2.0.X

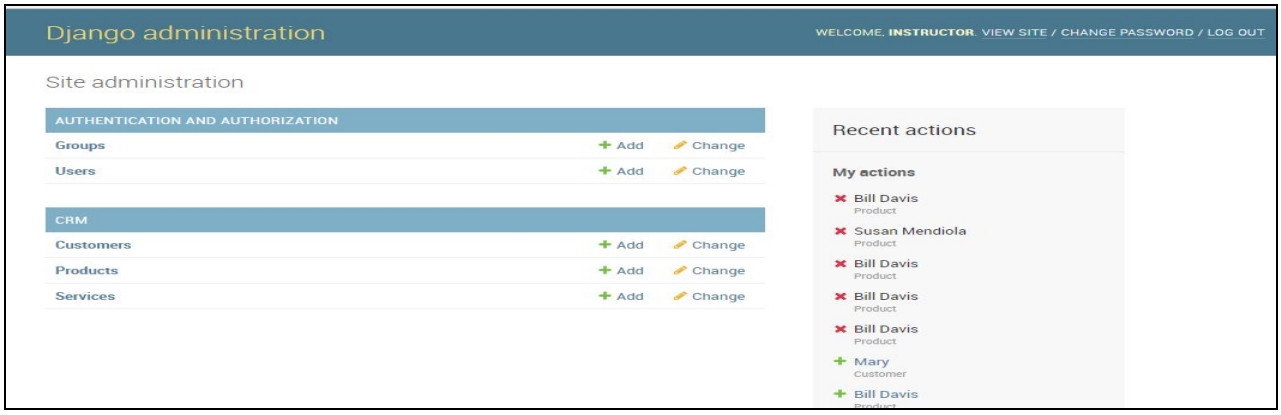
In this tutorial you will be developing a multi-table application using Python Django. Unlike the previous tutorial, this tutorial will give you a good start on the application but you will need to create several of the pages on your own. I use the Maverick Food Services as a name for a fictitious organization which provides food services in a campus setting. Imagine if you were a catering manager for the food services organization and needed to keep track of their customers and want to grow their profits by marketing to these customers and their colleagues. To do that you need some type of a database to track your customers and the products and services they acquire from you over time. This is sometime called a customer information database or a customer relationship management database or CRM for short. We will be building a CRM for the Maverick Food services Organization. I am providing an application shell which starts the application but you must complete it both on your local machine and deploy the application to the Heroku cloud hosting facility or to PythonAnywhere. Below are typical starter pages.

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

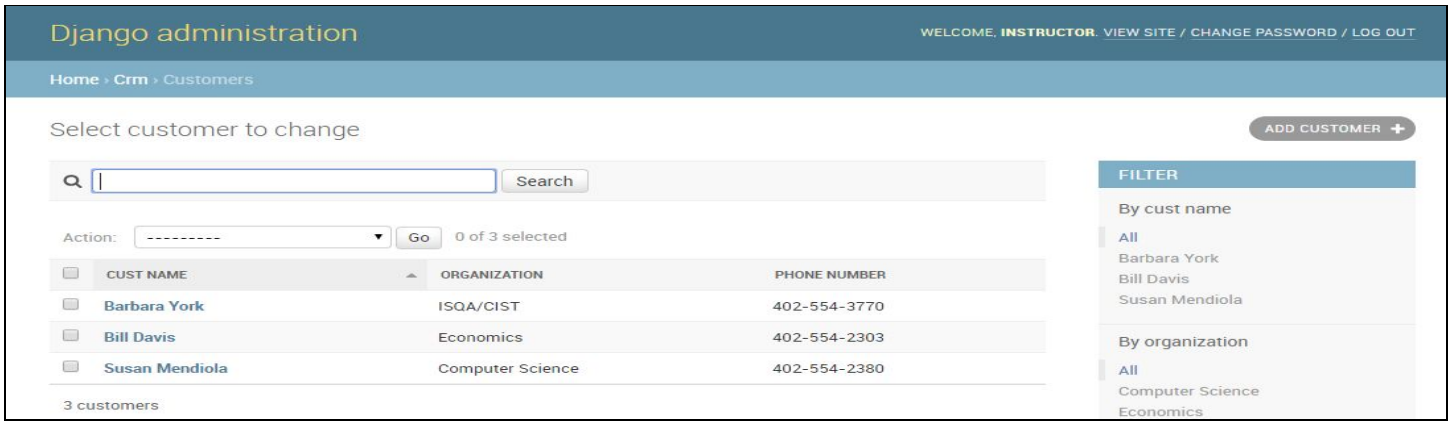
Sample Landing Page for the Maverick Food Service CRM



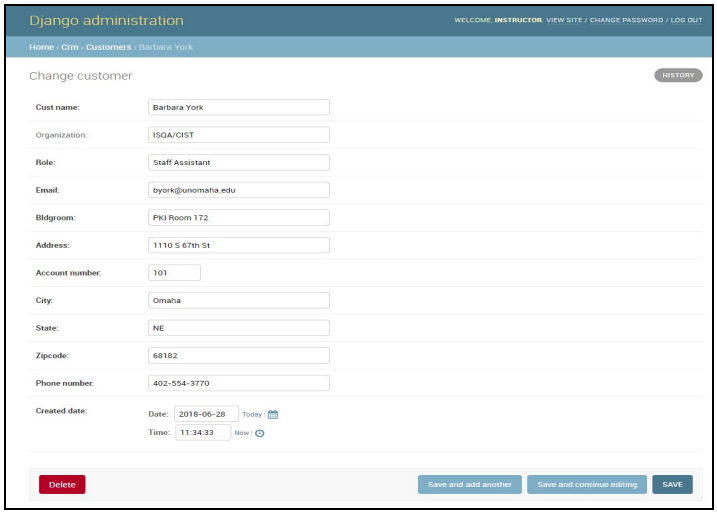
There are two roles in this sample application. The administrators have access to the the information for all customers, services and products. They can also create and delete accounts from the system. Employees of Maverick Food Service login in from this screen The administrators have access to the Admin section of the Django. They are capable of setting up new employee accounts. The employee of Maverick Food Service on the other hand uses the Maverick Food Service page to login. Below you see the Admin panel of the application available to administrators. As you learned in the blog you simply type in /admin to access the url.



You are familiar with the Groups and Users in the admin panel from first Blog exercise. In our application, a given customer can acquire many services and and products.Here is a view of the customers and the ability to add update and delete customers:

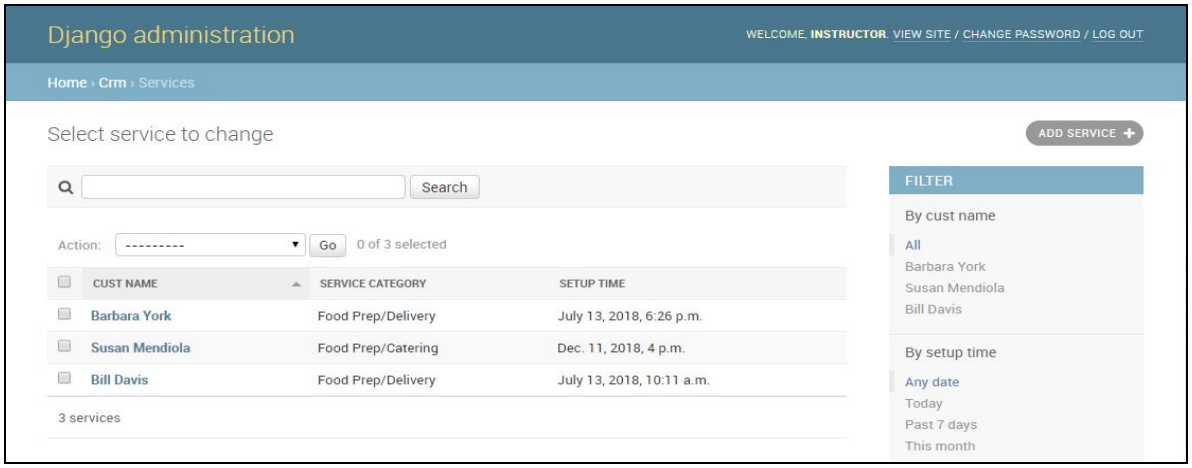


Here is an edit and update of one of the records:

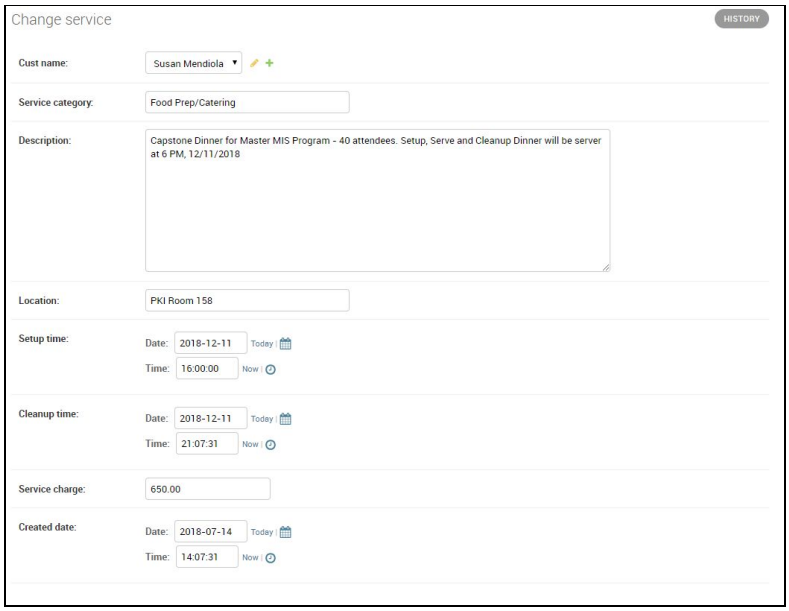


Note the history. This points out the logging capabilities provided by Django.

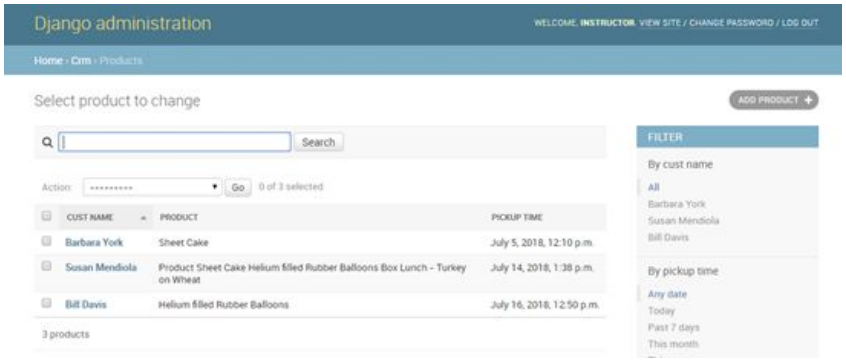
Below you find a list of the service transaction in the Service Table.



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

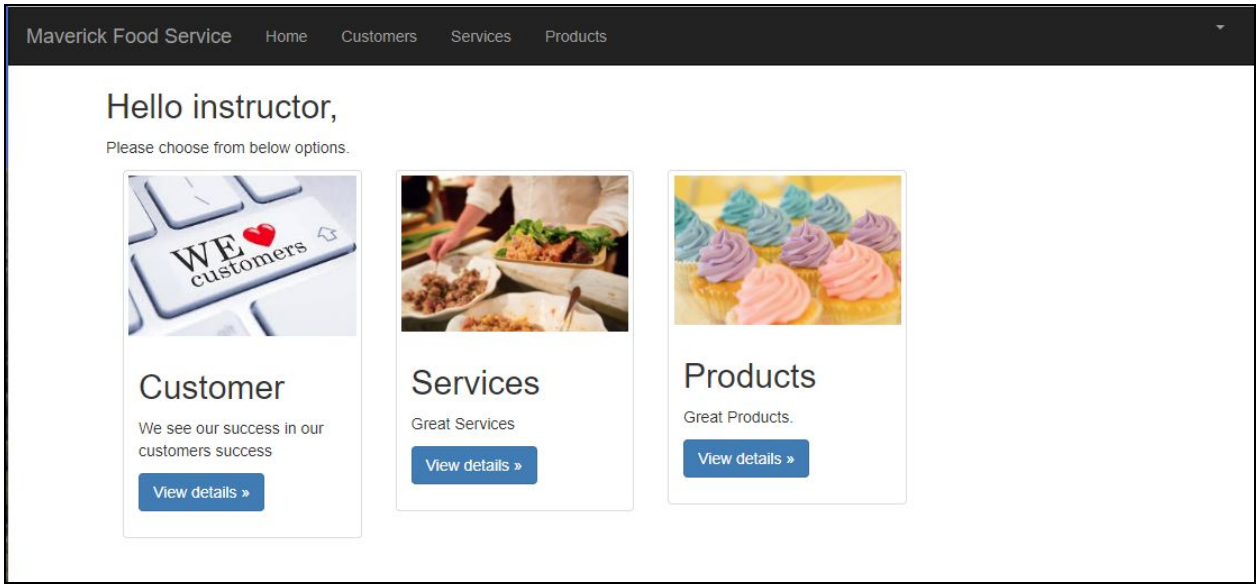


Next we have the page listing of the products purchased by the Maverick Food Service customers .

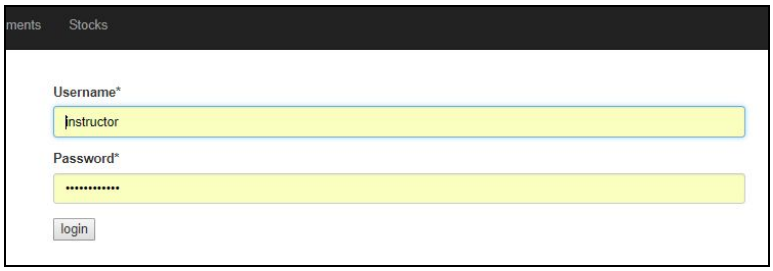


As with Services you can click on a Product and edit or delete the products.

Now let's take a look at screens That can be used by any employee of Maverick Food Service. In this scenario, we will assume that employees are provided their ID and PW by the administrator. Here we are back to the landing page for the employee:

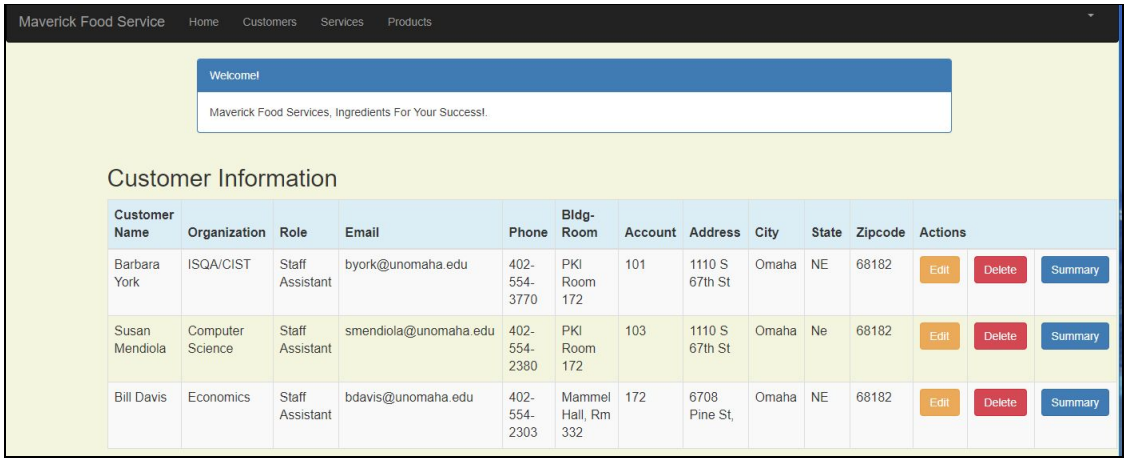


If they are not already logged in, they would log in by clicking the login button and receive this page:



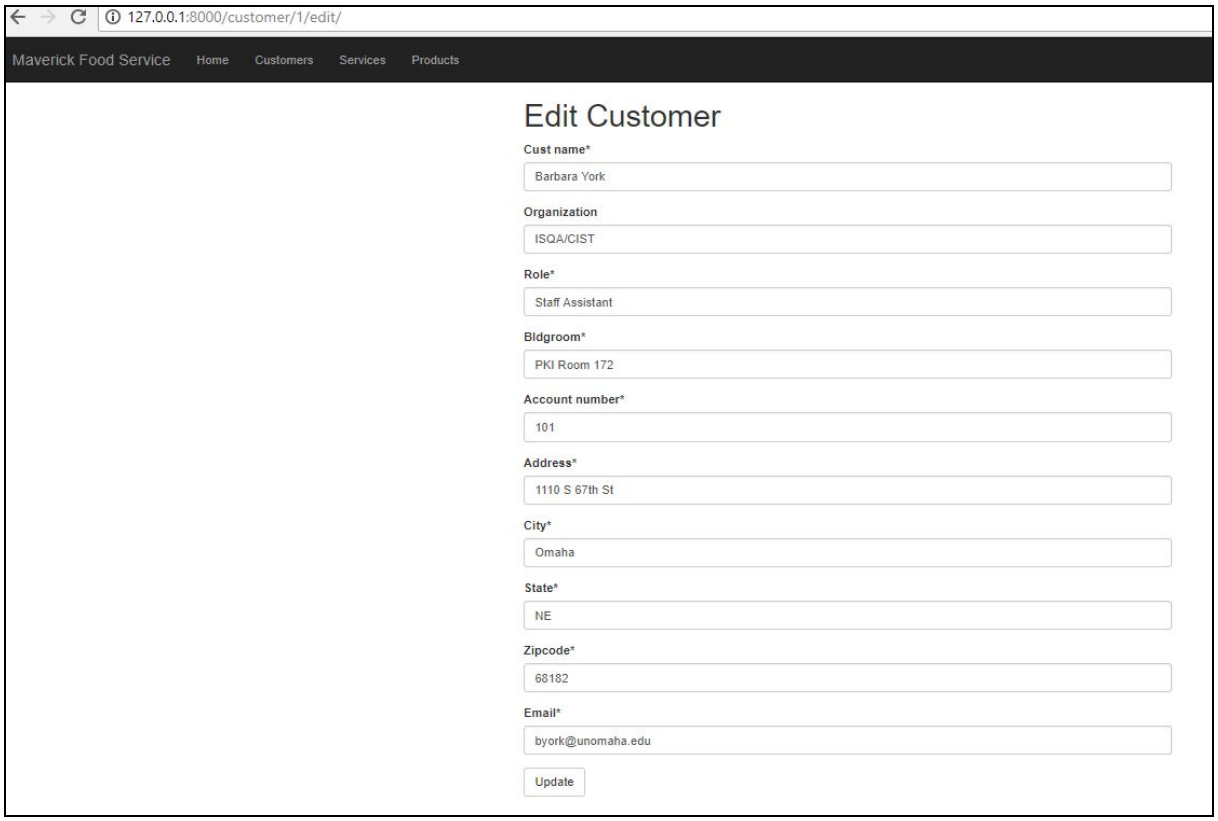
No other features such as “forget your password” in the assignment. This can again be something you can do to move your grade from B to A as discussed in the assignment. . Once they have signed in, an employee can see a list of customers and update customer information with a page similar to the one shown below.

Customer Screen



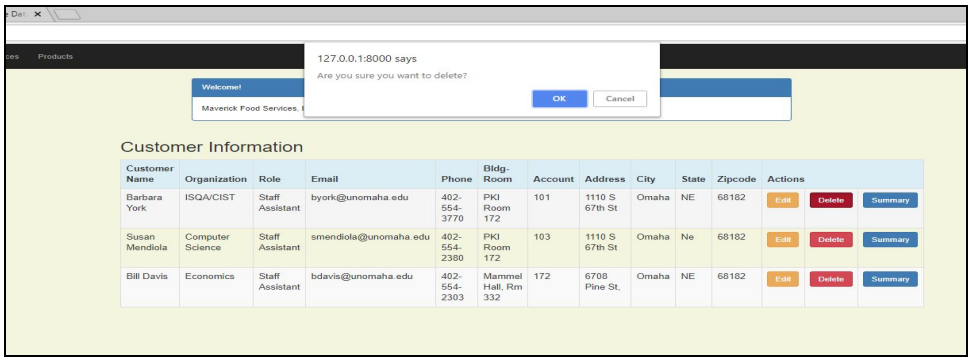
Employees can edit customer information with this screen.

Customer Edit



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

Customer delete



The summary function is actually a report of all services and products acquired by a customer. We will discuss this at the end of the tutorial.

Summary Report

Welcome!

Maverick Food services, Ingredients For Your Success!

Customer Summary

Customer Total

Total of Service Charges and Product Charges
1,180.00

Services Information

Customer	Service Category	Description	Location	SetUp Time	CleanUp Time	Service Charge
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
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

Total of Service Charge
1,100.00

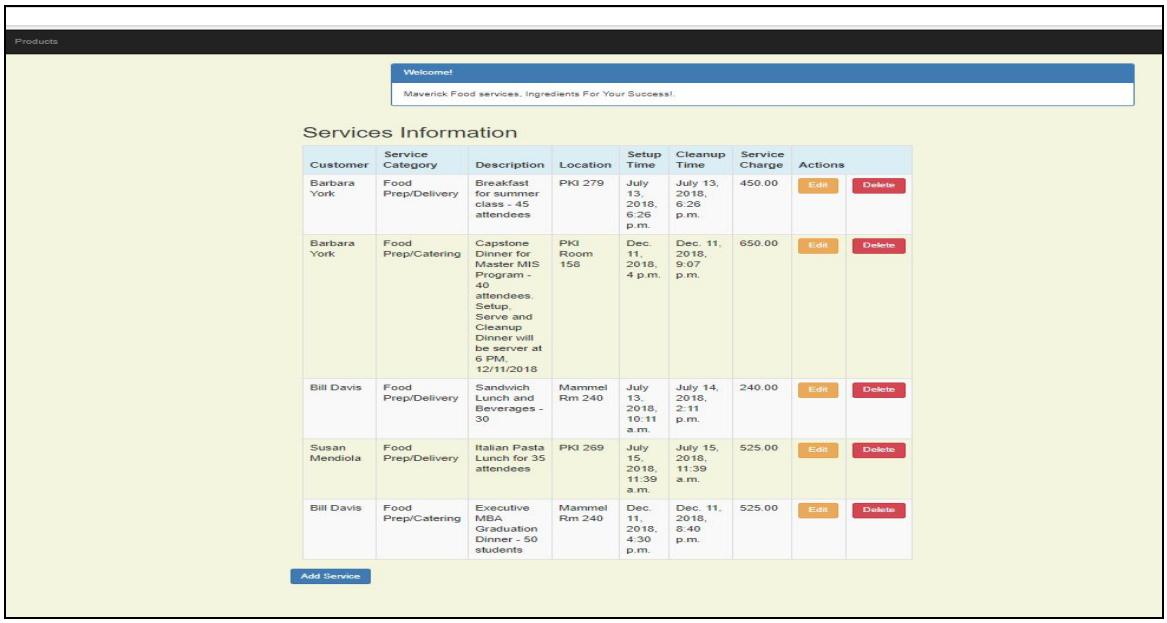
Product Information

Product	Description	Quantity	Pickup Time	Total Charge
Sheet Cake	Full Sheet Cake with wording: Congratulations Harvey Smith on Your Retirement!! Barbara will send a student assistant to pick up at Scott Cafe in PKI. Please have it wrapped for travel	1	July 5, 2018, 12:10 p.m.	80.00

Total of Product Charges
80.00

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

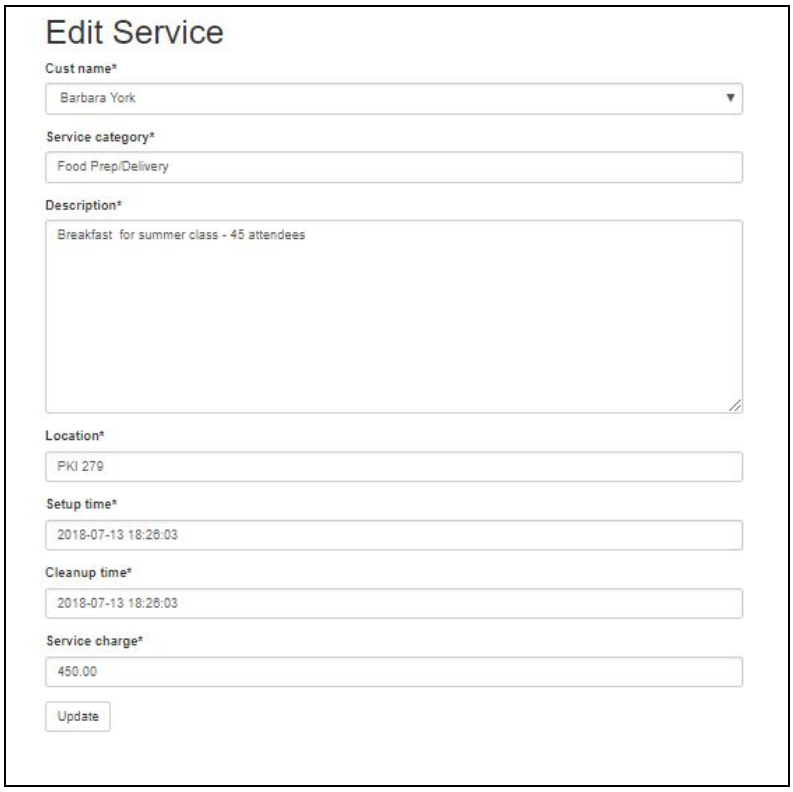
Service List



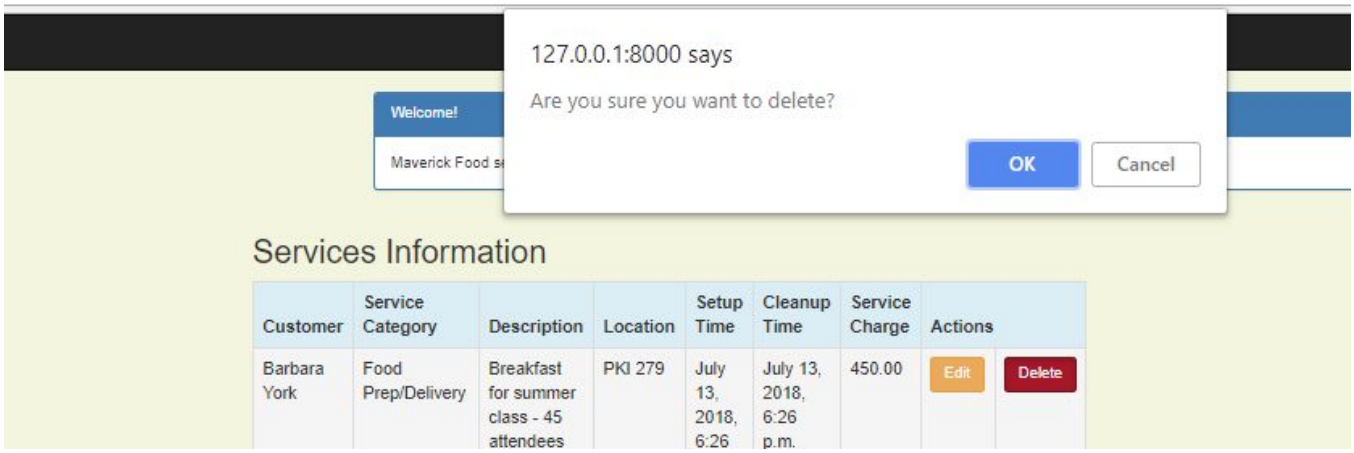
The service list screen allows employees of MFS (Maverick Food Service) add all services provided to customers including those working for the university and external parties that require food services for events they are hosting in a UNO building. This pages allows for adding a new service provided,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

Adding a new service looks like this:

In a similar way employees can also edit information in an existing record.

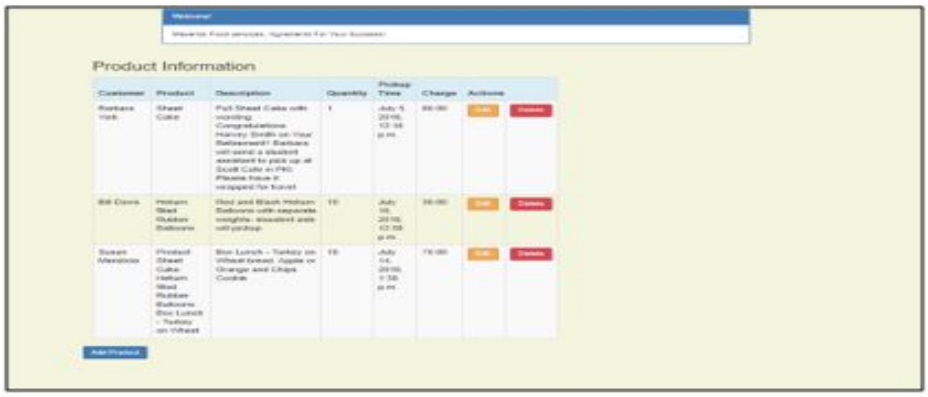


Deletes do have a warning as with the customer record but only delete that one record.



Below you will see that employees have the ability to add, update and delete products sold to customers.

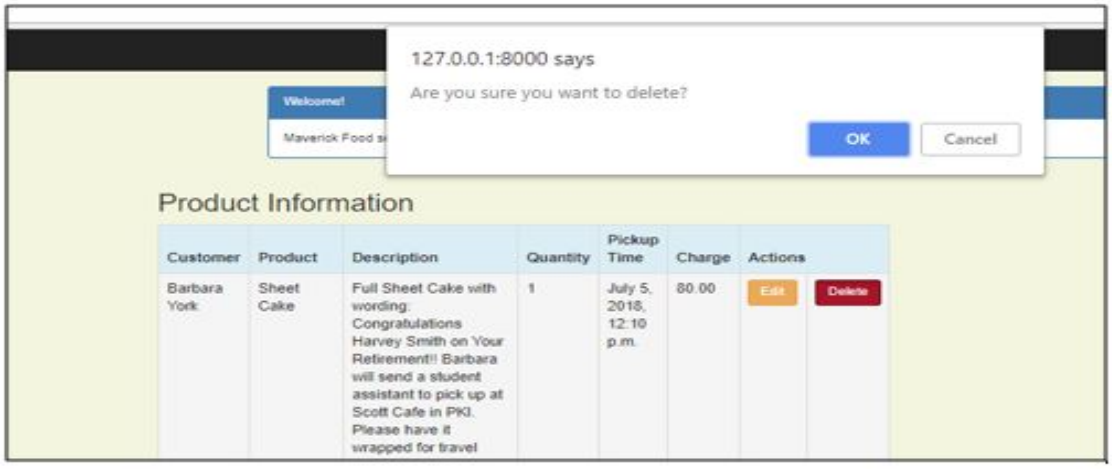
Product List



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 3.6.x and Django 2.0.5)
Step 1 - Just as we did in the previous tutorial, we need to create a new project directory that will host both the virtual environment files and the project code.

Let's start in your directory that hosts your django projects and create a directory called foodserv.
mkdir foodservice

Now inside this first foodserv directory we will create the files to create a virtual environment.

Run the following commands:
python -m venv c:\path\to\myenv
On MAC
python3 -m virtualenv myvenv

Next we must activate the virtual environment with

On Windows go to scripts directory and type **activate**

On Mac go to bin inside venv and issue the command **source activate**

In Windows this will look like this:

```
C:\apps\pythonsu2018\efs\myvenv\Scripts>activate

(myvenv) C:\apps\pythonsu2018\efs\myvenv\Scripts>cd..

(myvenv) C:\apps\pythonsu2018\efs\myvenv>
```

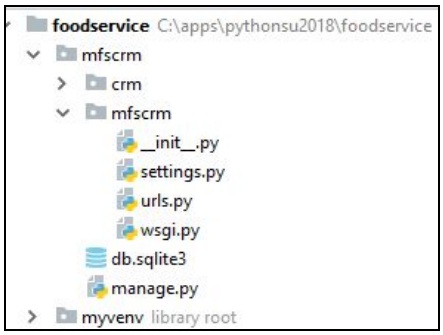
In a Mac it should look like this

```
Georges-MacBook-Pro:venv georgeroyce$ ls
bin             include         lib             pyvenv.cfg
Georges-MacBook-Pro:venv georgeroyce$ cd bin
Georges-MacBook-Pro:bin georgeroyce$ source activate
(venv) Georges-MacBook-Pro:bin georgeroyce$ cd..
```

As it mentions, any time you want to exit the virtual environment you can type deactivate.
Now that we have activated the virtual environment we will install django into the virtual environment using this command:
pip install django==2.0.5

Now it is time to create our project.
Run the following command.
django-admin startproject mfscrm

If you now open this project in your Pycharm editor, the directory structure will look like this:



To ensure that all is well so far with the application navigate the directory containing manage.py and run the following command:

python manage.py runserver

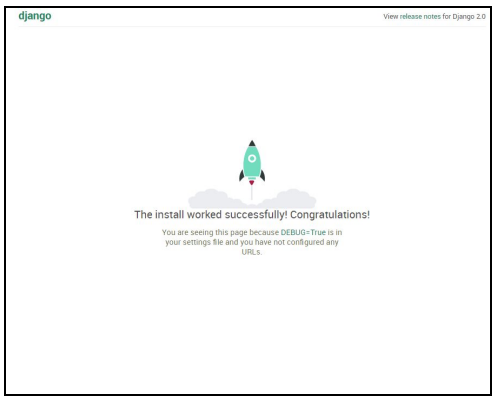
You will receive this on the command line terminal:

```
System check identified no issues (0 silenced).

You have 14 unapplied migration(s). Your project may not wor
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 28, 2018 - 15:44:33
Django version 2.0.5, using settings 'efs.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Don't worry about the 14 unapplied migrations. We will resolve this shortly.

In your browser go to http://127.0.0.1:8000/

You should see this:

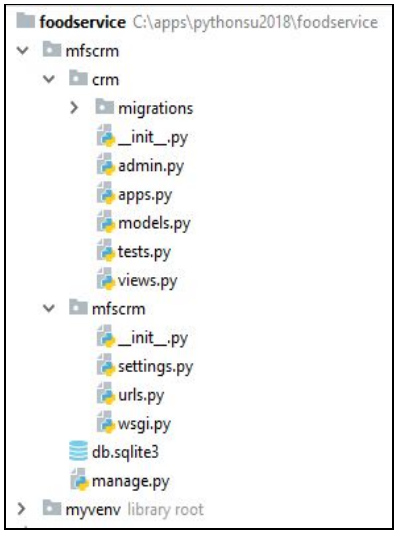


Section 3 - Creating the crm app

We are creating a customer relationship management application so we will call our app 'crm'. You should continue to be in the main project directory with manage.py application file. Now issue the following command:

python manage.py startapp crm

Now you should have the following app files:



Notice the difference in files Django created for us when looking at the project (mfscrm) versus app (crm). You can also see that we now have a db.sqlite3 database file as well.

Section 4 - Editing the settings.py file to connect the mfscrm project with the crm app

Now we need to let the mfscrm project know we have the 'crm app". To do that we edit the settings.py file in the mfscrm project directory.

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

1. We will leave DEBUG = True for now until we are ready to turn this into a production application.
2. Add the portfolio application under installed applications.
3. We can also change timezone from UTC to 'America/Chicago' for now.

Section 5 - Populating a Database with Base Django tables and a Customer Table

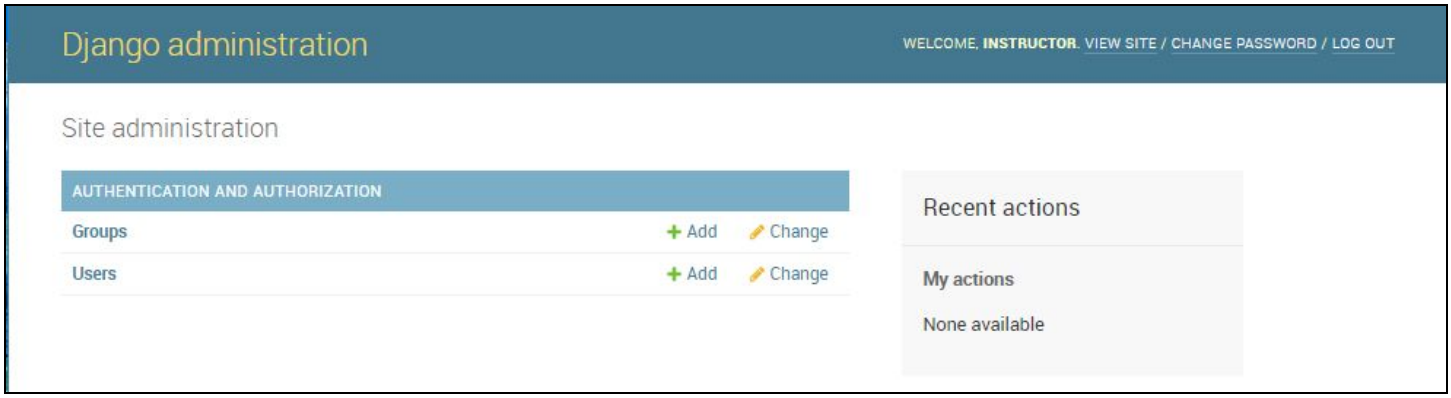
Step 2 - Next we will create the Django project tables by running our first migration of this project. Change directory in the command line tool to be in the same directory as manage.py and run the following command:
python manage.py migrate
This will create the default tables for any django project/application.

```
(myvenv) C:\pythonsu2018\efs\efs>python manage.py migrate
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 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 sessions.0001_initial... OK
```

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

Please use this for one of your superusers. It will facilitate grading of the assignment.
User = instructor
Email = your email address
Password = instructor1a

Now start the server again with the command: **python manage.py runserver.** Sign in with your new superuser credentials. You should see the admin panel like this come up when you add admin to our url like this: .



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 they simply start as an active user, not a superuser unless you change them to a superuser. This is also the place where you can set up groups of users with like permissions. More about this later.

Now it is time to create our own table of customers for the Maverick Food Services. This will allow us to easily manage contact information of the customers. We will need to add the following information about our customer table into the models.py file in the portfolio app.
Models.py

```
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)
```

To quickly sum up what we just added, the first line tells Django to also import its features: timezone and reverse. If you look inside the code, we have added *timezone.now()* and *return reverse ('customer_list')* to this model, so without importing the two utilities 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. The return reverse part grabs the page we will use to display Customer's data once we create its template and views. Customer_list is going to be the name of our template as well as the url that we'll use to display customer's information.
Now that we have a database for our customers, let's go ahead and add it to the *admin.py* file so that we can actually add our customer numbers, address, city, state, etc.

When you open up your *admin.py* file, you should see the following:

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

To continue on from here, you will need to add the following:

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)
```

Now we will create the first table beyond the standard Django Table for our by making and then running a migration. These 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 this data model. You do this by issuing the following command in the command line interface:

python manage.py makemigrations

The results should be:

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

Follow this with the command

python manage.py migrate

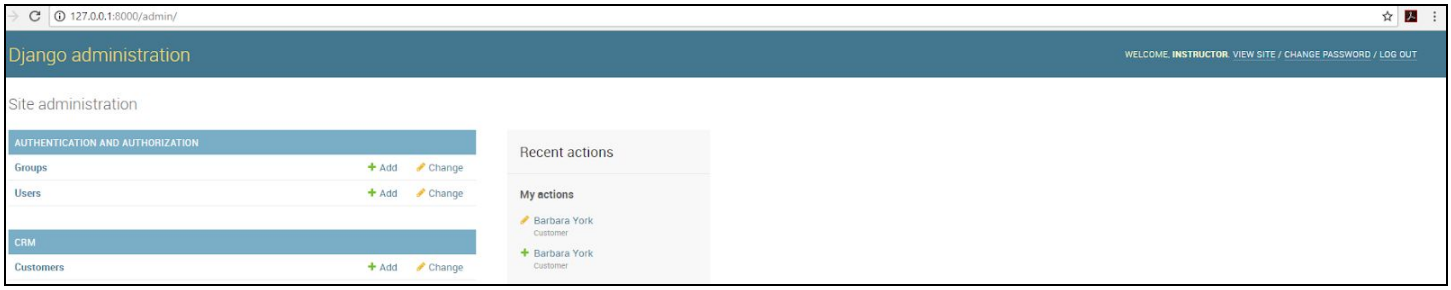
You should receive this message:

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

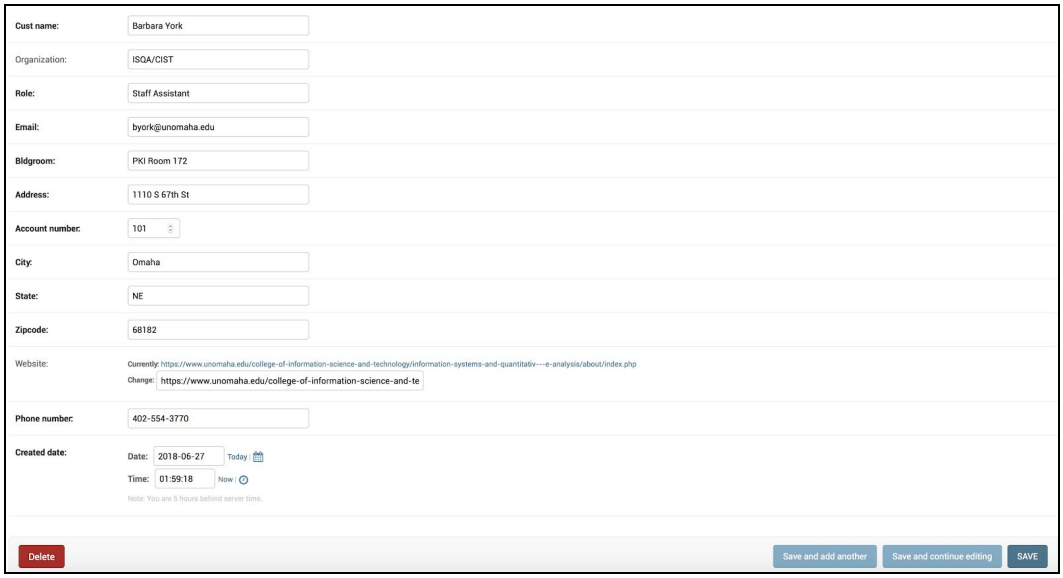
Now let's see what our table looks like in the admin panel. To do this startup the server with

python manage.py runserver

The new table will appear under the portfolio app listing



Let's add some customers to the application. Click here for MFSCRM Table data. Please add the 3 customers to your database. This and other information is used in grading.



If you have not already done so, check out the tables in the directory 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 in the project directory. Check out the system tables and the new customers table.

Section 5 - 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 page in the crm app called forms.py Add to the form the following:

forms.py

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

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

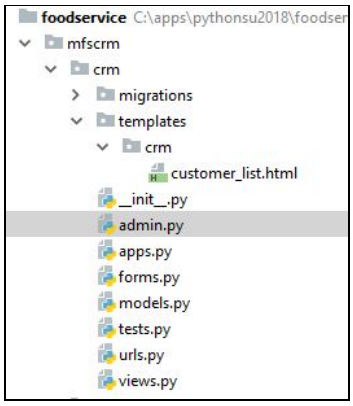
Since Django is a model-view-template framework we now must describe what we want to display in a template in the views.py file. Double click to open the views.py file in portfolio app, and enter the following:

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 is to indicate where Django needs to look for a template, to display our url for customer list. And that's it for the view!

As of right now, we do not have a template directory. Let's make one by right clicking on the crm directory in PyCharm, and selecting New, and then Directory. Name it templates. Inside of the templates directory create a directory 'crm'. Create an html file inside the templates directory and call it customer_list.html as shown below.



Add the following to the customer_list.html file:

customer_list.html

```
 {% 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 %}
```

Now we need to set up our url path for this web page. While we have an overall urls.py at the project level, we generally create one for the apps. In the crm folder, create a urls.py python file and add the following:

crm\urls.py

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

app_name = 'crm'
urlpatterns = [

    path('customer_list', views.customer_list, name='customer_list'),

]
```

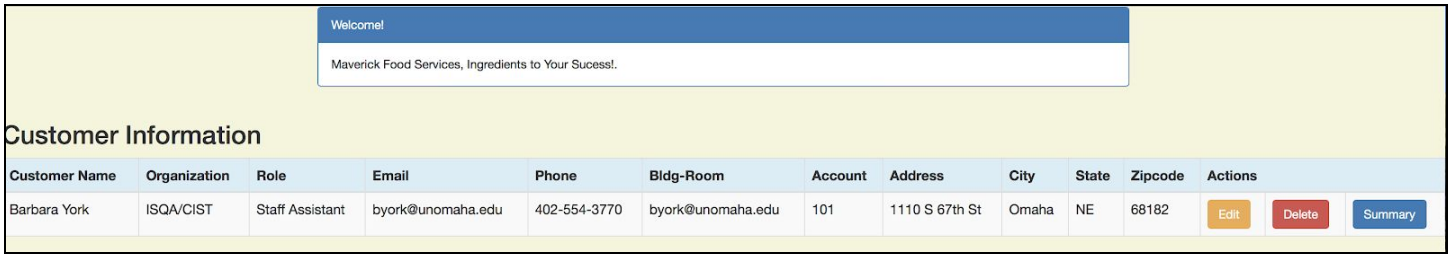
For this to work we need to tell the project urls.py file about our portfolio urls.py. We do this by updating the urls.py as shown below.

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

It should look like this:



Section 6 - 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 introduce the ability to edit a record in a template. There are a couple of ways to do this.

1. Default forms which work in all environments. You do not need to install anything.
2. Django-crispy forms which is a simple add in to django. I have had great success on Windows machine and in deployment environments like Heroku or PythonAnywhere but have encountered problems on my Mac development environment. Based on this, it is optional that you use this. (Also, if you have no problems on your Mac, let me know. To use Django Crispy forms run:
   pip install Django-crispy-forms

   You will also need to add this application to installed apps in settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'portfolio',
    'crispy_forms',
]
```

   At the very end of the settings.py file add the line below after static_url. Also add static_root if not already there.

```
STATIC_URL = '/static/'
CRISPY_TEMPLATE_PACK = 'bookstrap4'
```

2. Next, create a customer_edit.html file and place it in the template directory. By now you should already know that you need to right click on templates, 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. The html that goes inside of our customer_edit.html file is as simple as this:

customer_edit.html

```
{% 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 %}
```

With Django Crispy forms

customer_edit.html

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

3. Now that we have an html file for editing or updating customer information, let's configure our views. In *views.py* file in portfolio, go ahead and add the following code

views.py

```
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})

ALSO add at the top of views:
from django.shortcuts import render, get_object_or_404
```

4. Finally, we need to add to the urls.py.

crm/urls

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

app_name = 'crm'
urlpatterns = [

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

5. Need to update the customer list. Replace:

REPLACE:

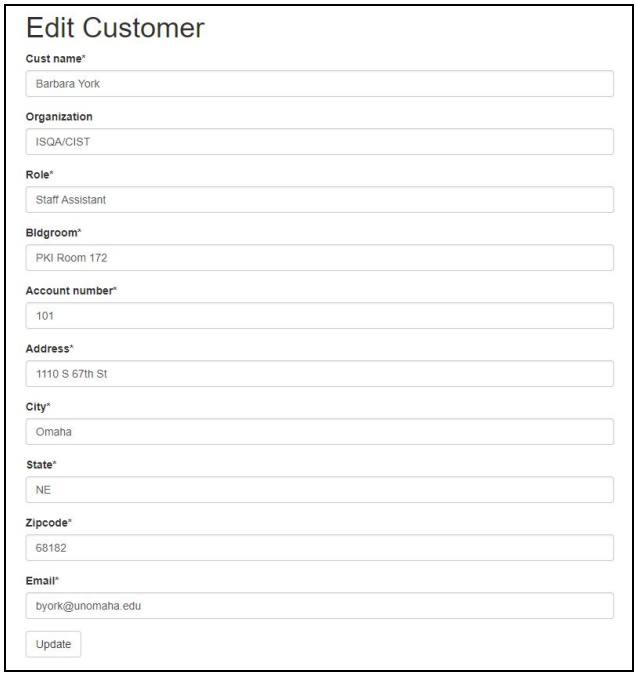
```
  <td><a href="" class="btn btn-warning">Edit</a>
```

WITH:

```
<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 customer listed, but when we're making changes to our customer table, we are only making the change for one customer 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 be able to edit a record. Select one and make an edit to the address. See if it updates back in the list.



Section 7 - Creating the ability to delete information in the list.

You will follow many of 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 templates.Inside of the file, add this code:

customer_delete.html

```
{% 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 %}
```

And change the 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 views.py in 2 ways. First, at the beginning of the file add or update these lines.

Views.py

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

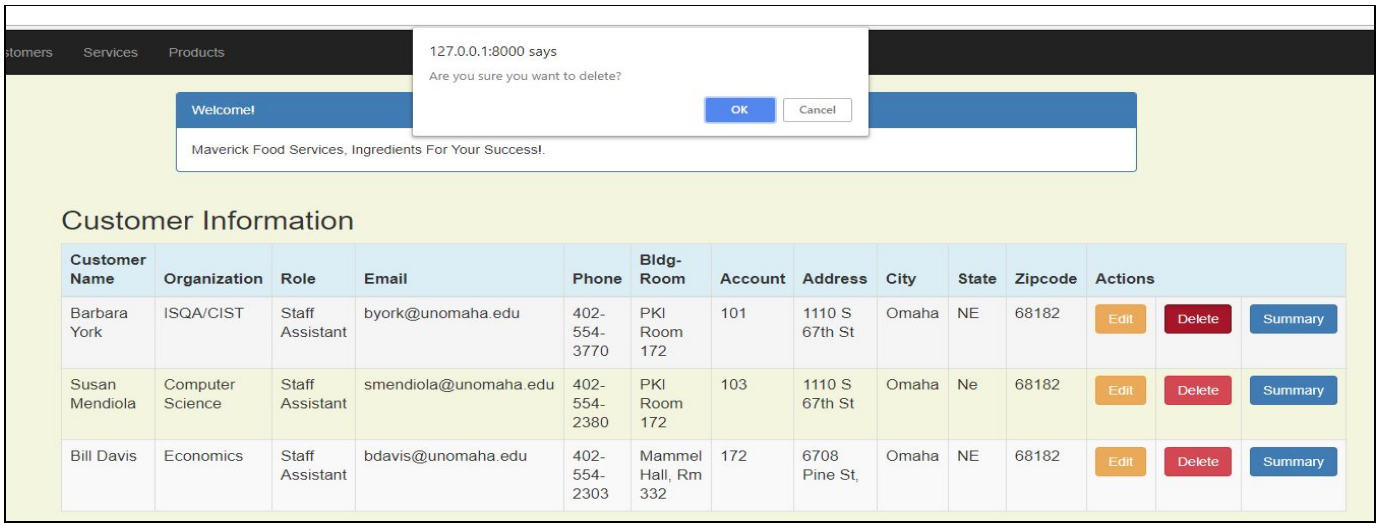
Also add this at the top of views.py:

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

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

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

When you attempt to delete a record you should receive this:



Section 8 - Adding Services and Product Data Models

We have now shown how to list customer information, edit the information and delete a record, all outside the admin panel which is really meant for administrators. 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 following information to your model so the new model looks like this:

```
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.acquired_date = timezone.now()
        self.save()

    def updated(self):
        self.recent_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.acquired_date = timezone.now()
        self.save()

    def updated(self):
        self.recent_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 much of a change. Even in the admin panel only the customer table shows up. What do we need to do?

I am hoping you said, we need to create and run migrations.
python manage.py makemigrations

Followed by:
python manage.py migrate

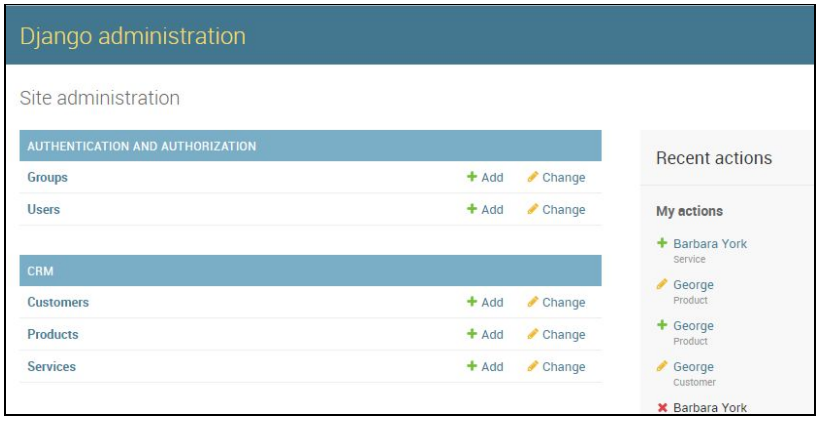
To see the new tables in the admin panel you will need to add the following to the admin.py.
admin.py

```
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. If you download the Db Browser for SQLite from: https://sqlitebrowser.org/ you will see the tables below. Each table created in Django comes with an auto-incrementing 'id' out of the box. You can see this in the illustration I captured using DB Browser for SQLite. You will see the ID field in customer, stock and investment table. It is a NOT NULL, AUTO-INCREMENTING, INTEGER, PRIMARY KEY field for all tables. There is a customer_id field in both the stock and investment tables which is a foreign key referencing the ID field in the customer table.

auth_user_user_permissions		CREATE TABLE "auth_user_user_permissic
crm_customer		CREATE TABLE "crm_customer" ("id" inte
id	integer	`id` integer NOT NULL PRIMARY KEY AUT
cust_name	varchar (50)	`cust_name` varchar (50) NOT NULL
organization	varchar (100)	`organization` varchar (100) NOT NULL
role	varchar (100)	`role` varchar (100) NOT NULL
email	varchar (100)	`email` varchar (100) NOT NULL
bldgroom	varchar (100)	`bldgroom` varchar (100) NOT NULL
address	varchar (200)	`address` varchar (200) NOT NULL
account_number	integer	`account_number` integer NOT NULL
city	varchar (50)	`city` varchar (50) NOT NULL
state	varchar (50)	`state` varchar (50) NOT NULL
zipcode	varchar (10)	`zipcode` varchar (10) NOT NULL
phone_number	varchar (50)	`phone_number` varchar (50) NOT NULL
created_date	datetime	`created_date` datetime NOT NULL
updated_date	datetime	`updated_date` datetime NOT NULL
crm_product		CREATE TABLE "crm_product" ("id" integ
id	integer	`id` integer NOT NULL PRIMARY KEY AUT
product	varchar (100)	`product` varchar (100) NOT NULL
p_description	text	`p_description` text NOT NULL
quantity	integer	`quantity` integer NOT NULL
pickup_time	datetime	`pickup_time` datetime NOT NULL
charge	decimal	`charge` decimal NOT NULL
created_date	datetime	`created_date` datetime NOT NULL
updated_date	datetime	`updated_date` datetime NOT NULL
cust_name_id	integer	`cust_name_id` integer NOT NULL
crm_service		CREATE TABLE "crm_service" ("id" intege
id	integer	`id` integer NOT NULL PRIMARY KEY AUT
service_category	varchar (100)	`service_category` varchar (100) NOT NU
description	text	`description` text NOT NULL
location	varchar (200)	`location` varchar (200) NOT NULL
setup_time	datetime	`setup_time` datetime NOT NULL
cleanup_time	datetime	`cleanup_time` datetime NOT NULL
service_charge	decimal	`service_charge` decimal NOT NULL
created_date	datetime	`created_date` datetime NOT NULL
updated_date	datetime	`updated_date` datetime NOT NULL
cust_name_id	integer	`cust_name_id` integer NOT NULL
django_admin_log		CREATE TABLE "django_admin_log" ("id"

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

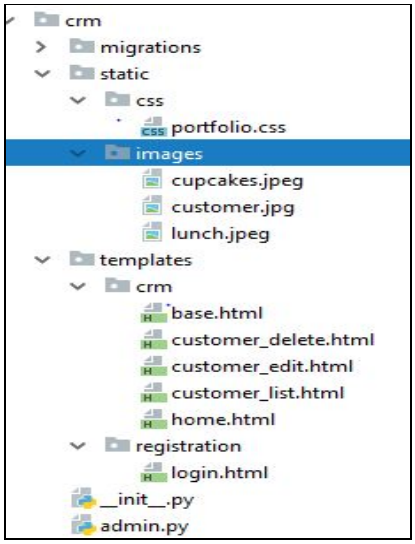


Go ahead and add the same data now to Services and Investment table. Click here for CRM Table data. For starters we just need to have 1 service and 1 product for each of the 3 customers.

Section 9 - Adding a Home Page and Base Template

I am hopeful you will begin to see a pattern as you begin to build the CRUD (create, read, update and delete) pages for services and products. I will show you how to add a new service and list services. You will then need to add the remaining delete and edit a service and then continue to add the product add, edit and delete on your own. Before we do this we will need to create a navigation page and add some simple security to these pages outside of the admin site.

We will begin by downloading static files (pictures, CSS. etc…) for the application. Click here to download the Static Files. Unzip the files and review the code. Here is what your directory structure should look like when you have placed all the files in the appropriate places:



The login.html that you placed in the registration directory is shown below:

login.html

```
{% extends "/base.html" %}

{% block content %}
   {% if form.errors %}
   <br />
   <p>Your username and password didn't match. Please try again.</p>
   {% endif %}
   <br />
   <br />
   <form method="post" action="{% url 'login' %}">
      {{ form.as_p }}
      {% csrf_token %}


      <input type="submit" value="login"/>
      <input type="hidden" name="next" value="{{ next }}"/>
   </form>
{% endblock %}
```

This is simply a login and logout page capability for a person that is not an administrator. You can use exercises in your Django text to add to this security capability to move your grade from a B to and A.

The next files we should review are the base.html file and the home.html files. These files will give our site a home page. Please all the these files in the correct directory as shown above. They will now reference the jpg's that are in the static files directory so be sure they are also present. Here is the base.html which provides a common look and file for all files that we will use.

base.html

```
{% load staticfiles %}

<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>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 class="col-md-3">
                    <div class="thumbnail">
                      <img src="{% static "images/lunch.jpeg" %}" alt="lunch"/>
                      <div class="caption">
                        <h2>Services</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>
              </div>
            {% endblock %}
          </div>
        </div>
</body>
</html>
```

You can see now that this page references the “summary:customer_list” where you might expect it to say service or product. This is because we have not completed these pages and the base would generate an error at this point. I will remind you to go back and change this when you add the additional pages. The home page is listed below and it also is references the “crm:customer_list” when it should be referencing service or product. Again, we will fix this shortly.

home.html

```
{% extends 'crm/base.html' %}
{% load staticfiles %}
{% block content %}
        {% 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>
        </div>
    </div>
```

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

Next we need to add a number of items to the views.py. There is the updated views.py:

```
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('portfolio:customer_list')
```

Next we need to update the urls files. Add the following to the crm/urls file
crm/urls

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

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

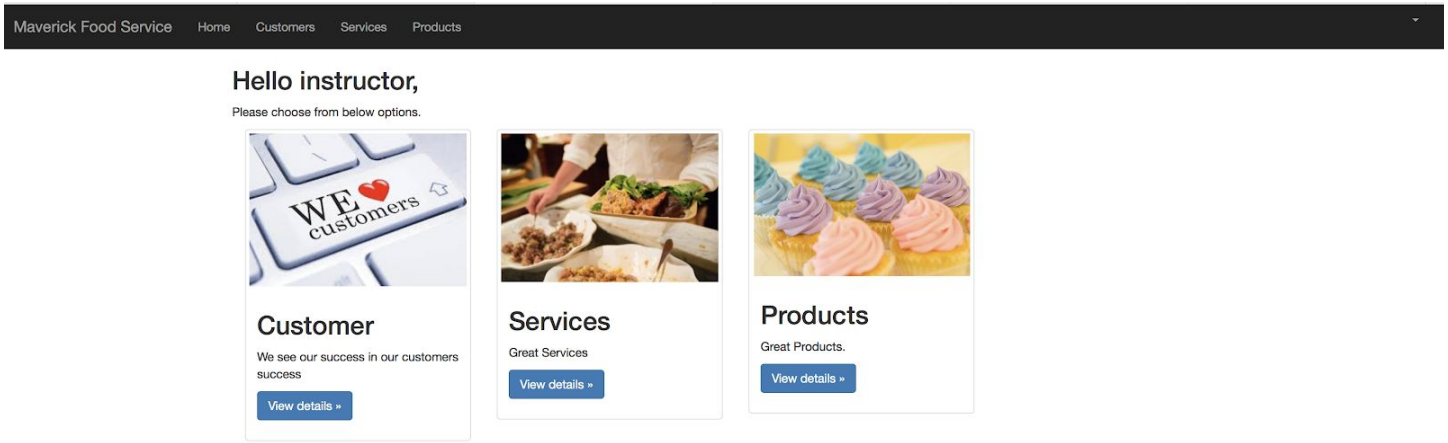
And add the following lines to the mfscrm\urls.py
mfscrm\urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('crm.urls')),
   url(r'^accounts/login/$', views.login, name='login'),
   url(r'^accounts/logout/$', views.logout, name='logout', kwargs={'next_page': '/'}),
]
```

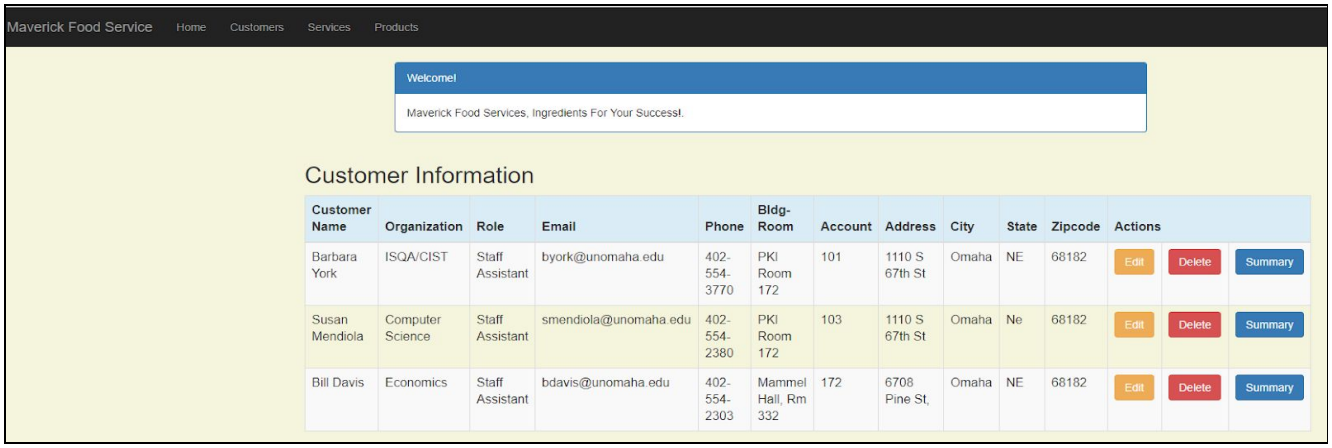
Also add this line in the settings.py

```
LOGIN_REDIRECT_URL = '/home'
```

Now it is time to check out what this looks like. Issue this command again: **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. I have selected a sample style and jpeg's but you are welcome to improve this this.

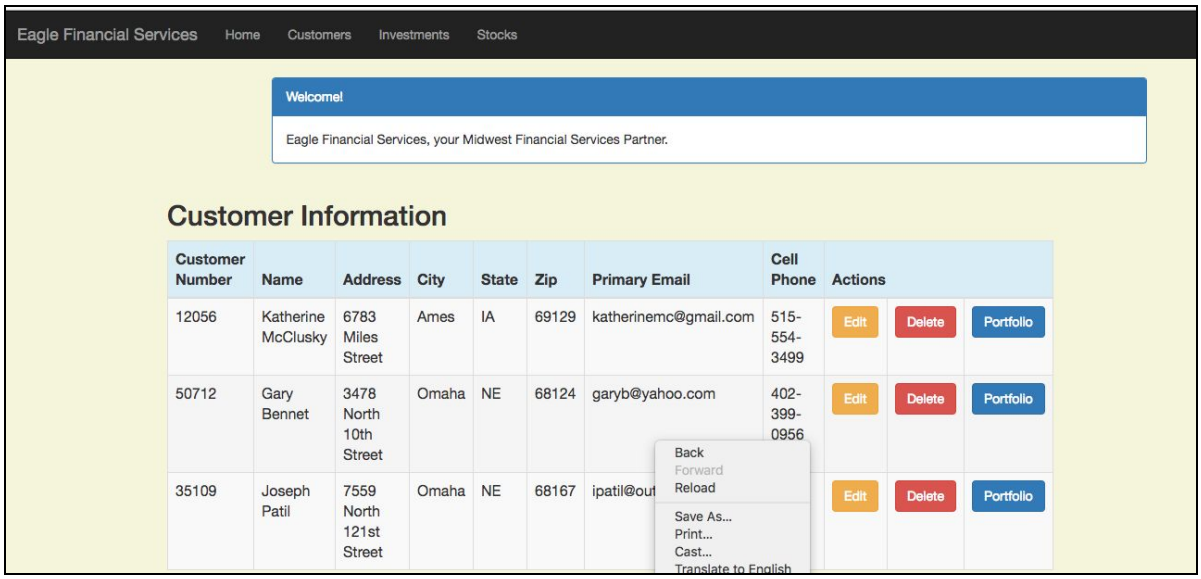


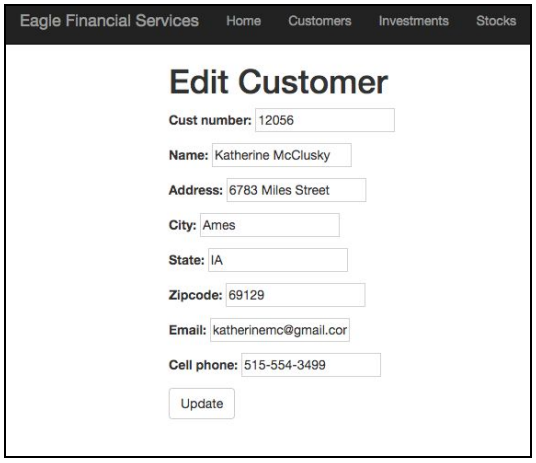
Now you can see the navigation provided by base.html. Right now when you click on services and products you will find that I have redirected the link to avoid an error to customers. When we add the code to work with services and products, we will replace the links. When you click on View details for customers you should see something like this:



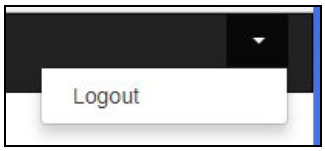
We need to add a line in the file to bring the navigation to this file. What would you add to do this? Hopefully you recognize that the following line should be place in each of the pages we have developed

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





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.



Now log back in. You should arrive back at the home page. If you attempt to access the data without logging in your 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 to move your grade from a B to A. Your textbook gives great examples for these features.

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

If you examine the fields for the services entries and how we created views entries, templates and url entries, you should have a good idea of how to add the services and product features. I will help you out with the services list and adding a new service. Your job will be to following 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 templates directory. Add the following code.

service_list.html

```
{% extends 'crm/base.html' %}
{% block content %}
    <html>
    <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 class="row">
          <a href=""><span
                class="btn btn-primary">Add Service</span></a>
        </div>
    </div>
    </body>
    </html>
{% endblock %}
```

Next we need to the following to the views.py

Add to **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})
```

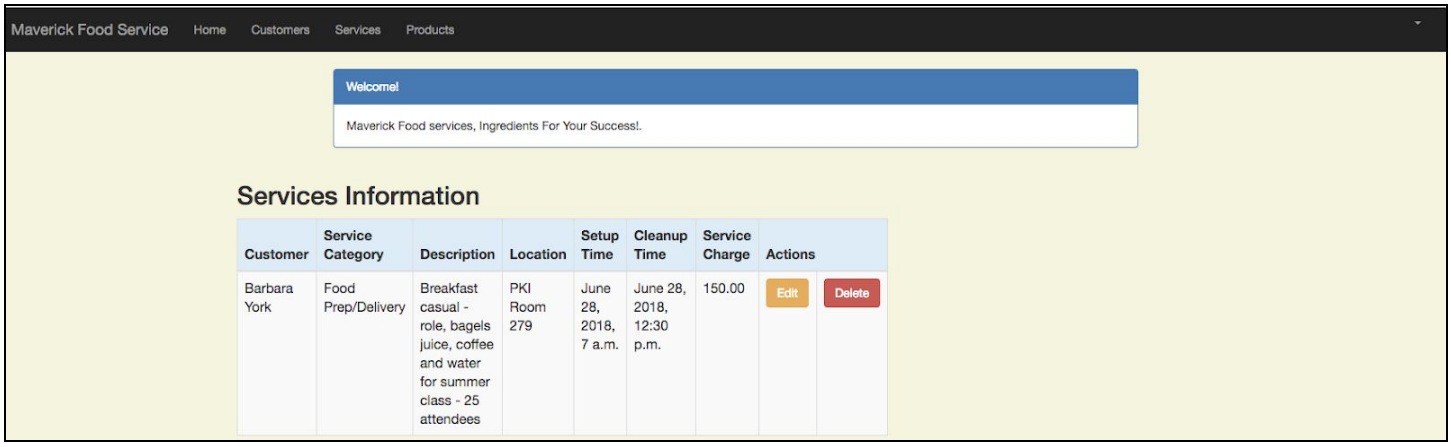
We also need to add the following line to the crm**/urls.py**.

```
path('service_list', views.service_list, name='service_list'),
```

Also, you will now need to go back to the home.html and base.html and change the **customer.list.html** to the **service.list.html** You will need to do this a total of 3 times between the two pages. Now it it time to test out our changes. Again start of web app server with the command:

Python manage.py runserver

You should see:



Of course the buttons do not function now but we can fix these just as we did with the customer page.

Next we will activate the Add Service Button.
Step 1 - To do this we will need the following template page:
service_new.html

```
{% extends 'crm/base.html' %}
{% load crispy_forms_tags %}
{% block content %}
   <h1>Add a New Service</h1>
   <form method="POST" class="service-form">{% csrf_token %}
        {{ form|crispy }}
       <button type="submit" class="save btn btn-default">Save</button>
   </form>
{% endblock %}
```

Step 2 - Since we need a form to add or edit records we need to add the following to the forms.py page.
Add to **forms.py**

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


Also add Service in the models you will import
from .models import Customer, Service
```

Step 3 - We need to add the following to the views.py page:

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})
```

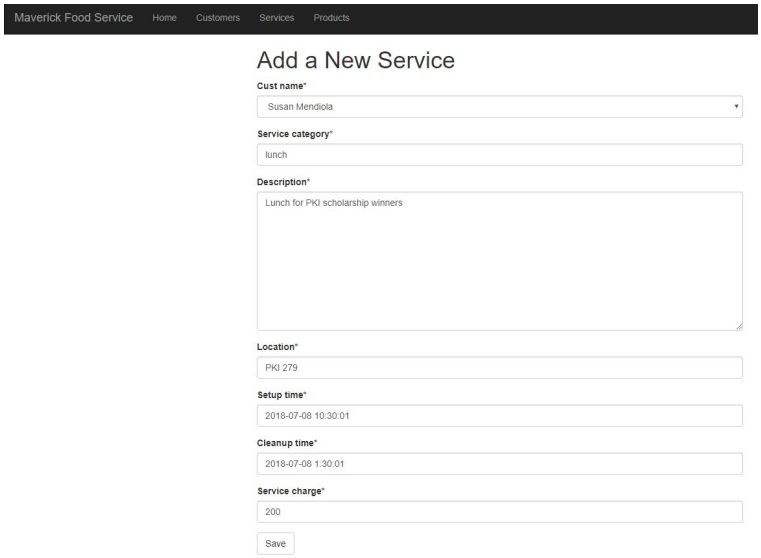
Step 4 - You guessed it, we need to add the following line to the portfolio/urls.py file

```
path('service/create/', views.service_new, name='service_new'),
```

Step 5 - So we did not have an error show up, I removed the URL for the Add stock button. Add the url as shown here to the existing div class in the **service_list.html**. :
Update the **service_list.html**

```
<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. When you press this Add Service button you should see:



The next logical form to do is the Edit a Service.
Step 1 - To do this we will need the following template page:

```
{% extends 'crm/base.html' %}
{% load crispy_forms_tags %}
{% 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 %}
```

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

Step 3 - Step 3 - We need to add the following to the views.py page:

```
@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})
```

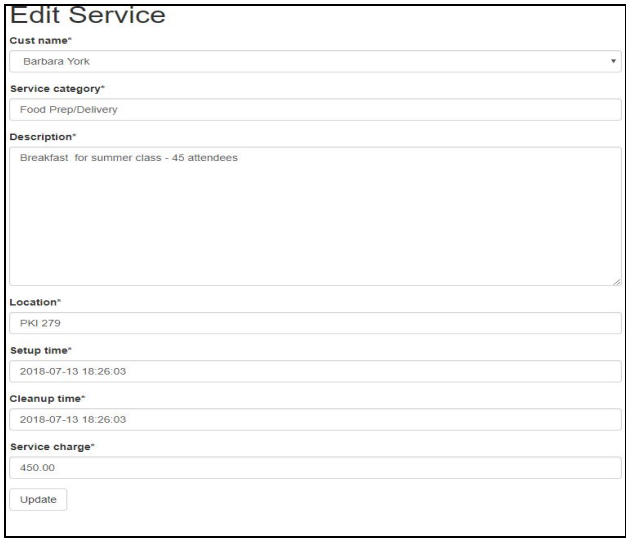
Step 4 - Next, we need to add the following line to the crm/urls.py file

```
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 service_list.html by using a blank URL. Now we must fill it in as shown below.

```
<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. When you press this Edit Service button you should see:



Good work!! You have learned how to create the essential elements of a typical CRUD Django application.

11. Now it is your turn to Complete Service and Products Pages

Now it is your turn to 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
6. A food service manager should be able to create a summary report of all services and products sold to a customer.

12. Adding the Summary Page to Our Project

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

Customer Information

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-	PKI Room	101	1110 S 67th St	Omaha	NE	68182	Edit	Delete	Summary

Once you click the summary button shown above we want the following report to be generated:

Welcome!

Maverick Food services, Ingredients For Your Success!

Customer Summary

Customer Total

Total of Service Charges and Product Charges
1,180.00

Services Information

Customer	Service Category	Description	Location	SetUp Time	CleanUp Time	Service Charge
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
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

Total of Service Charge
1,100.00

Product Information

Product	Description	Quantity	Pickup Time	Total Charge
Sheet Cake	Full Sheet Cake with wording: Congratulations Harvey Smith on Your Retirement!! Barbara will send a student assistant to pick up at Scott Cafe in PKI. Please have it wrapped for travel	1	July 5, 2018, 12:10 p.m.	80.00

Total of Product Charges
80.00

Step 1 - So start, we now need to install some additional packages. Be sure you have your virtual machine running. Once you have confirmed this then you will add the :Django Mathfilters. This component allows use to display and do simple math in a template. To install this issue the following command: **pip install django-mathfilters.** You should see something like this:

```
(myvenv) C:\apps\pythonsu2018\efsP2\efs>pip install django-mathfilters
Collecting django-mathfilters
  Using cached https://files.pythonhosted.org/packages/c6/fb/fa2fe3d531dc018d486b03c91461063e1
f/django_mathfilters-0.4.0-py2.py3-none-any.whl
Installing collected packages: django-mathfilters
Successfully installed django-mathfilters-0.4.0
```

Next add **'mathfilters',** to settings.py as shown below:

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

Notice You also need to add the **django.contrib.humanize**.but you do not need to pip install it, 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 do a good job of formatting numbers with placement of commas etc.

Step 2. - We will now use the two calculated values in the views and a summary.html template. In the **views.py** add the following:

from django.db.models import Sum

In your current **views.py** add the following to portray 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'))
    return render(request, 'crm/summary.html', {'customers': customers,
                                                 'products': products,
                                                 'services': services,
                                                 'sum_service_charge': sum_service_charge,
                                                 'sum_product_charge': sum_product_charge,})
```

This function makes all elements of the customer, service and product objects available. These are then summarized using the Django **aggregation function** called **Sum**. Now we are ready to develop the summary.html template.

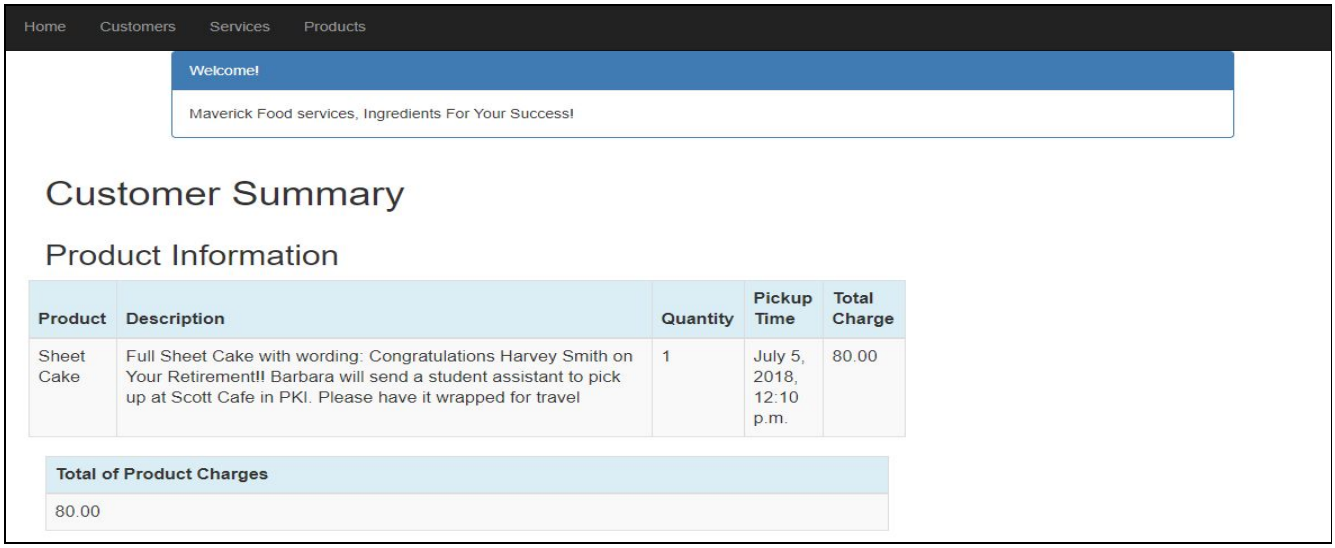
Step 4 - .The summary page is an HTML template that I have started for you below. You will create remaining parts of the page and place it in the templates directory. Below is the starter summary.html.

```
{% 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</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 %}
```

First you should note this is only a partial version of the customer summary



.

Using this starter, shell you need to develop the complete summary as 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

Step 5 - The summary.html will not be visible until we add the following line to the crm\urls.py

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

Step 6 - The summary.html also needs a button to activate it. Go back to the customer_list.html and add the following URL to the Summary button object:

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

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

Welcome!

Maverick Food services, Ingredients For Your Success!

Customer Summary

Customer Total

Total of Service Charges and Product Charges
1,180.00

Services Information

Customer	Service Category	Description	Location	SetUp Time	CleanUp Time	Service Charge
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
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

Total of Service Charge
1,100.00

Product Information

Product	Description	Quantity	Pickup Time	Total Charge
Sheet Cake	Full Sheet Cake with wording: Congratulations Harvey Smith on Your Retirement!! Barbara will send a student assistant to pick up at Scott Cafe in PKI. Please have it wrapped for travel	1	July 5, 2018, 12:10 p.m.	80.00

Total of Product Charges
80.00

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

1. Deploy your project to Github. Provide the link in the word document of the assignment.

2. Go to the **Canvas Module called General Resources Used Throughout Class** and select **Deploy to Heroku** or **Deploy to Pythonanywhere**. Deploy your application to one of these hosting facilities and

be sure to test it and make sure it has all the functions of the local application you developed. Add the sample records in each table to demonstrate you have successfully deployed and tested the application.

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