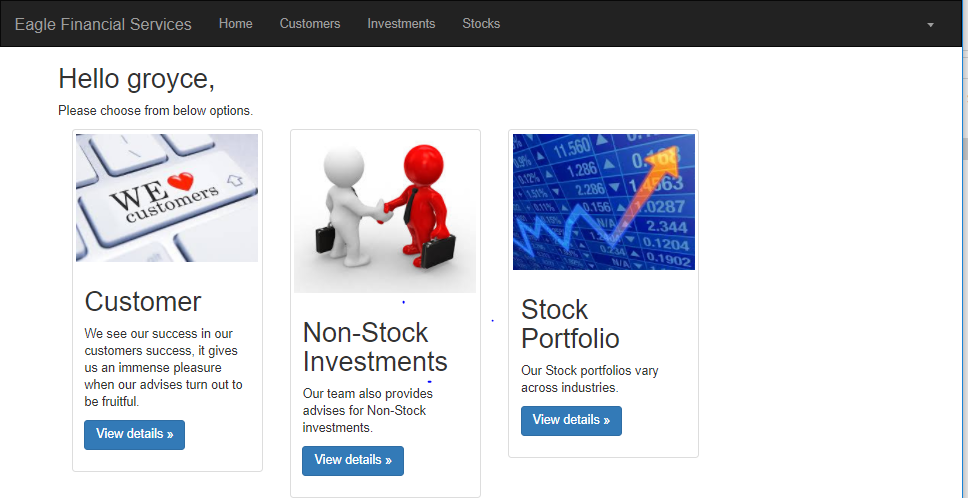
**Eagle Financial Services Tutorial (Assign 2P1)**

**Using Python Django**

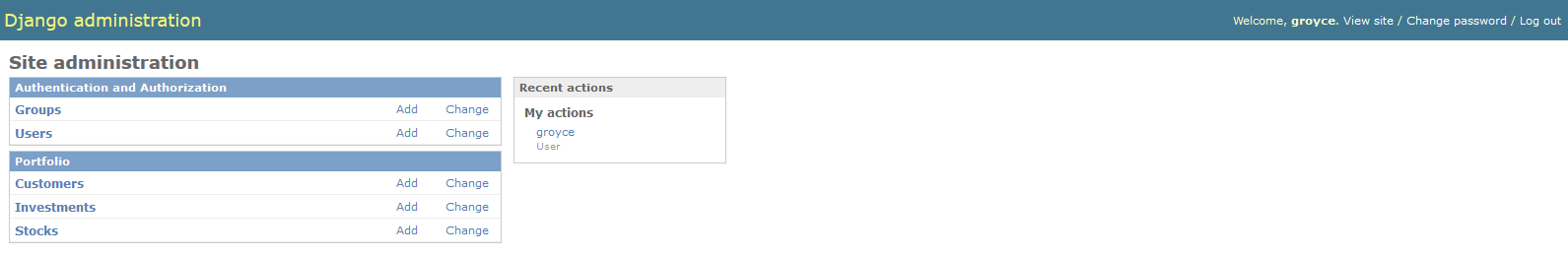
In this tutorial you will be developing a multi-table application using Python Django. Unlike the previous tutorial, this tutorial will give you a good start on the application but you will need to create several of the pages on your own. I use Eagle Financial Services as a name for a fictitious organization which provides financial services and advice to their customers on investing and growing their financial portfolio. Imagine if you were a Financial Advisor and needed to track the stock and other investments of your customers. The application is built to help the financial advisor do just this. From this information, the advisor can suggest changes to the portfolio of clients when they meet in person or over the phone. 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. You are welcome to add your own style to the application (Which may help you go from a grade of B to an A).

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

Sample Landing Page for the Eagle Financial Services Advisors

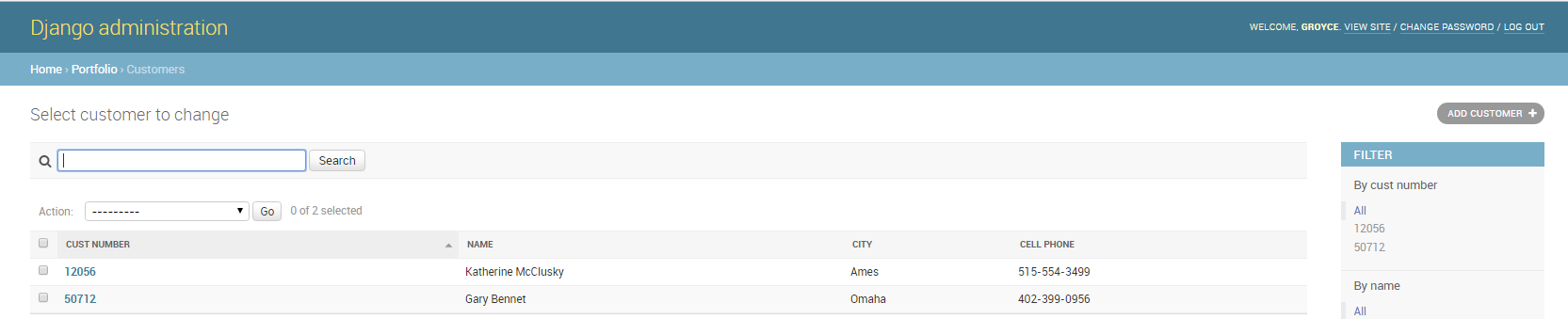


There are two roles in this sample application. The administrators have access to the the information for all customers and advisors. The administrators have access to the Admin section of the Django. They are capable of setting up new advisors and customer accounts. The advisor on the other hand uses EFS main 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 on the url to access.



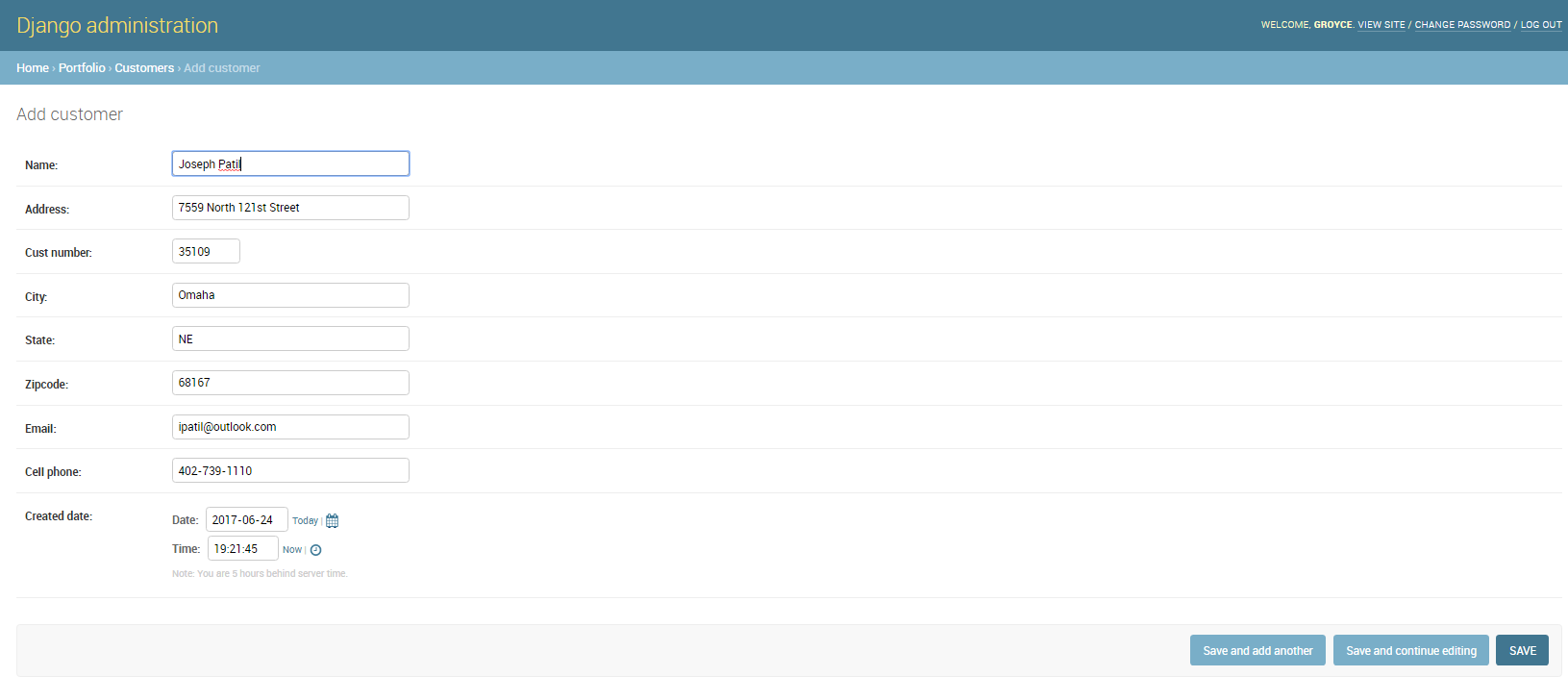
You are familiar with the Groups and Users in the admin panel from first Blog exercise. In our application, a given customer can own many stocks and investments.

Here is a view of the customers and the ability to add update and delete customers:

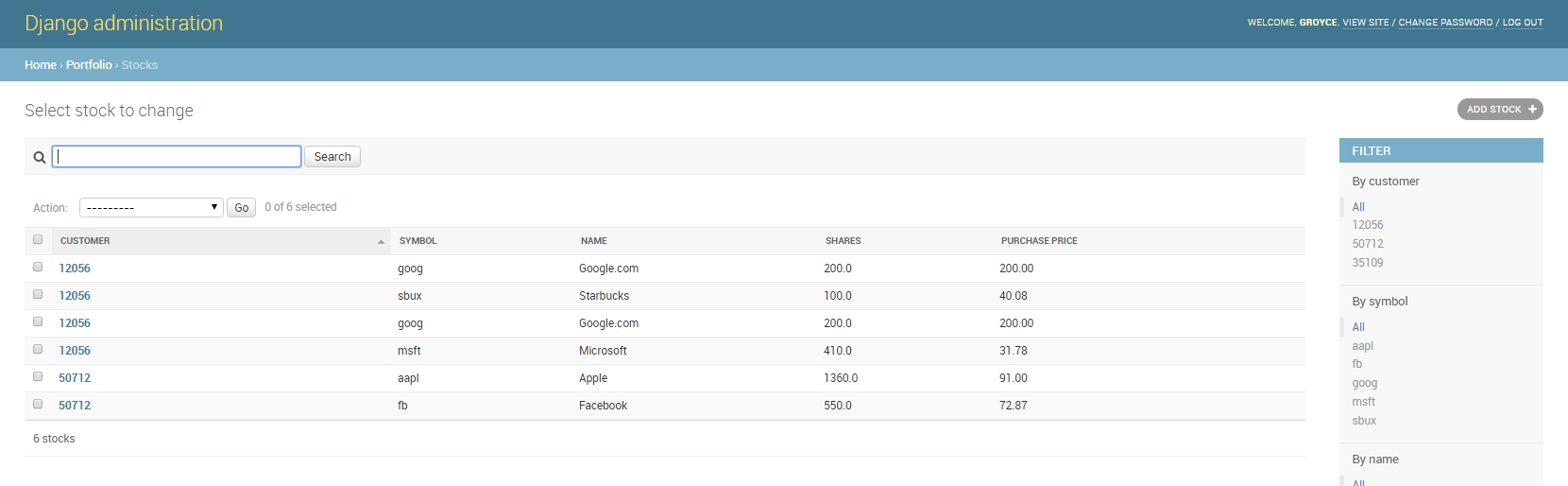


Here is an edit and update of one of the records:

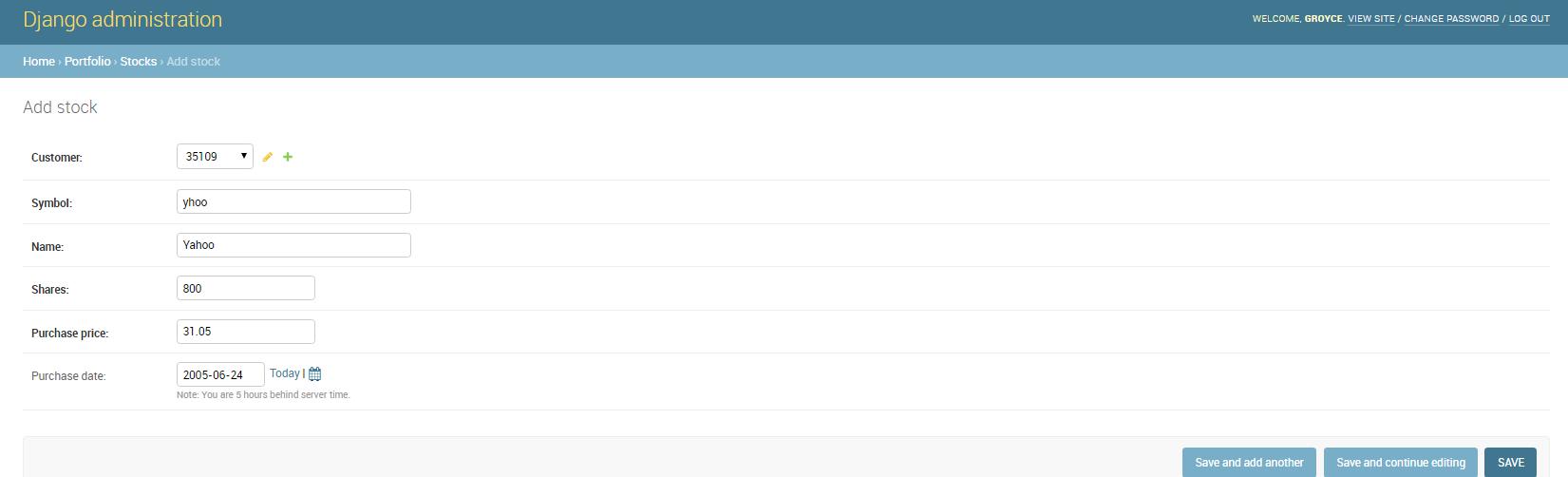
Note the history. This points out the logging capabilities provided by Django.



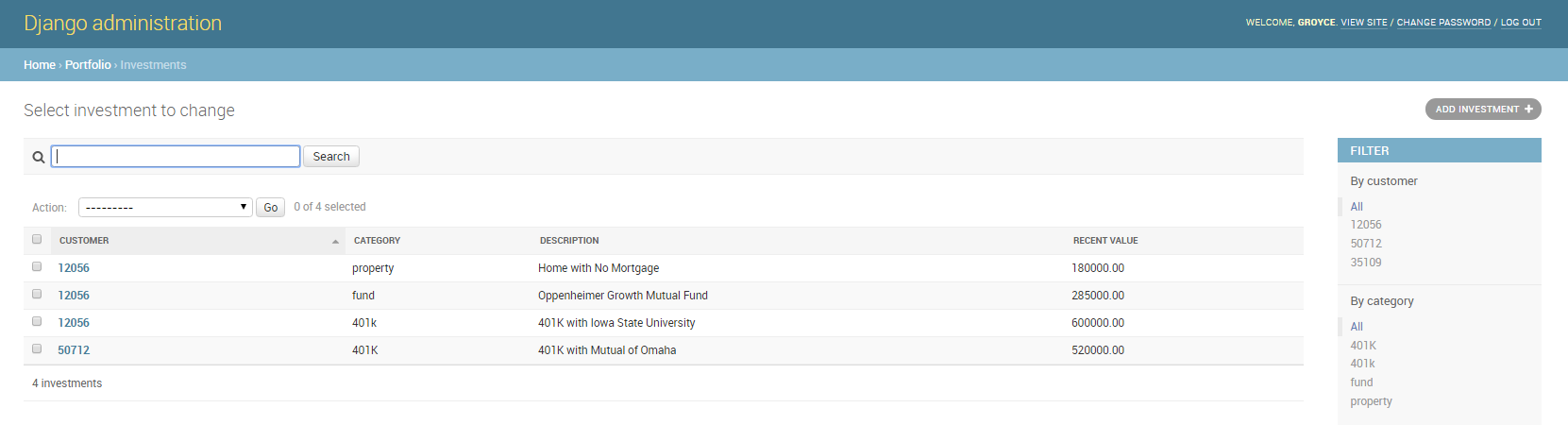
Below you find a list of the key values in the stock table.



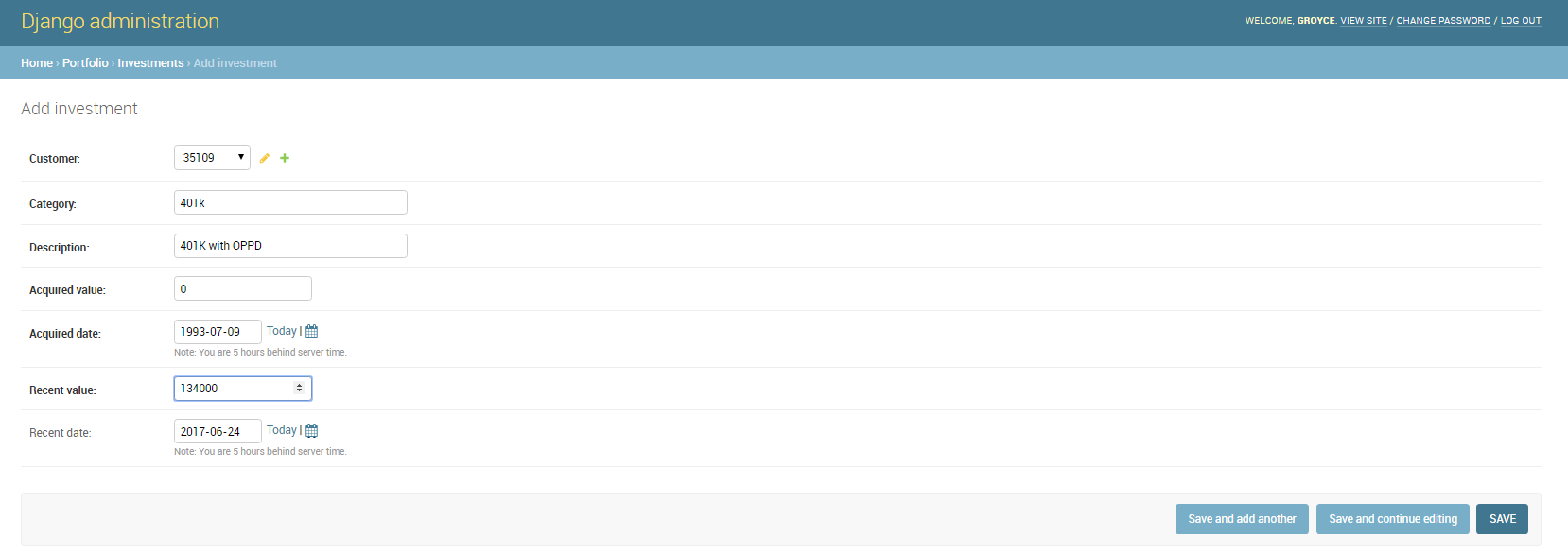
As with the customers, You can add, update and delete stocks for each customer.



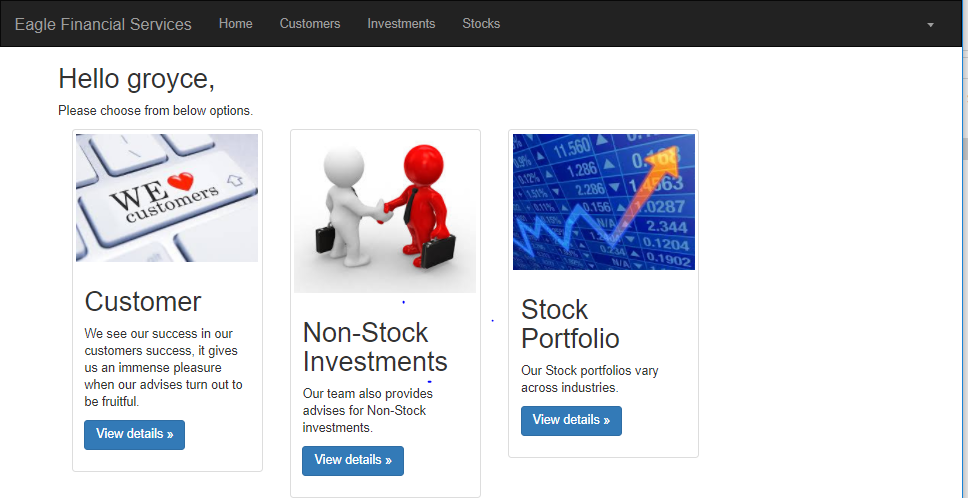
Next we have the page listing the investments. The field call “Results by Investment” is calculated field by finding the difference between the Recent Value from the Acquired Values.



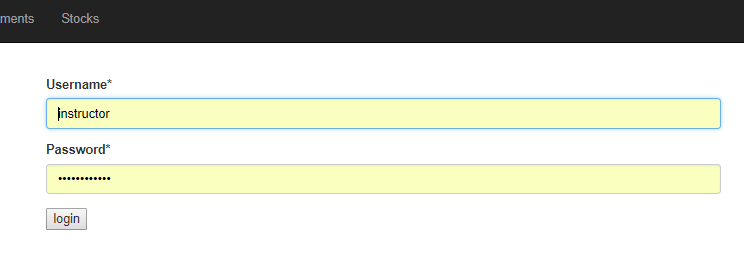
Here is screen showing the editing of investments:



As mentioned earlier, these screens can be accessed by those with administrative rights. Now let's take a look at screens the Financial Advisors might use in working with their customers. Let's assume that the customer information including are handled by the administrators. The Financial Advisor must be able to login into a system and review a list of his customers and the stocks and investments owned by the customer. Here we are back to the landing page for the advisor:

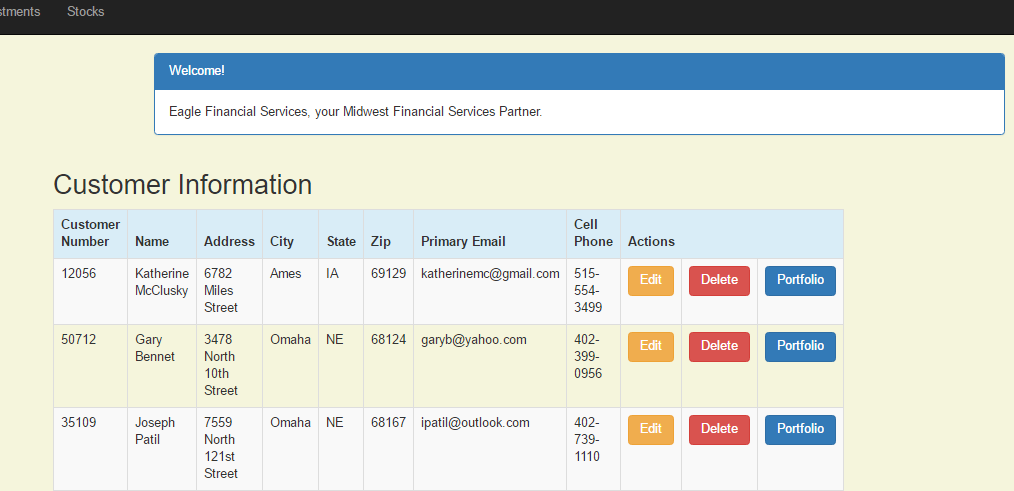


If they are not already logged in, they would log in by clicking the login button and receive this page:

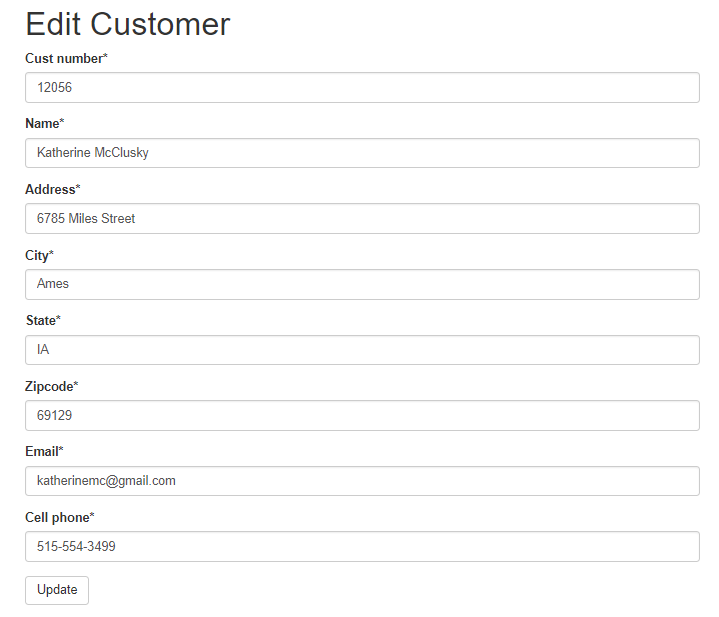


No other features such as “forget your password” in the assignment. This can again be something you can do to move your grade from B to A in part 3 of this assignment. Once they have signed in, an advisor can see a list of stocks and add or update stocks with a page similar to the one shown below.

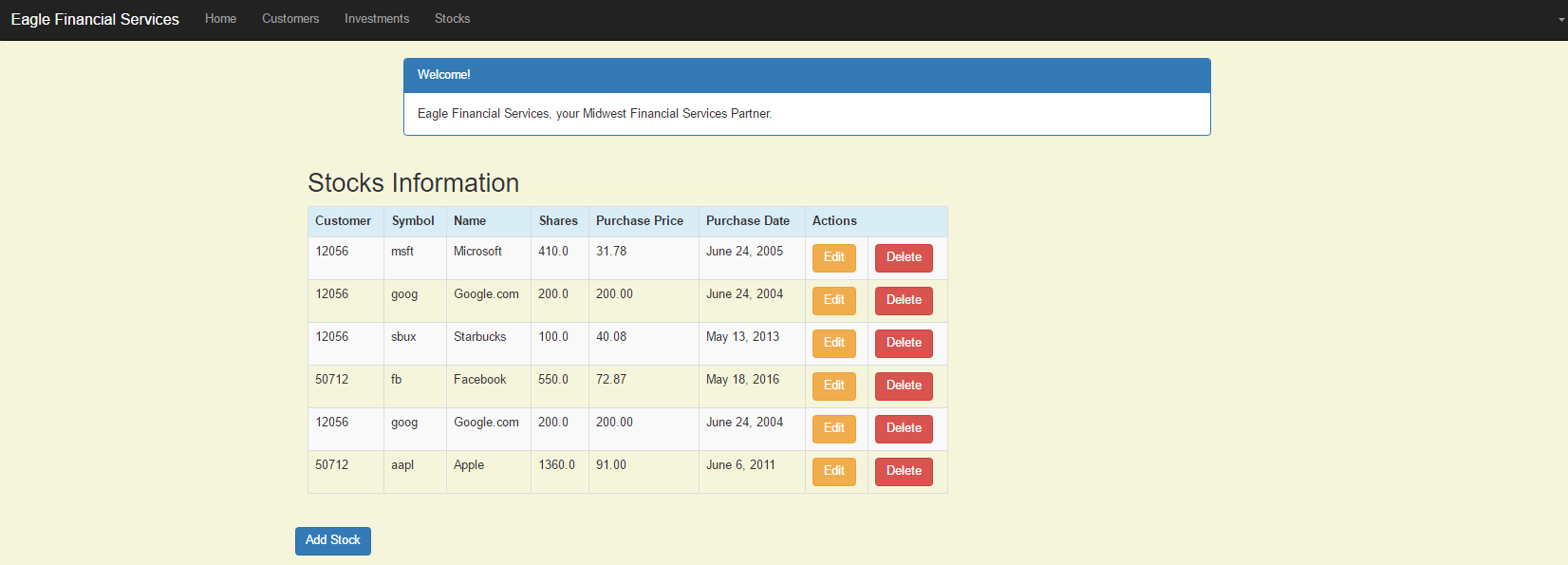
Customer Screen



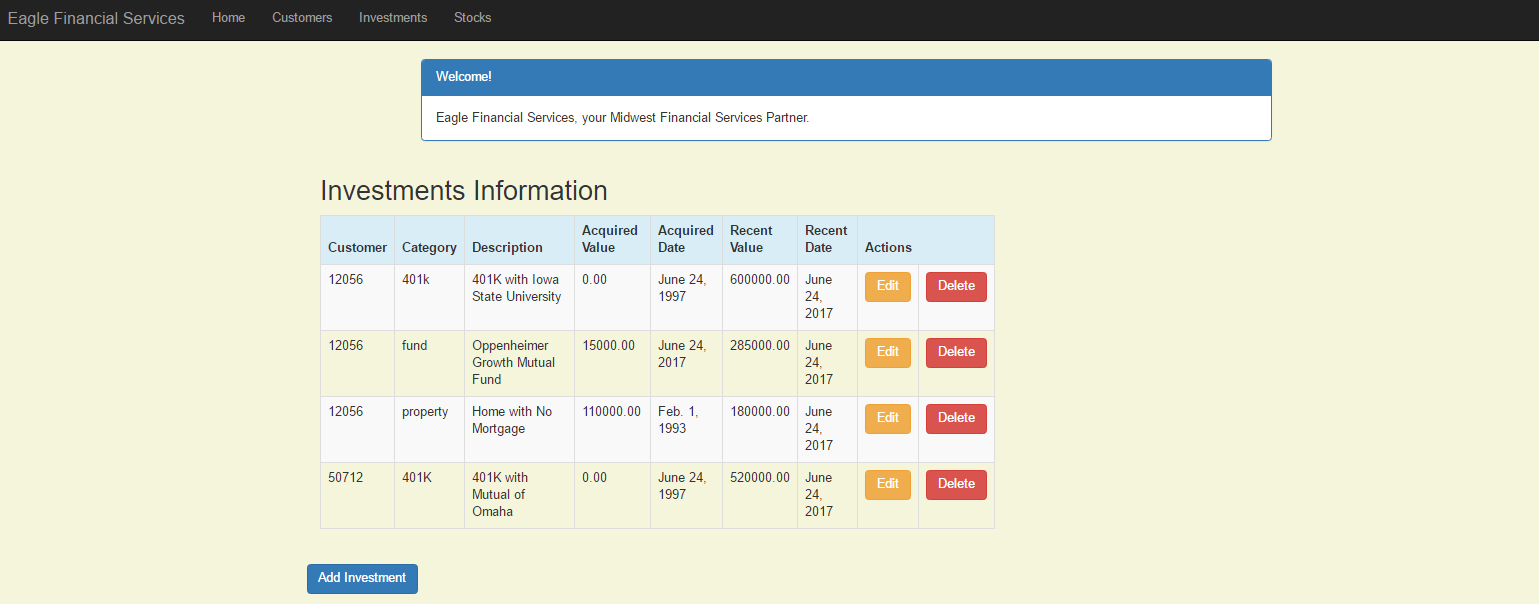
The page allows the advisor to update the address, phone, etc. and they can also delete a customer. If they request to edit a customer page like this comes up.



We also have pages that allow advisors to add, update and delete stocks. Below is the screen to edit a stock.



And we have a similar set of pages to add, update and delete investments.



As with the stock page, the edit option brings up a page similar to the edit of customer information shown earlier.

**Section 2 - Directions for Setting up the EFS 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 call efsd2.

**mkdir efsd2**

Now inside this first efsd2 directory we will create the files to create a virtual environment.

Run the following commands:

**python -m venv myvenv**

**On MAC**

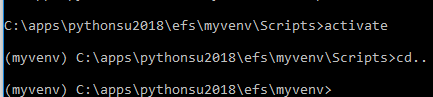
**python3 -m virtualenv myvenv**

Next we **must activate the virtual environment** with

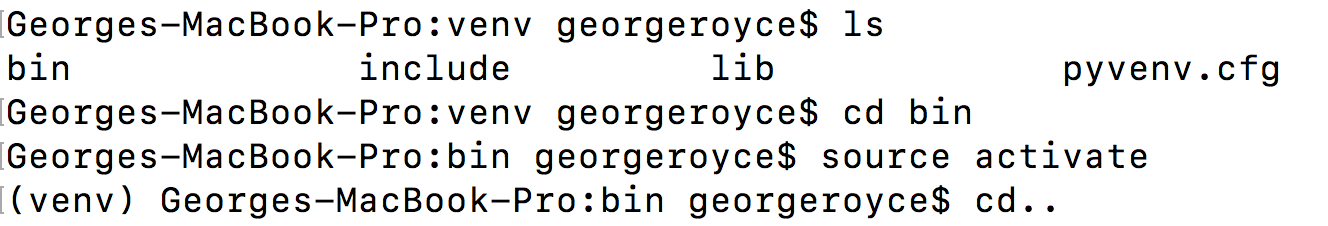
On Windows go to scripts directory and type **activate**

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

In Windows this will look like this:



In a Mac it should look like this



As it mentions, any time you want to exit the virtual environment you can type deactivate.

Now we will install django into the virtual environment using this command:

Make sure you have the latest version of pip,

python -m pip install --upgrade pip

**pip install** Django~=3.2.10

or

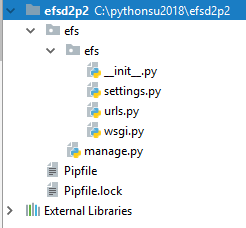
pip install -r requirements.txt

Now it is time to create our project.

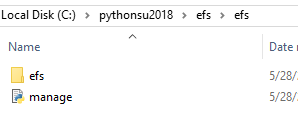
Run the following command.

**django-admin startproject efs .**

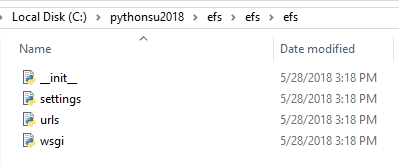
If you now open this project in your Pycharm editor, the directory structure will look like this:



One level deeper you will see the actual project directory which you can always tell since it contains manage.py



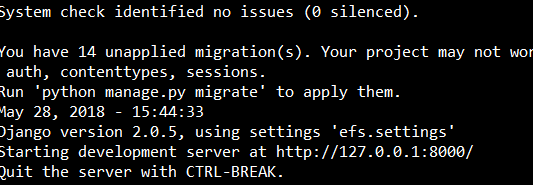
Finally, the efs project contains the contains these files:



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**

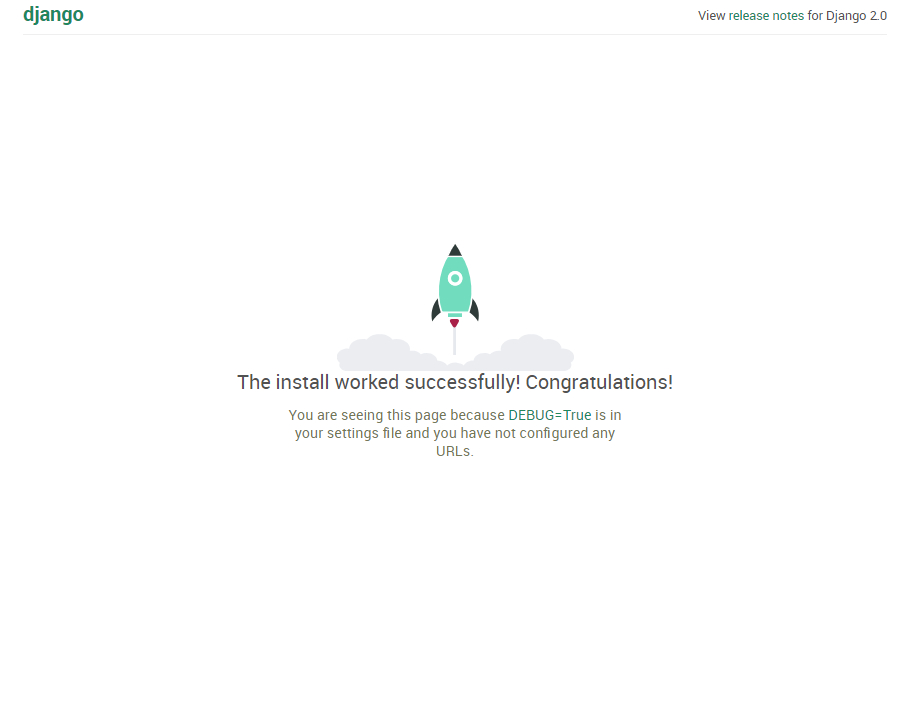
You will receive this on the command line terminal:

****

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:

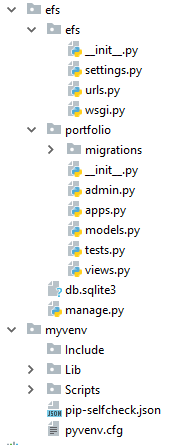


**Section 3 - Creating the portfolio app**

The main application of this project is called the portfolio app. You should continue to be in the main project directory with manage.py. Now issue the following command:

**python manage.py startapp portfolio**

Now you should have the following app files:



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

**Section 4 - Editing the settings.py file to connect the efs project with the portfolio app**

Now we need to let the efs project know we have the ‘portfolio app”. To do that we edit the settings.py file in the efs project directory.



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.

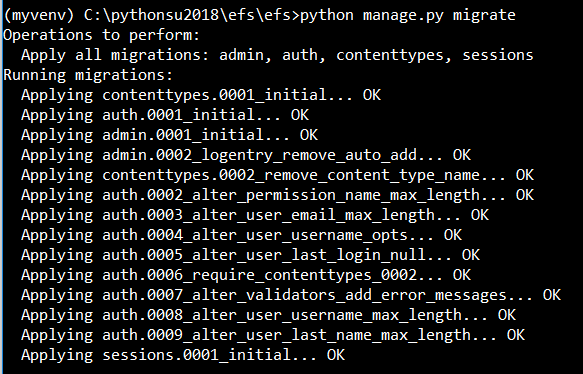
3. We can also change timezone from UTC to ‘America/Chicago’ for now.

**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.

****

Now create a superuser for the application using the following command:

**python manage.py createsuperuser**

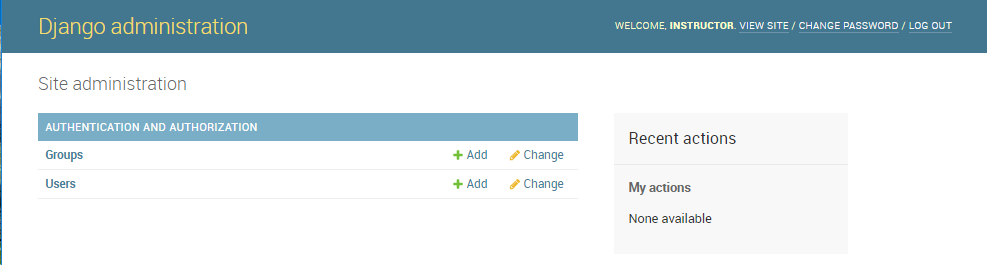
Please use this for one of your superusers. It will facilitate grading of the assignment.

User = smith

Email = jks11061@gmail.com

Password = Maverick27

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: 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 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 Eagle Financial Services Company. 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):  name = models.CharField(max\_length=50)  address = models.CharField(max\_length=200)  cust\_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)  email = models.EmailField(max\_length=200)  cell\_phone = 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\_number) |

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\_number', 'name', 'city', 'cell\_phone')  list\_filter = ('cust\_number', 'name', 'city')  search\_fields = ('cust\_number', 'name')  ordering = ['cust\_number']  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**

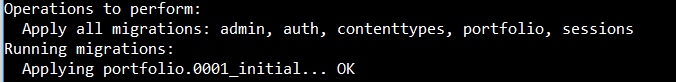
The results should be:

****

Follow this with the command

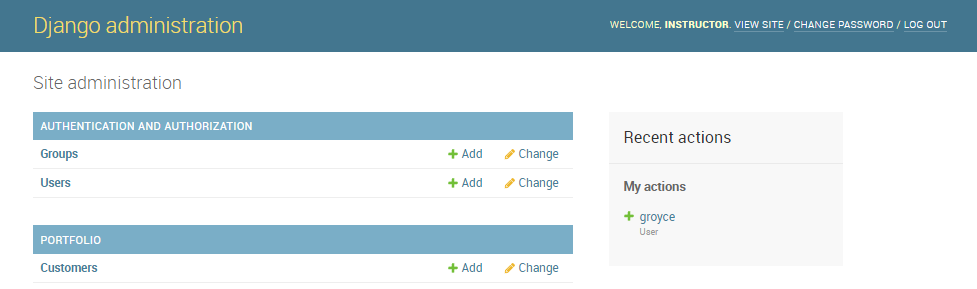
**python manage.py migrate**

You should receive this message:

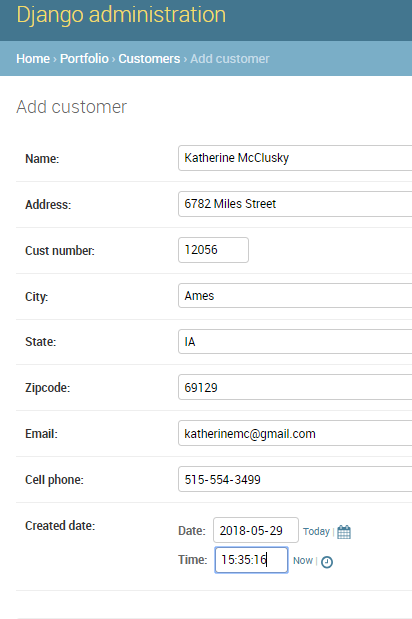


Now let’s see what our table looks like in the admin panel. To do this startup the server with **python manage.py runserver**

The new table will appear under the portfolio app listing



Let's add some customers to the application. [Click here for EFS Table data](https://docs.google.com/spreadsheets/d/1iPhE-5ijQkpe5Hhmegqiltm74LxEV5KmXBL5-_zUgok/edit#gid=0). 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.**

Customers of financial services companies may sign up in a variety of ways. They may sign up online and provide information about their current financial investments. They may also call an 800 number and signup and they may even signup through a paper mail campaign where they send in a form that is entered into a system. These potential customers are then contacted by a financial advisor to ensure the information is correct and to explain the services EFS provides to help the customer manage and grow their financial portfolio. 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 financial advisors 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 portfolio 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\_number', 'name', 'address', 'city', 'state', 'zipcode', 'email', 'cell\_phone',) |

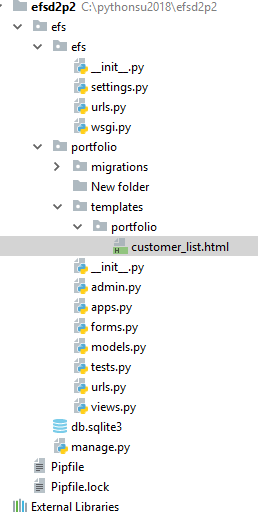
Since Django is a model-view-template framework we now must describe what we want to displate in a template in the views.py file. Double click to open the views.py file in portfolio app, and enter the following:

**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, 'portfolio/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 portfolio directory in PyCharm, and selecting New, and then Directory. Name it templates. Inside of the templates directory create a directory portfolio. 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>Eagle Financial</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">  Eagle Financial Services, your Midwest Financial Services Partner.  </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 Number</th>  <th>Name</th>  <th>Address</th>  <th>City</th>  <th>State</th>  <th>Zip</th>  <th>Primary Email</th>  <th>Cell Phone</th>  <th colspan="3">Actions</th>  </tr>  </thead>  <tbody>  {% for customer in customers %}  <tr>  <td>{{ customer.cust\_number }}</td>  <td>{{ customer.name }}</td>  <td>{{ customer.address }}</td>  <td>{{ customer.city }}</td>  <td>{{ customer.state }}</td>  <td>{{ customer.zipcode }}</td>  <td>{{ customer.email }}</td>  <td>{{ customer.cell\_phone }}</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">Portfolio</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 portfolio folder, create a urls.py python file and add the following:

**portfolio\urls.py**

|  |
| --- |
| from django.conf.urls import url  from . import views  from django.urls import path  app\_name = 'portfolio'  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.

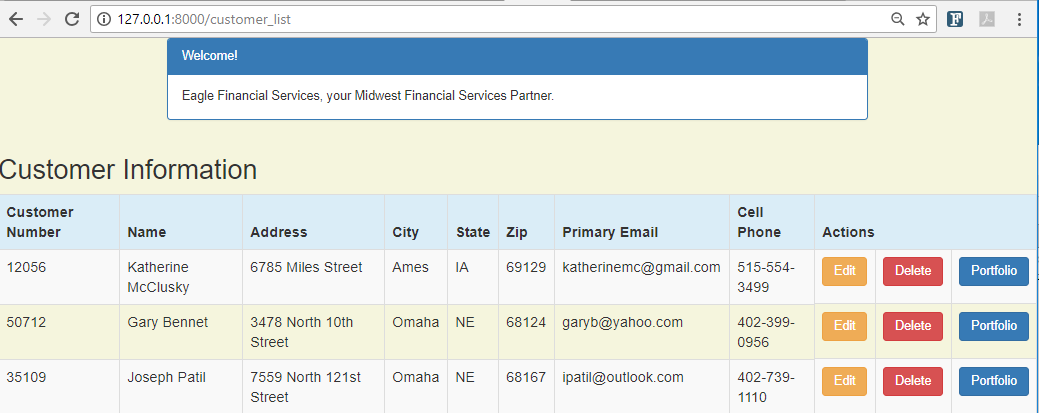
**efs\urls.py**

|  |
| --- |
| from django.contrib import admin  from django.urls import path, include  urlpatterns = [  path('admin/', admin.site.urls),  path('', include('portfolio.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:



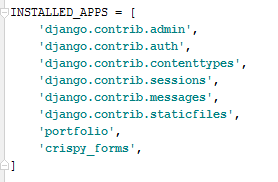
**Section 6 - Creating the ability to update information in the list.**

Now we will create the ability to edit a record using the edit view. The process is similar to the above process:

1. We need to add a form helper to simplify for creation. To do this you will need to install Crispy forms. Do this with the following command:\*\*\*

**pip install Django-crispy-forms**

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



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 = 'bootstrap4' |

**\*\*\* You can try to use Crispy forms on a MAC but I had issues with it. I was able to have it work on Windows and on server environments but my MAC OS had an issue. Not to work, just explain you are using standard forms since you are using a Mac. If you do get it to work, let me know. Thanks**

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**

|  |
| --- |
| {% 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 %} |

On a MAC, if you cannot use Crispy forms simply replace this line **{{ form|crispy }}** with {{ form.as\_p }} for standard form look and feel.

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, 'portfolio/customer\_list.html',  {'customers': customer})  else:  # edit  form = CustomerForm(instance=customer)  return render(request, 'portfolio/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.

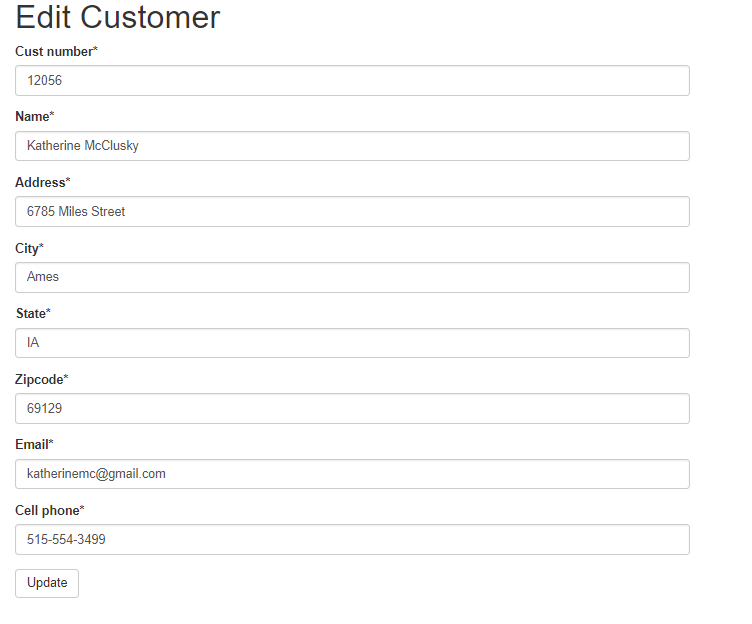
|  |
| --- |
| from django.conf.urls import url  from . import views  from django.urls import path  app\_name = 'portfolio'  urlpatterns = [  path('customer\_list', views.customer\_list, name='customer\_list'),  path('customer/<int:pk>/edit/', views.customer\_edit, name='customer\_edit'),  ] |

5. Need to update the customer list. Replace:

|  |
| --- |
| **REPLACE:**  <td><a href="" class="btn btn-warning">Edit</a>  **WITH:**  <**td**><**a href="{% url 'portfolio: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 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 'portfolio: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 'portfolio: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.

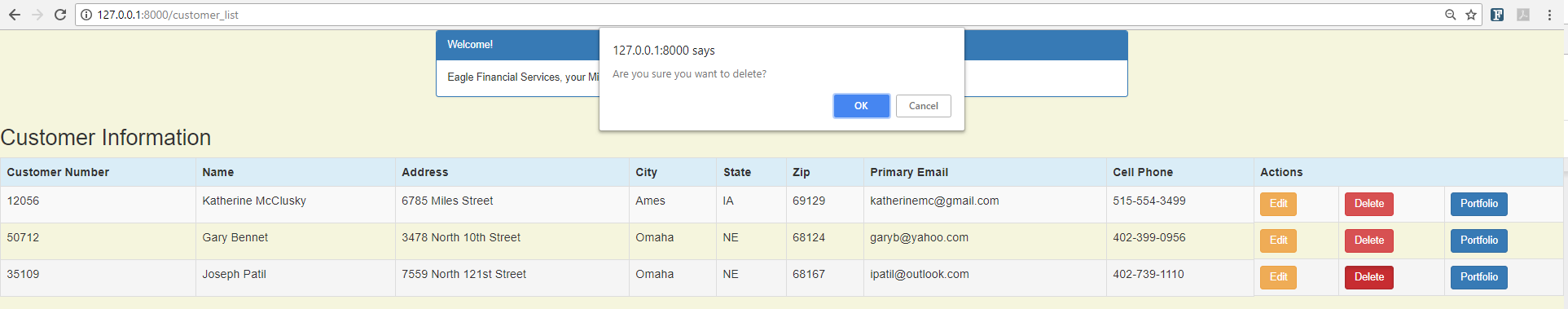
**Views.py**

|  |
| --- |
| def customer\_delete(request, pk):  customer = get\_object\_or\_404(Customer, pk=pk)  customer.delete()  return redirect('portfolio:customer\_list')  **Also add this above in views.pye:**  from django.shortcuts import redirect |

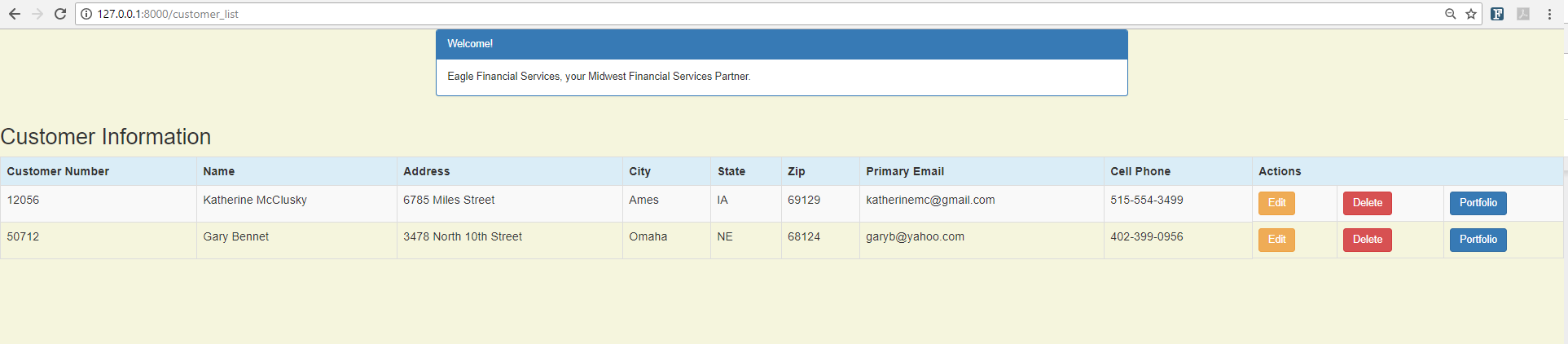
3. We then add the following to the portfolio urls.py file

|  |
| --- |
| path('customer/<int:pk>/delete/', views.customer\_delete, name='customer\_delete'), |

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



If you answer OK, then you should see this:



**Section 8 - Adding Stocks and Investments 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. We will assume administrators will add a new customer for the financial advisors and they will only have security to use the pages we create outside the admin panel. However, advisors will need to be able to add or change stocks and investments. Let's move on this this part of the site now.

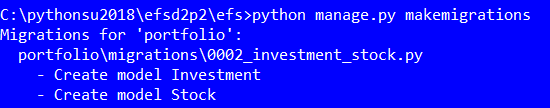
To begin we need to start by adding to the model. Add the following information to your model so the new model looks like this:

|  |
| --- |
| from django.db import models from django.utils import timezone from django.contrib.auth.models import User #import requests  # Create your models here. class Customer(models.Model):  name = models.CharField(max\_length=50)  address = models.CharField(max\_length=200)  cust\_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)  email = models.EmailField(max\_length=200)  cell\_phone = 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\_number)   class Investment(models.Model):  customer = models.ForeignKey(Customer, on\_delete=models.CASCADE, related\_name='investments')  category = models.CharField(max\_length=50)  description = models.CharField(max\_length=200)  acquired\_value = models.DecimalField(max\_digits=10, decimal\_places=2)  acquired\_date = models.DateField(default=timezone.now)  recent\_value = models.DecimalField(max\_digits=10, decimal\_places=2)  recent\_date = models.DateField(default=timezone.now, blank=True, null=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.customer)   def results\_by\_investment(self):  return self.recent\_value - self.acquired\_value  class Stock(models.Model):  customer = models.ForeignKey(Customer, on\_delete=models.CASCADE, related\_name='stocks')  symbol = models.CharField(max\_length=10)  name = models.CharField(max\_length=50)  shares = models.DecimalField (max\_digits=10, decimal\_places=1)  purchase\_price = models.DecimalField(max\_digits=10, decimal\_places=2)  purchase\_date = models.DateField(default=timezone.now, blank=True, null=True)   def created(self):  self.recent\_date = timezone.now()  self.save()   def \_\_str\_\_(self):  return str(self.customer)   def initial\_stock\_value(self):  return self.shares \* self.purchase\_price |

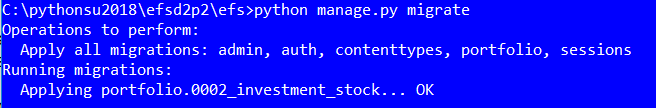
After you have added these new tables to the model you won’t see much of a change. Even in the admin panel only the customer table shows up. What do we need to do?

I am hoping you said, we need to create and run migrations.

**python manage.py makemigrations**

****

**python manage.py migrate**

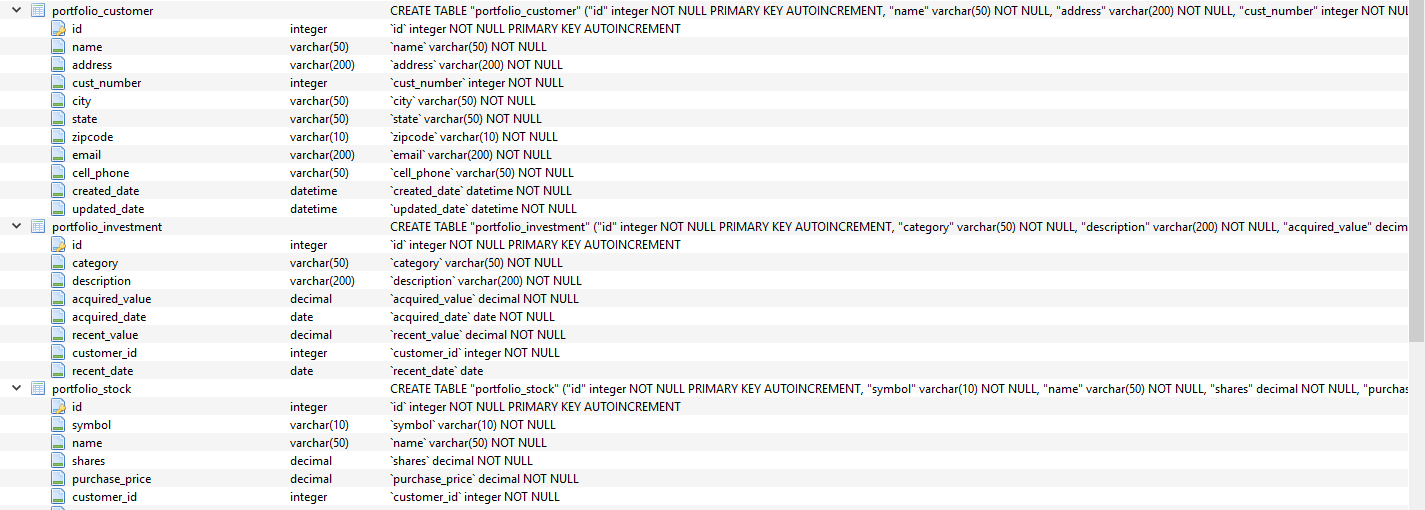


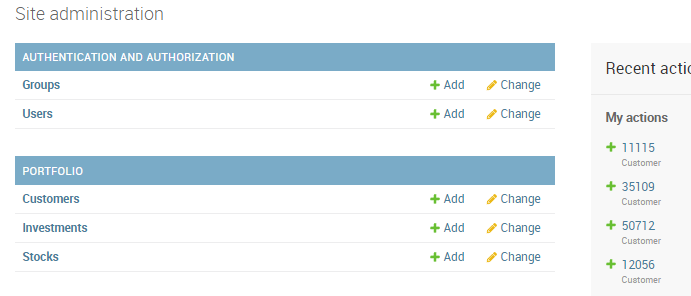
To see the new tables in the admin panel you will need to add the following to the admin.py.

**admin.py**

|  |
| --- |
| from django.contrib import admin  from .models import Customer, Investment, Stock  class CustomerList(admin.ModelAdmin):  list\_display = ('cust\_number', 'name', 'city', 'cell\_phone')  list\_filter = ('cust\_number', 'name', 'city')  search\_fields = ('cust\_number', 'name')  ordering = ['cust\_number']  class InvestmentList(admin.ModelAdmin):  list\_display = ('customer', 'category', 'description', 'recent\_value')  list\_filter = ('customer', 'category')  search\_fields = ('customer', 'category')  ordering = ['customer']  class StockList(admin.ModelAdmin):  list\_display = ('customer','symbol', 'name', 'shares', 'purchase\_price')  list\_filter = ('customer','symbol', 'name')  search\_fields = ('customer','symbol', 'name')  ordering = ['customer']  admin.site.register(Customer, CustomerList)  admin.site.register(Investment, InvestmentList)  admin.site.register(Stock, StockList) |

Let's take a minute and understand the relationship between tables. As you can imagine, customer and stocks and customer and investments are 1 to many relationships. If you download the Db Browser for SQLite from: <https://sqlitebrowser.org/> you will see the table 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.



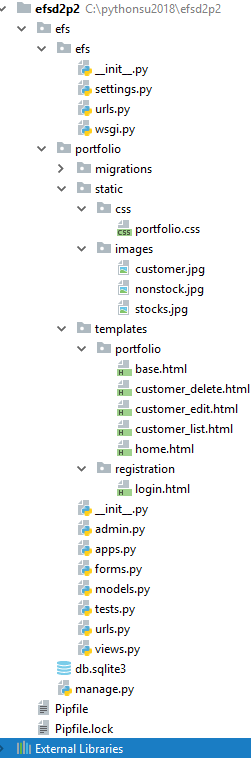
Now you should see this when you restart the application and look at the admin panel:.

Go ahead and add the same data now to Investments and Stocks table. [Click here for Sample EFS Table data](https://docs.google.com/spreadsheets/d/1iPhE-5ijQkpe5Hhmegqiltm74LxEV5KmXBL5-_zUgok/edit#gid=0). For starters we just need to have 1 stock and 1 investment for each customer.

**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 pages for stocks and investments. I will show you how to add a new stock and list stocks. You will then need to add the remaining delete and edit a stock and then continue to add the investment list, 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 these files.](https://drive.google.com/open?id=14AbeBDKmNww2rgk6MAETrLC08vFify23) 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 "portfolio/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 in part 3 of this assignment.

The next files we should review are the base.html file and the home.html files. These files will give our site a homepage. 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>Eagle Financial 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="/">  Eagle Financial Services  </a>  </div>  <div class="collapse navbar-collapse" id="myNavbar">  <ul class="nav navbar-nav">  <li><a href="{% url 'portfolio:home' %}">Home</a></li>  <li><a href="{% url 'portfolio:customer\_list' %}">Customers</a></li>  <li><a href="{% url 'portfolio:customer\_list' %}">Investments</a></li>  <li><a href="{% url 'portfolio:customer\_list' %}">Stocks</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>We see our success in our customers success, it gives us an immense pleasure when  our  advises  turn out to be fruitful. </p>  {% if user.is\_authenticated %}  <p><a class="btn btn-default" href="{% url 'portfolio: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/nonstock.jpg" %}" alt="non-stock"/>  <div class="caption">  <h2>Non-Stock Investments</h2>  <p>Our team also provides advice for Non-Stock investments.</p>  <p><a class="btn btn-default" href="{% url 'portfolio:customer\_list' %}"  role="button">View  details &raquo;</a></p>  </div>  </div>  </div>  <div class="col-md-3">  <div class="thumbnail">  <img src="{% static "images/stocks.jpg" %}" alt="stocks"/>  <div class="caption">  <h2>Stock Portfolio</h2>  <p>Our Stock portfolios vary across industries.</p>  <p><a class="btn btn-default" href="{% url 'portfolio: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 “portfolio:customer\_list” where you might expect it to say investment or stock. 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 “portfolio:customer\_list” when it should be referencing stock or investment. Again, we will fix this shortly.

**home.html**

|  |
| --- |
| {% extends 'portfolio/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, it gives us an immense  pleasure when our  advises  turn out to be fruitful. </p>  <p><a class="btn btn-primary" href="{% url 'portfolio:customer\_list' %}"  role="button">View  details &raquo;</a></p>  </div>  </div>  </div>  <div class="col-md-3">  <div class="thumbnail">  <img src="{% static "images/nonstock.jpg" %}" alt="non-stock"/>  <div class="caption">  <h2>Non-Stock Investments</h2>  <p>Our team also provides advice for Non-Stock investments.</p>  <p><a class="btn btn-primary" href="{% url 'portfolio:customer\_list' %}"  role="button">View  details &raquo;</a></p>  </div>  </div>  </div>  <div class="col-md-3">  <div class="thumbnail">  <img src="{% static "images/stocks.jpg" %}" alt="stocks"/>  <div class="caption">  <h2>Stocks</h2>  <p>Our Stock portfolios vary across industries.</p>  <p><a class="btn btn-primary" href="{% url 'portfolio: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, 'portfolio/home.html',  {'portfolio': home})  @login\_required  def customer\_list(request):  customer = Customer.objects.filter(created\_date\_\_lte=timezone.now())  return render(request, 'portfolio/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, 'portfolio/customer\_list.html',  {'customers': customer})  else:  # edit  form = CustomerForm(instance=customer)  return render(request, 'portfolio/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 portfolio/urls file

|  |
| --- |
| from django.conf.urls import url  from . import views  from django.urls import path  app\_name = 'portfolio'  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'), |

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

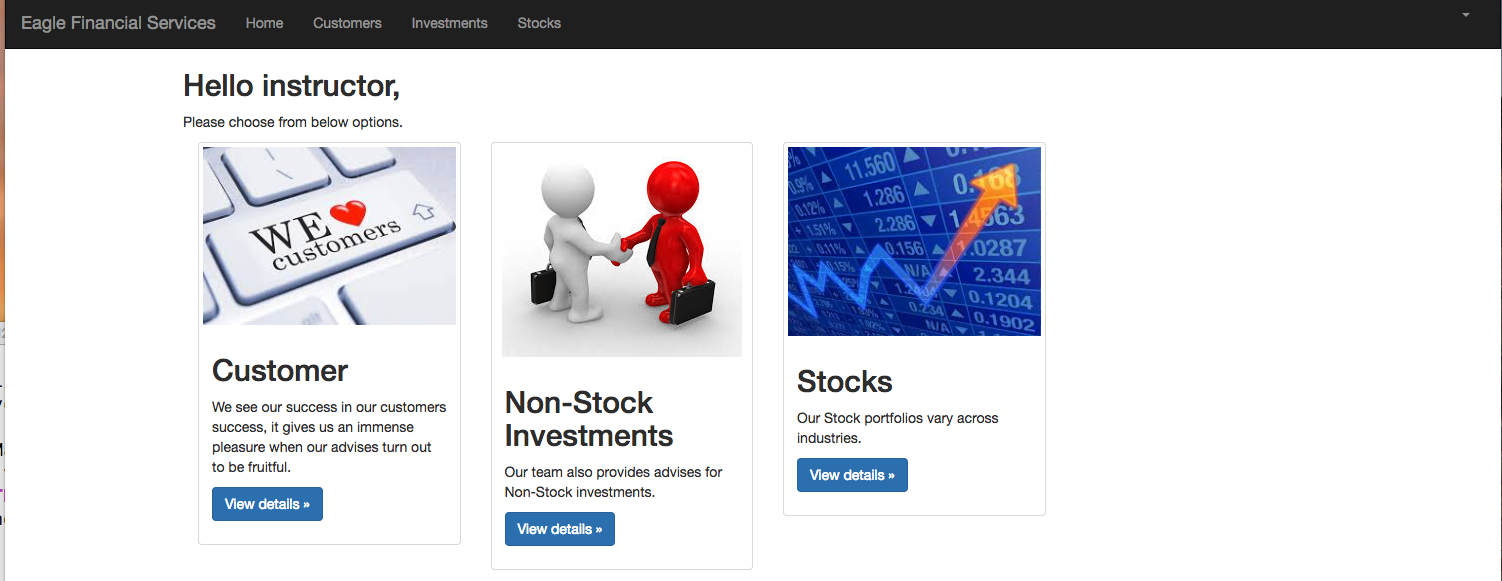
**efs\urls.py**

|  |
| --- |
| from django.contrib import admin  from django.conf.urls import include,url  from django.urls import path  from django.contrib.auth import views  urlpatterns = [  path('admin/', admin.site.urls),  path('', include('portfolio.urls')),  url(r'^accounts/login/$', views.login, name='login'),  url(r'^accounts/logout/$', views.logout, name='logout', kwargs={'next\_page': '/'}),  ] |

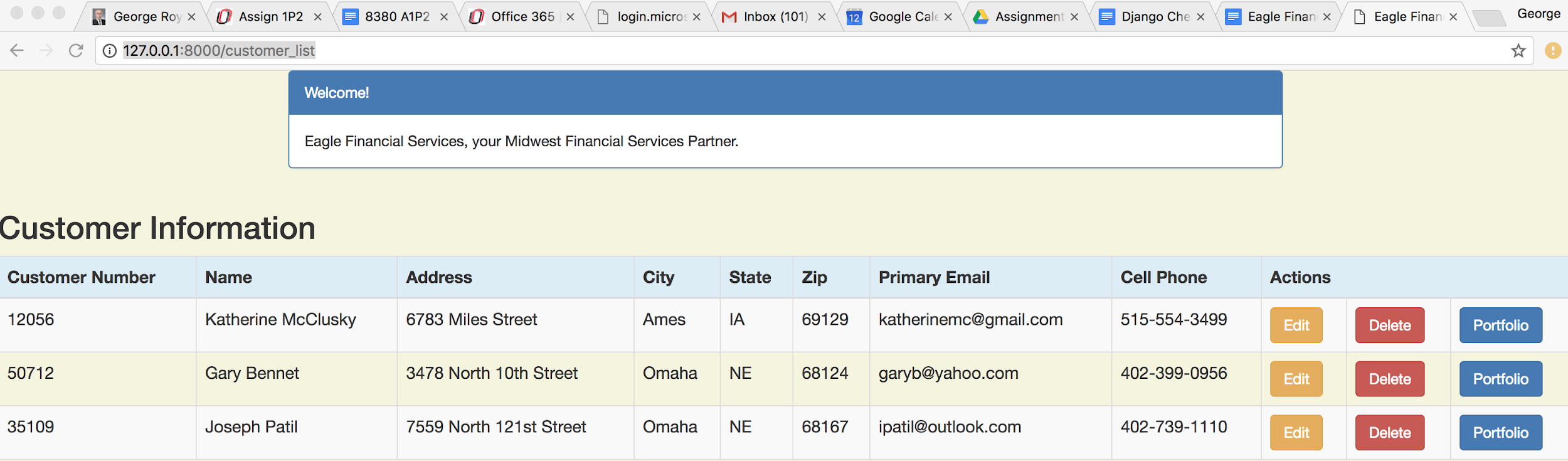
Also add this line in the settings.py

|  |
| --- |
| 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 jpg’s but you are welcome to improve this this.

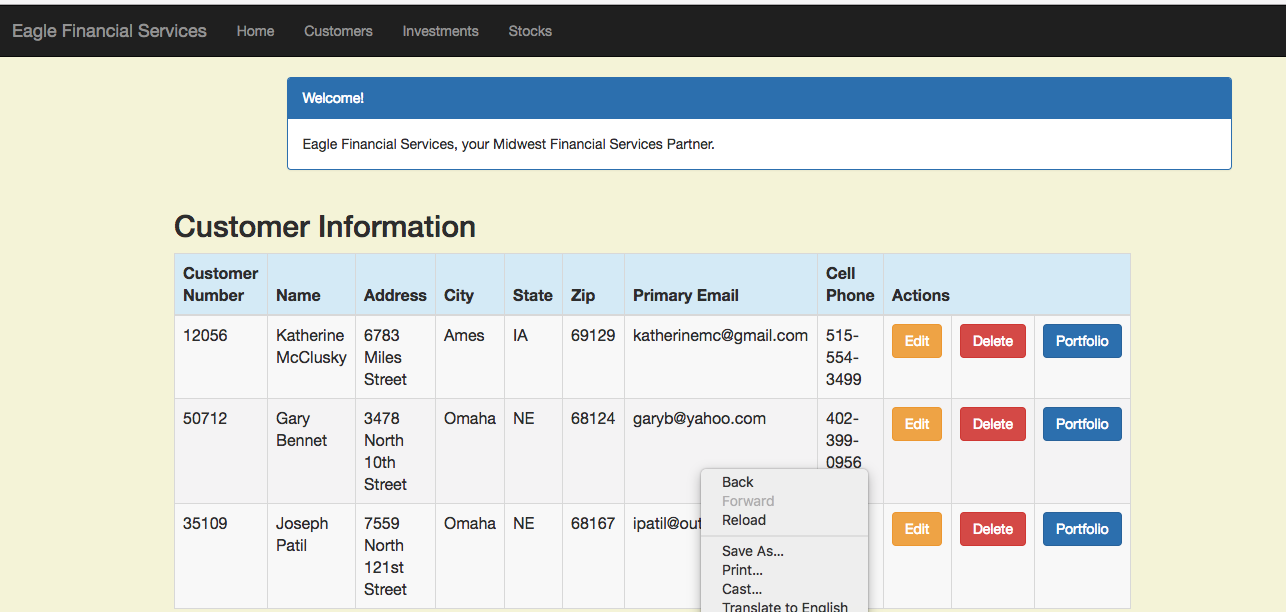


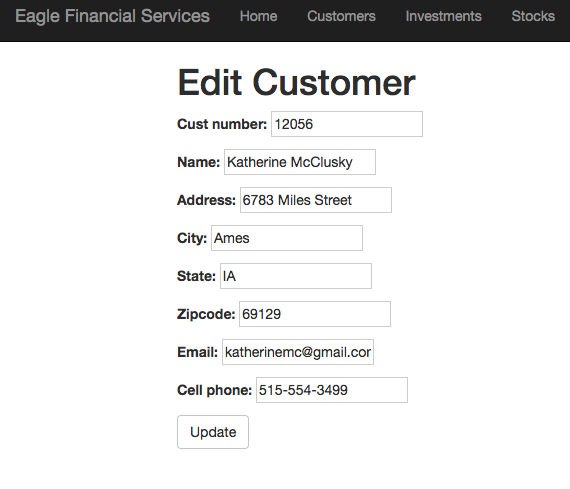
Now you can see the navigation provided by base.html. Right now when you click on non-stock investments and stocks you will find that I have redirected the link to avoid an error. When we add the code to work with these financial instruments, we will replace the links. When you click on View details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



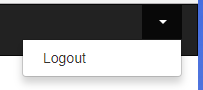
We need to add a line the file to bring the navigation to this file. What would you add to do this. Hopefully you recognize that the following line should be place in each of the pages we have developed

|  |
| --- |
| {% extends 'portfolio/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 you 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 stocks for a customer outside the admin panel**

If you examine the fields for the stock entries and how we created views entries, templates and url entries, you should have a good idea of how to add the stock and investment features. I will help you out with the stock list and adding a new stock. Your job will be to following the pattern and create the ability to edit and delete a stock for a customer outside the admin panel. Let's begin by adding the stock list template. Create a stock\_list.html file in the templates directory. Add the following code.

stock\_list.html

|  |
| --- |
| {% extends 'portfolio/base.html' %}  {% block content %}  <html>  <head>  <meta charset="UTF-8">  <title>Eagle Financial</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">  Eagle Financial Services, your Midwest Financial Services Partner.  </div>  </div>  </div>  </div>  </div>  <div class="row">  <h2 style="padding-left: 15Px">Stocks Information</h2>  </div>  <div>  <table class="table table-striped table-bordered table-hover">  <thead>  <tr class="bg-info">  <th>Customer number</th>  <th>Symbol</th>  <th>Name</th>  <th>Shares</th>  <th>Purchase Price</th>  <th>Purchase Date</th>  <th colspan="3">Actions</th>  </tr>  </thead>  <tbody>  {% for stock in stocks %}  <tr>  <td>{{ stock.customer }}</td>  <td>{{ stock.symbol }}</td>  <td>{{ stock.name }}</td>  <td>{{ stock.shares }}</td>  <td>{{ stock.purchase\_price }}</td>  <td>{{ stock.purchase\_date }}</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 Stock</span></a>  </div>  </div>  </body>  </html>  {% endblock %} |

Next we need to the following to the views.py

Add to **views.py**

|  |
| --- |
| @login\_required  def stock\_list(request):  stocks = Stock.objects.filter(purchase\_date\_\_lte=timezone.now())  return render(request, 'portfolio/stock\_list.html', {'stocks': stocks}) |

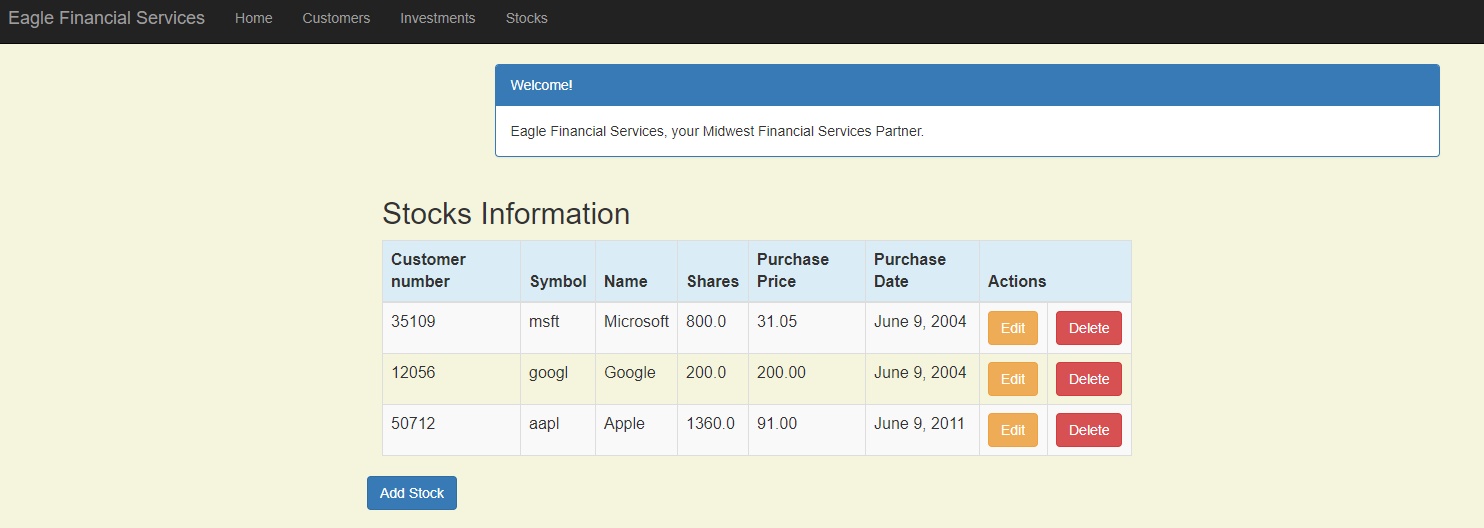
We also need to add the following line to the **portfolio/urls.py**.

|  |
| --- |
| path('stock\_list', views.stock\_list, name='stock\_list'), |

Also, you will now need to go back to the home.html and base.html and change the **customer.list.html** to the **stock.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 Stock Button.

Step 1 - To do this we will need the following template page:

**stock\_new.html**

|  |
| --- |
| {% extends 'portfolio/base.html' %}  {% load crispy\_forms\_tags %}  {% block content %}  <h1>Add a New Stock</h1>  <form method="POST" class="stock-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 StockForm(forms.ModelForm):  class Meta:  model = Stock  fields = ('customer', 'symbol', 'name', 'shares', 'purchase\_price', 'purchase\_date',)  Also add stock in the models you will import  from .models import Customer, Stock |

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

|  |
| --- |
| @login\_required def stock\_new(request):  if request.method == "POST":  form = StockForm(request.POST)  if form.is\_valid():  stock = form.save(commit=False)  stock.created\_date = timezone.now()  stock.save()  stocks = Stock.objects.filter(purchase\_date\_\_lte=timezone.now())  return render(request, 'portfolio/stock\_list.html',  {'stocks': stocks})  else:  form = StockForm()  # print("Else")  return render(request, 'portfolio/stock\_new.html', {'form': form}) |

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

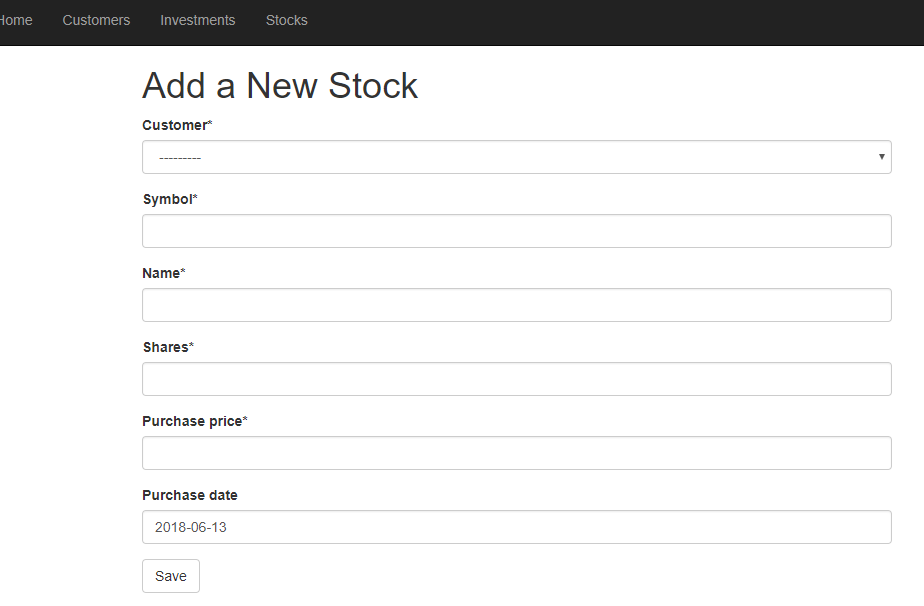
|  |
| --- |
| path('stock/create/', views.stock\_new, name='stock\_new'), |

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 **stock\_list.html**. :

Update the **stock\_list.html**

|  |
| --- |
| <div class="row">  <a href="{% url 'portfolio:stock\_new' %}" class="row"><span  class="btn btn-primary">Add Stock</span></a>  </div> |

You should now be able to add a new stock to your current clients. When you press this Add Stock button you should see:



The next logical form to do is the Edit a Stock.

Step 1 - To do this we will need the following template page:

|  |
| --- |
| {% extends 'portfolio/base.html' %}  {% load crispy\_forms\_tags %}  {% block content %}  <h1>Edit Stock</h1>  <form method="POST" class="stock-form">{% csrf\_token %}  {{ form|crispy }}  <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 stock\_edit(request, pk):  stock = get\_object\_or\_404(Stock, pk=pk)  if request.method == "POST":  form = StockForm(request.POST, instance=stock)  if form.is\_valid():  stock = form.save()  # stock.customer = stock.id  stock.updated\_date = timezone.now()  stock.save()  stocks = Stock.objects.filter(purchase\_date\_\_lte=timezone.now())  return render(request, 'portfolio/stock\_list.html', {'stocks': stocks})  else:  # print("else")  form = StockForm(instance=stock)  return render(request, 'portfolio/stock\_edit.html', {'form': form}) |

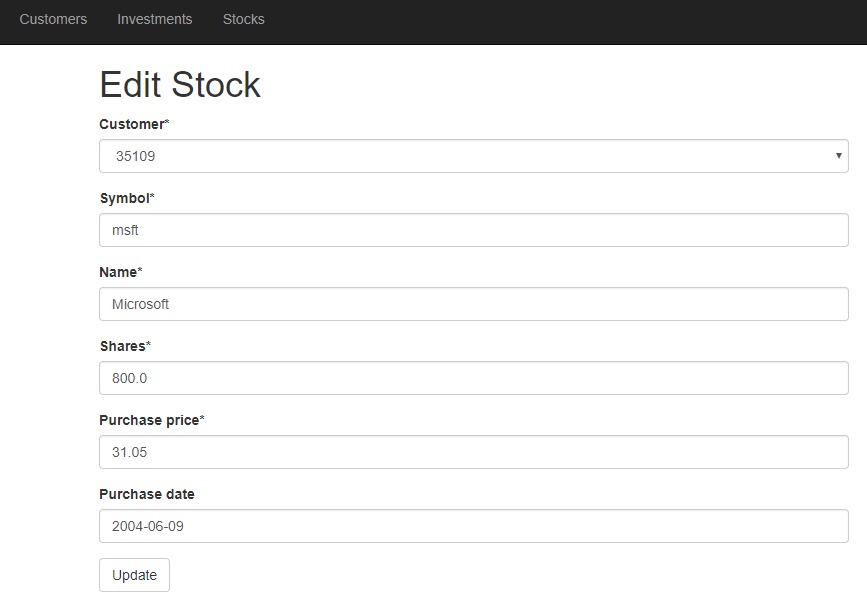
Step 4 - Next, we need to add the following line to the portfolio/urls.py file

|  |
| --- |
| path('stock/<int:pk>/edit/', views.stock\_edit, name='stock\_edit'), |

Step 5 - Don’t forget, we allowed the buttons to show up on the stock\_list.html by using a blank URL. Now we must fill it in as shown below.

|  |
| --- |
| <td><a href="{% url 'portfolio:stock\_edit' pk=stock.pk %}"  class="btn btn-warning">Edit</a></td> |

You should now be able to edit an existing stock for your current clients. When you press this Edit Stock button you should see:

****

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

**12. Now it is your turn to Complete Stocks and Investments and push your code using git to GitHub and a Hosting site like Heroku or Python Anywhere.**

**Now it is your turn to complete the following on your project:**

1. A financial advisor should be able to delete stocks of their customers.

2. A financial advisor should be able to list all the investments of their customers

3. A financial advisor should be able to add a new investment for their customers

4. A financial advisor should be able to edit a current investment for their customers

5. A financial advisor should be able to delete a current investment for their customers

Once you have done the items listed above and tested to ensure everything is working on your own local machine

1. it is now time to deploy your project to Github. Provide the link in the word document of the assignment.

2. Deploy and TEST on either Heroku or Python Anywhere. Provide the link in the word document of the assignment.

\_\_\_\_\_\_

Acknowledgement: Sandra Dizdarevic provided substantial help with the conversion of the part of Assignment 1 to Django 2 during a Spring 2018 independent study.