**Eagle Financial Services API Tutorial (Assign 2P2)**

**Using Python Django 2**

In Assignment 1 Part 2 you developed a 3 table Django application called **portfolio** which allowed administrators and financial advisors to make recommendations on a customer's financial portfolio. You created the application from directions I provided and finished the stock section and completed the Investments section of the application, tested it and deployed it to Heroku or PythonAnywhere. This tutorial will assume you have a functioning application that you submitted for PART 2. You have an automatic backup since you have been using Github. However, some of you are new to Github. If you are new to the use of Github, it is a good idea to also simply make a copy of your existing working application before you proceed with adding the changes to the application.

In this section of the application you will add four major features to the application and have an opportunity to additional features to move your grade from B to an A. The features are:

1. Integrate with the Alpha Vantage web API to provide a current stock value for the customer stock purchases.

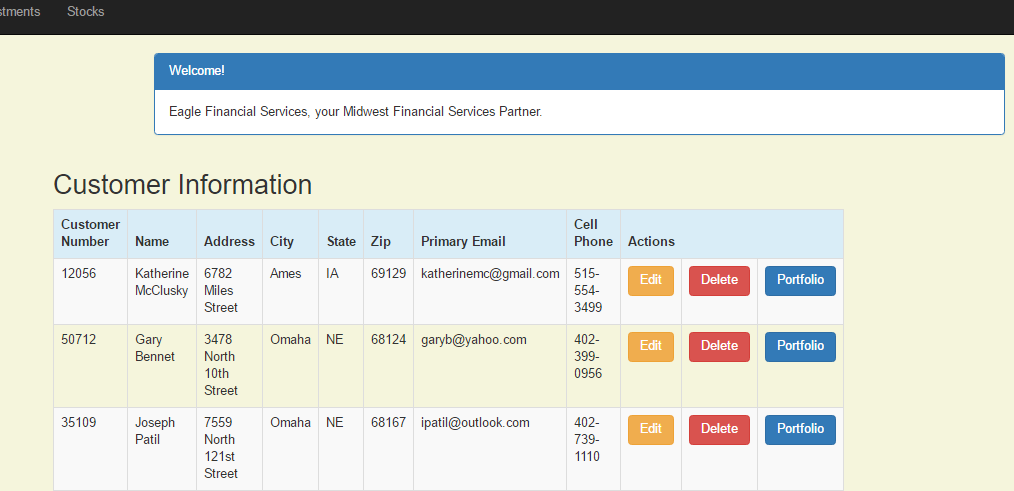
2. Create a single page view of all assets of a customer and the current value of the assets.

3. Use the Django REST framework to expose a JSON API from your application that might be consumed by other applications in the mythical Eagle Financial Services and/or be used in the server side of a Mobile App or an app built using a Javascript framework like [AngularJS](https://angularjs.org/), [ReactJS](https://reactjs.org/) ot ueJS..

4. You will consume JSON API from your application using either Microsoft Excel 2016 or Excel in Microsoft Office 365. We will also discuss a way to do this using Google Sheets.

**Step 1**

You have a portfolio button on the Customer\_list.html page shown below.



We now want to put something behind that button. Below is a sample portfolio template page we will create. Notice that it includes the current stock values. Also note that filters have been applied to the result so that only the results for a given customer is shown in the portfolio.



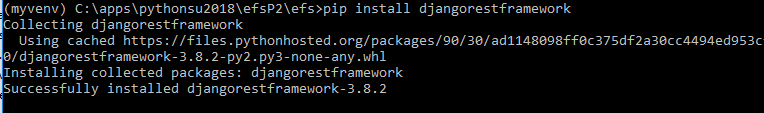
We now need to install some additional packages. Be sure you have your virtual machine running. Once you have confirmed this then you will add the following:

1. Django rest framework

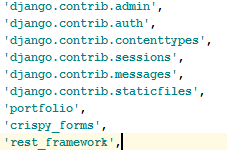
We will need this to create our own web API that exposes the data and functions of this application as a RESTful service. Issue the following command:

**pip install djangorestframework**

You should see something like this:



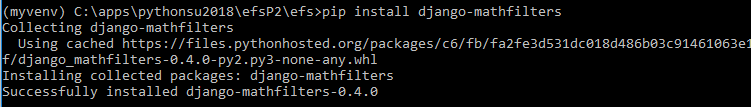
Next add **‘rest\_framework’,** to settings.py as shown below



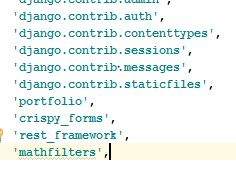
2. Django Mathfilters

This little 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

****

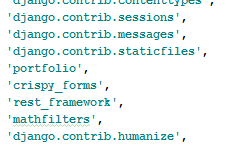
Next add **‘mathfilters’,** to settings.py as shown below:

****

To use this time templates we need to add:

You can check out how to use it at: <https://github.com/dbrgn/django-mathfilters> (scroll dorn to read me)

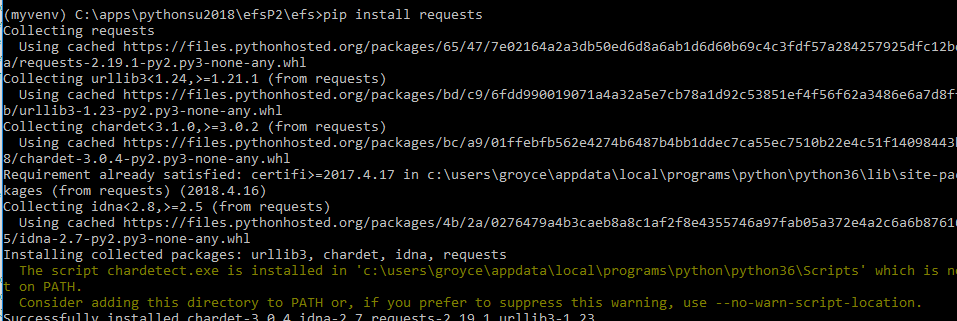
3. Next we will a tool to improve the look and feel of numbers in your app called Django Humanize. This is a part of Django so no need to install it. Simply add: 'django.contrib.humanize' to the installed apps. Your installed apps should look like:

****

Check out how to use it here: <https://docs.djangoproject.com/en/2.0/ref/contrib/humanize/>

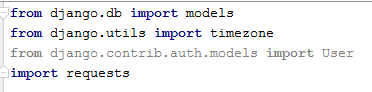
4. Requests is a library to connect to many API’s.

Be sure you are in your VM. To install use the command pip install requests. You should see:



**Step 2** - First you must sign up as a developer needing apis from Alpha Vantage at <https://www.alphavantage.co/> and obtain an API key. Now we will add the following two functions under the Stock class in the **models.py** to retrieve the current\_stock\_ price and to calculate the current\_stock\_value so they can be displayed and used in our portfolio summary page.

To models.py add **import requests** at the top of the page as shown below:



Now add the following code to the end of the Models.py page)

|  |
| --- |
| def current\_stock\_price(self):  symbol\_f = str(self.symbol)  main\_api = 'https://www.alphavantage.co/query?function=BATCH\_STOCK\_QUOTES&symbols='  api\_key = '&apikey= **YOUR API KEY HERE!!**'  url = main\_api + symbol\_f + api\_key  json\_data = requests.get(url).json()  open\_price = float(json\_data["Stock Quotes"][0]["2. price"])  share\_value = open\_price  return share\_value  def current\_stock\_value(self):  return float(self.current\_stock\_price()) \* float(self.shares) |

**Step 3** - We will now use the two calculated values in the views and a portfolio.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 portfolio:

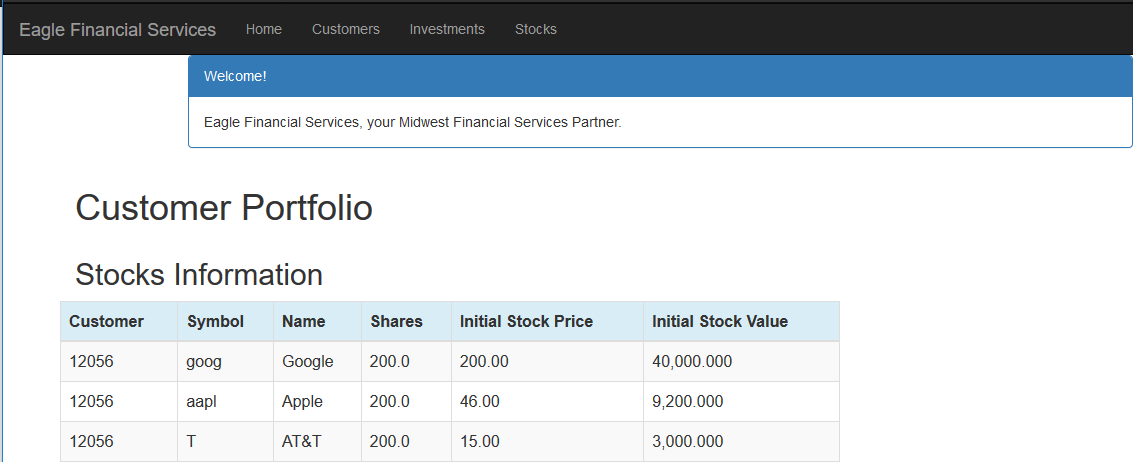
|  |
| --- |
| @login\_required  def portfolio(request,pk):  customer = get\_object\_or\_404(Customer, pk=pk)  customers = Customer.objects.filter(created\_date\_\_lte=timezone.now())  investments =Investment.objects.filter(customer=pk)  stocks = Stock.objects.filter(customer=pk)  sum\_recent\_value = Investment.objects.filter(customer=pk).aggregate(Sum('recent\_value'))  sum\_acquired\_value = Investment.objects.filter(customer=pk).aggregate(Sum('acquired\_value'))  #overall\_investment\_results = sum\_recent\_value-sum\_acquired\_value  # Initialize the value of the stocks  sum\_current\_stocks\_value = 0  sum\_of\_initial\_stock\_value = 0  # Loop through each stock and add the value to the total  for stock in stocks:  sum\_current\_stocks\_value += stock.current\_stock\_value()  sum\_of\_initial\_stock\_value += stock.initial\_stock\_value()  return render(request, 'portfolio/portfolio.html', {'customers': customers,  'investments': investments,  'stocks': stocks,  'sum\_acquired\_value': sum\_acquired\_value,  'sum\_recent\_value': sum\_recent\_value,  'sum\_current\_stocks\_value': sum\_current\_stocks\_value,  'sum\_of\_initial\_stock\_value': sum\_of\_initial\_stock\_value,}) |

This function makes all elements of the customer, stock and invest objects available. These are then summarized using the Django **aggregation function** called **Sum**. Now we are ready to develop the portfolio page.

**Step 4** - The portfolio 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 portfolio.html.

|  |
| --- |
| </div> <div class="row">  <h1 style="padding-left: 15Px">Customer Portfolio</h1> </div>  <div class="row">  <h2 style="padding-left: 15Px">Stocks Information</h2> </div> <div class="row">  <table class="table table-striped table-bordered table-hover">  <thead>  <tr class="bg-info">  <th>Customer</th>  <th>Symbol</th>  <th>Name</th>  <th>Shares</th>  <th>Initial Stock Price</th>  <th>Initial Stock Value</th>   </tr>  </thead>  <tbody>  {% for stock in stocks %}  <tr>  <td>{{ stock.customer }}</td>  <td>{{ stock.symbol }}</td>  <td>{{ stock.name }}</td>  <td>{{ stock.shares|intcomma }}</td>  <td>{{ stock.purchase\_price|intcomma }}</td>  <td>{{ stock.initial\_stock\_value|intcomma}}   </tr>  {% endfor %}  </tbody>  </table> {% endblock %} |

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



Using this starter, shell you need to develop the complete portfolio as shown below. It does NOT need to look exactly like the one below. There are many ways to summarize a portfolio. 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 portfolio.html will not be visible until we add the following line to the portfolio\urls.py

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

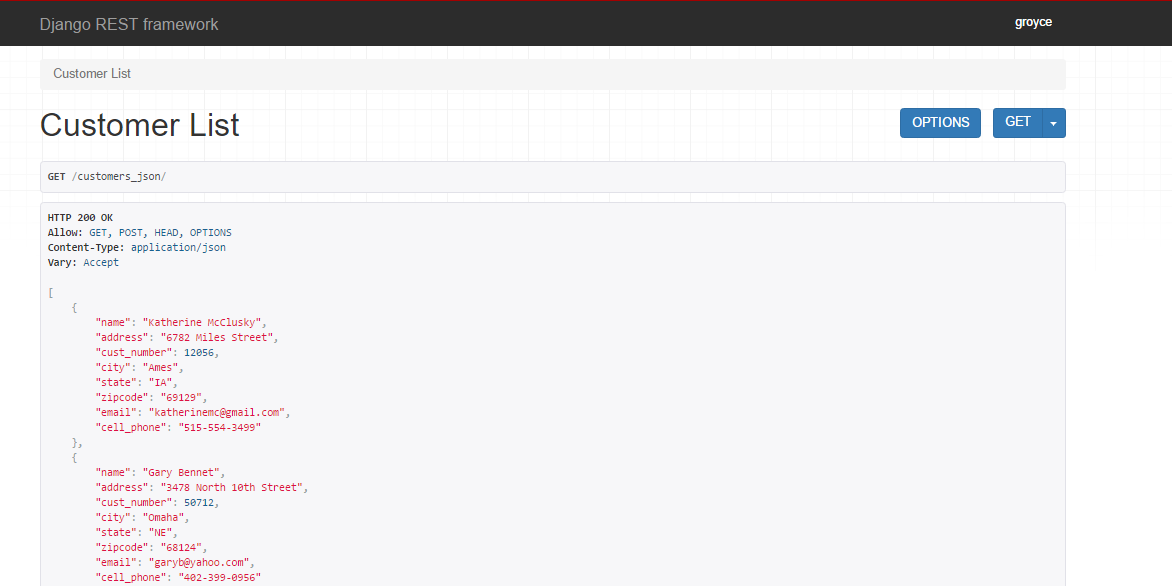
**Step 5** - The portfolio.html also needs a button to activate it. Go back to the customer\_list.html and add the following URL to the Portfolio button object:

|  |
| --- |
| <td><a href="{% url 'portfolio:portfolio' pk=customer.pk %}"  class="btn btn-primary">Portfolio</a> |

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



**Step 5** - Now it is time to create our own REST API of customer customer records in our application. It actually could be any information in our application. We are simply using customer information for convenience. These REST API’s can be used by other applications used by EFS, a Mobile App or even a Javascript framework like Angularjs. When we are done we will have a new view called customer\_json that looks like this in the browser:



You should have already installed the django rest framework. If not, you can install it now by issuing the command: **pip install djangorestframework**

We also need to tell the project that the rest framework has been installed. Add the **'**rest\_framework**'**,

’ to the app section of **settings.py.** You may have done this already in part 2.

**Step 6 -** Next we will create a serializer class which takes the data that would normally go into web pages and reformats it into JSON. Create a page in the portfolio directory where you already have the models and views. Call it **serializers.py.** Add the following to this **serializers page:**

|  |
| --- |
| from rest\_framework import serializers  from .models import Customer  class CustomerSerializer(serializers.ModelSerializer):  class Meta:  model = Customer  fields = ('name', 'address', 'cust\_number', 'city', 'state', 'zipcode', 'email', 'email', 'cell\_phone') |

**Step 7 -** Now go to the **views.py** and add the following information

|  |
| --- |
| from rest\_framework.views import APIView  from rest\_framework.response import Response  from rest\_framework import status  from .serializers import CustomerSerializer  # List at the end of the views.py  # Lists all customers  class CustomerList(APIView):  def get(self,request):  customers\_json = Customer.objects.all()  serializer = CustomerSerializer(customers\_json, many=True)  return Response(serializer.data) |

This above class based view now makes the serialized information ready be exposed to the internet if it is requested.

**Step 8 -** Even though there are no templates that are being made available to the internet, we still must add information in the portfolio/urls.py file so the information in JSON format can be provided provided when requested:

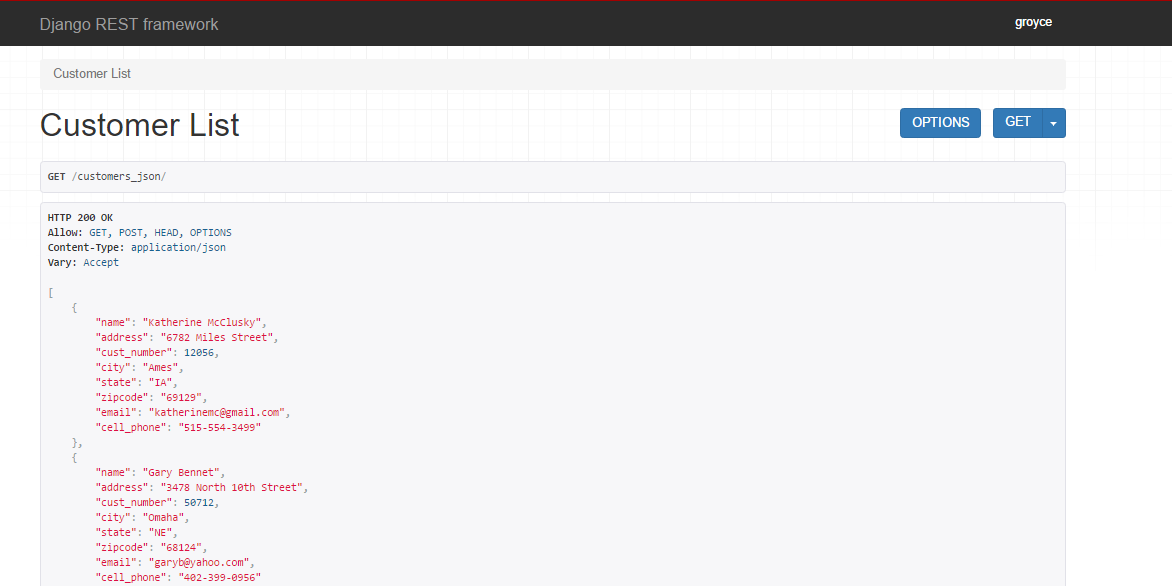
|  |
| --- |
| from django.conf.urls import url  from . import views  **from rest\_framework.urlpatterns import format\_suffix\_patterns**  urlpatterns = [  url(r'^$', views.home, name='home'),  url(r'^home/$', views.home, name='home'),  url(r'^customer/$', views.customer\_list, name='customer\_list'),  url(r'^customer/(?P<pk>\d+)/delete/$', views.customer\_delete, name='customer\_delete'),  url(r'^customer/(?P<pk>\d+)/edit/$', views.customer\_edit, name='customer\_edit'),  url(r'^investment/$', views.investment\_list, name='investment\_list'),  url(r'^investment/(?P<pk>\d+)/delete/$', views.investment\_delete, name='investment\_delete'),  url(r'^investment/(?P<pk>\d+)/edit/$', views.investment\_edit, name='investment\_edit'),  url(r'^investment/create/$', views.investment\_new, name='investment\_new'),  url(r'^stock/$', views.stock\_list, name='stock\_list'),  url(r'^stock/(?P<pk>\d+)/delete/$', views.stock\_delete, name='stock\_delete'),  url(r'^stock/(?P<pk>\d+)/edit/$', views.stock\_edit, name='stock\_edit'),  url(r'^stock/create/$', views.stock\_new, name='stock\_new'),  url(r'^customer/(?P<pk>\d+)/portfolio/$', views.portfolio, name='portfolio'),  **url(r'^customers\_json/', views.CustomerList.as\_view()),**  ]  **urlpatterns = format\_suffix\_patterns(urlpatterns)** |

Add the three lines in green to your existing **portfolio/urls.py file.**

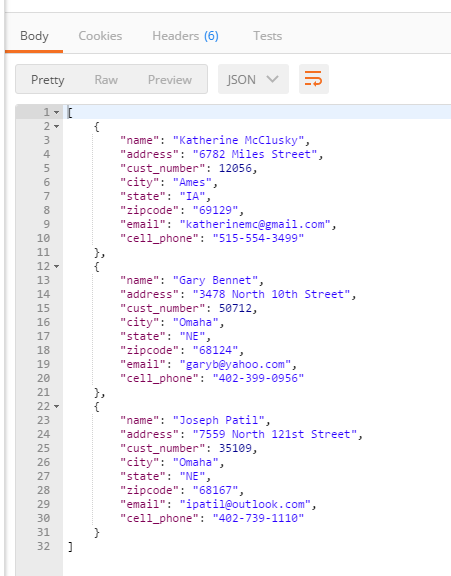
**Step 9 -** Restart your app server by redoing the command: **python manage.py runserver**

In the browser type the following URL: <http://127.0.0.1:8000/customers_json/>

The results should look like this:



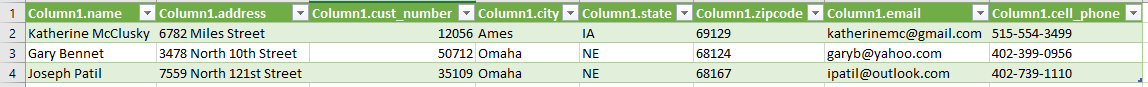
Here is a cleaner view of the same data in [Postman](https://www.getpostman.com/) tool which is great in working with RESTful API’s.



**Step 10 - Consuming our RESTful web API with Excel.**

There are a variety of RESTful API tools and applications that can consume our new API. We will consume our JSON data coming from our API using a spreadsheet. You have two options. Here is a great demo of how you would consume the JSON API using Microsoft Excel. I am using Excel 2016 but I believe older versions have the same or similar capabilities. [Click here for a video demonstrating consuming JSON API with Excel](http://roycesite.com/george/media/JSONAPI2EXCEL/JSONAPI2EXCEL.html).

The results should look something like this:



The advantage of using Excel is that you can test it on your PC or Mac with both your local application server and also when you deploy your application to Heroku or PythonAnywhere. You can also do this with Google Sheets but it only works with your deployed version of the application since Google Sheets cannot easily interact with your localhost environment.

**Step 10 Alternative - Consuming our RESTful web API with Google Sheets.**

Please note that the directions below ONLY work with a deployed app on Heroku or PythonAnywhere.

Our goal is to get an understanding of how we can consume a REST service with a JSON payload.

First, be sure that your application is working properly and providing JSON data when requested.

.

Launch a Google spreadsheet. Even if you don’t have a personal account, the college uses gmail for student accounts so you have easy access to create a Google spreadsheet.

Google does have a built-in capability for importing XML but Not JSON (Yet!). I found a custom function I used. I found it in an article called: “How to import JSON data into Google Spreadsheets in less than 5 minutes.” Here is the link to the article.<https://medium.com/@paulgambill/how-to-import-json-data-into-google-spreadsheets-in-less-than-5-minutes-a3fede1a014a>

This described a script called ImportJSON which as available at Github at this location:<https://gist.github.com/paulgambill/cacd19da95a1421d3164>

The details of this script are described in details at:

<http://blog.fastfedora.com/projects/import-json#ImportJSON>

Once you have the script loaded into your Google spreadsheet, consume the records. Then save the file as a Microsoft Excel format so you can hand in. Also, take a screenshot of your file in Google Sheets.

**Step 11.** You have likely modified EFS project you handed in in part 2 of this assignment. You now need to push your changes to your GitHub repository so you can hand it in as a part of this assignment.

**Step 12.** Also push your code and deploy your update to either Heroku or PythonAnywhere.