Lab 9 - Django Rest Framework

Dependencies

1. Virtualenv
2. Python3

Description

This lab would use Url parameters and perform limit and offset views and a 3rd party plugin django data table would be used to perform pagination.

Secondly we will introduce the concept of RESTful services through the use of the 3rd party django plugin django rest framework.

Setup

1. Create a virtual envrionment called lab9

```
$ virutalenv lab9
```

1. Download pokedex.zip, move pokedex folder and requirement.txt file into root of lab9 folder
2. Open lab9 folder as visual studio code project
3. Open terminal in lab9 and activate envionment

```
user:~/lab9$ . bin/activate
```

1. Install pip requirements

```
(lab9)user:~/lab9$ pip install -r requirements.txt
```

1. Finally cd into the django project and run the django server

```
user:~/lab9$ cd pokemon
user:~/lab9/pokemon$ python manage.py runserver
```

Lab

When you run the server you would be greeted with two views, listing and table. The goal is to have the pokemon data in pokedata.json to be presented in these views.

Firstly we need to get the pokedata from the json file into our project's database. import.py is the script that will get the job done.

Located at pokedex/pokelist/pokemon is a django model created specifically for the fields in our data set.

```
from django.db import models
from django.db.models import CharField
from django.db.models import IntegerField

class Pokemon(models.Model):
    name = CharField(max_length=100)
    hp = IntegerField()
    TYPES = (
        ("Normal", "Normal"),
        ("Fire", "Fire"),
        ("Water", "Water"),
        ("Electric", "Electric"),
        ("Grass", "Grass"),
        ("Ice", "Ice"),
        ("Fighting", "Fighting"),
        ("Poison", "Poison"),
        ("Ground", "Ground"),
        ("Flying", "Flying"),
        ("Psychic", "Psychic"),
        ("Bug", "Bug"),
        ("Ghost", "Ghost"),
        ("Dragon", "Dragon"),
        ("Dark", "Dark"),
        ("Steel", "Steel"),
        ("Fairy", "Fairy"),
    )
    type_1 = CharField(max_length = 20, choices=TYPES)
    type_2 = CharField(max_length = 20, choices=TYPES, null=True
    attack = IntegerField()
```

```
    defense = IntegerField()
    sp_atk = IntegerField()
    sp_def = IntegerField()
    total = IntegerField()
    speed = IntegerField()
```

1. Import the data

Take a look at import.py to see how the import process is done. Because of its django dependencies, import.py is a django script, that means it can only run in the django CLI envrionment as opposed to the Python CLI. To run the script enter the following command.

```
user:~/lab9/pokemon$ python manage.py shell < import.py
```

This executes the script in django's shell.

If that does not work you can open the shell manually then run the exec() command as follows.

```
user:~/lab9/pokemon$ python manage.py shell
>>> exec(open('import.py').read())
```

3. Check the data

We need to create a superuser login to view our data in the django's admin interface.

```
user:~/lab9/pokemon$ python manage.py createsuperuser
```

Then enter your admin credentials(Try to remember them nuh).

If your server is still running you can point your browser to http://localhost:8000/admin, login with your credentials and see the newly imported pokemon data.

You can also see that the views are now populated with data. The implementation would not be covered in depth in this lab but you can check the following files for details.

1. pokedex/pokelist/pokemonTable.py

2. pokedex/pokelist/views.py
3. pokedex/pokelist/templates/datatable.html
4. pokedex/pokelist/urls.py

4. Adding Django Rest Framework to project

DRF is aleady installed as it was listed in requirements.txt and added to the applicaiton. This was done by simply adding it to installed apps and a REST_FRAMEWORK configuration dictonary to settings.py.

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

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAdminUser',
    ],
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.Limit
    'PAGE_SIZE': 10
}
```

We also want to add a new set of urls provided by DRF by adding the following file to urls.py (this has been done already)

```
from django.conf.urls import url, include

from rest_framework import routers

router = routers.DefaultRouter()

urlpatterns = [
    url(r'^api-auth/', include('rest_framework.urls', namespace=
    url(r'^api/', include(router.urls)),
]
```

All of these steps are specifed in DRF's docs.

If you point your browser to http://localhost:8000/api and login with your admin credentials you can see DRF's inreactive API for user objects.

5. REST & Serializers

In very layman's terms a restful api is an appliacation wich presents data in urls for the use in other applications. Well be using DRF to expose our pokemon data in urls as JSON.

DRF requires us to define something they call Serializers which simply tells django how to display our model's data as json.

The following steps would build out our rest api

1. Serialize Pokemon model and update its package
2. pokemon/pokelist/pokemonSerializer.py

```
# Serialize Pokemon has HyperlinkedModel
from . import Pokemon
from rest_framework import serializers

class PokemonSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Pokemon
        fields = ('id', 'name', 'hp', 'type_1', 'type_2', 'attac
```

- pokemon/pokelist/ __init__.py

```
from . pokemon import Pokemon
from . pokemonSerializer import PokemonSerializer
from . pokemonTable import PokemonTable

#allows other files to import the classes above from pokelist.po
```

1. Next we want to create a ViewSet which links our objects to our serializer we also need to import PokemonSerializer in our views
2. add the following topokemon/pokelist/views.py

```
from .pokemon import PokemonSerializer
```

```
class PokemonViewSet(viewsets.ModelViewSet):
    queryset = Pokemon.objects.all()
    serializer_class = PokemonSerializer
```

1. Now lets link our new view to a url, simply uncomment line7 in pokemon/pokelist/urls.py. It would add our pokemon viewset to a pokemon route url.

```
router.register(r'pokemon', views.PokemonViewSet)
```

Now if you navigate to http://localhost:8000/api/pokemon you now have all of the pokemon data in JSON!

1. Next we'd like to create a trainer model. We create a one to one mapping for every user to trainer. Meaning that every user account is tied to only one trainer maximum.
2. edit pokedex/pokelist/trainer/trainer.py

```
# create Trainer Model which maps one to one to User
#  override __str__ method to display name property
from django.db import models
from django.contrib.auth.models import User

class Trainer(models.Model):
    name = models.CharField(max_length=30)
    user = models.OneToOneField(User)
    num_badges = models.IntegerField()

    # This selects the name attribute of the trainer for display
    def __str__(self):
        return self.name

* Add our trainer model to models.py
* pokemon/pokelist/models.py should now look like this
```python
    from django.contrib.auth.models import User
    from .pokemon import Pokemon
    from .trainer import Trainer
    # import other models here
```

1. You can't be a trainer without any pokemon so we now want to make models which show wich pokemon are in a trainer's team.
2. Create a trainerPokemon model
```

3. edit pokemon/pokelist/trainerPokemon.py

```
from django.db import models
from pokelist.models import Pokemon
from pokelist.trainer.trainer import Trainer #uncomment when Tra

#create TrainerPokemon model links to pokemon and trainer
# override __str__ method to display nickname
from django.db import models
from pokelist.models import Pokemon
from pokelist.trainer.trainer import Trainer


class TrainerPokemon(models.Model):
    nickname = models.CharField(max_length=10)
    poke_stats = models.ForeignKey(Pokemon)
    trainer = models.ForeignKey(Trainer, related_name='party')

    def __str__(self):
        return self.nickname

    class Meta:
        ordering = ('nickname',)
```

1. Now we need to serialize these models as well

2. pokemon\pokelist\trainer\trainerSerializer.py

```
# Serialize Trainer
from . import Trainer # uncomment after Trainer model is made

from . import Trainer
from pokelist.trainerPokemon import TrainerPokemon, TrainerPokem
from rest_framework import serializers


class TrainerSerializer(serializers.HyperlinkedModelSerializer):
    # party = TrainerPokemonSerializer(many=True, read_only=True
    # party = serializers.SlugRelatedField(
    #     many=True,
    #     read_only=True,
    #     slug_field='nickname'
    # )
    party = serializers.StringRelatedField(many=True)
```

```
    class Meta:
        model = Trainer
        fields = ('id', 'name', 'num_badges', 'user', 'party')
```

- pokemon/pokelist/trainerPokemon/trainerPokemonSerializer.py

```
# create serializer
# add a new readonly field to serializer which diplays the id fr
from . import TrainerPokemon
from rest_framework import serializers
from pokelist.pokemon import PokemonSerializer


class TrainerPokemonSerializer(serializers.HyperlinkedModelSeria

    # poke_stats = PokemonSerializer()
    # custom field, read only based on another attribute in mode
    poke_id = serializers.SerializerMethodField()

    class Meta:
        model = TrainerPokemon
        fields = ('id', 'nickname', 'poke_stats', 'trainer', 'po

    def get_poke_id(self, obj):
        return obj.poke_stats.id
```

- We need to update our packages for imports to work
- Remove the comments from
    i. pokelist/pokemon/__init__.py
    ii. pokelist/trainer/__init__.py
    iii. pokelist/trainerPokemon/__init__.py
    iv. pokelist/trainer/__init__.py
- Now our packages are setup we would like to add our new models to the app and migrate them to the database
- update pokelist/models.py to look like this

```
from django.contrib.auth.models import User
from .pokemon import Pokemon
```

```
from .trainer import Trainer
from .trainerPokemon import TrainerPokemon
```

- run the following commands

```
user:~/lab9/pokemon$ python manage.py makemigrations
user:~/lab9/pokemon$ python manage.py migrate
```

1. Were almost there, we need to create views which would send our data to our serializers, luckily these all look the same.

2. update pokelist/views.py

```
from django.shortcuts import render
from rest_framework import viewsets
from django.contrib.auth.models import User
from .serializers import UserSerializer
from .pokemon import PokemonTable, Pokemon, PokemonSerializer
from .trainer import Trainer, TrainerSerializer
from .trainerPokemon import TrainerPokemon, TrainerPokemonSerial

# Create your views here.

def get_pokemon(request):
    pokemon = PokemonTable(Pokemon.objects.all())
    return render(request, 'datatable.html', {
        'pokemon':pokemon
    })

def get_pokemon_range(request, offset, limit):
    offset = int(offset)
    limit = int(limit)
    print(limit)
    pokemon = Pokemon.objects.all()[offset:limit]
    return render(request, 'listing.html', {
        'pokemon':pokemon
    })

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer

# make ModelViewSets for Pokemon Trainer and TrainerPokemon
class PokemonViewSet(viewsets.ModelViewSet):
    queryset = Pokemon.objects.all()
    serializer_class = PokemonSerializer
```

```
class TrainerViewSet(viewsets.ModelViewSet):
    queryset = Trainer.objects.all()
    serializer_class = TrainerSerializer

class TrainerPokemonViewSet(viewsets.ModelViewSet):
    queryset = TrainerPokemon.objects.all()
    serializer_class = TrainerPokemonSerializer
```

1. Finally we want to add the routes to our views.
2. Update pokelist/urls.py (uncomment lines 7 and 8)

```
from django.conf.urls import url, include
from . import views
from rest_framework import routers

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'pokemon', views.PokemonViewSet)
router.register(r'trainer', views.TrainerViewSet)
router.register(r'trainer_pkmn', views.TrainerPokemonViewSet)

urlpatterns = [
    url(r'^$', views.get_pokemon, name="poke_table"),
    url(r'^listing/offset/(?P<offset>\d+)/limit/(?P<limit>\d+)/'
    url(r'^api-auth/', include('rest_framework.urls', namespace=
    url(r'^api/', include(router.urls)),
]
```

Now we finally have a browseable api. Point your browser to http://localhost:8000/api login and interact with your app data! You can create users, then trainers then add pokemon to a trainer's team.