Create REST API With Django and Neo4j Database Using Django_nemodel

In this tutorial, I'll show you how to create a REST API using Django and Neo4j database through a simple example. In order to create this API, we need to use an Object Graph Mapper (OGM) to request the graph database, that's why we will use django_neomodel which is a Django integration of the awesome OGM neomodel.

What will we use ?

- Django : **Django** is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.
- Neo4j : Neo4j is a NoSQL graph database management system (DBMS).
- Django_neomodel : a module that allows you to use the neo4j graph database with Django using neomodel.

To do list

1. Installing dependencies.
2. Create and set up the project.
3. Create the graph structure that Django_neomodel ORM will manage.
4. Create and manage some objects using Django shell.
5. Create views and urls patterns.
6. Request our API.

1. Installing dependencies

Before jumping into the installation, you should have already downloaded python. You can check the actual version for python by this command:

```
$ python --version
```

The first step is to create a virtual environment that will will englobe all the requirements. So let's start by installing virtualenv and activate the virtual environment:

```
$ pip install virtualenv
$ virtualenv env
$ env\Scripts\activate
```

Then we install django and django_neomodel:

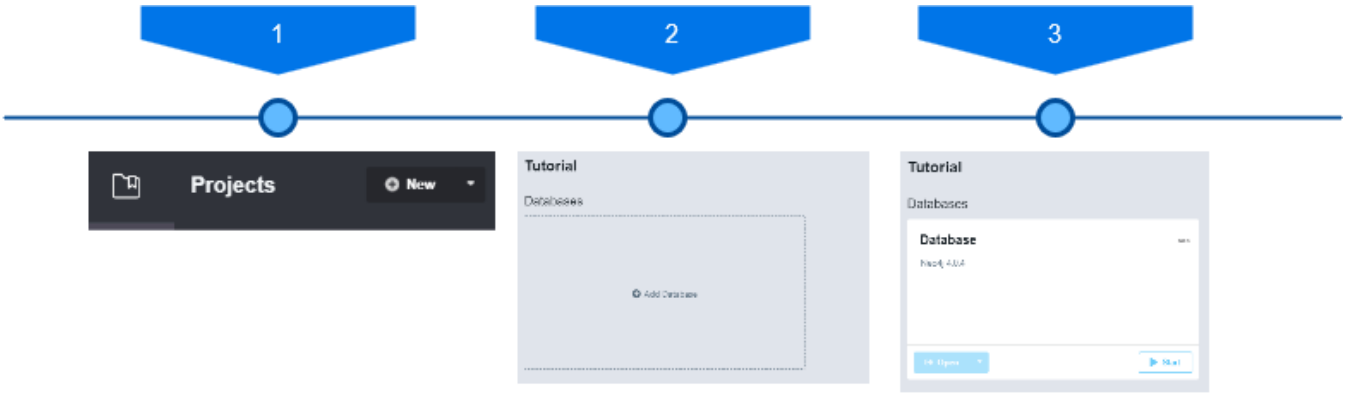
```
$ pip install Django
$ pip install django_neomodel
```

Finally, we install Neo4j. For that i recommend you to use the official documentation of Neo4j. You can find it here.

We are ready now to start creating our django API.

2. Create and set up the project

First of all, we start by creating a neo4j project named "Tutorial". Then we create a graph database inside it. (look at the figure below)



Create Neo4j database process

Now, it's time to create django project. For that, we use this command:

```
$ django-admin startproject myproject
```

Then, according to the django project structure we must create an application inside this project:

```
$ cd myproject
$ python manage.py startapp myapi
```

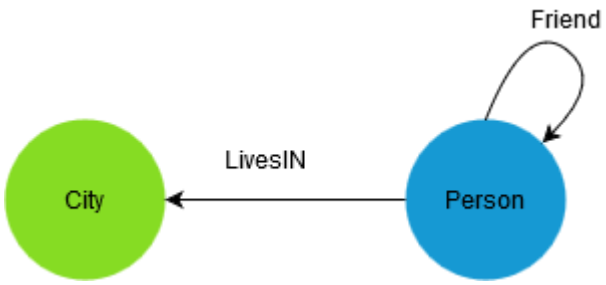
After that, we need to register our application in myproject/settings.py file so Django can recognize this new application. We must also register django_neomodel and set up the connexion to our neo4j database.

```
# Application definition

INSTALLED_APPS = [
    # django.contrib.auth etc
    'myapi.apps.MyapiConfig',
    'django_neomodel'
]

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases


NEOMODEL_NEO4J_BOLT_URL =
os.environ.get('NEO4J_BOLT_URL','bolt://username:password@localhos
t:7687')
```

```
# you are free to add this configurations
NEOMODEL_SIGNALS = True
NEOMODEL_FORCE_TIMEZONE = False
NEOMODEL_ENCRYPTED_CONNECTION = True
NEOMODEL_MAX_POOL_SIZE = 50
```

3. Create the graph structure that Django_neomodel ORM will manage

Our example is very simple. We will have two entities Person and City and two relationships between them(LivesIn and Friend). see the structure below:



Graph structure of our example

Let's define our graph structure inside myapi/models.py file like that :

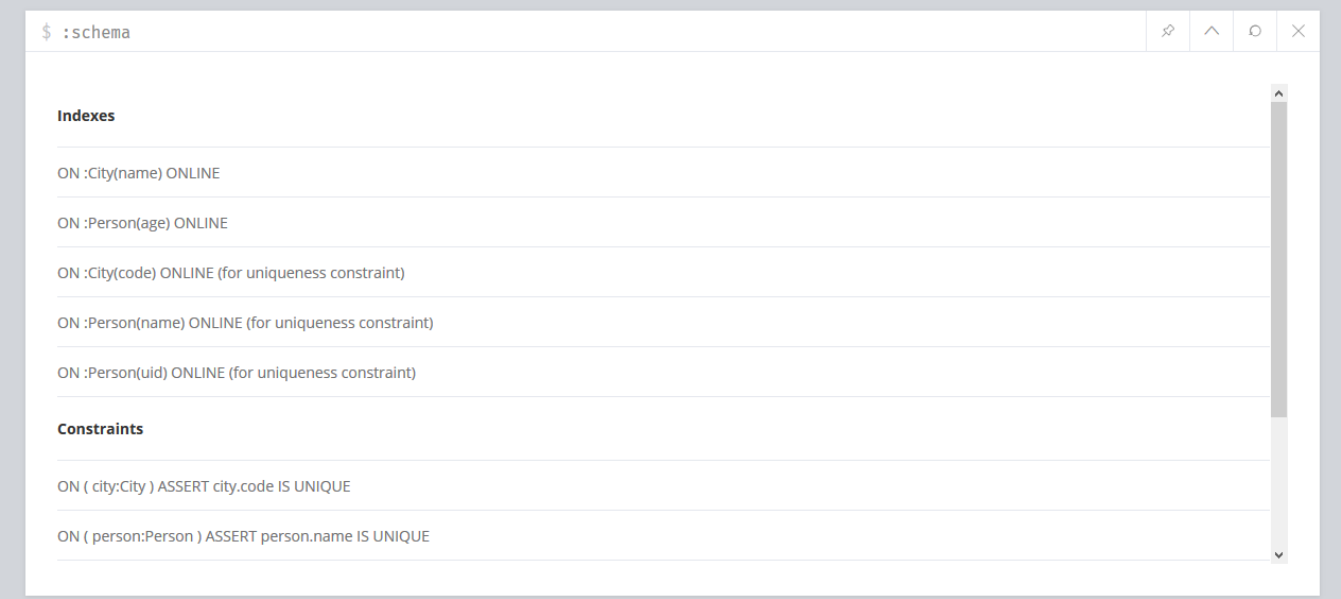
```
from neomodel import StructuredNode, StringProperty,
IntegerProperty,UniqueIdProperty, RelationshipTo

# Create your models here.


class City(StructuredNode):
    code = StringProperty(unique_index=True, required=True)
    name = StringProperty(index=True, default="city")

class Person(StructuredNode):
    uid = UniqueIdProperty()
    name = StringProperty(unique_index=True)
    age = IntegerProperty(index=True, default=0)

    # Relations :
    city = RelationshipTo(City, 'LIVES_IN')
    friends = RelationshipTo('Person','FRIEND')
```

You can find more details about how to define node entities and relationships in the neomodel official documentation.

After that, we must create constraints and indexes for our labels to ensure that everything is right by typing this command :

```
$ python manage.py install_labels
```

You can ensure that constraints and were applied in the Neo4j browser like this :



Checking our database schema

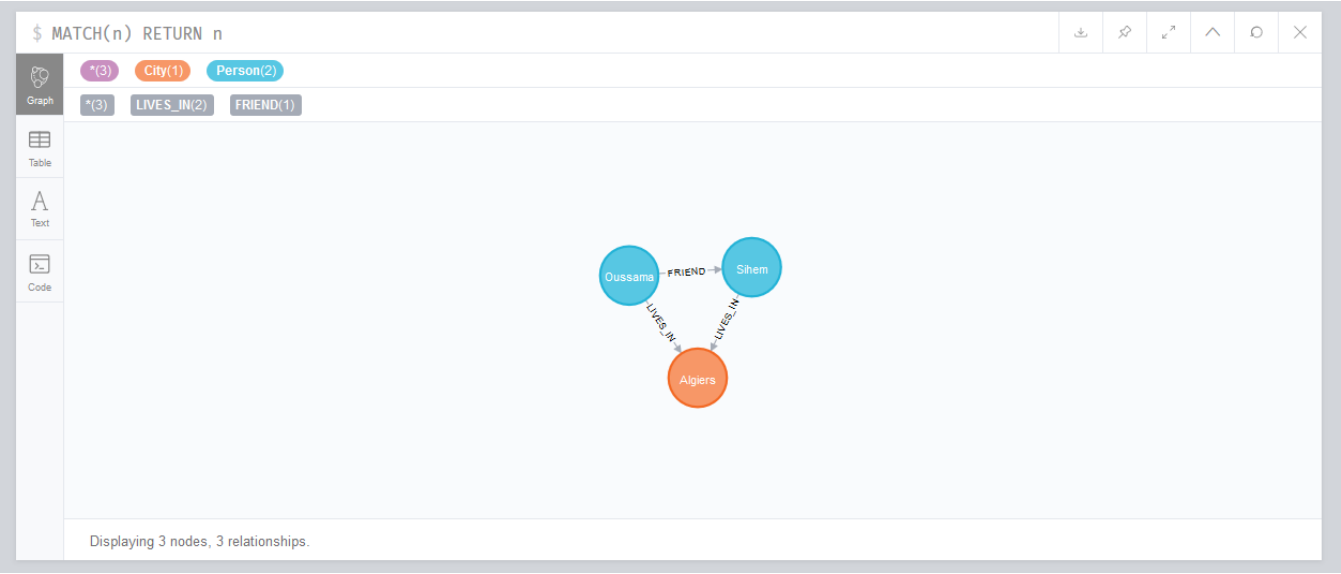
4. Create and manage some objects using Django shell

Now, let's ensure that everything is all right by populating our database by some persons and cities. For that, we will use simply Django shell:

```
$ python manage.py shell
Type "help", "copyright", "credits" or "license" for more
information.
(InteractiveConsole)
>> from myapi.models import *
>> all_persons = Person.nodes.all()
[]
>> all_cities = City.nodes.all()
[]
```

```
>> algiers = City(code="ALG",name="Algiers")
>> print(algiers)
{code: "ALG", name: "Algiers"}
>> algiers.save()
<City: {code: "ALG", name: "Algiers", id: 0}>
>> sihem = Person(name='Sihem', age=24)
>> print(sihem)
{'uid': '56329dc', 'name': 'Sihem', 'age': 24}
>> sihem.save()
<Person: {'uid': '56329dc', 'name': 'Sihem', 'age': 24, 'id': 1}>
>> sihem.city.connect(algiers)
True
>> if sihem.city.is_connected(algiers):
...     print("sihem lives in Algeirs")
...
sihem lives in Algeirs
>> oussama= Person(name='Oussama', age=22)
>> oussama.save()
<Person: {'uid': 'c139f04', 'name': 'Oussama', 'age': 22, 'id':
21}>
>> oussama.city.connect(algiers)
True
>> oussama.friends.connect(sihem)
True
```

After that, you can check our objects in Neo4j database by using Cypher queries.



Showing all created nodes and relationships

You can create more objects using either Neo4j browser or Django shell.

5. Create views and urls patterns

Arriving at this stage, it's time to create our person and city views. So let's delete myapi/views file and use instead of it a new package named myapi/views. We wil have the following application structure:

```
myapi
|_migrations
|_models
| |_ __init__.py
| |_person.py
| |_city.py
|_views
| |_ __init__.py
| |_ person.py
| |_ city.py
| |_ connectors.py
|_ ......
```

Now, lets define the content of each file of mayapi/views package :

- ***person.py :*** this file contains the following methods ***:***

1. personDetails : in which we will put our CRUD operations (create new person, update, delete and get details of a person).
2. getAllPersons : which will render a list of existing Person instances.

```
from django.http import JsonResponse
from myapi.models import Person
from django.views.decorators.csrf import csrf_exempt
import json


def getAllPersons(request):
    if request.method == 'GET':
        try:
            persons = Person.nodes.all()
            response = []
            for person in persons :
                obj = {
                    "uid": person.uid,
                    "name": person.name,
                    "age": person.age,
                }
                response.append(obj)
            return JsonResponse(response, safe=False)
        except:
```

```
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
@csrf_exempt
def personDetails(request):
    if request.method == 'GET':
        # get one person by name
        name = request.GET.get('name', ' ')
        try:
            person = Person.nodes.get(name=name)
            response = {
                "uid": person.uid,
                "name": person.name,
                "age": person.age,
            }
            return JsonResponse(response, safe=False)
        except :
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

    if request.method == 'POST':
        # create one person
        json_data = json.loads(request.body)
        name = json_data['name']
        age = int(json_data['age'])
        try:
            person = Person(name=name, age=age)
            person.save()
            response = {
                "uid": person.uid,
            }
            return JsonResponse(response)
        except :
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

    if request.method == 'PUT':
        # update one person
        json_data = json.loads(request.body)
        name = json_data['name']
        age = int(json_data['age'])
        uid = json_data['uid']
        try:
            person = Person.nodes.get(uid=uid)
            person.name = name
            person.age = age
            person.save()
            response = {
                "uid": person.uid,
                "name": person.name,
                "age": person.age,
            }
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
```

```
    if request.method == 'DELETE':
        # delete one person
        json_data = json.loads(request.body)
        uid = json_data['uid']
        try:
            person = Person.nodes.get(uid=uid)
            person.delete()
            response = {"success": "Person deleted"}
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
```

- ***city.py :*** this file contains the following methods :

1. cityDetails : in which we will put our CRUD operations (create new city, update, delete and get details of a city).
2. getAllCities : which will render a list of existing City instances.

```
from django.http import JsonResponse
from myapi.models import City
from django.views.decorators.csrf import csrf_exempt
import json


def getAllCities(request):
    if request.method == 'GET':
        try:
            cities = City.nodes.all()
            response = []
            for city in cities:
                obj = {
                    "code": city.code,
                    "name": city.name,
                }
                response.append(obj)
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
@csrf_exempt
def cityDetails(request):
    if request.method == 'GET':
        # get one city by name
        name = request.GET.get('name', ' ')
        try:
            city = City.nodes.get(name=name)
            response = {
```

```
                "code": city.code,
                "name": city.name,
            }
            return JsonResponse(response, safe=False)
        except :
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

    if request.method == 'POST':
        # create one city
        json_data = json.loads(request.body)
        name = json_data['name']
        code = json_data['code']
        try:
            city = City(name=name, code=code)
            city.save()
            response = {
                "code": city.code,
            }
            return JsonResponse(response)
        except :
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

    if request.method == 'PUT':
        # update one city
        json_data = json.loads(request.body)
        name = json_data['name']
        code = json_data['code']
        try:
            city = City.nodes.get(code=code)
            city.name = name
            city.save()
            response = {
                "code": city.code,
                "name": city.name,
            }
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

    if request.method == 'DELETE':
        # delete one city
        json_data = json.loads(request.body)
        code = json_data['code']
        try:
            city = City.nodes.get(code=code)
            city.delete()
            response = {"success": "City deleted"}
            return JsonResponse(response)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
```

- ***connectors.py :*** this file contains the following methods :

1. connectPaC : which will create a connexion between a person and a city.
2. connectPaP: which will create a connexion between two persons.

```
from django.http import JsonResponse
from myapi.models import *
from django.views.decorators.csrf import csrf_exempt
import json


@csrf_exempt
def connectPaC(request):
    if request.method == 'PUT':
        json_data = json.loads(request.body)
        uid = json_data['uid']
        code = json_data['code']
        try:
            person = Person.nodes.get(uid=uid)
            city = City.nodes.get(code=code)
            res = person.city.connect(city)
            response = {"result": res}
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)

@csrf_exempt
def connectPaP(request):
    if request.method == 'PUT':
        json_data = json.loads(request.body)
        uid1 = json_data['uid1']
        uid2 = json_data['uid2']
        try:
            person1 = Person.nodes.get(uid=uid1)
            person2 = Person.nodes.get(uid=uid2)
            res = person1.friends.connect(person2)
            response = {"result": res}
            return JsonResponse(response, safe=False)
        except:
            response = {"error": "Error occurred"}
            return JsonResponse(response, safe=False)
```

- __init__.py : this file makes **Python** treat myapi/views directory as one module.

```
from myapi.views.person import *
from myapi.views.city import *
from myapi.views.connectors import *
```

After that, we create new file inside myapi/ directory called urls in which we will register our list routes URLs to our views.

```
from django.conf.urls import url
from myapi.views import *
from django.urls import path
urlpatterns = [
    path('person',personDetails),
    path('getAllPersons',getAllPersons),
    path('city',cityDetails),
    path('getAllCities',getAllCities),
    path('connectPaC',connectPaC),
    path('connectPaP',connectPaP)
]
```

Then, we must register this list routes URLs inside myproject/urls.py file like that :

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

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

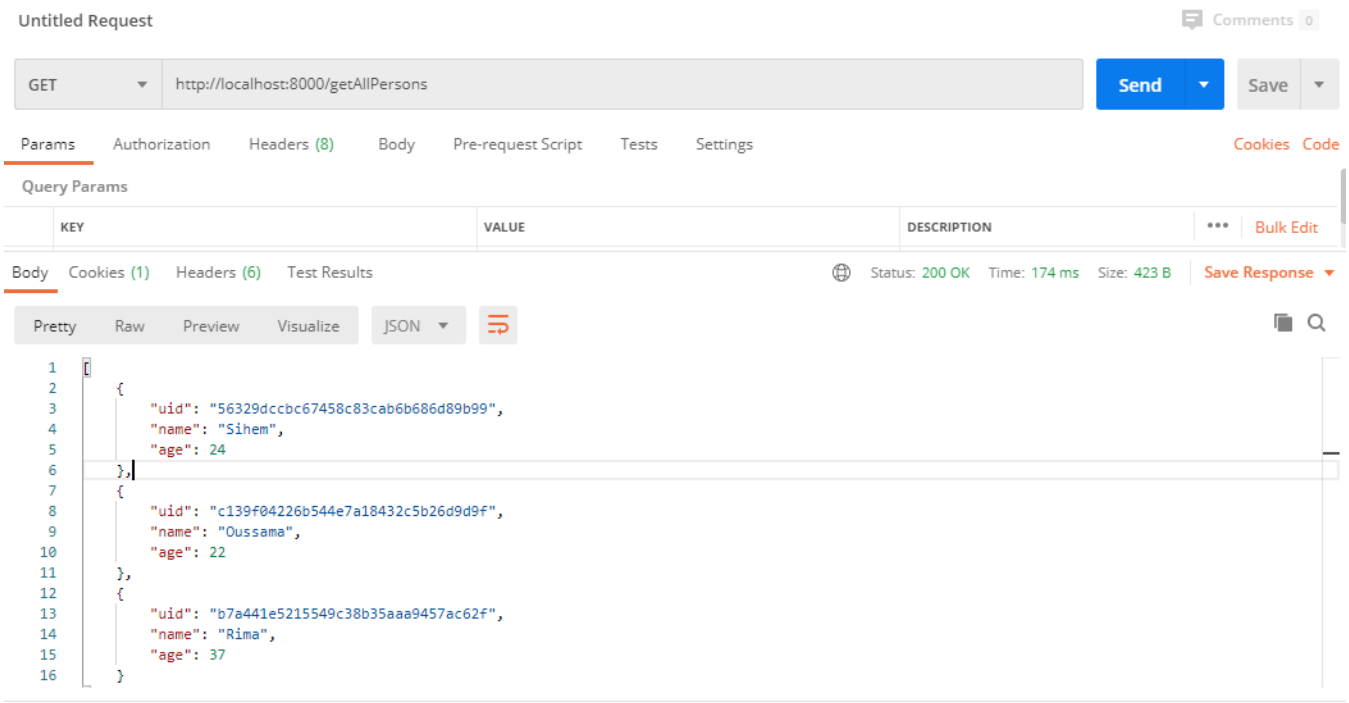
6. Request our API

Finally, we will request our API to ensure that everything works correctly. so let's start our API by runing this command :

```
$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
```

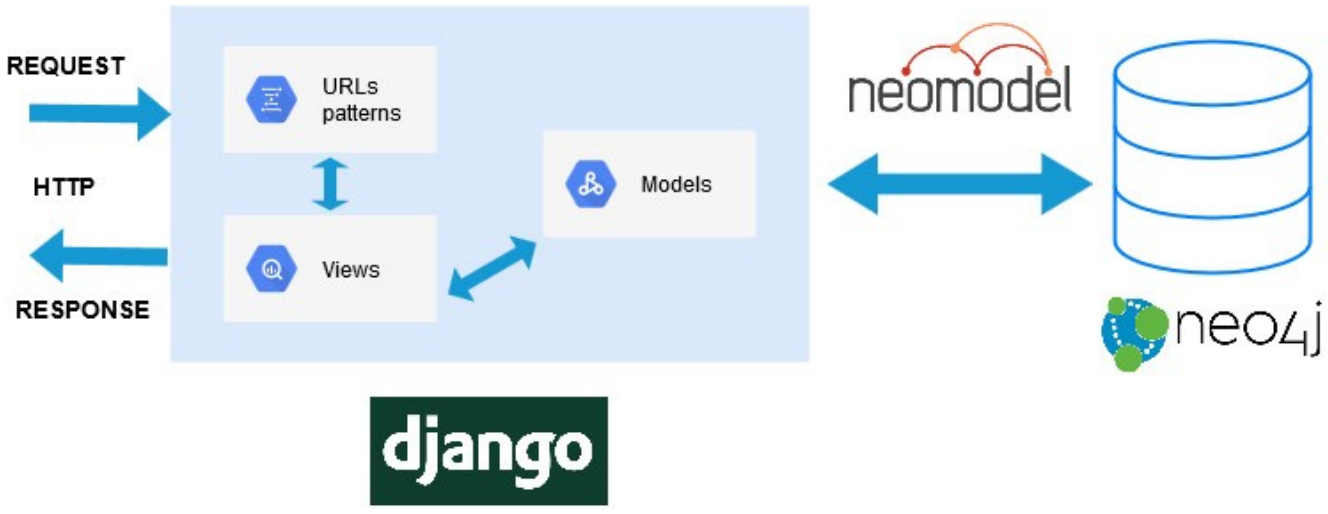
```
System check identified no issues (0 silenced).
July 29, 2020 - 16:48:13
Django version 3.0.8, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
```

To request the API, i used Postman which simplify making CRUD requests.



List of all persons

At the end, we can illustrate the architeture of our API like that :



Architecture of our API

Thank you for reading, if you have any questions or remarks please do not hesitate to leave a comment below

Here is the repository of the whole project in this github link :

SihemBouhenniche/NeomodelAPI

REST API created by Django and Neo4j. Contribute to SihemBouhenniche/NeomodelAPI development by creating an…

github.com