Django tutorial

Contents

1 Prerequisite **2**

2 Installation **2**

3 Project creation **3**

4 Launch the webpage **4**

5 Let's create an Application **5**

6 Django_neomodel **8**

7 Creating a simple model **11**

1 Prerequisite

1. Python >= 3.5
2. Pip == 3
3. Neo4j >= 4.2

2 Installation

- Its better to create a virtual environment before installing django.
- Create the virtual environment with the following command

```
$ python3 -m venv env
$ source env/bin/activate
```

- Virtual environment env (in the screenshot it has named as dj) is created and activated, now install django as shown in Figure 1

```
$ pip install django
```

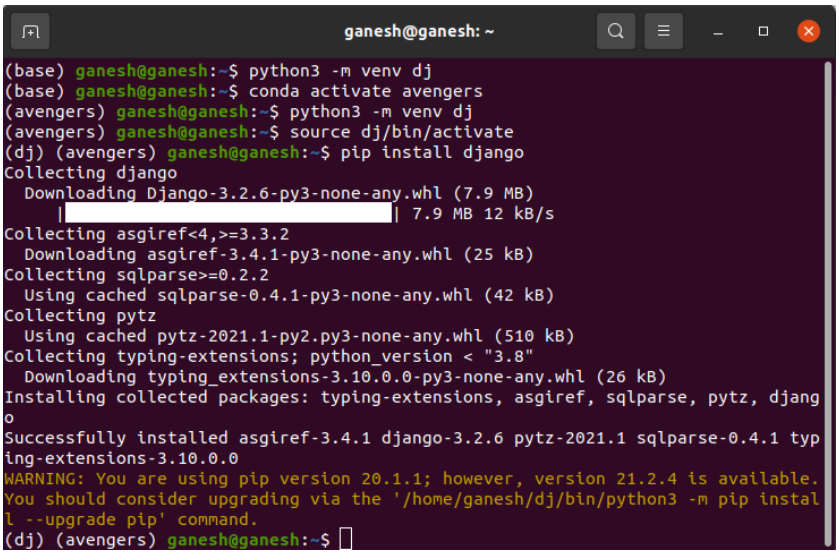


Figure 1: *Django installation using pip*

- To verify the installation type the following command this should display the version of the django

```
$ python -m django --version
```

3 Project creation

- Create a project in django using the following command

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

- After executing the command a folder is created that can be navigated and visualized using tree (should have been installed [not necessary]). Refer Figure 2

```
$ cd myproject
$ tree
```

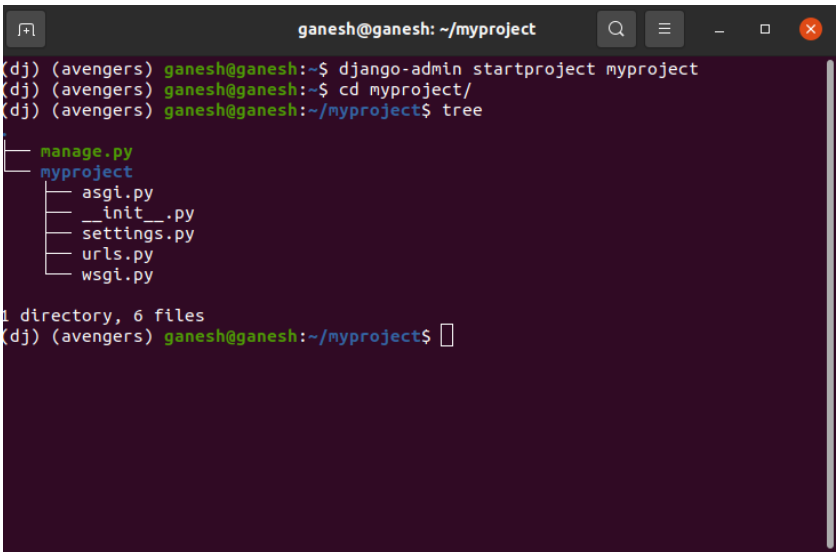


Figure 2: *Django project creation*

- The folder contains manage.py and another folder of the project name itself.

- **Manage.py** used to execute the command line commands which is unedited majorly.
- The myproject folder consist of 5 files namely asgi.py, __init__.py, settings.py, urls.py, wsgi.py.
- **asgi.py** supply an application callable which the application server uses to communicate with your code
- **__init__.py** is an empty file that tells the python that the project is a python package.
- **settings.py** used to do configuration for the project that contains security key, debug mode, database settings.
- **urls.py** used to map certain urls with the project that consists of default url to the admin.
- **wsgi.py** is python web application where the web-server communicate.

4 Launch the webpage

- To launch the webpage run the following command as shown in Figure 3

```
$ python3 manage.py runserver
```

- Do not terminate the command, the server should run to visualize the webpage in the browser.
- Open the browser and paste the url "http://127.0.0.1:8000/" to open the webpage.
- The initial webpage looks as presented in Figure 4.

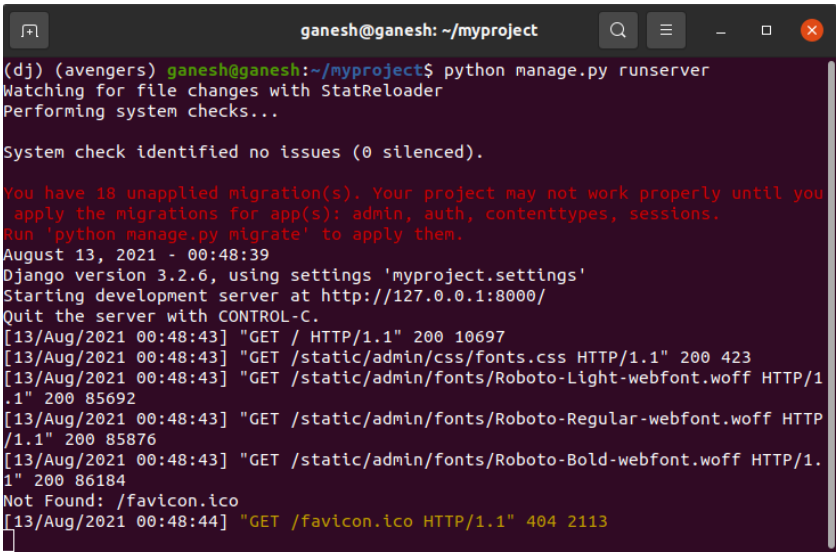


Figure 3: *Launching the webpage*

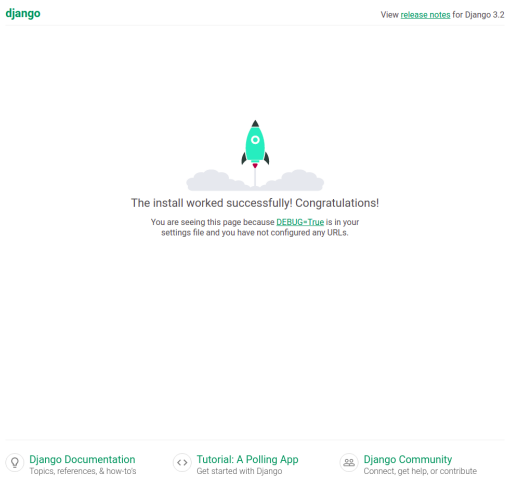


Figure 4: *Viewing the webpage*

5 Let's create an Application

- To create application the following command is used (myapp is the name for the application and the preceding is the command to create the app)

```
$ python3 manage.py startapp myapp
```

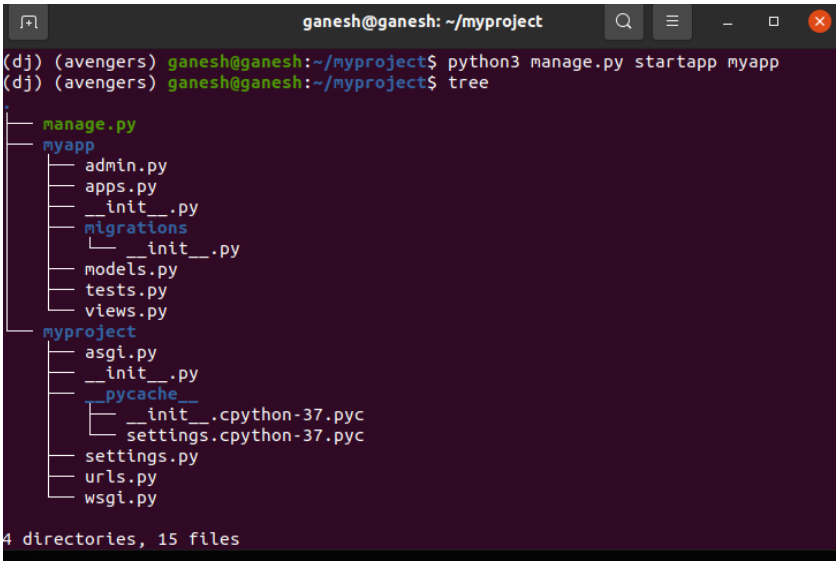


Figure 5: *Creation of the app*

- Figure 5 shows the tree structure after creating the app where the app is shown along with the predefined created files.
- The application is to create a page that displays a heading of the page. To do this open the project is Vs-code or other code editor.
  - Open the views.py from the myapp directory. Type the following

```
from django.shortcuts import render
from django.http import HttpResponse

def home(request):
    return HttpResponse('<h1> Myapp Home </h1>')

def about(request):
    return HttpResponse('<h1> Myapp About </h1>')
```

  - The above program consists of the content that has to be displayed in the page.
  - Now create a urls.py under myapp (Figure 6) directory similar to the one in myproject (Figure 5)

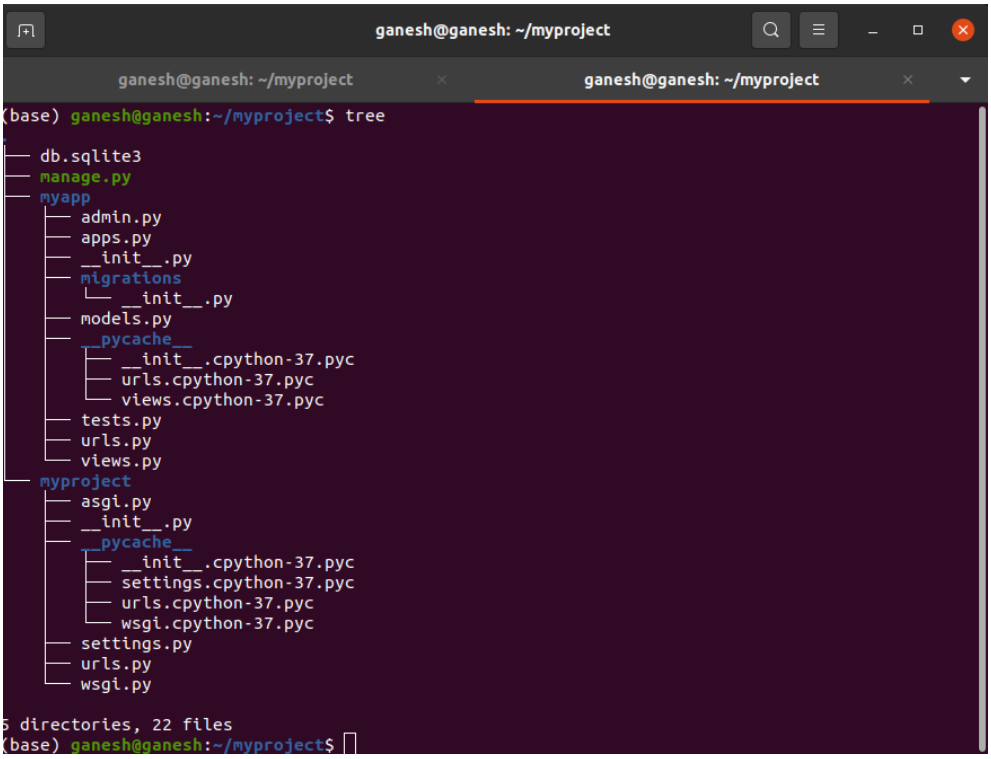


Figure 6: *Urls.py is created under myapp*

– In the new urls.py type the following

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

urlpatterns = [
    path('', views.home, name='Myapp-Home'),
    path('about/', views.about, name='Myapp-about'),

]
```

- Modify the old urls.py as shown below

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

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

- When the server is ran using the below command

```
$ python3 manage.py runserver
```

- It displays an error page having admin and Myapp as written in the old urls.py file.The following URL "http://localhost:8000/Myapp/" should make the page as shown in Figure 7

Figure 7: *Output of myapp*

- Therefore once the serve is ran, it looks for the URL to direct since there is no main URL it waits for the trigger either admin/ or Myapp/ should be navigated manually. Once Myapp is included in the URL along with the localhost this redirects to url.py in the app which calls the home function from the view.

6 Django_neomodel

- Install django_neomodel using the following command

```
$ pip3 install django_neomodel
```

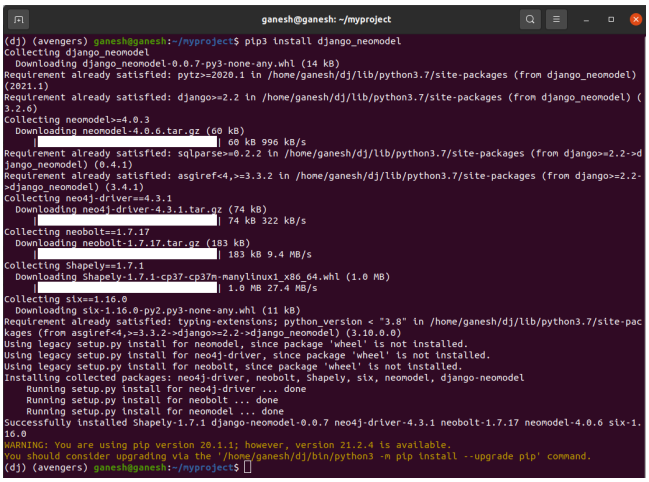
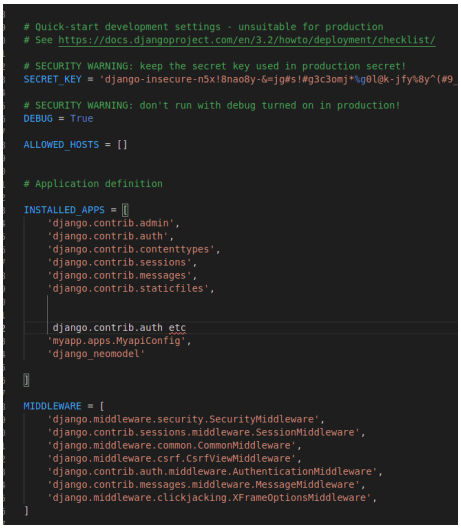


Figure 8: *Django_neomodel installation using pip*

- The project and application is already being created. Therefore, 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 connection to our neo4j database.
- Open myproject/settings.py and add the following
  - Under the application definition in the installed_apps add the following lines

```
# django.contrib.auth etc
'myapp.apps.MyappConfig',
'django_neomodel'
```

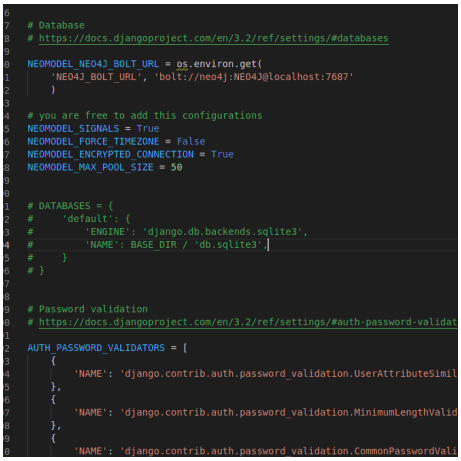


```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-n5x!8nao8y-&=jg#s!#g3c3omj*%g0l@k-jfy%8y^(#9

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

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


    django.contrib.auth etc
    'myapp.apps.MyapiConfig',
    'django_neomodel'

]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Figure 9: *Application definition*

- Under the database add the following lines and comment the existing lines mentioned with sqlite. If "os" is not imported, import it at the start

```
NEOMODEL_NEO4J_BOLT_URL = os.environ.get(
'NEO4J_BOLT_URL', 'bolt://neo4j:NEO4J@localhost:7687'
)

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



```
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

NEOMODEL_NEO4J_BOLT_URL = os.environ.get(
    'NEO4J_BOLT_URL', 'bolt://neo4j:NEO4J@localhost:7687'
    )

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


# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': BASE_DIR / 'db.sqlite3',
#     }
# }


# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validat

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimil
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValid
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordVali
```

Figure 10: *Database*

7 Creating a simple model

- Will have two entities Person and City and two relationships between them(LivesIn and Friend)
- Open myapp/models.py and type the following

```python
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')
```

- To verify that everything works correctly create constraints and indexes by typing the following command

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

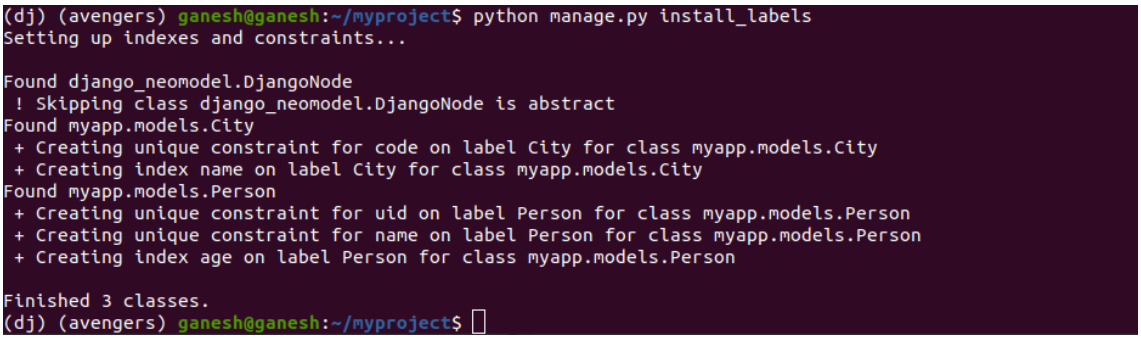


Figure 11: *Creating constraints*

- After creating constraints now its possible to create nodes from django shell
- To activate the shell type the following commands in the terminal

```
$ python manage.py shell
```

- The above commands open the shell, type the following code line by line as shown in the Figure 12

```
from myapp.models import *

all_persons = Person.nodes.all()
all_cities = City.nodes.all()
Bonn = City(code="BN", name="Bonn")
print(Bonn)
Bonn.save()
Ganesh = Person(name="Ganesh", age=27)
print(Ganesh)
Ganesh.save()
```

```python
Ganesh.city.connect(Bonn)
if Ganesh.city.is_connected(Bonn):
    print("Ganesh lives in Bonn")
tony = Person(name="tony", age=25)
tony.save()
tony.city.connect(Bonn)
tony.friends.connect(Ganesh)
```

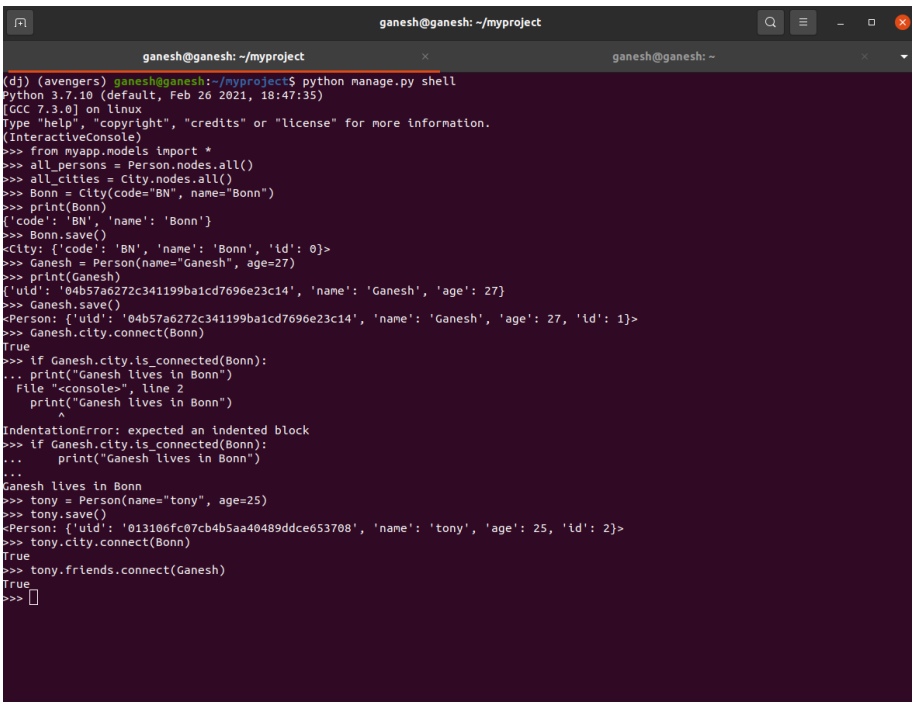


Figure 12: *Creating nodes from django shell*

- In parallel the nodes are created in the Neo4j database that has been linked to django that is shown in the below Figure 13

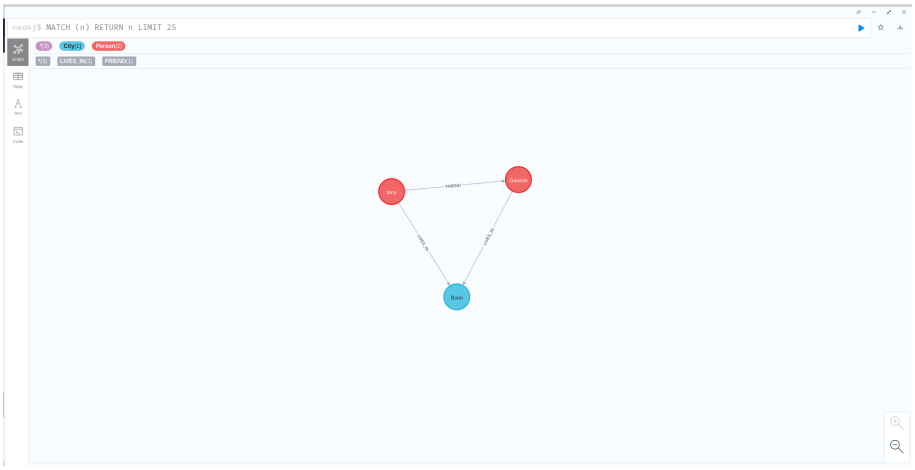


Figure 13: *Created nodes in Neo4j*