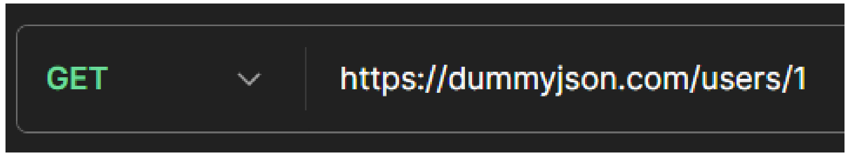
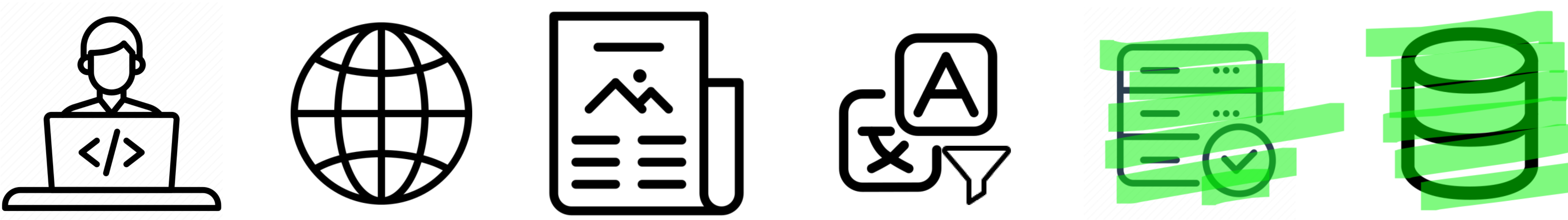
@fpftech.educacional

FPFtech
Escola Tecnológica

FPFtech
Escola Tecnológica

Ciclo de Requests


GET https://dummyjson.com/users/1



request

endpoint
urls

viewsets

Serializers
JSON=Django

models

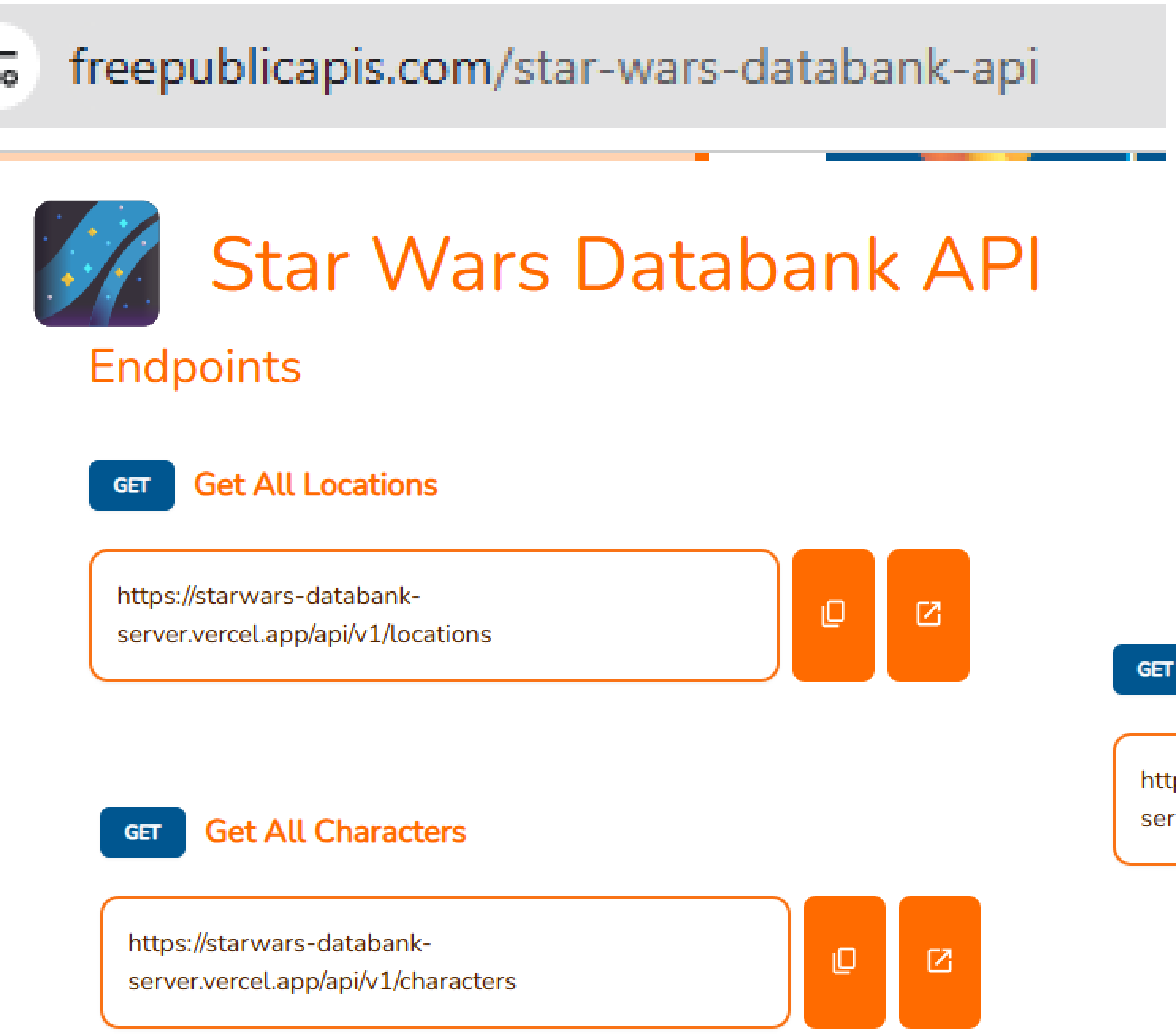
database

```python
from rest_framework import serializers
from .models import MockUser

class MockUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = MockUser
        fields = ['id', 'username', 'email']
```

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import MockUserViewSet

router = DefaultRouter()
router.register(r'mock-users', MockUserViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

```python
class MockUserViewSet(viewsets.ModelViewSet):
    queryset = MockUser.objects.all()
    serializer_class = MockUserSerializer
```

```python
from django.db import models

class MockUser(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

    def __str__(self):
        return self.username
```



freepublicapis.com/star-wars-databank-api

Star Wars Databank API

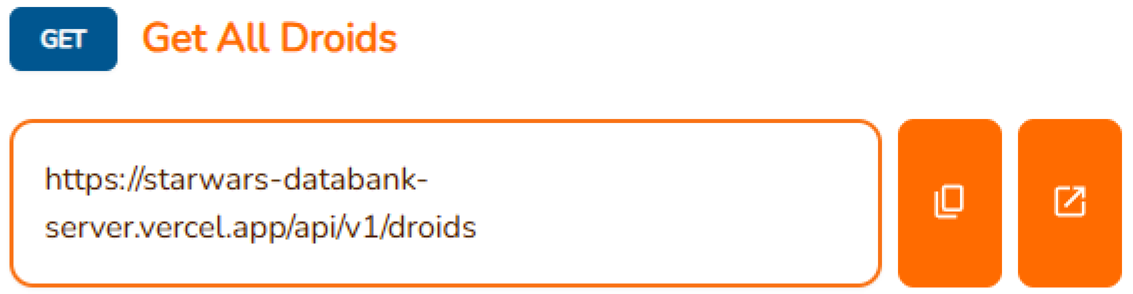
Endpoints

GET Get All Locations

https://starwars-databank-server.vercel.app/api/v1/locations

GET Get All Characters

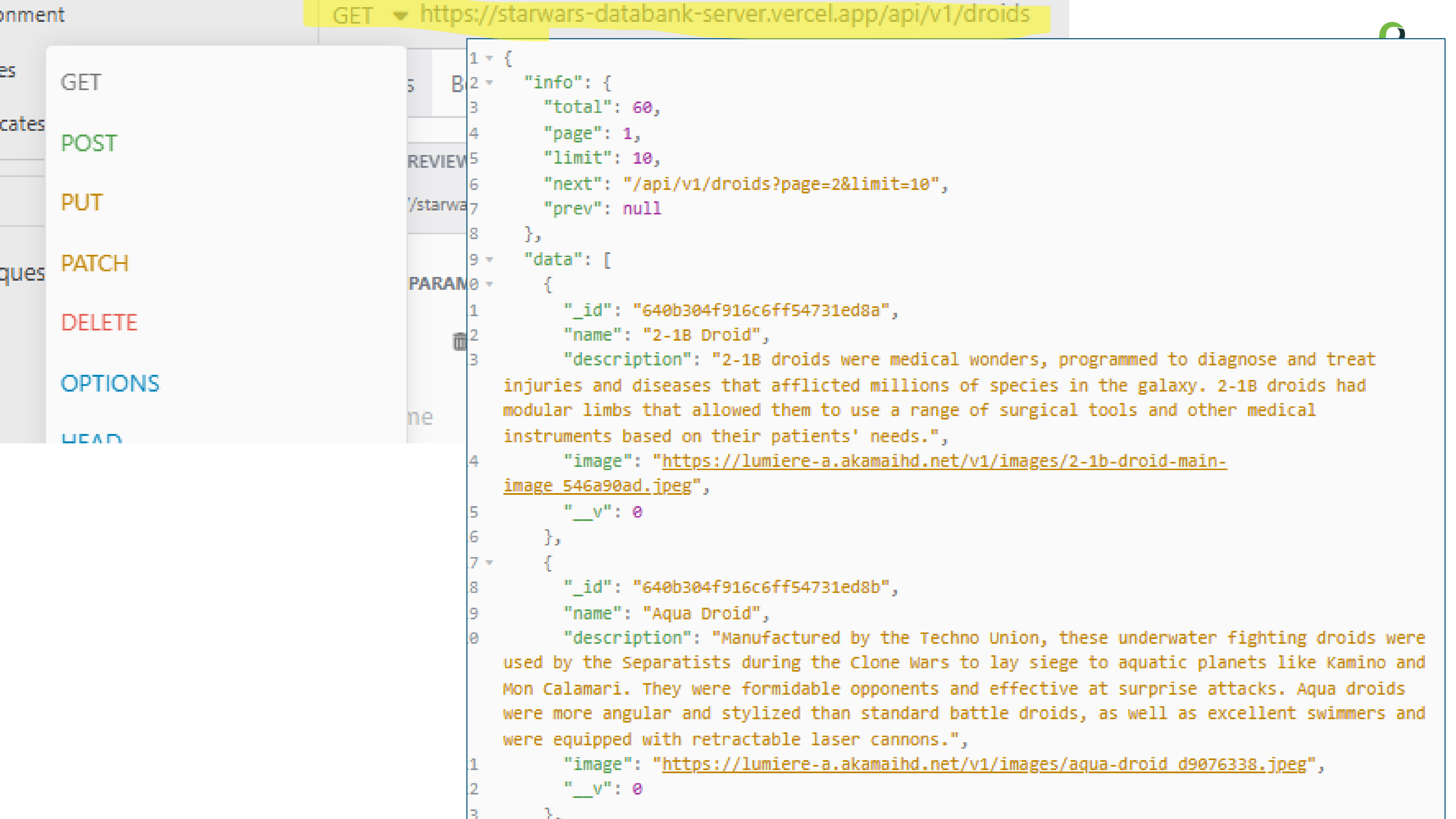
https://starwars-databank-server.vercel.app/api/v1/characters



GET Get All Droids

https://starwars-databank-server.vercel.app/api/v1/droids

FPFtech Escola Tecnológica

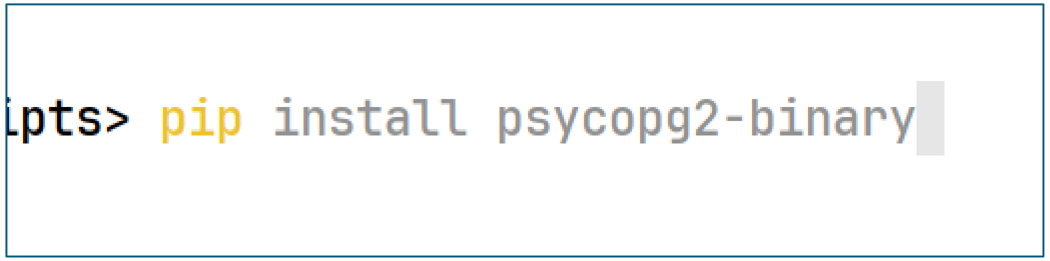


GET https://starwars-databank-server.vercel.app/api/v1/droids

```
{
  "info": {
    "total": 60,
    "page": 1,
    "limit": 10,
    "next": "/api/v1/droids?page=2&limit=10",
    "prev": null
  },
  "data": [
    {
      "_id": "640b304f916c6ff54731ed8a",
      "name": "2-1B Droid",
      "description": "2-1B droids were medical wonders, programmed to diagnose and treat injuries and diseases that afflicted millions of species in the galaxy. 2-1B droids had modular limbs that allowed them to use a range of surgical tools and other medical instruments based on their patients' needs.",
      "image": "https://lumiere-a.akamaihd.net/v1/images/2-1b-droid-main-image_546a90ad.jpeg",
      "__v": 0
    },
    {
      "_id": "640b304f916c6ff54731ed8b",
      "name": "Aqua Droid",
      "description": "Manufactured by the Techno Union, these underwater fighting droids were used by the Separatists during the Clone Wars to lay siege to aquatic planets like Kamino and Mon Calamari. They were formidable opponents and effective at surprise attacks. Aqua droids were more angular and stylized than standard battle droids, as well as excellent swimmers and were equipped with retractable laser cannons.",
      "image": "https://lumiere-a.akamaihd.net/v1/images/aqua-droid_d9076338.jpeg",
      "__v": 0
    },
```

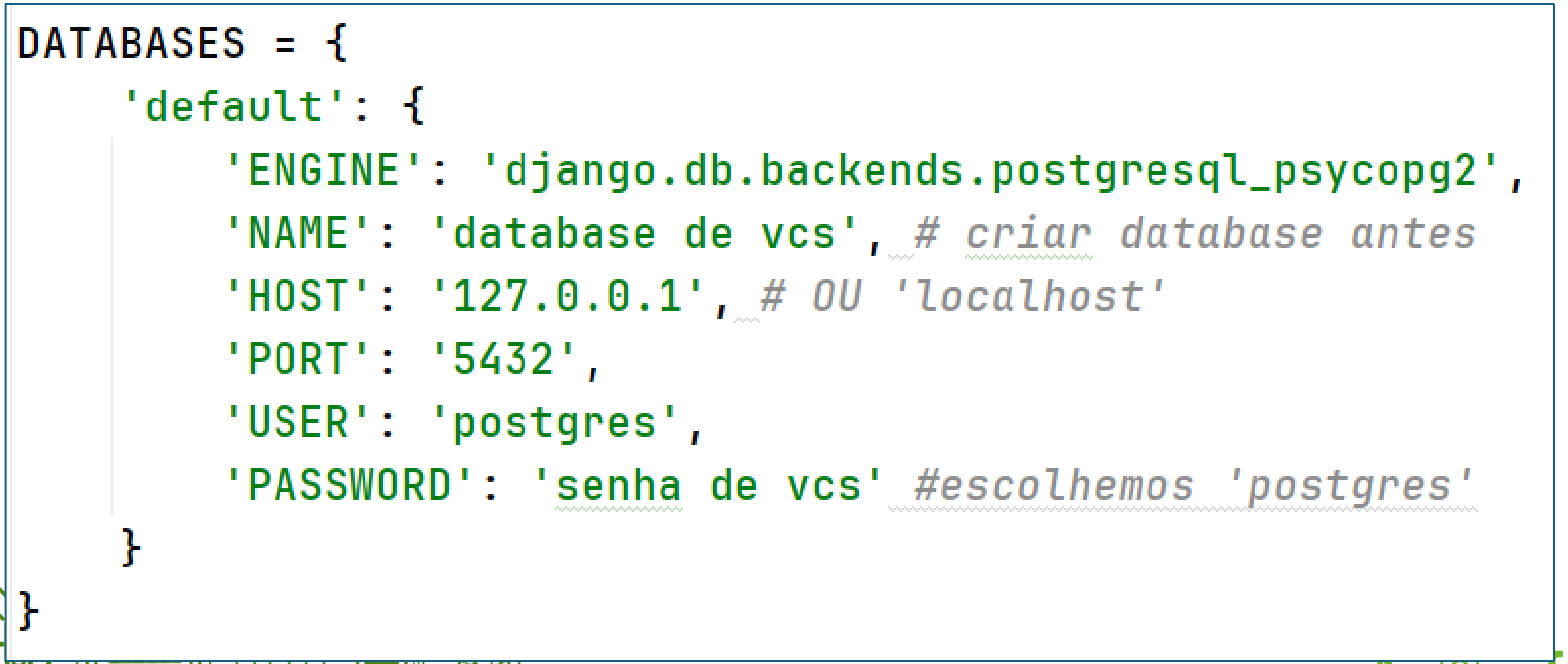
Conectando postgres no django

Primeiramente necessita instalar a biblioteca psycopg2-binary



```
ipts> pip install psycopg2-binary
```

Modificar no settings.py a info da DATABASE:



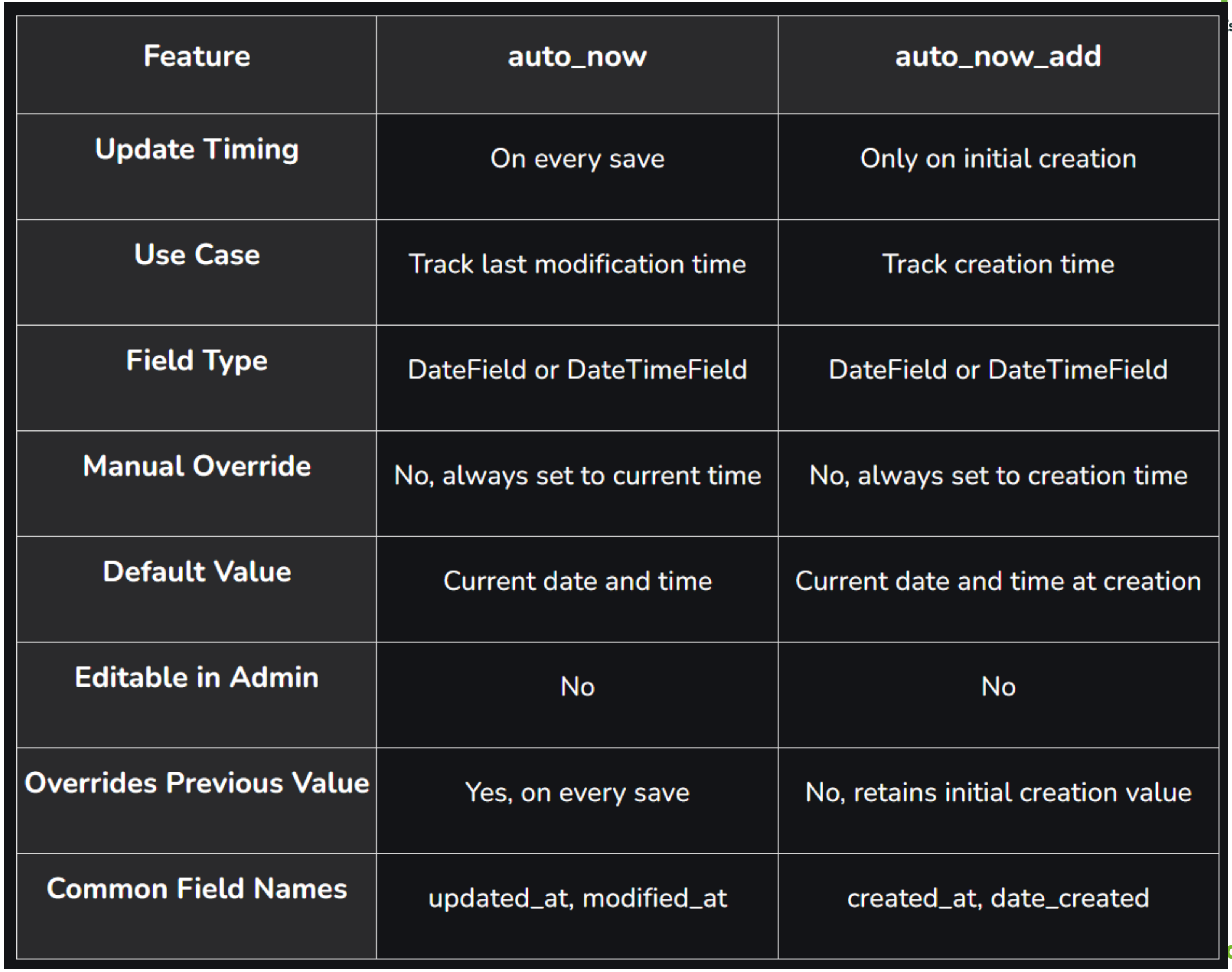
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'database de vcs', # criar database antes
        'HOST': '127.0.0.1', # OU 'localhost'
        'PORT': '5432',
        'USER': 'postgres',
        'PASSWORD': 'senha de vcs' #escolhemos 'postgres'
    }
}
```

Entendendo o arquivo models.py

```python
from django.db import models


class ModelBase(models.Model):
    id = models.BigAutoField(
        db_column='id',
        null=False,
        primary_key=True
    )
    created_at = models.DateTimeField(
        db_column='dt_created',
        auto_now_add=True,
        null=True
    )
    modified_at = models.DateTimeField(
        db_column='dt_modified',
        auto_now=True,
        null=True
    )
    active = models.BooleanField(
        db_column='cs_active',
        null=False,
        default=True
    )

    class Meta:
        abstract = True
        managed = True
```

Entendendo o arquivo models.py

https://www.geeksforgeeks.org/



Feature	auto_now	auto_now_add
Update Timing	On every save	Only on initial creation
Use Case	Track last modification time	Track creation time
Field Type	DateField or DateTimeField	DateField or DateTimeField
Manual Override	No, always set to current time	No, always set to creation time
Default Value	Current date and time	Current date and time at creation
Editable in Admin	No	No
Overrides Previous Value	Yes, on every save	No, retains initial creation value
Common Field Names	updated_at, modified_at	created_at, date_created

Entendendo o arquivo models.py

```
class Client(ModelBase):

    name = models.CharField(
        db_column='description',
        max_lenght=70,
        null=False
    )
    age = models.IntegerField(
        db_column='age',
        null=False
    )
    rg = models.CharField(
        db_column= 'rg',
        max_lenght=12,
        null=False
    )
    cpf = models.CharField(
        db_column= 'cpf',
        max_lenght=12,
        null=False
    )
```

Entendendo o arquivo models.py

```
class Product(ModelBase):

    description = models.TextField(
        db_column='description',
        null=False
    )
    quantity = models.IntegerField(
        db_column='quantity',
        null=False,
        default=0
    )
```

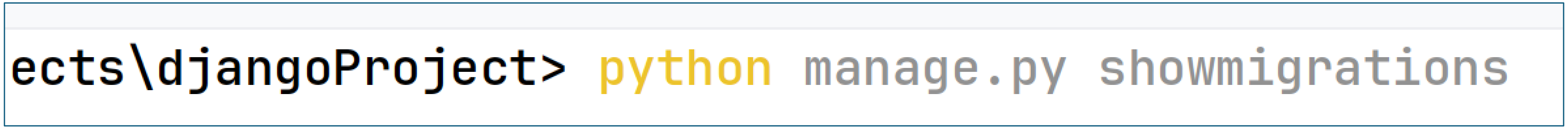
```
class Employee(ModelBase):

    name = models.CharField(
        db_column=‘tx_nome’,
        max_length=70,
        null=False
    )
    registration = models.CharField(
        db_column=‘tx_registro’,
        max_length=15,
        null=False
    )
```

```
class Sale(ModelBase):

    product = models.ForeignKey(
        Product,
        db_column='id_product',
        null=False,
        on_delete=models.DO_NOTHING
    )

    client = models.ForeignKey(
        Client,
        db_column='id_client',
        null=False,
        on_delete=models.DO_NOTHING
    )

    employee = models.ForeignKey(
        Employee,
        db_column='id_employee',
        null=False,
        on_delete=models.DO_NOTHING
    )
    nrf = models.CharField(
        db_column=‘tx_nrf',
        max_length=255,
        null=False
    )
```

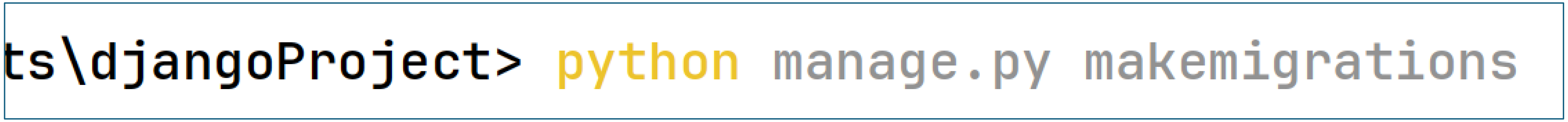
Salvar no banco postgres

Django ORM (Object-Relational Mapping) é uma ferramenta que permite interagir com o banco de dados utilizando código Python ao invés de escrever SQL diretamente.
Trabalha-se com objetos Python (modelos) em vez de tabelas e consultas SQL.

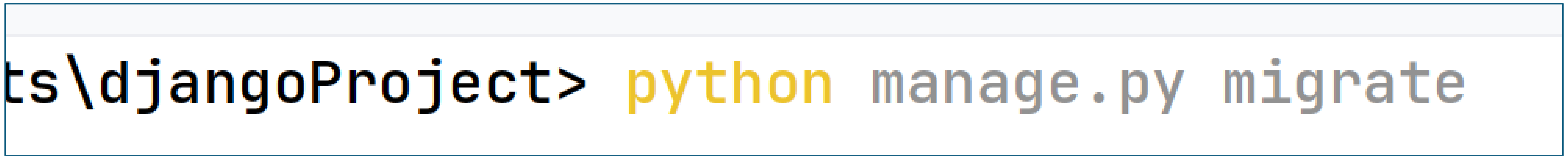
Django ORM (Object-Relational Mapping) é uma ferramenta que permite interagir com o banco de dados utilizando código Python ao invés de escrever SQL diretamente.
Trabalha-se com objetos Python (modelos) em vez de tabelas e consultas SQL.



```
ects\djangoProject> python manage.py showmigrations
```

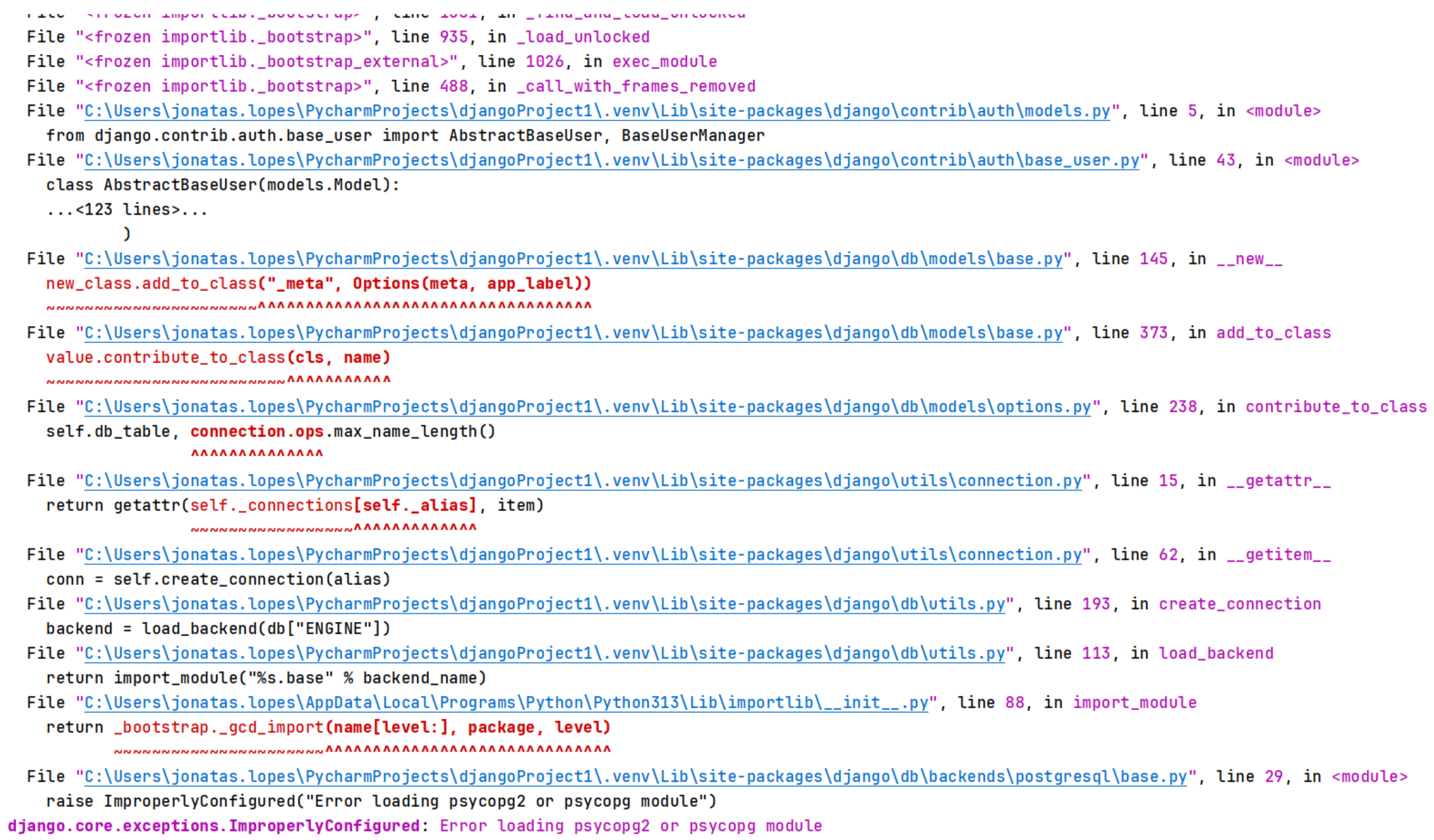


```
ts\djangoProject> python manage.py makemigrations
```



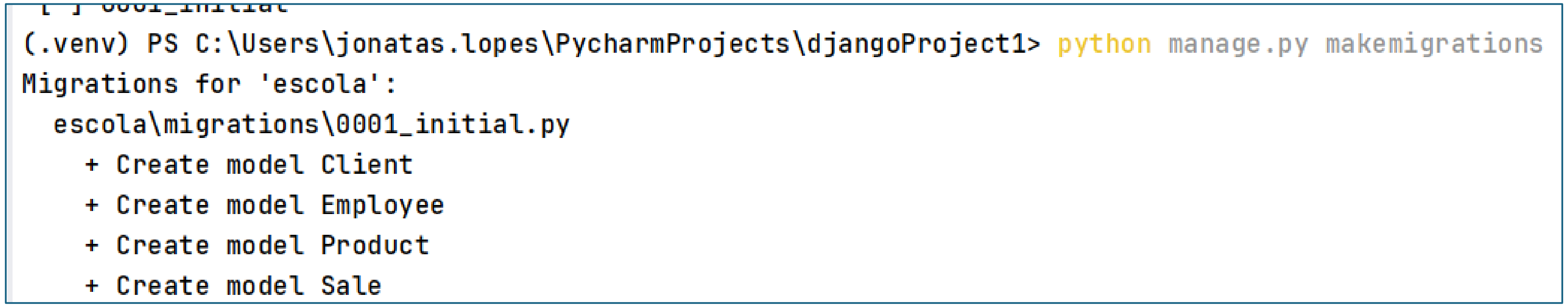
```
ts\djangoProject> python manage.py migrate
```

Salvar no banco postgres

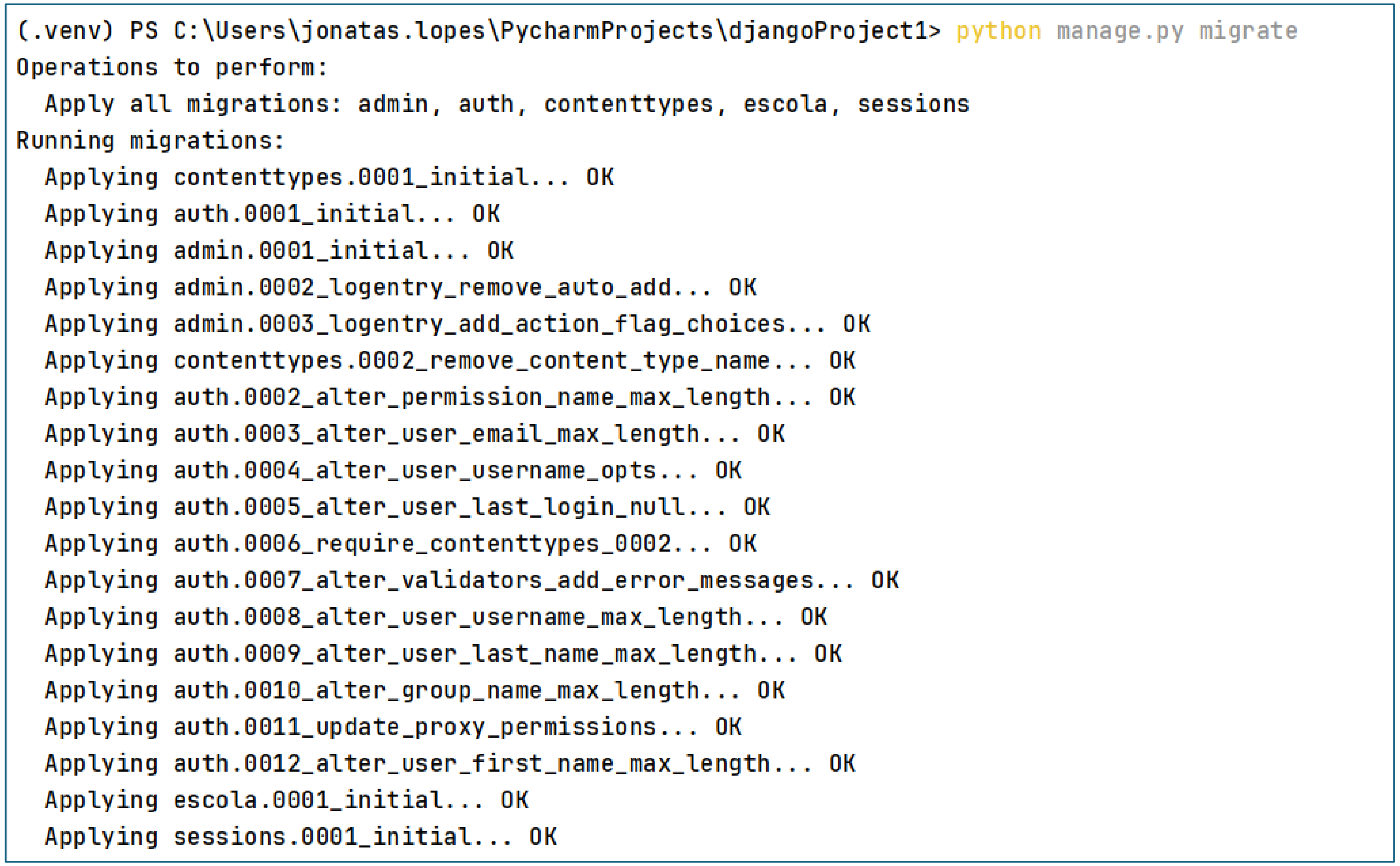


```
  File "<frozen importlib._bootstrap>", line 935, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 1026, in exec_module
  File "<frozen importlib._bootstrap>", line 488, in _call_with_frames_removed
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\contrib\auth\models.py", line 5, in <module>
    from django.contrib.auth.base_user import AbstractBaseUser, BaseUserManager
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\contrib\auth\base_user.py", line 43, in <module>
    class AbstractBaseUser(models.Model):
    ...<123 lines>...
            )
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\models\base.py", line 145, in __new__
    new_class.add_to_class("_meta", Options(meta, app_label))
    ~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\models\base.py", line 373, in add_to_class
    value.contribute_to_class(cls, name)
    ~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\models\options.py", line 238, in contribute_to_class
    self.db_table, connection.ops.max_name_length()
                   ^^^^^^^^^^^^^^
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\utils\connection.py", line 15, in __getattr__
    return getattr(self._connections[self._alias], item)
                   ~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\utils\connection.py", line 62, in __getitem__
    conn = self.create_connection(alias)
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\utils.py", line 193, in create_connection
    backend = load_backend(db["ENGINE"])
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\utils.py", line 113, in load_backend
    return import_module("%s.base" % backend_name)
  File "C:\Users\jonatas.lopes\AppData\Local\Programs\Python\Python313\Lib\importlib\__init__.py", line 88, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
           ~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\jonatas.lopes\PycharmProjects\djangoProject1\.venv\Lib\site-packages\django\db\backends\postgresql\base.py", line 29, in <module>
    raise ImproperlyConfigured("Error loading psycopg2 or psycopg module")
django.core.exceptions.ImproperlyConfigured: Error loading psycopg2 or psycopg module
```

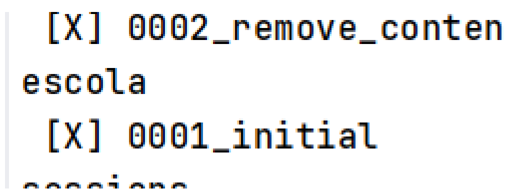
Faltou instalar o psycopg2-binary



```
  [ ] 0001_initial
(.venv) PS C:\Users\jonatas.lopes\PycharmProjects\djangoProject1> python manage.py makemigrations
Migrations for 'escola':
  escola\migrations\0001_initial.py
    + Create model Client
    + Create model Employee
    + Create model Product
    + Create model Sale
```

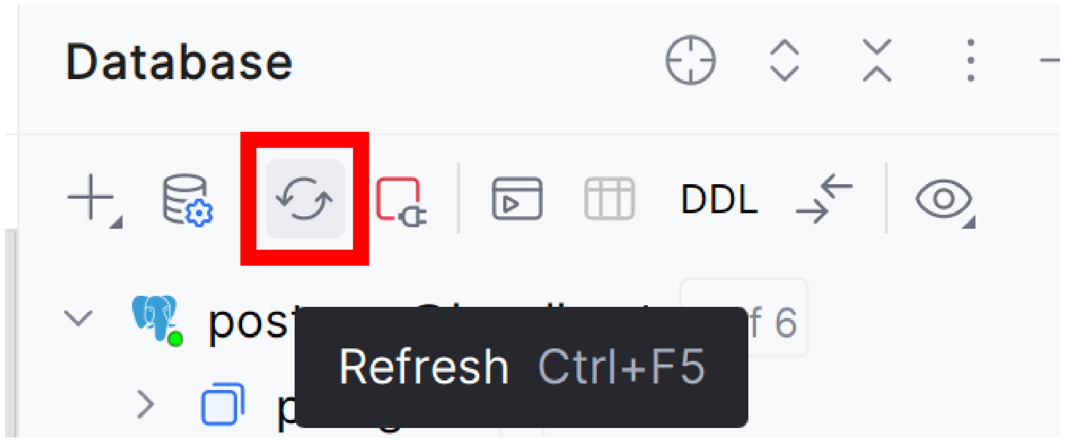


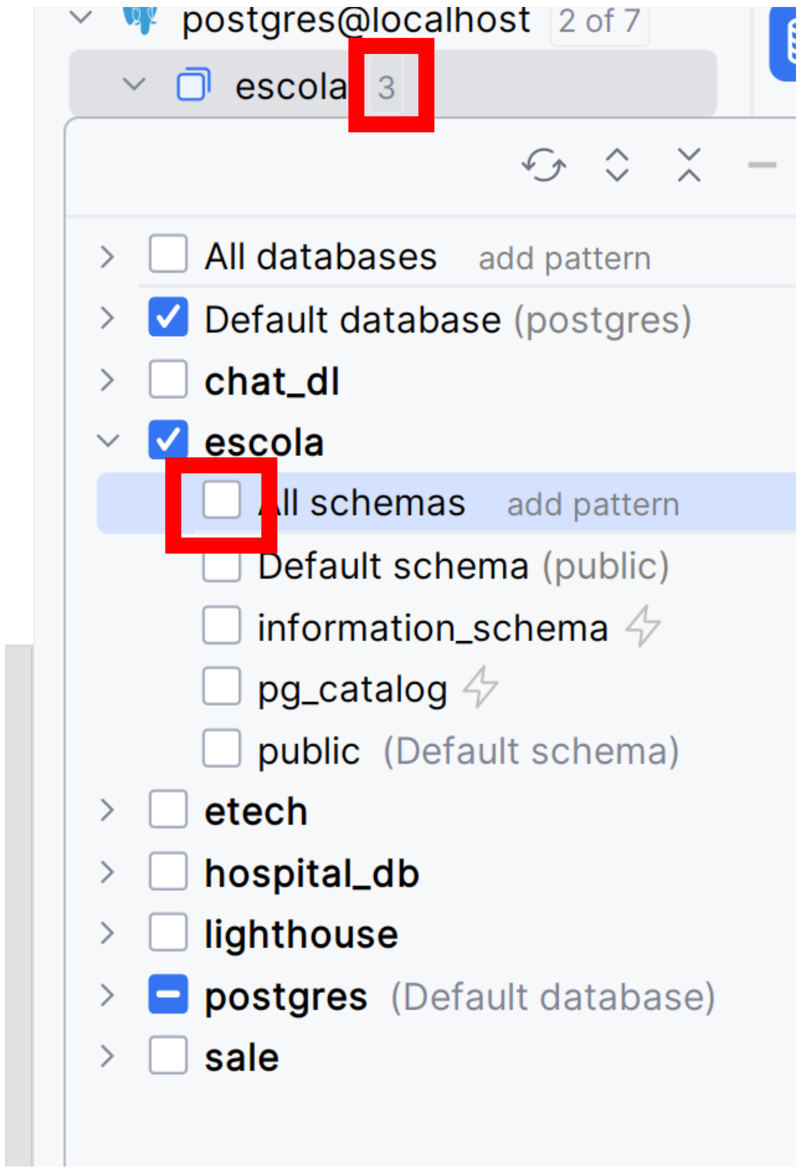
```
(.venv) PS C:\Users\jonatas.lopes\PycharmProjects\djangoProject1> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, escola, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying escola.0001_initial... OK
  Applying sessions.0001_initial... OK
```



```
 [X] 0002_remove_conten
escola
 [X] 0001_initial
```

Ver alterações no banco de dados postgres.

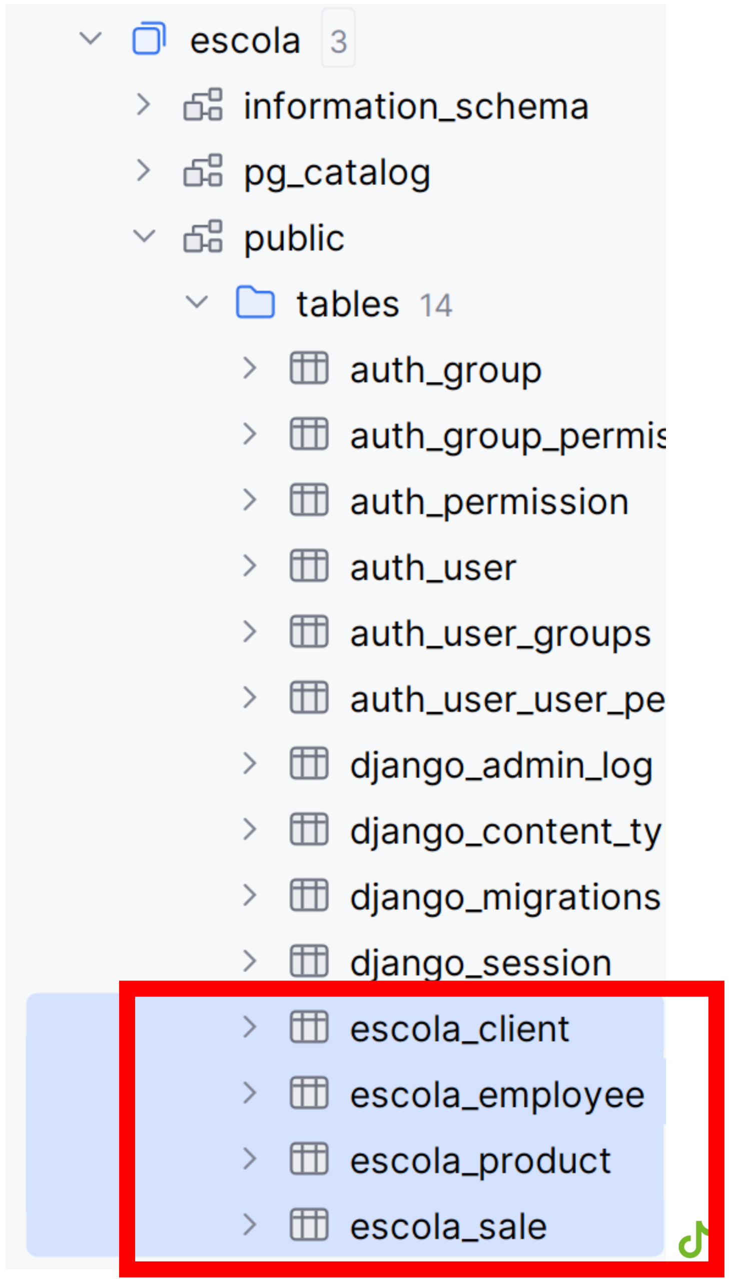



postgres@localhost 2 of 7
escola 3

All databases add pattern
✓ Default database (postgres)
chat_dl
✓ escola
All schemas add pattern
Default schema (public)
information_schema
pg_catalog
public (Default schema)
etech
hospital_db
lighthouse
postgres (Default database)
sale

Ver alterações no banco de dados postgres.

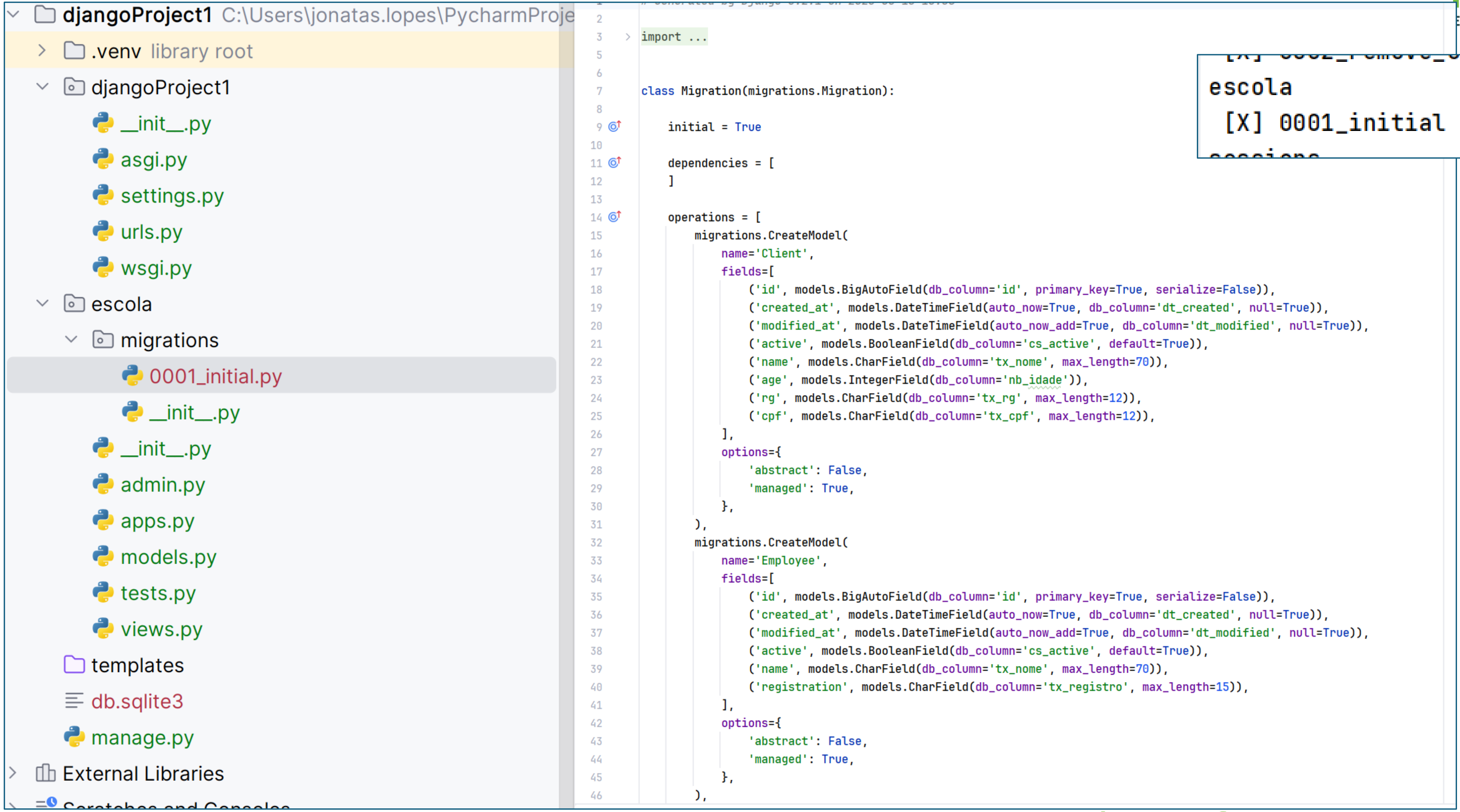
Novas tabelas estão aqui (limpas ainda):



```
escola 3
  information_schema
  pg_catalog
  public
    tables 14
      auth_group
      auth_group_permis
      auth_permission
      auth_user
      auth_user_groups
      auth_user_user_pe
      django_admin_log
      django_content_ty
      django_migrations
      django_session
      escola_client
      escola_employee
      escola_product
      escola_sale
```

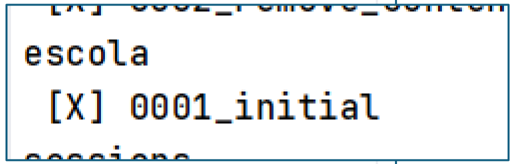
<-tabelas do django

Ver alterações no banco de dados postgres.



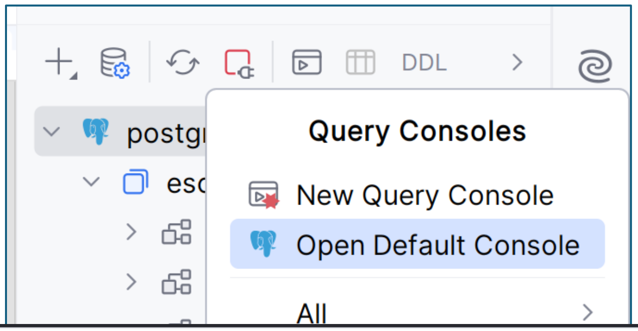
```
djangoProject1  C:\Users\jonatas.lopes\PycharmProje
  > .venv  library root
  v djangoProject1
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
  v escola
    v migrations
        0001_initial.py
        __init__.py
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      views.py
    templates
    db.sqlite3
    manage.py
> External Libraries
```

```python

import ...


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Client',
            fields=[
                ('id', models.BigAutoField(db_column='id', primary_key=True, serialize=False)),
                ('created_at', models.DateTimeField(auto_now=True, db_column='dt_created', null=True)),
                ('modified_at', models.DateTimeField(auto_now_add=True, db_column='dt_modified', null=True)),
                ('active', models.BooleanField(db_column='cs_active', default=True)),
                ('name', models.CharField(db_column='tx_nome', max_length=70)),
                ('age', models.IntegerField(db_column='nb_idade')),
                ('rg', models.CharField(db_column='tx_rg', max_length=12)),
                ('cpf', models.CharField(db_column='tx_cpf', max_length=12)),
            ],
            options={
                'abstract': False,
                'managed': True,
            },
        ),
        migrations.CreateModel(
            name='Employee',
            fields=[
                ('id', models.BigAutoField(db_column='id', primary_key=True, serialize=False)),
                ('created_at', models.DateTimeField(auto_now=True, db_column='dt_created', null=True)),
                ('modified_at', models.DateTimeField(auto_now_add=True, db_column='dt_modified', null=True)),
                ('active', models.BooleanField(db_column='cs_active', default=True)),
                ('name', models.CharField(db_column='tx_nome', max_length=70)),
                ('registration', models.CharField(db_column='tx_registro', max_length=15)),
            ],
            options={
                'abstract': False,
                'managed': True,
            },
        ),
```

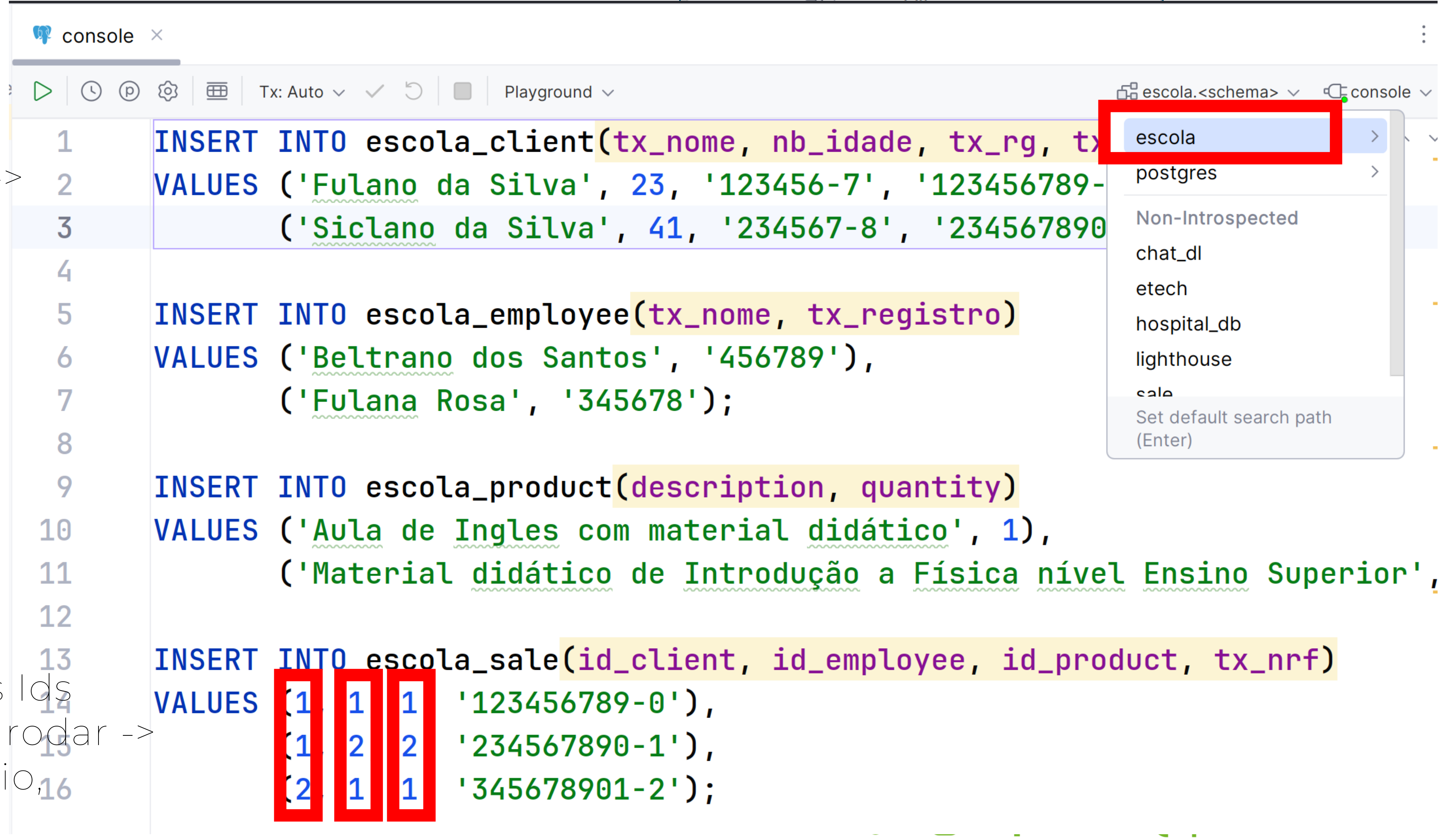


```
escola
 [X] 0001_initial
```

Adicionando dados



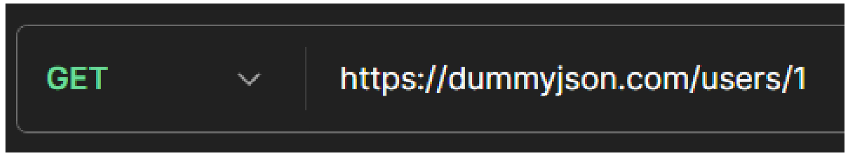
Lembre-se:
para rodar cada um, ->
selecione e ‘Ctrl + Enter’

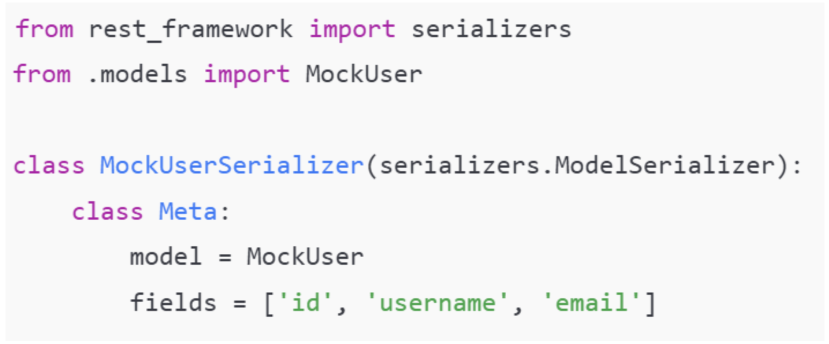


```sql
INSERT INTO escola_client(tx_nome, nb_idade, tx_rg, tx
VALUES ('Fulano da Silva', 23, '123456-7', '123456789-
       ('Siclano da Silva', 41, '234567-8', '234567890

INSERT INTO escola_employee(tx_nome, tx_registro)
VALUES ('Beltrano dos Santos', '456789'),
       ('Fulana Rosa', '345678');

INSERT INTO escola_product(description, quantity)
VALUES ('Aula de Ingles com material didático', 1),
       ('Material didático de Introdução a Física nível Ensino Superior',

INSERT INTO escola_sale(id_client, id_employee, id_product, tx_nrf)
VALUES (1, 1, 1, '123456789-0'),
       (1, 2, 2, '234567890-1'),
       (2, 1, 1, '345678901-2');
```

Importante:
verifique se esses Ids existem antes de rodar -> esse. Se necessário, mude os ids.

Ciclo de Requests


GET https://dummyjson.com/users/1


```python
from rest_framework import serializers
from .models import MockUser

class MockUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = MockUser
        fields = ['id', 'username', 'email']
```

request

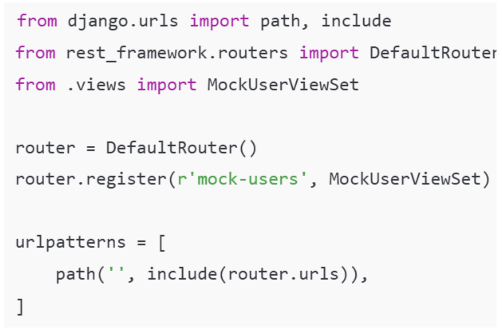
endpoint
urls

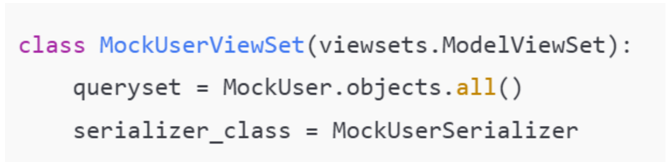
viewsets

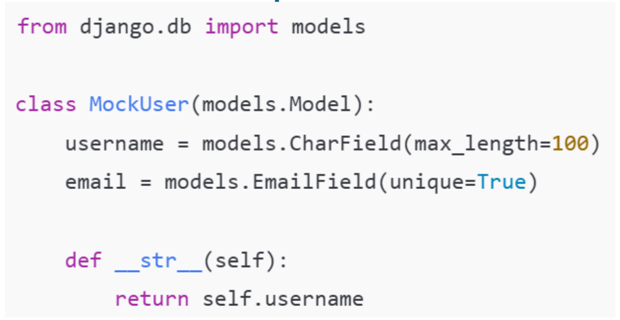
Serializers
JSON=Django

models

database


```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import MockUserViewSet

router = DefaultRouter()
router.register(r'mock-users', MockUserViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```


```python
class MockUserViewSet(viewsets.ModelViewSet):
    queryset = MockUser.objects.all()
    serializer_class = MockUserSerializer
```


```python
from django.db import models

class MockUser(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

    def __str__(self):
        return self.username
```

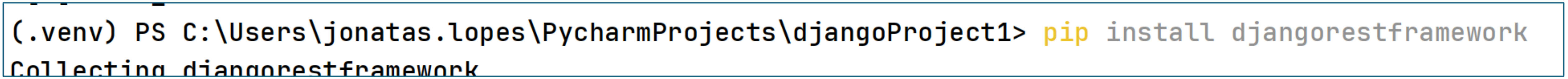
@fpftech.educacional

Framework DRF
Configurando o Django REST Framework no projeto.

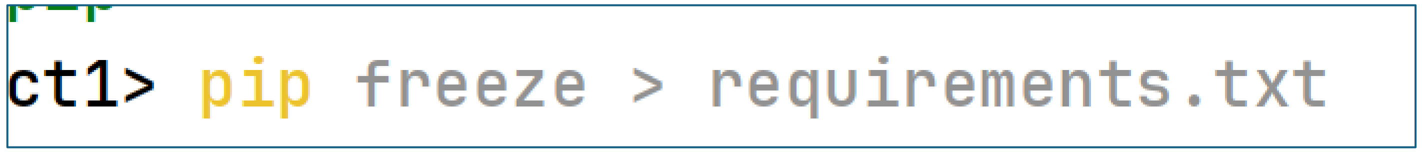
Framework REST

O Django Rest Framework é uma biblioteca do Django que facilita a criação de APIs RESTful. Ele é amplamente utilizado para o desenvolvimento de APIs na web, permitindo que desenvolvedores construam rapidamente Endpoints que seguem o padrão REST, de forma eficiente e organizada.

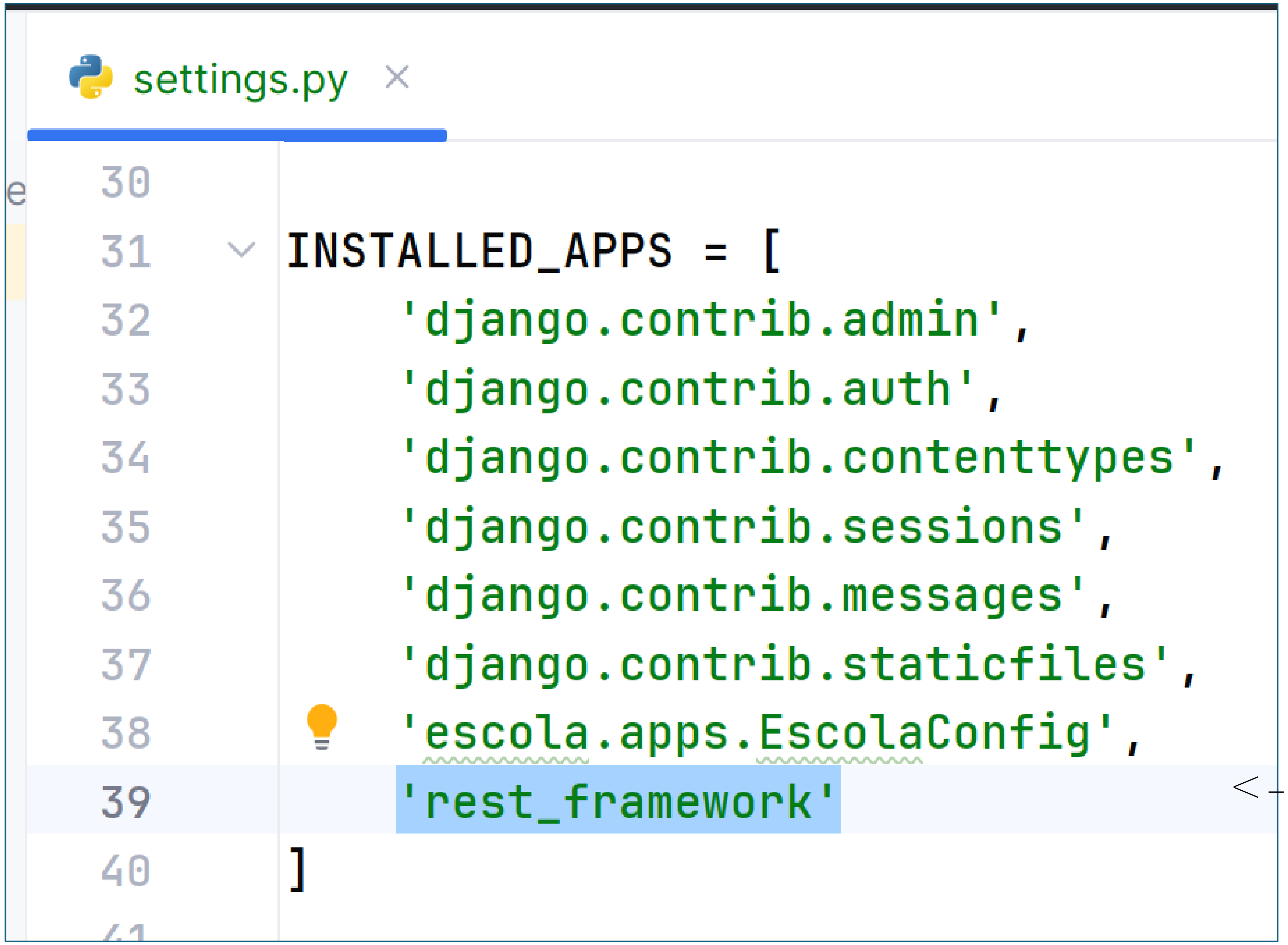
- Serialização
- Viewsets prontos para uso.
- Interface browsable API
- Bom padrões
- Autenticação e Permissões
- Paginação



```
(.venv) PS C:\Users\jonatas.lopes\PycharmProjects\djangoProject1> pip install djangorestframework
Collecting djangorestframework
```



```
ct1> pip freeze > requirements.txt
```


settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'escola.apps.EscolaConfig',
    'rest_framework'
]
```

<- Adicionar

PRONTO!

Serializers

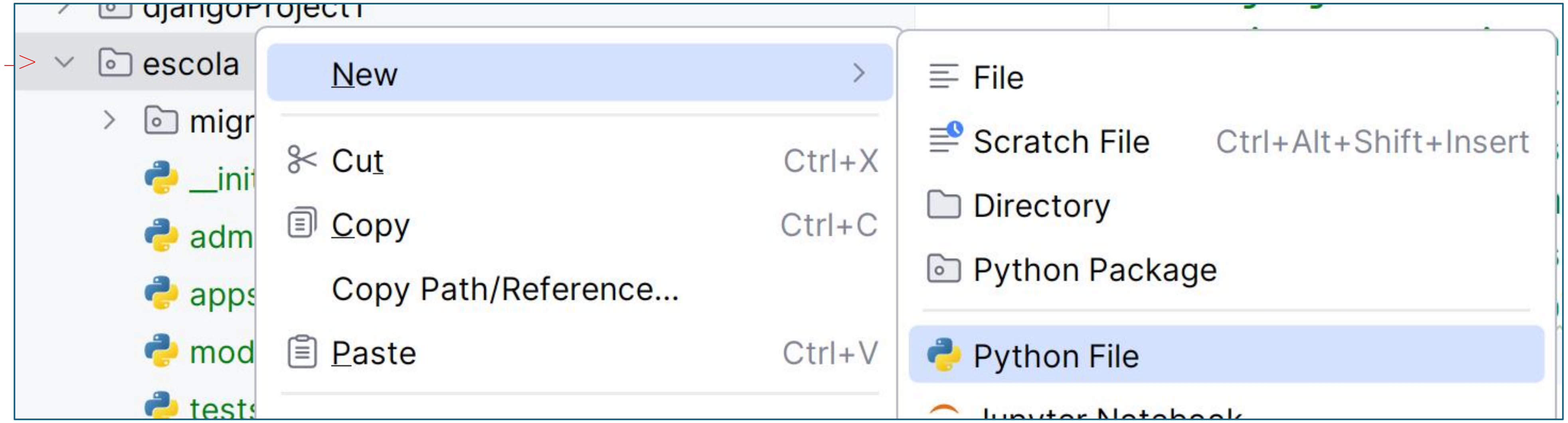
normalizar e realizar validações com DjangoRestFramework.

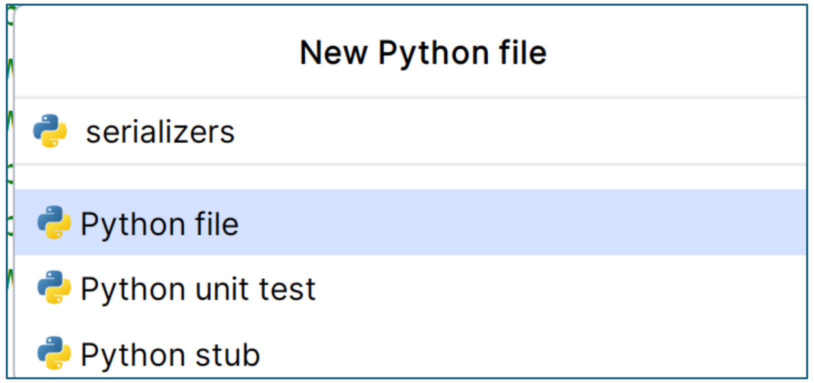
Código Python ⇔ JSON

Oferece ferramentas para serializar dados, o que significa transformar objetos do banco de dados (modelos Django) em formatos como JSON ou XML, e vice-versa. Além disso serve para algumas validações.

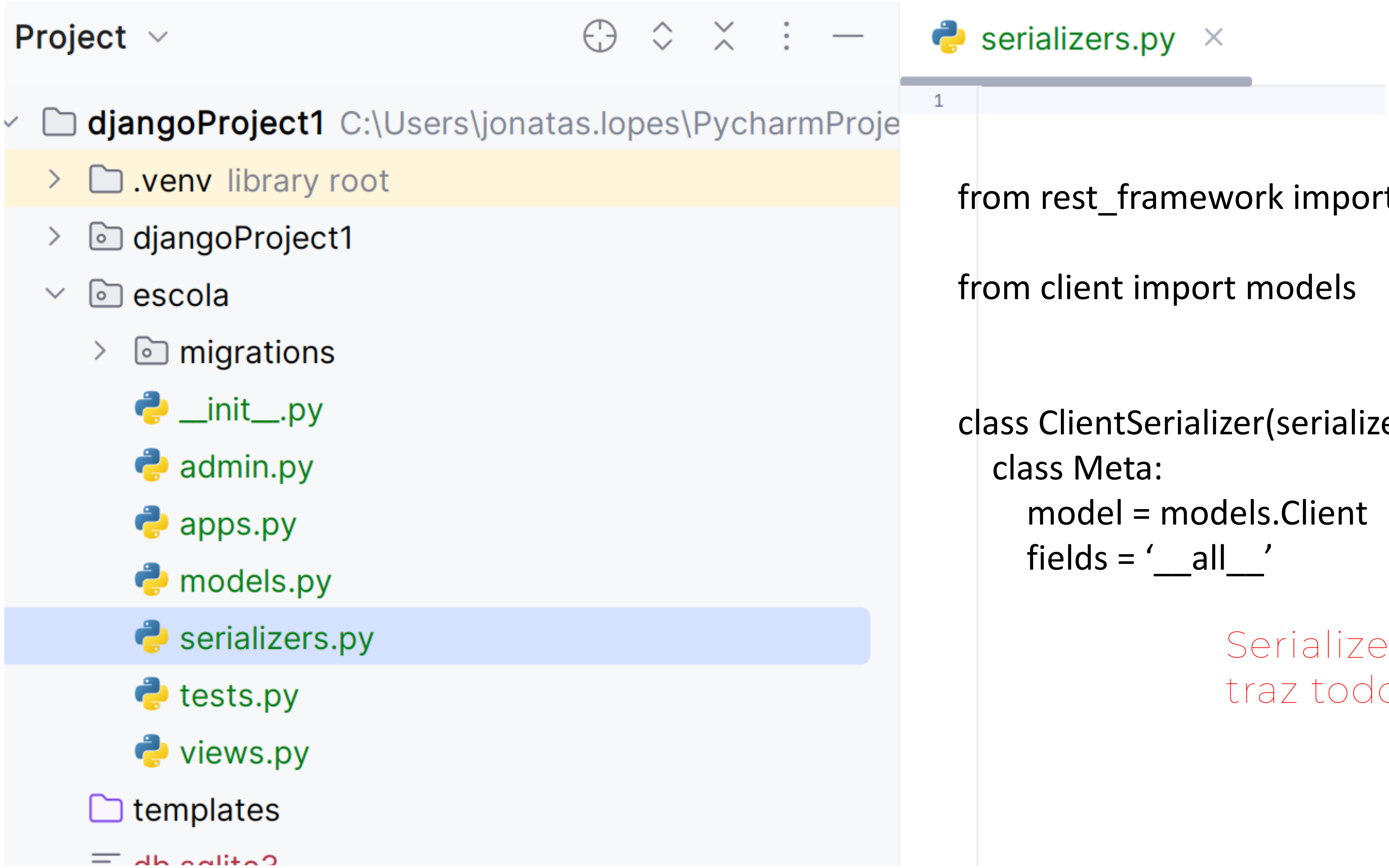
1) Criar arquivo serializers.py dentro da sua Aplicação:

Botão direito aqui ->





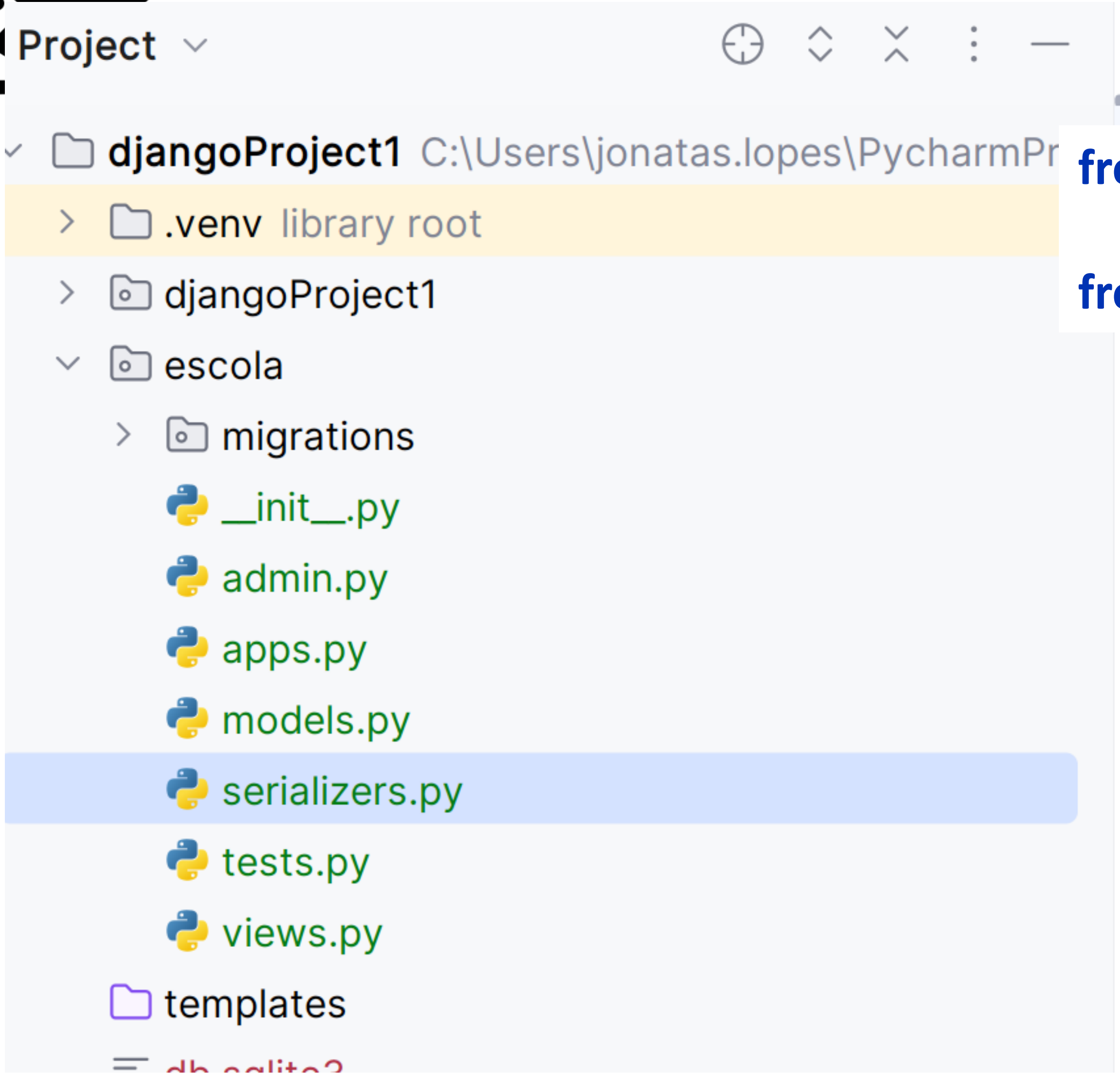
Serializers



```
from rest_framework import serializers

from client import models


class ClientSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.Client
        fields = ‘__all__’
```

Serializer do Client:
traz todos os campos

Serializers



serializers.py

```
from rest_framework import serializers

from escola.models import Client, Product, Employee, Sale


class ClienteSerializer(serializers.ModelSerializer):
  id = serializers.CharField(read_only=True)
  name = serializers.CharField(max_length=70)
  age = serializers.IntegerField(min_value=18, max_value=100)

  class Meta:
    model = models.Client
    fields = ['id', 'name', 'age', 'created_at']
```

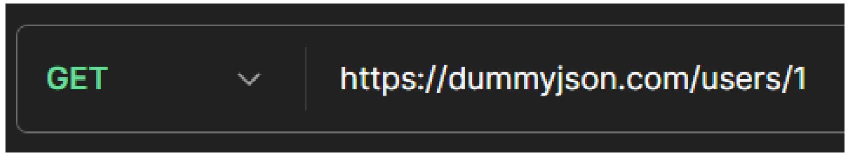
Serializer do Client:
validando campos id, name, age
Filtrando campos trazendo só alguns

```
class ClientSerializer(serializers.ModelSerializer):
     age = serializers.IntegerField(min_value=18, max_value=100)
    class Meta:
        model = Client
        fields = ['id', 'name', 'age', 'rg', 'cpf']

class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ['id', 'description', 'quantity']

class EmployeeSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = ['id', 'name', 'registration']
```

FPFtech
Escola Tecnológica

```
class SaleSerializer(serializers.ModelSerializer):
   product = ProductSerializer(read_only=True)
   product_id = serializers.PrimaryKeyRelatedField(queryset=Product.objects.all(),
source='product', write_only=True)
   client = ClientSerializer(read_only=True)
   client_id = serializers.PrimaryKeyRelatedField(queryset=Client.objects.all(),
source='client', write_only=True)
   employee = EmployeeSerializer(read_only=True)
   employee_id = serializers.PrimaryKeyRelatedField(queryset=Employee.objects.all(),
source='employee', write_only=True)

   class Meta:
      model = Sale
      fields = ['id', 'product_id', 'product', 'client_id', 'client', 'employee_id', 'employee', 'nrf']
```

Ciclo de Requests



GET https://dummyjson.com/users/1

```python
from rest_framework import serializers
from .models import MockUser

class MockUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = MockUser
        fields = ['id', 'username', 'email']
```

request

endpoint
urls

viewsets

Serializers
JSON=Django

models

database

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import MockUserViewSet

router = DefaultRouter()
router.register(r'mock-users', MockUserViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

```python
class MockUserViewSet(viewsets.ModelViewSet):
    queryset = MockUser.objects.all()
    serializer_class = MockUserSerializer
```

```python
from django.db import models

class MockUser(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

    def __str__(self):
        return self.username
```

FPFtech
Escola Tecnológica

Obrigado!

FPFtech
Escola Tecnológica

@fpftech.educacional