django

Inicio

Overview do Framework

Cases de Sucesso

Insta

coursera

cartola FC

SporTV

globo.com

Bitbucket

HOST

- Amazon
- Heroku
- Pythonanywhere
- Docker

TEMPLATE

- Built-in System
- Jinja2

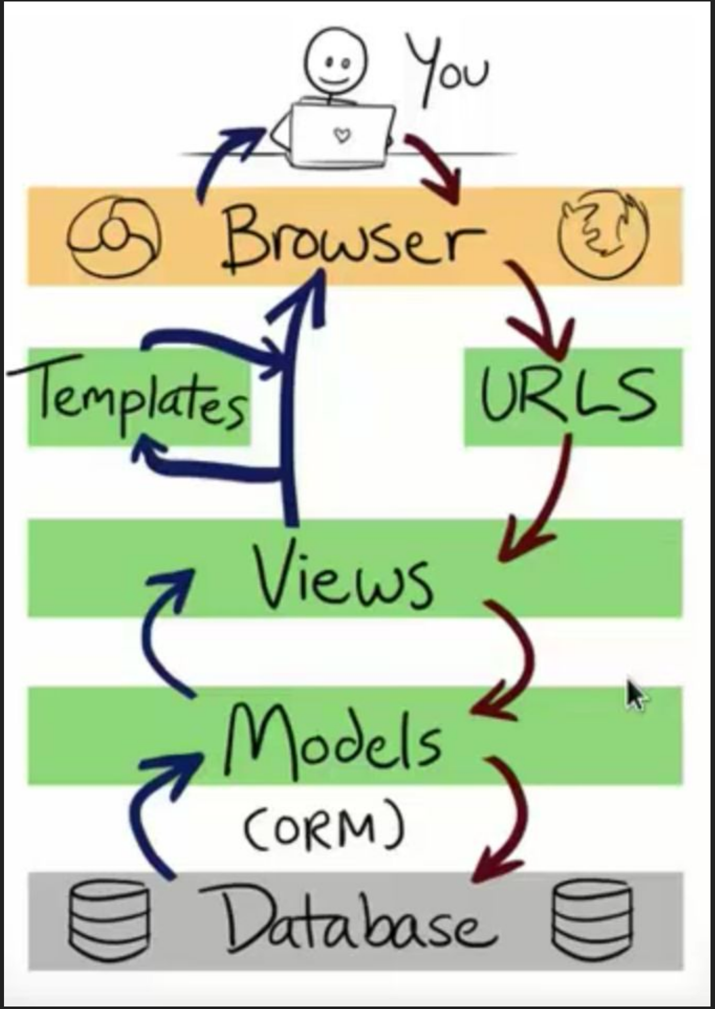
Parte 1

INTRODUÇÃO DO DJANGO

MVT o MVC no Django

Django MTV		**Modelo MVC**
MODEL	⟷	MODEL
TEMPLATE		VIEW
VIEW		CONTROLLER

Arquitetura Django



Novo **Projeto** e **App**

```
$ django-admin startproject teacherfeed

$ cd teacherfeed

$ python manage.py runserver

$ python manage.py migrate

$ python manage.py startapp core
```

Gestão de Migrações com Banco de dados

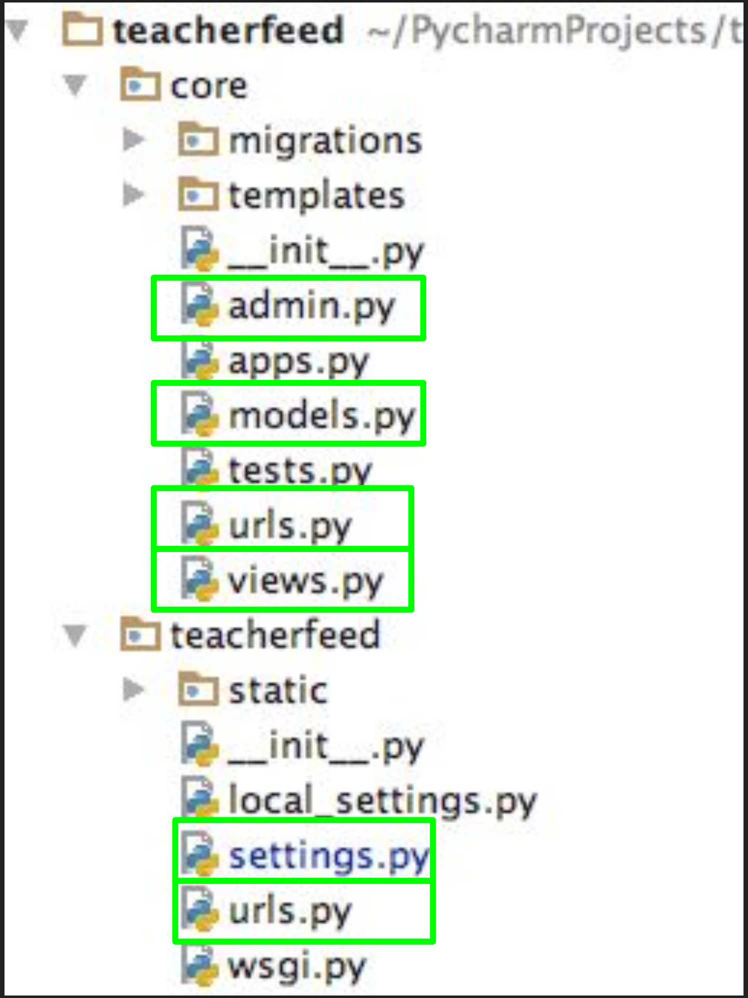
Ao modificar algum Model devemos:

```
$ python manage.py makemigrations

$ python manage.py migrate

$ python manage.py showmigrations
```

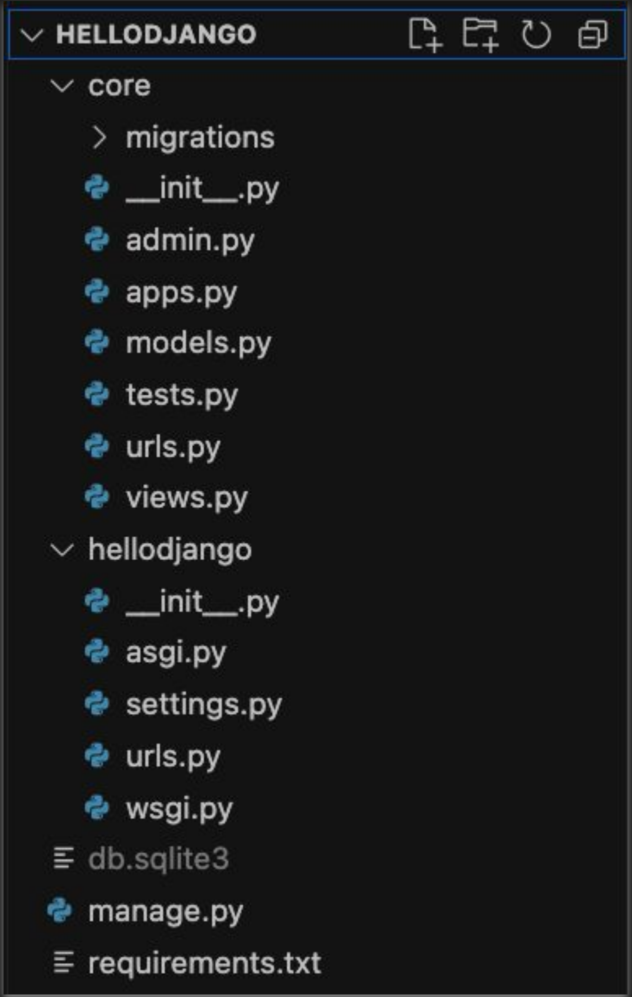
Criar Super Usuário

```
$ python manage.py createsuperuser
```

Project



```
teacherfeed ~/PycharmProjects/t
  core
    migrations
    templates
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
  teacherfeed
    static
    __init__.py
    local_settings.py
    settings.py
    urls.py
    wsgi.py
```



```
HELLODJANGO
  core
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
  hellodjango
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  db.sqlite3
  manage.py
  requirements.txt
```

“Instalar” a App no Projeto

settings.py

```
....

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

Rotas (`urls.py`)

App urls

```python
from django.urls import path

from .views import hello, hello_rosa, index,
milhao

urlpatterns = [
      path('', index),
      path('hello', hello),
      path('rosa', hello_rosa),
      path('milho', milhao)
]
```

Project urls

```python
from django.contrib import admin
from django.urls import include, path
from core.urls import urlpatterns as core_urls

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

Views e Templates

Boa prática criar subpasta com nome da App dentro do diretório templates

core/views.py

```python
def index(request):
    #return HttpResponse('TeacherFeed...')
    return render(request, 'core/index.html')
```

core/templates/core/index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>TeacherFeed</title>
</head>
<body>

<h1>Index TeacherFeed</h1>

</body>
</html>
```

Herança de Template

base.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>TeacherFeed</tit
    le>
    </head>
<body>
    {% block content %}
    {% endblock %}
</body>
</html>
</html>
```

index.html

```
{% extends 'core/base.html' %}


{% block content %}
    <h1>Index TeacherFeed</h1>
{% endblock %}
```

Fim - Parte 1

INTRODUÇÃO DO DJANGO

Parte 2

Modelos e Admin

Django Models.......... ModelFields Types

core/models.py

```
class Professor(models.Model):
      nome = models.CharField(verbose_name='Nome', max_length=100)
      email = models.EmailField()
```

- AutoField
- BigIntegerField
- BinaryField
- BooleanField
- CharField
- CommaSeparatedIntegerField
- DateField
- DateTimeField
- DecimalField
- DurationField
- EmailField
- FileField
- FileField and FieldFile
- FilePathField
- FloatField
- ImageField
- IntegerField
- GenericIPAddressField
- NullBooleanField
- PositiveIntegerField
- PositiveSmallIntegerField
- SlugField
- SmallIntegerField
- TextField
- TimeField
- URLField
- UUIDField

Django Models.......... ModelFields Types

core/models.py

```python
class Professor(models.Model):
    nome = models.CharField(verbose_name='Nome', max_length=100)
    email = models.EmailField()

class Disciplina(models.Model):
    professor = models.ForeignKey(Professor, on_delete=models.CASCADE, related_name='disciplinas')
    nome = models.CharField(max_length=100, blank=False)
    sigla = models.CharField(max_length=10, blank=False, null=True)
    instituicao = models.CharField(max_length=100, blank=False, null=False)
    curso = models.CharField(max_length=100, blank=False, null=False)

class Aluno(models.Model):
    nome = models.CharField(verbose_name='Nome', max_length=100)
    email = models.EmailField()
    user = models.OneToOneField(User, on_delete=models.CASCADE)
```

Django ORM

```
$ python manage.py <command>
```

Comandos:

- makemigrations: gerar migração/sql
- migrate: executar
- sqlmigrate: mostrar o sql da migração
- shell: ambiente interativo com BD/App

Admin

“O Admin não é o Django” é apenas mais uma app plugável

http://localhost:8000/admin

```
$ python manage.py createsuperuser
```

Registro de Models para o Admin

core/admin.py

```
from django.contrib import admin
from .models import *


admin.site.register(Professor)
admin.site.register(Disciplina)
admin.site.register(Aluno)
```

Ajustes no Admin

core/admin.py

```
class ProfessorAdmin(admin.ModelAdmin):
    list_display = ('nome', 'email', )
    search_fields = ('nome',)

admin.site.register(Professor, ProfessorAdmin)
```

core/models.py

```
class Professor(models.Model):
    nome = models.CharField(verbose_name='Nome', max_length=100)
    email = models.EmailField()
    user = models.OneToOneField(User, on_delete=models.CASCADE)

    class Meta():
        verbose_name = 'Professor'
        verbose_name_plural = 'Professores'
        ordering = ['-nome', ]
```

Fim - Parte 2

Modelos e Admin

Parte 3

Trabalhando com Views, Forms, Templates

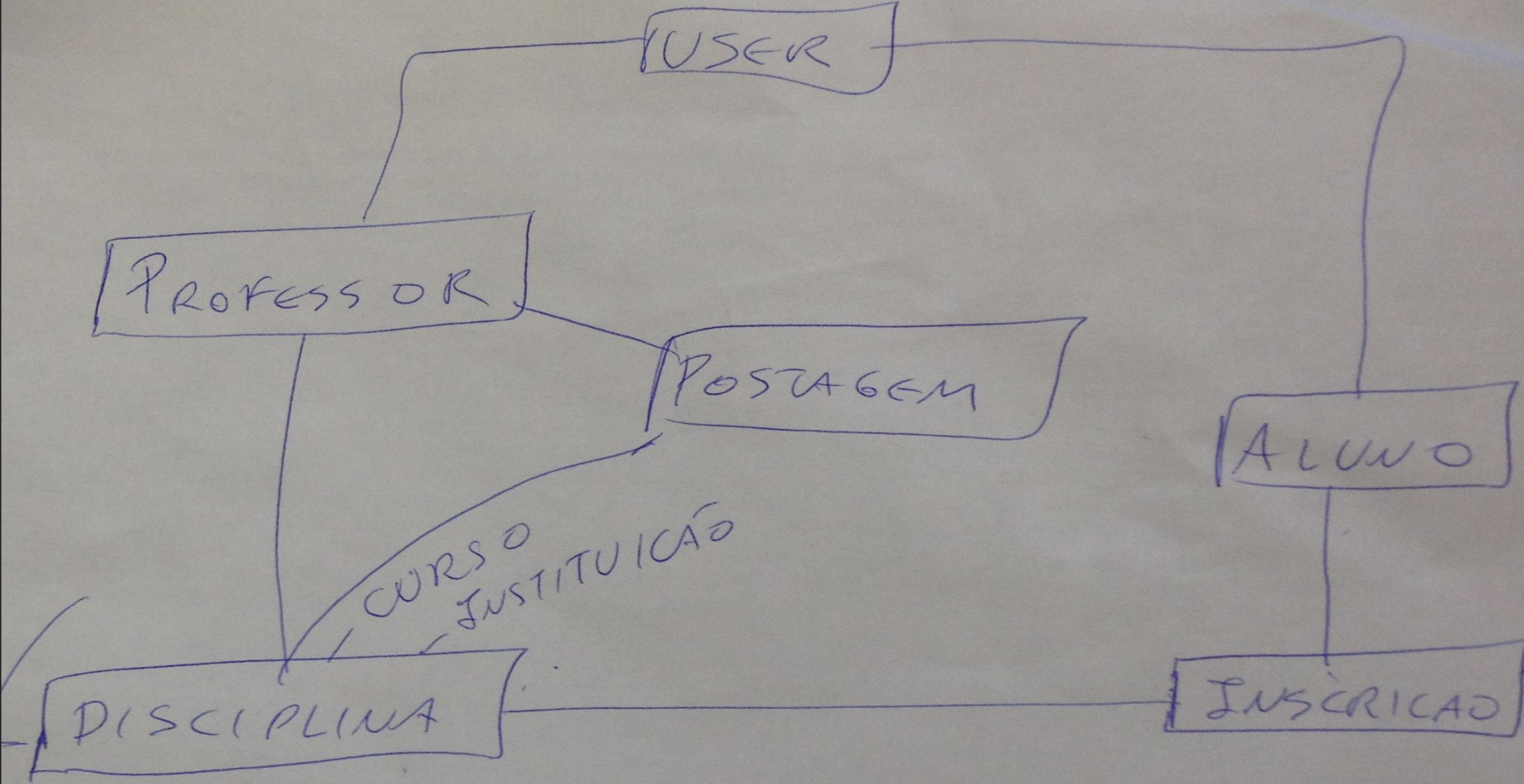
TeacherFeed

Features:

- Auto signup Professor e Aluno
- Professor: Adicionar Disciplinas, Aceitar alunos e Fazer Postagens
- Alunos: Pesquisar por Disciplinas, Solicitar Inscrição

Fluxo de Desenvolvimento:

- Rotas
- View
  - Opções: Function View, Class-Based View, Generic Views
- Form
- Template
- Ou, Usar o Admin Customizado(ou não.)
- Usar REST


USER
PROFESSOR
POSTAGEM
ALUNO
CURSO
INSTITUICÃO
DISCIPLINA
INSCRICAO

Componentes:

- URL → /professor/signup
- TEMPLATE → professor_signup.html
- VIEW → ProfessorSignup.py
- FORM → ProfessorSignupForm.py

Django 'Function-Based' Views

```
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # <view logic>
        return HttpResponse('result')
```

Django Generic Class-Based Views

```python
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponse('result')
```

Django CBV: 'Class-Based Views'

```
class ProfessorSignup(View):

    template_name = 'core/formulario_professor.html'

    def get(self, request):
        return render(request, template_name=self.template_name)

    def post(self, request):
        # view login
```

VIEW: Registrar Professor

core/views.py

```
class ProfessorSignup(View):
    template_name = 'core/formulario_professor.html'

    def get(self, request):
        return render(request, template_name=self.template_name)

    def post(self, request):
        nome = request.POST['professor.nome']
        email = request.POST['professor.email']
        senha = request.POST['professor.user.password']

        if User.objects.filter(username=email).exists():
            return HttpResponse('Usuario ja existe!!')

        user = User.objects.create_user(username=email, email=email, password=senha)
        professor = Professor.objects.create(nome=nome, email=email, user=user)

        return HttpResponse('Professor Salvo com sucesso! %d' % professor.id)
```

core/urls.py

```
from core import views


urlpatterns = [
    path('/', views.index, name='index'),
    path('professor/signup', views.ProfessorSignup.as_view(),
name='professor-signup'),
]
```

core/template/core/formulario_professor.html

TEMPLATE: Registrar Professor

```
{% extends 'core/base_accounts.html' %}

{% block action %} Registro de Professor {% endblock %}

{% block content %}

<form action="{% url 'core:professor-signup' %}" method="post">

    {% csrf_token %}

    <input name="professor.nome" type="text" placeholder="Nome Professor" />
    <input name="professor.email" type="email" placeholder="email" />
    <input name="professor.user.password" type="password" placeholder="Senha">

    <button type="submit">Registrar</button>
</form>

{% endblock %}
```

FORM: ModelForm

core/forms.py

```
class ProfessorSignupForm(forms.ModelForm):
    senha = forms.CharField(required=True, widget=forms.PasswordInput())

    class Meta:
        model = Professor
        fields = ['nome', 'email', ]
        #exclude = ['user']
```

NOVO TEMPLATE: Registrar Professor

core/template/core/form_professor.html

```
{% extends 'core/base_accounts.html' %}

{% block action %} Registro de Professor {% endblock %}

{% block content %}

<form action="{% url 'core:professor-signup' %}" method="post">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Registrar</button>
</form>

{% endblock %}
```

Novo método 'post' da VIEW

core/views.py

```
def post(self, request):
        form = ProfessorSignupForm(request.POST)

        if form.is_valid():
                nome = form.cleaned_data['nome']
                email = form.cleaned_data['email']
                senha = form.cleaned_data['senha']

                if User.objects.filter(username=email).exists():
                        form._errors[NON_FIELD_ERRORS] = form.error_class(['Usuário já existe'])
                        ctx = {'form': form}
                        return render(request, self.template_name, ctx)

                user = User.objects.create_user(username=email, email=email, password=senha)
                professor = Professor.objects.create(nome=nome, email=email, user=user)

                return HttpResponse('Professor Salvo com sucesso! %d' % professor.id)
        else:
                ctx = {'form': form}
                return render(request, self.template_name, ctx)
```

Login e Logout: Views from Django Auth

core/urls.py

```
from django.contrib.auth import views as auth_views


urlpatterns = [
path('', views.index, name='index'),
path('professor/signup', views.ProfessorSignup.as_view(), name='professor-signup'),
path('login/', auth_views.login, {'template_name': "core/login.html"}, name='login'),
path('logout/', auth_views.logout, {'next_page': 'core:index'}, name='logout'),
]
```

settings.py

```
...
LOGIN_URL = 'core:login'
LOGOUT_URL = 'core:logout'
LOGIN_REDIRECT_URL = 'core:index'
...
```

Template Login: core/login.html

Deve conter um formulário com action para a URL de Login e com dois inputs: username e password

Fim - Parte 3

Trabalhando com Views, Forms, Templates

Parte 4

REST API: O que de FATO importa para um Framework MVC, junto com ORM, atualmente.

Simple JSON Result

core/views.py

```python
from django.core import serializers
from .models import Disciplina

class GerenciarDisciplina(View):

    def get(self, request):
        disciplinas = Disciplina.objects.all()
        dados = serializers.serialize('json', disciplinas)
        return HttpResponse(dados, content_type='application/json')
```

Django REST Framework

django-rest-framework.org

```
$ pip install djangorestframework
```

Instalar no Projeto

core/urls.py

```
....

INSTALLED_APPS = (
    ...
    'rest_framework',
)


..
```

Generic-Views para Listar e Criar Disciplinas

core/views.py

```python
from rest_framework.generics import ListCreateAPIView
from rest_framework import serializers

class DisciplinaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Disciplina
        fields = ('id', 'nome', 'sigla', 'instituicao', 'professor')

class DisciplinaAPI(ListCreateAPIView):
    queryset = Disciplina.objects.all()
    serializer_class = DisciplinaSerializer
```

Acessando a API (Obs: Registrar a rota)

GET /api/disciplinas

(Incluso uma Browser Tool)

```
[
  {
    "id": 2,
    "nome": "PROGRAMACAO CORPORATIVA",
    "sigla": "PC",
    "instituicao": "IFPI",
    "professor": 2
  },
  {
    "id": 3,
    "nome": "PROGRAMAÇÃO ORIENTADA A OBJETOS",
    "sigla": "POO",
    "instituicao": "IFPI",
    "professor": 2
  }
]
```

Acessando a API (Obs: Registrar a rota)

POST /api/disciplinas

PAYLOAD:

```
{
   "nome": "ALGORITMOS E PROGRAMACAO",
   "sigla": "ALG",
   "instituicao": "IFPI",
   "professor": 2
 }
```

Criando a API diretamente nas Rotas

core/urls.py

```
from rest_framework.generics import ListCreateAPIView
from rest_framework.serializers import ModelSerializer

class ProfessorSerializer(ModelSerializer):
   class Meta:
       model = Professor
       fields = ('id', 'nome', 'titulo','email')

urlpatterns = [
       ...
   url(r'^api/teacher/',ListCreateAPIView.as_view(queryset=Professor.objects.all(),
serializer_class=ProfessorSerializer),  name='teacher-list'),
]
```

Serializando as relações

core/views.py

```python
class ProfessorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Professor
        fields = ('id', 'nome', 'email')

class DisciplinaSerializer(serializers.ModelSerializer):

    professor = ProfessorSerializer(read_only=True)

    class Meta:
        model = Disciplina
        fields = ('id', 'nome', 'sigla', 'instituicao', 'professor')

class DisciplinaAPI(ListCreateAPIView):
    queryset = Disciplina.objects.all()
    serializer_class = DisciplinaSerializer
```

Acessando a API (http://localhost:8000/api/disciplinas)

```
[
  {
    "id": 2,
    "nome": "PROGRAMACAO CORPORATIVA",
    "sigla": "PC",
    "instituicao": "IFPI",
    "professor": {
      "id": 2,
      "nome": "Rogério da Silva",
      "email": "rogerio410@gmail.com"
    }
  },
  {
    "id": 3,
    "nome": "PROGRAMAÇÃO ORIENTADA A OBJETOS",
    "sigla": "POO",
    "instituicao": "IFPI",
    "professor": {
      "id": 2,
      "nome": "Rogério da Silva",
      "email": "rogerio410@gmail.com"
    }
  }
]
```

Django-Rest GenericViews

- **CreateAPIView**
- **ListAPIView**
- **RetrieveAPIView**
- **DestroyAPIView**
- **UpdateAPIView**
- **ListCreateAPIView**
- **RetrieveUpdateAPIView**
- **RetrieveDestroyAPIView**
- **RetrieveUpdateDestroyAPIView**

Completando a API para Disciplinas

core/urls.py

```
from core import views


urlpatterns = [
    ...
    url(r'^api/disciplinas$', views.DisciplinaAPI.as_view()),
    url(r'^api/disciplinas/(?P<pk>\d+)/$', views.DisciplinaDetalhesAPI.as_view()),
]
```

Completando a API para Disciplinas

core/views.py

```
from rest_framework.generics import \
    ListCreateAPIView, RetrieveUpdateDestroyAPIView


class DisciplinaAPI(ListCreateAPIView):
    queryset = Disciplina.objects.all()
    serializer_class = DisciplinaSerializer


class DisciplinaDetalhesAPI(RetrieveUpdateDestroyAPIView):
    queryset = Disciplina.objects.all()
    serializer_class = DisciplinaSerializer
```

View REST costumizadas

```python
class SnippetList(APIView):
    """
    List all snippets, or create a new snippet.
    """
    def get(self, request, format=None):
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        serializer = SnippetSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

View REST costumizadas

```python
class SnippetDetail(APIView):
    """ Retrieve, update or delete a snippet instance. """
    def get_object(self, pk):
        try:
            return Snippet.objects.get(pk=pk)
        except Snippet.DoesNotExist:
            raise Http404

    def get(self, request, pk, format=None):
        snippet = self.get_object(pk)
        serializer = SnippetSerializer(snippet)
        return Response(serializer.data)

    def put(self, request, pk, format=None):
        snippet = self.get_object(pk)
        serializer = SnippetSerializer(snippet, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk, format=None):
        snippet = self.get_object(pk)
        snippet.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Demais Funcionalidades

- Autenticação:
  - BASIC, TOKEN E SESSION
- Autorização: Inclusive a nível de objetos
- Serialização e Deserialização de Relações
- Filtering: querystring
- Paginação
- Helper para Status Code
- Testável

Listando as disciplinas na index

# views.py

```python
def index(request):

    disciplinas = Disciplina.objects.all()
    ctx = {'disciplinas':disciplinas}
    return render(request, template_name='core/index.html', context=ctx)
```

# core/index.html

```html
{% block content %}
    <h1>Index TeacherFeed</h1>
    <h2>Disciplinas Disponíveis</h2>

    <ul>
        {% for d in disciplinas %}
            <li>{{ d }}</li>
        {% endfor %}
    </ul>

{% endblock %}
```