



Mini Curso de Django

10º WTADS - IFRN

Material deste mini curso

[Código-fonte](#)



[Esta apresentação](#)



Objetivos

- Mostrar o que é o Django e seus principais conceitos.
- Como fazer sua instalação e configuração;
- Apresentar o modelo MVT (Model, View, Template);
- Por a mão na massa (Aí sim!).

Pré-Requisitos

- Conhecimento básico de programação orientação a objetos (POO);
- Conhecimento básico em Python.

POO no modo "caldo de cana + pastel"



O que é?

Classificar, organizar e abstrair as coisas do mundo real (classes) em uma coleção de objetos que possuem características (dados) e realizam ações (métodos).

POO no modo "caldo de cana + pastel"

Como fazer isso?

É bem simples:

- Primeiro classifique;
- Depois organize;
- E por fim, abstraia (aqui começa a viagem...).



POO no modo "caldo de cana + pastel"

Cenário exemplo

João foi ao supermercado e botou na cesta 1 kg de arroz, 1kg de feijão preto, um pacote de 500g de linguiça toscana e 6 garrafas de cerveja. Ao final, foi ao caixa e pagou um total de 25 reais.

POO no modo "caldo de cana + pastel"

Classificando e organizando...

- João é um cliente do supermercado;
- 1 kg de arroz, ... 6 garrafas de cerveja são produtos;
- No caso há 2 tipos de produto no exemplo: alimentos e bebidas;
- João pagou pelos produtos no supermercado. No caso trata-se de uma compra na perspectiva de João e de uma venda na perspetiva do supermercado;
- Os itens comprados por João já não podem ser vendidos a outro cliente.

O que abstrair daí?

A intensidade da viagem vai depender da necessidade do demandante do sistema, ou seja, do que ele deseja controlar.

Atenção: Cuidado com a palavra SISTEMINHA! É mentira!

O que abstrair daí?

- Classes:
 - Cliente, Produto, TipoProduto, Venda e Estoque.
- Objetos:
 - do tipo Cliente: João;
 - do tipo Produto: Arroz, feijão, linguiça e cerveja;
 - do tipo TipoProduto: Alimento e bebida.
 - do tipo Venda: A venda realizada pelo supermercado a João;
 - do tipo Estoque: A baixa dos itens vendidos a João.

O que abstrair daí?

Modelagem básica:

- Cliente (nome, cpf, estado_civil, data_nascimento, e-mail, fone);
- Produto (descricao, preco);
- TipoProduto (descricao)
- Venda (data, cliente);
- Vendaltem (venda, produto, quantidade, preco);
- Estoque (produto, quantidade, tipo_registro).

Python no modo "caldo de cana"



O que é?

É uma linguagem orientada a objetos, com tipagem dinâmica e fortemente tipada.

Python no modo "caldo de cana"

O que eu faço com ela?

Faz muita coisa, nas mais diversas áreas, mas hoje vamos nos focar no Django e em escrever uma aplicação usando a abstração definida nos slides anteriores.

Python no modo "caldo de cana"



Como instalar?

- Linux (Debian):
 - \$ apt-get install python3
- Mac OS:
 - \$ brew install python3
- Windows:
 - Baixar o instalador e execute-o:
 - <https://www.python.org/downloads/>



Como instalar?

Além do python, precisamos instalar o pip, que é um gerenciador de pacotes python.

- Linux (Debian), Mac OS ou Windows:
 - `$ curl https://bootstrap.pypa.io/get-pip.py | python3`

Houston, are you there?

```
$ python3 --version  
$ pip3 --version
```



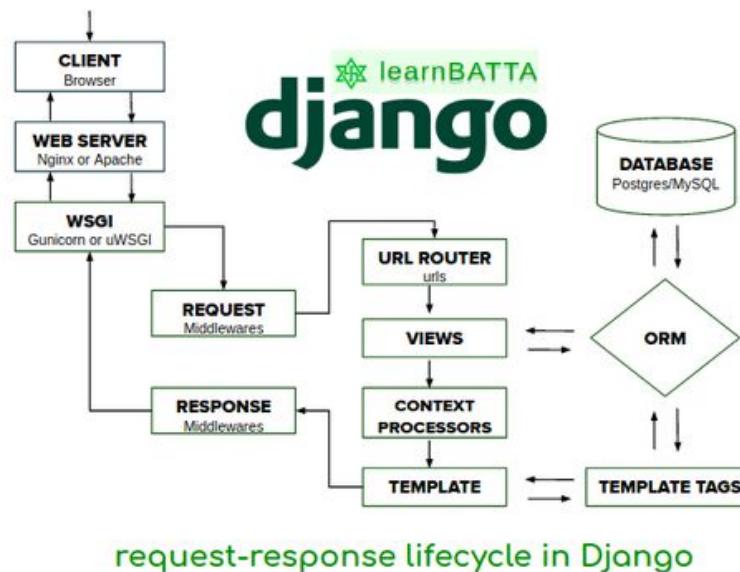
O que é o Django?

- É um framework python voltado para a criação de aplicações web, de maneira simples e ágil.
- Trabalha com o conceito de aplicações plugáveis, ou seja, reutilizáveis.
- Utiliza o padrão **MVC**, através da sigla **MVT (model, view, template)**;
 - **model** = **model**
 - **view** = **template**
 - **controller** = **view**

O que é o Django?

- Tem uma poderosa ORM;
- Oferece uma aplicação "Django Admin" bem legal e que pode ser customizada.
- Oferece um servidor de aplicação interno (`runserver`), muito útil na fase de desenvolvimento e testes. **Atenção: Ele não foi projetado para ser usado em produção, então não faça isso!**

Ciclo de vida



[Fonte da Imagem](#)

Como instalar?

\$ pip3 install django

- Caso queira especificar uma versão:
 - \$ pip3 install django==2.2.2
- Verificando os pacote que foram instalados:
 - \$ pip freeze

Obs: No caso estamos fazendo uma instalação básica. Em casa, estude sobre como criar ambiente isolados: [VirtualEnv & VirtualEnvWrapper](#).

Houston, do you hear me?

```
$ python3 -c "import django;  
print(django.get_version())"
```



Projeto

O Django trabalha com o conceito de projetos e aplicações.

Um projeto pode conter "n" aplicações.

A ideia é de que cada aplicação possa ser plugável, ou seja, reutilizável.

É no projeto que ficam as configurações "globais".

Criando um projeto

```
$ django-admin startproject mini_curso_django
```

```
$ cd mini_curso_django
```

```
$ tree
```

```
.  
└── manage.py <<< Utilitário de linha de comando do Django.  
└── mini_curso_django <<< Diretório com as configurações do projeto.  
    ├── __init__.py <<< Arquivo vazio que indica ao Python que esse diretório é um pacote.  
    ├── settings.py <<< Arquivo de configuração do projeto.  
    ├── urls.py <<< Arquivo com as urls do projeto.  
    └── wsgi.py <<< Um ponto de integração para servidores WEB compatíveis com WSGI.
```

Criando um projeto

Principais configurações...

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]  
  
# Internationalization  
# https://docs.djangoproject.com/en/2.1/topics/i18n/  
# Language code for this installation. All choices  
# http://www.i18nguy.com/unicode/language-identifiers/  
# LANGUAGE_CODE = 'en-us'  
LANGUAGE_CODE = 'pt-BR'  
  
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}  
  
# Local time zone for this installation. Choices can be found here:  
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name  
# Although not all choices may be available on all operating systems.  
# In a Windows environment this must be set to your system time zone.  
# TIME_ZONE = 'UTC'  
TIME_ZONE = 'America/Recife'
```

Criando um projeto

Antes de fazer o primeiro teste com o projeto, vamos criar toda a estrutura de banco necessária para as aplicações que já vêm definidas, por padrão em `settings.INSTALLED_APPS`:

```
$ python3 manage.py migrate
```

- ▶  auth_group
- ▶  auth_group_permissions
- ▶  auth_permission
- ▶  auth_user
- ▶  auth_user_groups
- ▶  auth_user_user_permissions
- ▶  django_admin_log
- ▶  django_content_type
- ▶  django_migrations
- ▶  django_session

... three, two, one! Go!



\$ python3 manage.py runserver 8000

Django: o framework web para perfe*x*

localhost:8000

120% Ver as notas de lançamento do Django 2.2

Ver as notas de lançamento do Django 2.2



A instalação foi com sucesso! Parabéns!

Você está vendo esta página pois possui DEBUG=True no seu arquivo de configurações e não configurou nenhuma URL.

 [Documentação do Django](#)
Tópicos, referências, & how-to's

 [Tutorial: Um aplicativo de votação](#)
Comece a usar Django

 [Comunidade Django](#)
Conecte-se, obtenha ajuda ou contribua

X pt-BR ▲ ▼ Destacar todas Diferenciar maiúsculas/minúsculas Palavras completas

Aplicação

Como já foi dito, a ideia é que cada aplicação seja plugável, ou seja, possa ser utilizando em conjunto com outras aplicações.

Levando em conta a nossa abstração, poderíamos imaginar, por exemplo, 3 aplicações:

- Atendimento: Módulo voltando inicialmente ao controle dos clientes;
- Vendas: Módulo focado na parte do registro das vendas;
- Estoque: Módulo focado no controle dos produtos e seus estoques.

Criando uma aplicação

Uma vez dentro do diretório do projeto `mini_curso_django`, execute:

```
$ django-admin startapp atendimento
```

```
$ tree
```

```
.  
└── atendimento  
    ├── __init__.py  
    ├── admin.py      <<< Local aonde ficam as definições de admin sobre cada modelo.  
    ├── apps.py       <<< Local aonde fica a classe de configuração da aplicação.  
    ├── migrations   <<< Aqui ficam todos os "scripts" acerca do banco de dados da aplicação.  
    │   └── __init__.py  
    ├── models.py     <<< Local aonde ficam as classes do modelo.  
    ├── tests.py  
    ├── views.py      <<< Aqui fica os métodos do controle da aplicação.  
    └── db.sqlite3    <<< Banco de dados do projeto, utilizando, no caso, SQLite.  
    └── manage.py  
    └── mini_curso_django <<< Diretório com as configurações do projeto.  
        └── ...
```

Criando uma aplicação

Uma vez criada a aplicação, se faz necessário informar ao Django que esta aplicação existe e que ela vai ser "plugada" ao projeto.

`mini_curso_django/mini_curso_django/settings.py`

```
# Application definition

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

Criando e entendendo uma view

Como já dissemos anteriormente, a camada **view** do Django (modelo MVT) equivale a camada **controller** do modelo MVC. Abaixo segue exemplos de duas views bem simples:

`mini_curso_django/atendimento/views.py`

```
from django.http import HttpResponse

# Exemplo de controle que retorna como resposta um texto simples.
def index(request):
    return HttpResponse('Hi Houston! It s all right!')

# Exemplo de controle que retorna como resposta um html simples.
def index2(request):
    html = '<html><body style="color: red">Hi Houston! It s all right!</body></html>'
    return HttpResponse(html)
```

Criando e entendendo uma view

Para que o Django saiba onde está o recurso solicitado na requisição, é preciso fazer o devido mapeamento da url.

`mini_curso_django/urls.py`

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

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

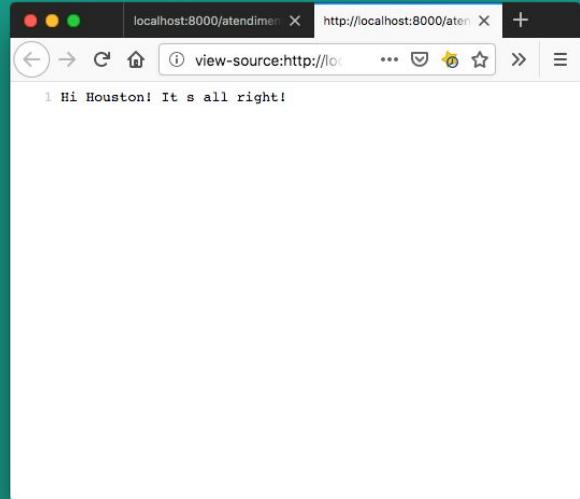
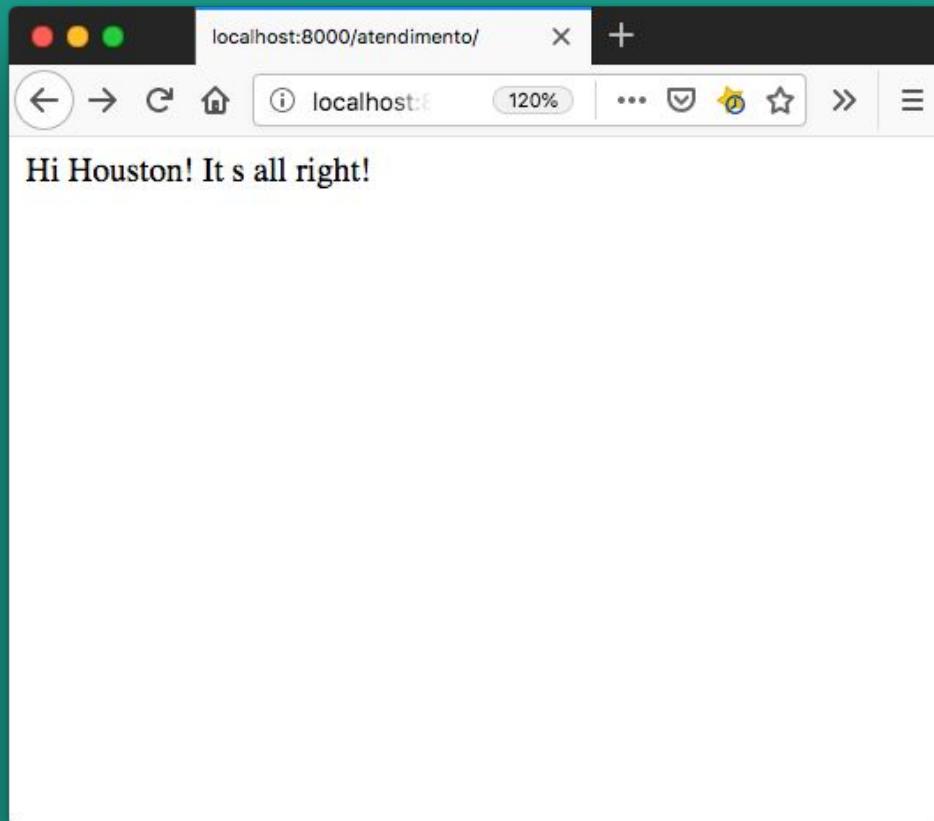
`mini_curso_django/atendimento/urls.py`

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

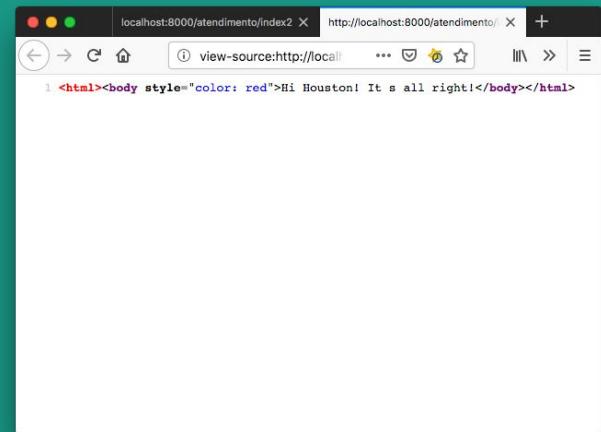
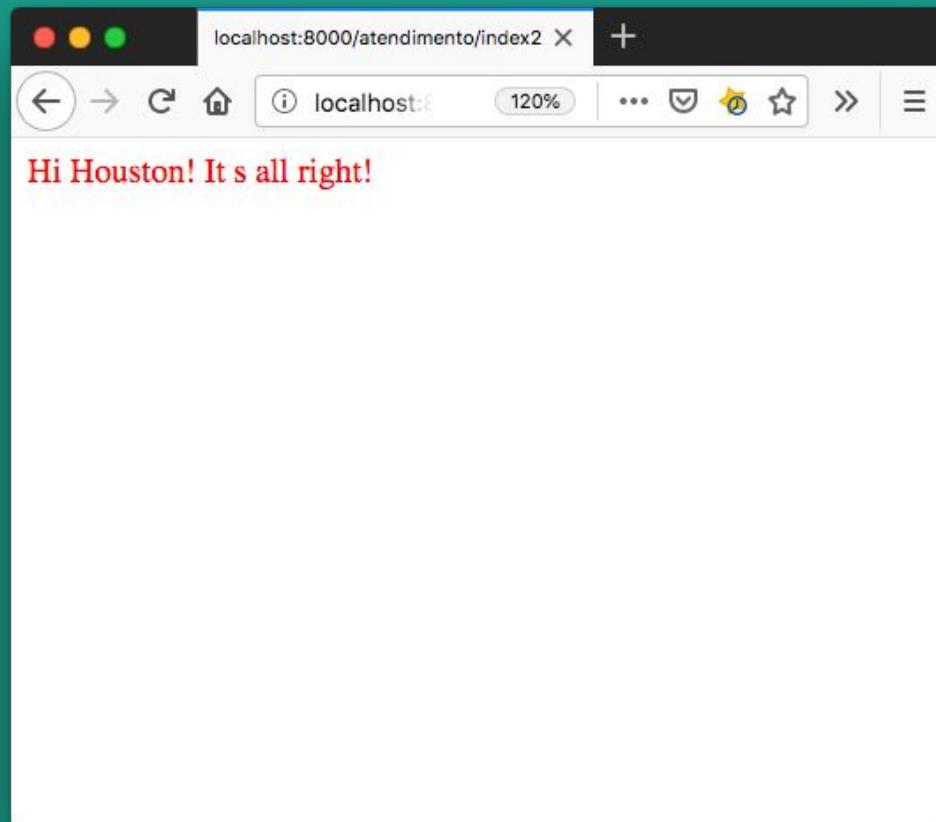
urlpatterns = [
    path('', views.index, name='index'),
    path('index', views.index, name='index'),
    path('index2', views.index2, name='index2'),
]
```

Let's view the view?

```
$ python3 manage.py runserver 8000
```



http://localhost:8000/atendimento/



<http://localhost:8000/atendimento/index2>



Fixando o funcionamento de uma view

- Browser dispara a requisição (HttpRequest) para determinada url do servidor;
- A engine do Django faz o direcionamento para a view correta, de acordo com o urls.py do projeto e da aplicação;
- A view é processada;
- A resposta (HttpResponse) é retornada. No nosso exemplo um retorno foi um texto simples e outro retorno foi um html.
- Uma view pode ou não estar associada a um template.

Todo mundo tranquilo?

\$ telnet myhead 8000

Trying 127.0.0.1...

$6 \times 9 = 54\dots$

Connected e entendendo....

Escape character is '^]'.



Criando e entendendo uma template

No caso do Django (modelo MVT), a camada **template** equivale a camada **view** do modelo MVC.

Um template nada mais é que um arquivo texto (geralmente .html) que possui variáveis e um conjunto de template tags que podem ser usadas.

Criando e entendendo uma template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>Houston, now is {{ current_date }}.</h1>

    {% if idade %}
        <p>Minha idade é {{ idade }}.</p>
    {% endif %}

    <p>Obs: Template em arquivo.</p>
</body>
</html>
```

Variáveis:

{{ current_date }}

{{ idade }}

Template Tags:

{% if %}

Criando e entendendo uma template

mini_curso_django/atendimento/views.py

```
import datetime

from django.http import HttpResponse
from django.template import Template, Context, loader
from django.shortcuts import render

# Exemplo de controle que retorna como resposta um html simples através de um template criado em tempo de execução.
def index3(request):
    now = datetime.datetime.now()

    # Obs: A data já é exibida de acordo com a configuração realizada no
    # Django: It timeeeeeee 3 de Julho de 2014 às 11:59.
    template = Template('<html><body>Houston, now is {{ current_date }}.</body></html>')
    # Contexto representa o conjunto de variáveis que serão processadas pelo template.
    context = Context({'current_date': now})
    html = template.render(context)

    return HttpResponse(html)
```

Criando e entendendo uma template

mini_curso_django/atendimento/views.py

```
# Exemplo de controle que retorna como resposta um html simples através de um template armazenado em arquivo.
def index4a(request):
    # Obs: Para o loader funcionar, se faz necessário que a aplicação esteja listada no settings.INSTALLED_APPS.
    template = loader.get_template('atendimento/index4.html')
    context = {
        'current_date': datetime.datetime.now(),
    }
    return HttpResponseRedirect(template.render(context, request))

# Exemplo de controle, que recebe opcionalmente o parâmetro idade retorna como resposta um html simples através de um
# template armazenado em arquivo.
def index4b(request, idade=None):
    current_date = datetime.datetime.now()
    # locals() retorna um dicionário contendo todos as variáveis locais disponíveis.
    # print(locals())
    return render(request, 'atendimento/index4.html', locals())
```

Criando e entendendo uma template

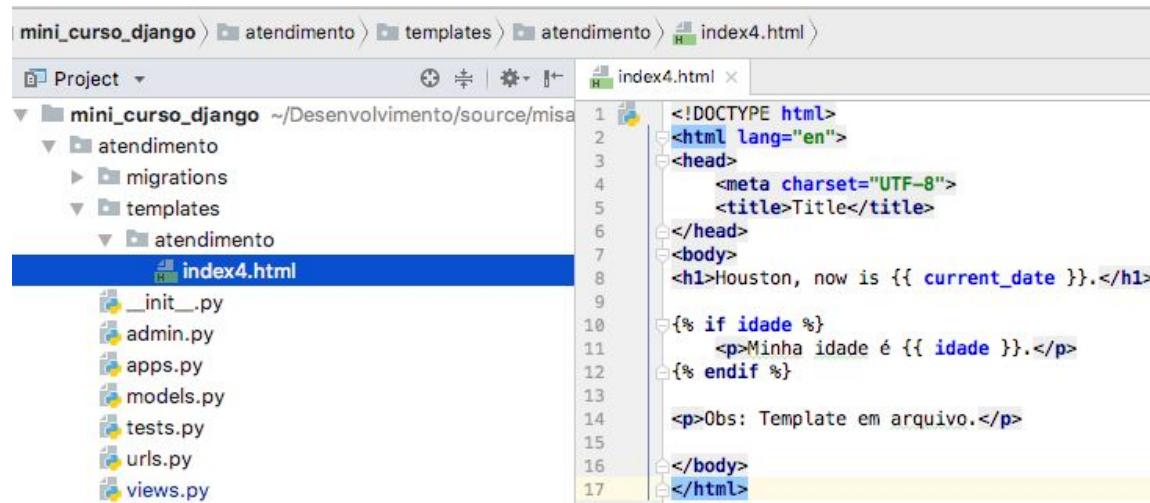
mini_curso_django/atendimento/urls.py

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

urlpatterns = [
    path('', views.index, name='index'),
    path('index', views.index, name='index'),
    path('index2', views.index2, name='index2'),
    path('index3', views.index3, name='index3'),
    path('index4a', views.index4a, name='index4a'),
    path('index4b/<int:idade>', views.index4b, name='index4b'),
```

Criando e entendendo uma template

mini_curso_django/atendimento/templates/atendimento/index4.html



The screenshot shows a code editor interface with a navigation bar at the top displaying the file path: mini_curso_django > atendimento > templates > atendimento > index4.html. The main area has two panes: a tree view on the left and the code editor on the right.

Tree View (Left):

- mini_curso_django (~/Desenvolvimento/source/misa)
 - atendimento
 - migrations
 - templates
 - atendimento
- index4.html

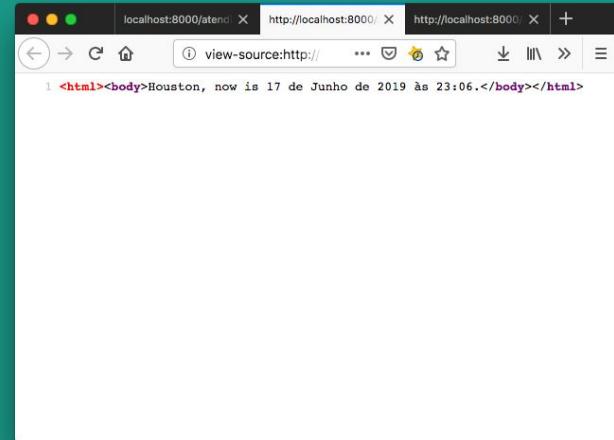
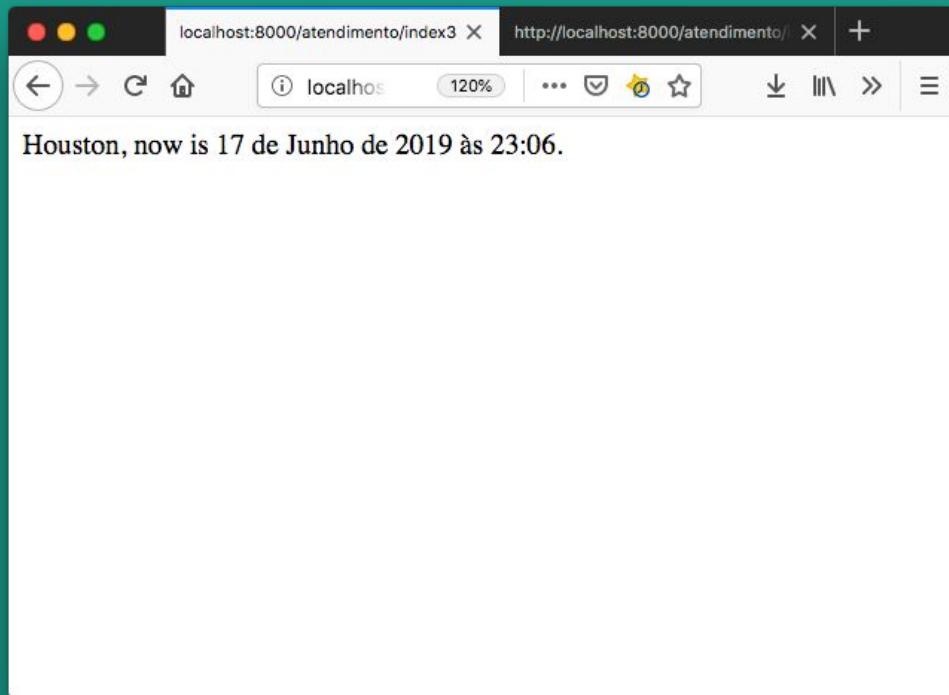
Code Editor (Right):

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6   </head>
7   <body>
8     <h1>Houston, now is {{ current_date }}.</h1>
9
10    {% if idade %}
11      <p>Minha idade é {{ idade }}.</p>
12    {% endif %}
13
14    <p>Obs: Template em arquivo.</p>
15
16  </body>
17 </html>
```

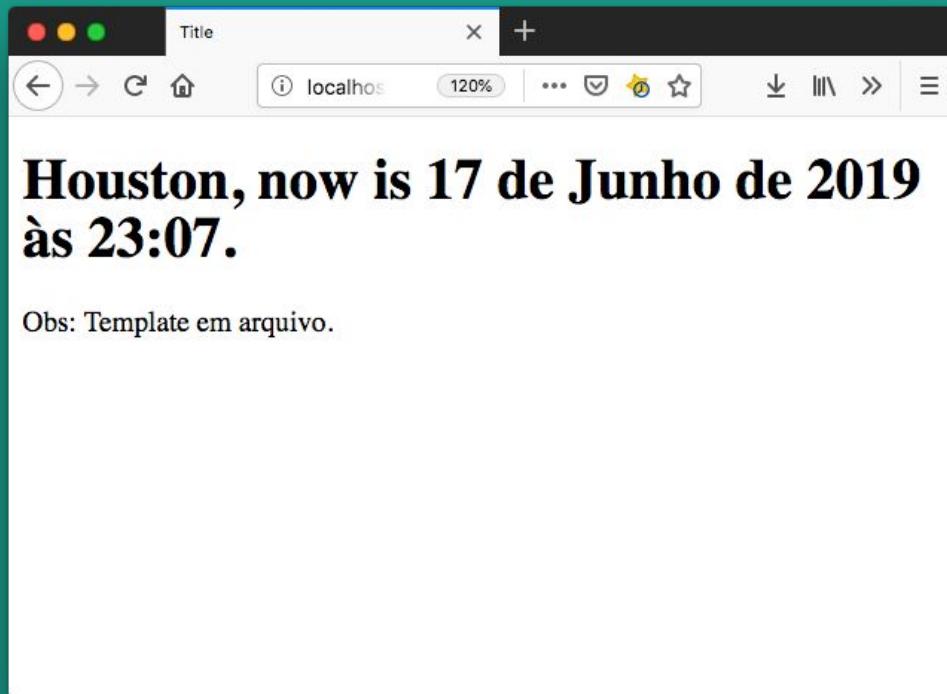
View + Template...
"Juntos e shallow now..."

\$ python3 manage.py runserver 8000





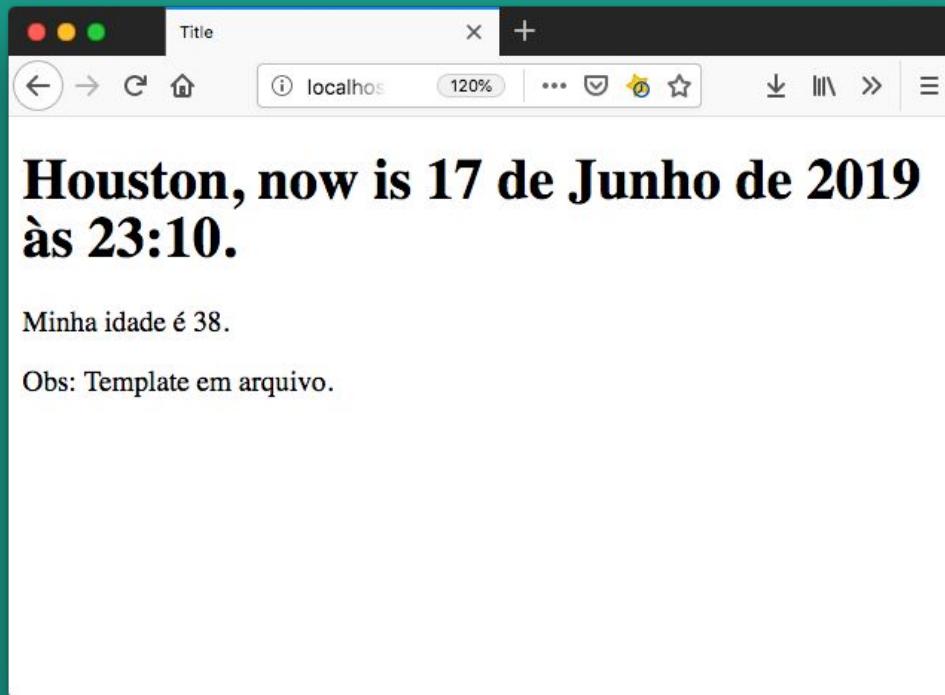
http://localhost:8000/atendimento/index3



A screenshot of a browser window showing the page source code at "http://localhost:8000/atendimento/index4a". The source code is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>Houston, now is 17 de Junho de 2019 às 23:07.</h1>
    <p>Obs: Template em arquivo.</p>
</body>
</html>
```

<http://localhost:8000/atendimento/index4a>



A screenshot of a browser window showing the page source code at "http://localhost:8000/atendimento/index4b/?". The source code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <h1>Houston, now is 17 de Junho de 2019 às 23:10.</h1>
9
10  <p>Minha idade é 38.</p>
11
12
13
14 <p>Obs: Template em arquivo.</p>
15
16 </body>
17 </html>
```

<http://localhost:8000/atendimento/index4b/?>

Fixando o funcionamento de uma template

- Browser dispara a requisição (HttpRequest) para determinada url do servidor;
- A engine do Django faz o direcionamento para a view correta, de acordo com o urls.py do projeto e da aplicação;
- Inicia-se o processamento da view.
- Durante o processamento da view, o template que será utilizado é identificado, um conjunto de variáveis é fornecido, ele é processado e retorna um html.
- A resposta (HttpResponse) é retornada. No caso é html gerado via template.

Fixando o funcionamento de uma template

- Os templates podem ser **criados em tempo de execução** (não recomendado) ou **carregados a partir de arquivos** (recomendado).
- Além de podermos criar templates, o Django nos permite sobrescrever os templates de outras aplicações, como por exemplo os do "Django Admin" (`django.contrib.admin`) que veremos mais à frente.

Mas e aí major? Entendi bem o conceito da view, do template, mas...

Cadê o meu sistema, rodando, de maneira simples é ágil?

Calma aí "dezanos"! Ainda falta a camada dos modelos. Rapadura é doce mas né mole não!



Criando e entendendo um model

No caso do Django (modelo MVT), a camada **model** equivale a camada **model** do modelo MVC. *Essa foi fácil hein?*

É nessa camada que iremos definir todas as classes do domínio do negócio da aplicação.

Criando e entendendo um model

mini_curso_django/atendimento/models.py

```
from django.db import models

# Create your models here.

class Cliente(models.Model):
    ESTADO_CIVIL_CHOICES = (
        ('SOLTEIRO', u'Solteiro'),
        ('CASADO', u'Casado'),
        ('DIVORCIADO', u'Divorciado'),
        ('VIUV', u'Viúvo'),
    )

    nome = models.CharField(verbose_name='Nome', max_length=100)
    estado_civil = models.CharField(verbose_name='Estado Civil', max_length=10, choices=ESTADO_CIVIL_CHOICES)
    cpf = models.CharField(verbose_name='CPF', max_length=14)
    data_nascimento = models.DateField(blank=True, null=True)
    email = models.EmailField(max_length=100, blank=True, null=True)

    class Meta:
        ordering = ['nome']

    # Label padrão que será exibido em toda o admin do Django.
    def __str__(self):
        return self.nome
```



Criando e entendendo um model

Uma vez definido o model, precisamos criar o "script" que cria a tabela do banco de dados que irá armazenar suas informações. Para isso basta executar:

```
$ python3 manage.py makemigrations atendimento
```

Uma vez executado o comando acima, o Django vai detectar as mudanças realizadas no *models.py* da aplicação e gerar o devido "script" (migration). A ORM do Django se encarregará de traduzir esses scripts em SQLs, de acordo com o banco utilizado. Ex: SQLite, PostgreSql, MySql, Microsoft Sql Server, Oracle.

Criando e entendendo um model

```
# Generated by Django 2.2.2 on 2019-06-15 23:04

from django.db import migrations, models

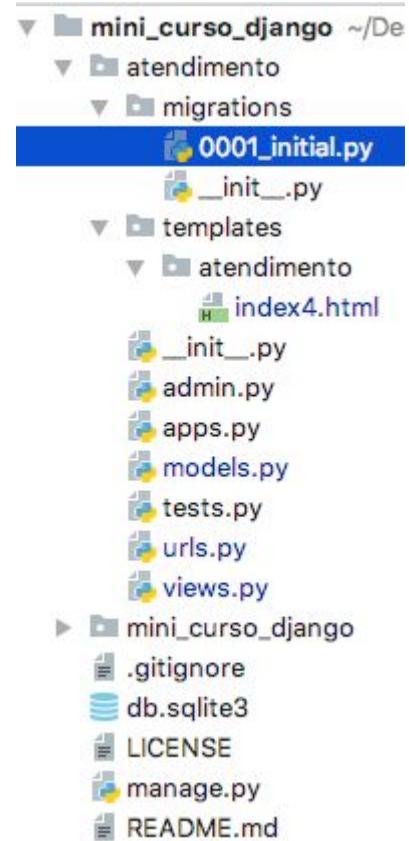
class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Cliente',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('nome', models.CharField(max_length=100, verbose_name='Nome')),
                ('estado_civil', models.CharField(choices=[('SOLTEIRO', 'Solteiro'), ('CASADO', 'Casado'), ('DIVORCIADO', 'Divorciado')], max_length=100)),
                ('cpf', models.CharField(max_length=14, verbose_name='CPF')),
                ('data_nascimento', models.DateField(blank=True, null=True)),
                ('email', models.EmailField(blank=True, max_length=100, null=True)),
            ],
            options={
                'ordering': ['nome'],
            },
        ),
    ],

```





Criando e entendendo um model

Para ver o SQL que será executado, quando a migration for aplicada, use o comando:

```
$ python3 manage.py sqlmigrate atendimento 0001
```

Sintaxe: \$ python3 manage.py sqlmigrate [aplicacao] [número ou nome da migration]

Para aplicar as alterações no banco, execute:

```
$ python3 manage.py migrate
```

Pronto, a partir de agora temos o nosso modelo pronto para ser usado!



Cadastrando um cliente via terminal

Agora que o *model* Cliente está pronto, vamos fazer o nosso primeiro cadastro, via terminal, já utilizando um pouco do poderio da ORM do Django.

Obs: Para entender mais sobre a ORM do Django, como fazer consultas, inserts, updates, deletes etc, procure sobre Querysets:

<https://docs.djangoproject.com/en/2.2/topics/db/queries/>

<https://docs.djangoproject.com/en/2.2/ref/models/querysets/>

<https://docs.djangoproject.com/en/2.2/ref/models/lookups/>

Cadastrando um cliente via terminal

```
$ python3 manage.py shell
```

```
Python 3.7.3 (default, Mar 27 2019, 09:23:39)
```

```
[Clang 10.0.0 (clang-1000.11.45.5)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
(InteractiveConsole)
```

```
>>> from atendimento.models import Cliente
```

```
>>> Cliente.objects.all()
```

```
<QuerySet []>
```



Cadastrando um cliente via terminal

...

```
>>> c = Cliente(nome='Jhon Rambo', estado_civil='SOLTEIRO', cpf='871.536.330-90')
```

```
>>> type(c)
```

```
<class 'atendimento.models.Cliente'>
```

```
>>> c.id
```

(Não retorna nada pois o objeto ainda não foi persistido)



Cadastrando um cliente via terminal

```
...
```

```
>>> c.save()
```

```
>>> c.id
```

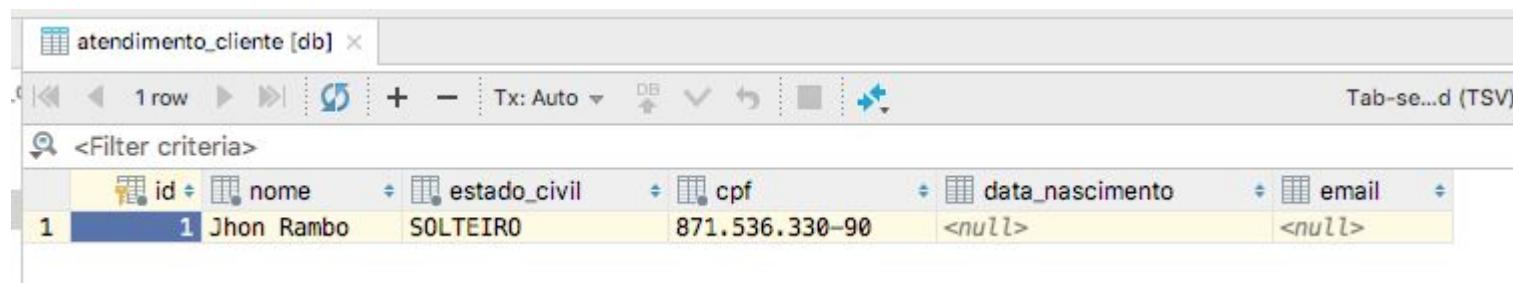
```
1
```

```
>>> Cliente.objects.all()
```

```
<QuerySet [<Cliente: Jhon Rambo>]>
```

Cadastrando um cliente via terminal

Depois de executado o método "save()", o objeto cliente foi persistindo no banco de dados:



The screenshot shows a MySQL Workbench interface with a database named 'atendimento_cliente'. A single row has been inserted into the table, which contains columns for id, nome, estado_civil, cpf, data_nascimento, and email. The row details are: id=1, nome='Jhon Rambo', estado_civil='SOLTEIRO', cpf='871.536.330-90', data_nascimento='<null>', and email='<null>'.

| | id | nome | estado_civil | cpf | data_nascimento | email |
|---|----|------------|--------------|----------------|-----------------|--------|
| 1 | 1 | Jhon Rambo | SOLTEIRO | 871.536.330-90 | <null> | <null> |

**Major, cadê o sistema? "Avie homi"
que eu tô com fome e ainda vou
pegar o "buzão"...**

**Deixe de aperreio "dezanos"! Se
aquele que vou te mostrar agora!**



Django Admin

Trata-se de uma aplicação web (site) através do qual será possível gerenciar os seus modelos. Esta aplicação, associada a outras, também oferecidas pelo Django, irão disponibilizar a você:

- Um site administrativo (`django.contrib.admin`);
- Um sistema de autenticação (`django.contrib.auth`);
- Um framework para tipos de conteúdo (`django.contrib.contenttypes`);
- Um framework de sessão (`django.contrib.sessions`);
- Um framework de envio de mensagem (`django.contrib.messages`);
- Um framework para gerenciamento de arquivos estáticos (`django.contrib.staticfiles`).



Django Admin

Por padrão ele já vem plugado, ou seja, já está definido em `settings.INSTALLED_APPS`, logo para poder utilizá-lo basta criar um superusuário (administrador geral do projeto):

```
$ python3 manage.py createsuperuser
```

```
Username (leave blank to use 'so_username'): admin
```

```
Email address:
```

```
Password: senha
```

```
Password (again): senha
```

```
This password is too short. It must contain at least 8 characters.
```

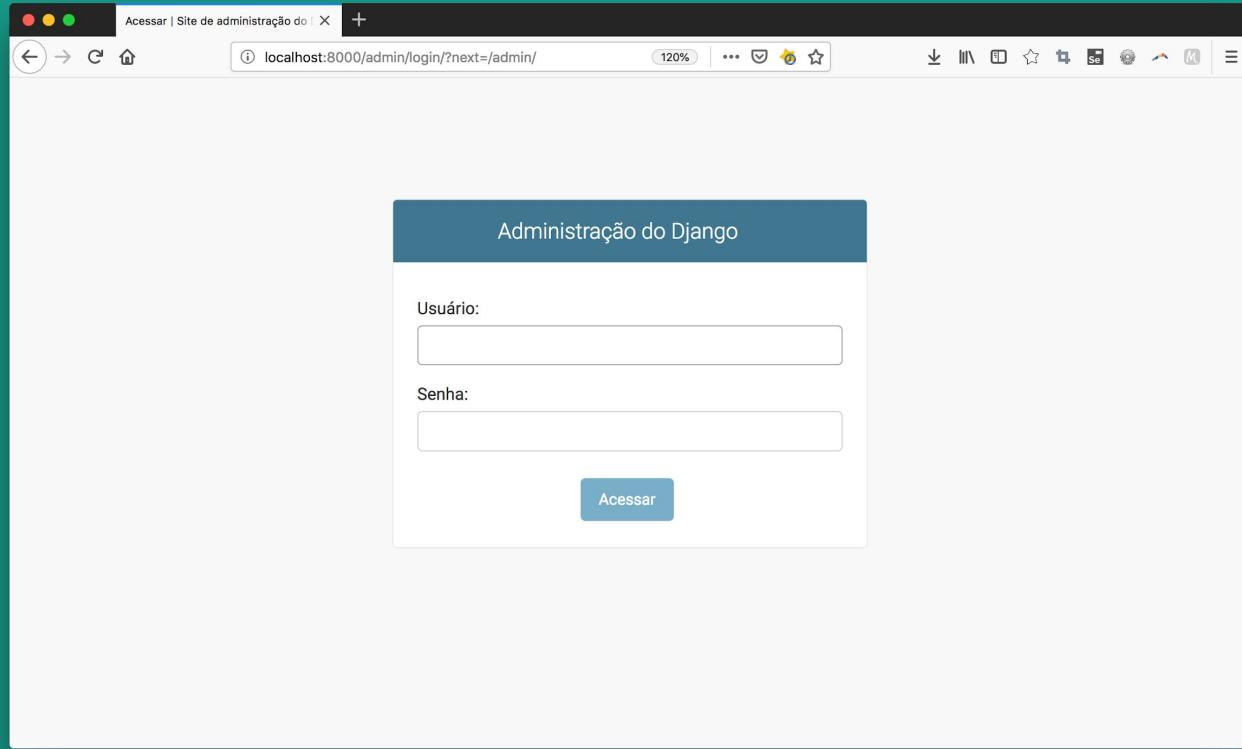
```
This password is too common.
```

```
Bypass password validation and create user anyway? [y/N]: y
```

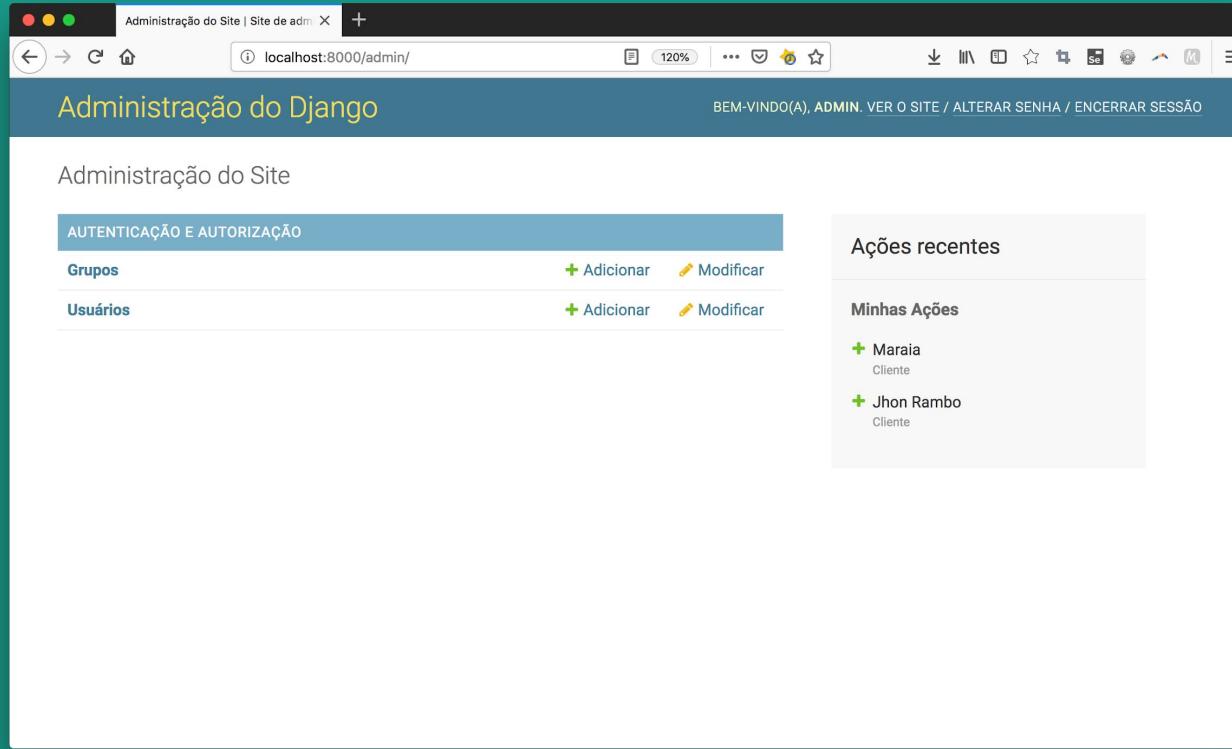
Run as administrator...



\$ python3 manage.py runserver 8000



http://localhost:8000/admin/



`http://localhost:8000/admin/`

Selecionar usuário para modificar | X +

localhost:8000/admin/auth/user/ 120% ... 🔍 M

Administração do Django

BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Início > Autenticação e Autorização > Usuários

Selecionar usuário para modificar

Pesquisar

Ação: 0 de 1 selecionados

| <input type="checkbox"/> | USUÁRIO | ENDEREÇO DE EMAIL | PRIMEIRO NOME | ÚLTIMO NOME | MEMBRO DA EQUIPE |
|-------------------------------------|---------|-------------------|---------------|-------------|-------------------------------------|
| <input checked="" type="checkbox"/> | admin | | | | <input checked="" type="checkbox"/> |

1 usuário

FILTRO

Por membro da equipe

- Todos
- Sim
- Não

Por status de superusuário

- Todos
- Sim
- Não

Por ativo

- Todos
- Sim
- Não

ADICIONAR USUÁRIO +

<http://localhost:8000/admin/auth/user/>

Modificar usuário | Site de administ... X +

localhost:8000/admin/auth/user/1/change/

Filtro

admin | entrada de log | Can add log entry
admin | entrada de log | Can change log entry
admin | entrada de log | Can delete log entry
admin | entrada de log | Can view log entry
atendimento | cliente | Can add cliente
atendimento | cliente | Can change cliente
atendimento | cliente | Can delete cliente
atendimento | cliente | Can view cliente
auth | grupo | Can add group
auth | grupo | Can change group
auth | grupo | Can delete group
auth | grupo | Can view group
auth | permissão | Can add permission
auth | permissão | Can change permission

Escolher todos 

Remover todos 

Permissões específicas para este usuário. Mantenha pressionado o "Control", ou "Command" no Mac, para selecionar mais de uma opção.

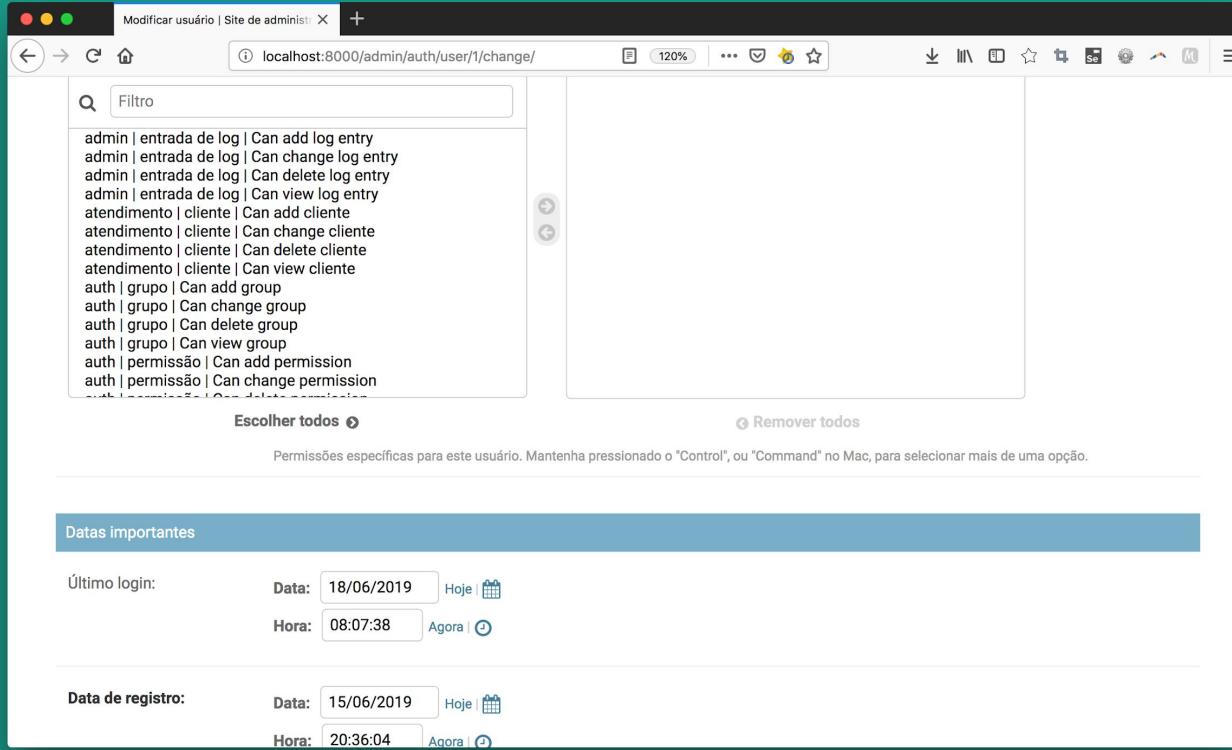
Datas importantes

Último login: Data: 18/06/2019 Hoje 

Hora: 08:07:38 Agora 

Data de registro: Data: 15/06/2019 Hoje 

Hora: 20:36:04 Agora 



<http://localhost:8000/admin/auth/user/1/change/>



Django Admin

De cara, ganhamos uma autenticação via banco. Caso queira, outros tipos de autenticação podem ser realizadas, com por exemplo via LDAP (Active Directory, OpenLDAP, Apache Directory Server...) ou integrado com outras aplicações, como por exemplo o "social login" (Facebook, Instagram, Twitter, Github, Linkedin...).

Quanto à autorização, ou seja, o que o usuário poderá fazer em cada aplicação, o Django Admin oferece um controle de grupos e usuários muito bom e fácil de utilizar. Além das permissões padrão de cada modelo (add, change, delete e view), é possível criar permissões personalizadas (Ex: usuario_pode_extornar_cobranca).

Gerenciando um model no "Django Admin"

Para que possamos utilizar o Django Admin para gerenciar os nossos models, precisamos primeiro informar a ele que models queremos que ele gerencie. Para isso, basta registrar o respectivo model junto ao Django Admin.

`mini_curso_django/atendimento/admin.py`

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

# Register your models here.

# Exemplo 1: Registrar um modelo junto ao Django Admin.
admin.site.register(Cliente)
```

The screenshot shows the Django Admin interface running on a local host. The top navigation bar includes the title "Administração do Site | Site de adm", the URL "localhost:8000/admin/", and various browser control icons. The main content area has a dark blue header with the text "BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO". Below this, the page is titled "Administração do Site".

ATENDIMENTO

| Clientes | + Adicionar | Modificar |
|----------|-----------------------------|---------------------------|
|----------|-----------------------------|---------------------------|

AUTENTICAÇÃO E AUTORIZAÇÃO

| Grupos | + Adicionar | Modificar |
|----------|-----------------------------|---------------------------|
| Usuários | + Adicionar | Modificar |

Ações recentes

Minhas Ações

- [+ Maraia
Cliente](#)
- [+ Jhon Rambo
Cliente](#)

<http://localhost:8000/admin/>

The screenshot shows a web browser window with the title "Selezione cliente para modificar | S X". The address bar displays "localhost:8000/admin/atendimento/cliente/". The page header reads "Administração do Django" and "BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO". The breadcrumb navigation shows "Início > Atendimento > Clientes". The main content area is titled "Selezione cliente para modificar" and features a "ADICIONAR CLIENTE +". Below this, there is a search bar with "Ação: -----" and a "Ir" button, followed by the message "0 de 1 selecionados". A table lists one client: "CLIENTE" and "Jhon Rambo". A note at the bottom states "1 cliente".

<http://localhost:8000/admin/atendimento/cliente/>

Modificar cliente | Site de administração

localhost:8000/admin/atendimento/cliente/1/change/ 120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes > Jhon Rambo

Modificar cliente

HISTÓRICO

Nome: Jhon Rambo

Estado Civil: Solteiro

CPF: 871.536.330-90

Data nascimento: Hoje

Email:

[Apagar](#) [Salvar e adicionar outro\(a\)](#) [Salvar e continuar editando](#) [SALVAR](#)

The screenshot shows a Django Admin modification page for a client named 'Jhon Rambo'. The page includes fields for Name, Civil Status (set to 'Solteiro'), CPF ('871.536.330-90'), Birth Date ('Hoje'), and Email. At the bottom, there are four buttons: 'Apagar' (Delete), 'Salvar e adicionar outro(a)' (Save and add another), 'Salvar e continuar editando' (Save and continue editing), and a large blue 'SALVAR' button.

<http://localhost:8000/admin/atendimento/cliente/1/change/>

Adicionar cliente | Site de administração

localhost:8000/admin/atendimento/cliente/add/ 120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes > Adicionar cliente

Adicionar cliente

Por favor corrija o erro abaixo

Nome: Maria da Silva

Estado Civil: Divorciado

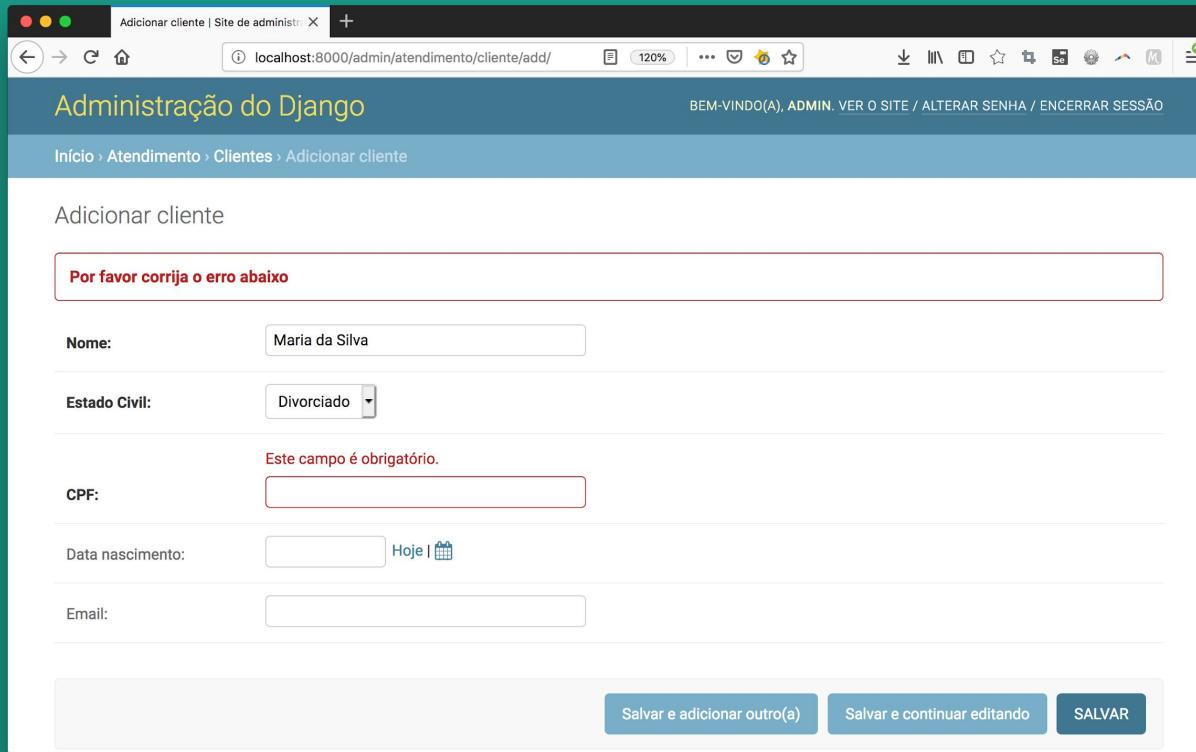
Este campo é obrigatório.

CPF:

Data nascimento: Hoje | 

Email:

Salvar e adicionar outro(a) Salvar e continuar editando SALVAR



The screenshot shows a Django Admin interface for adding a client. The page title is 'Adicionar cliente | Site de administração'. The URL in the address bar is 'localhost:8000/admin/atendimento/cliente/add/'. The top right corner shows the message 'BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO'. The main header is 'Administração do Django'. Below it, the breadcrumb navigation is 'Início > Atendimento > Clientes > Adicionar cliente'. The main title is 'Adicionar cliente'. A red box highlights an error message: 'Por favor corrija o erro abaixo'. The 'Nome' field contains 'Maria da Silva'. The 'Estado Civil' dropdown is set to 'Divorciado'. The 'CPF' field has a red border, indicating it is a required field. The 'Data nascimento' field has a red border and contains 'Hoje | '. The 'Email' field has a red border. At the bottom, there are three buttons: 'Salvar e adicionar outro(a)' (Save and add another), 'Salvar e continuar editando' (Save and continue editing), and a large blue button labeled 'SALVAR' (Save).

<http://localhost:8000/admin/atendimento/cliente/add/>

Ah major! Agora sim, eu vi vantagem!

**Mas major, e se eu quiser fazer alguma
customização, eu posso?**



"Oxe", é claro "dezanos"! Espia aí...



Gerenciando um model no "Django Admin"

Com o Django Admin é possível realizar diversos níveis de customização, desde de alterar a disposição das informações, definir o que deve ou não ser exibido, filtros, quais campos devem ser utilizados na busca, até a nível de apresentação do site (html, css).

A seguir vamos apresentar o primeiro nível de customização, realizado através de uma simples classe que herda de *admin.ModelAdmin*.

Para ver mais opções de customização via *admin.ModelAdmin*, acesse:

<https://docs.djangoproject.com/en/2.2/intro/tutorial07/>

Gerenciando um model no "Django Admin"

mini_curso_django/atendimento/admin.py

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

# Register your models here.

# Exemplo 2: Registrar um modelo junto ao Django Admin e fazendo customizações básicas.
class ClienteAdmin(admin.ModelAdmin):
    fields = ['nome', 'cpf', 'estado_civil']
    search_fields = ['nome', 'cpf']
    list_filter = ['estado_civil']
    list_display = ('nome', 'cpf', 'estado_civil', 'data_nascimento', 'email')
    list_display_links = ('nome',)

admin.site.register(Cliente, ClienteAdmin)
```

Selecionar cliente para modificar | S X +

localhost:8000/admin/atendimento/cliente/ 120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes

Selecionar cliente para modificar

Ação: ----- Ir 0 de 1 selecionados

| <input type="checkbox"/> | NOME | CPF | ESTADO CIVIL | DATA NASCIMENTO | EMAIL |
|--------------------------|------------|----------------|--------------|-----------------|-------|
| <input type="checkbox"/> | Jhon Rambo | 871.536.330-90 | Solteiro | - | - |

FILTRO

Por Estado Civil

Todos
Solteiro
Casado
Divorciado
Viúvo

1 cliente

<http://localhost:8000/admin/atendimento/cliente/>

Modificar cliente | Site de administração

localhost:8000/admin/atendimento/cliente/1/change/ 120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes > Jhon Rambo

Modificar cliente

HISTÓRICO

Nome: Jhon Rambo

CPF: 871.536.330-90

Estado Civil: Solteiro

[Apagar](#) [Salvar e adicionar outro\(a\)](#) [Salvar e continuar editando](#) [SALVAR](#)

The screenshot shows a Django Admin interface for modifying a client. The top navigation bar includes links for 'Início', 'Atendimento', 'Clientes', and the specific record 'Jhon Rambo'. The main title is 'Modificar cliente'. Below the title, there's a 'HISTÓRICO' button. The form contains three fields: 'Nome' (Name) with value 'Jhon Rambo', 'CPF' (Document Number) with value '871.536.330-90', and 'Estado Civil' (Civil Status) with value 'Solteiro' (Single). At the bottom of the form are four buttons: a red 'Apagar' (Delete) button, and three blue 'Salvar' (Save) buttons labeled 'Salvar e adicionar outro(a)', 'Salvar e continuar editando', and a primary 'SALVAR' button.

<http://localhost:8000/admin/atendimento/cliente/1/change/>

Gerenciando um model no "Django Admin"

Além das validações básicas, como campos obrigatórios, únicos, é possível também realizar validações customizadas. Abaixo temos um exemplo disso:

Classe Cliente - mini_curso_django/atendimento/models.py

```
# Método que faz validações customizadas. O ideal é que toda a validação do modelo se concentre aqui, e não em views
# ou forms, assim temos o código em um só local e facilitamos a sua manutenção.
# ModelForms invocam o método clean() antes de chamar o método save(). Obs: Admin utiliza internamente um ModelForm.
def clean(self):
    if self.nome:
        self.nome = self.nome.strip()
        if len(self.nome.split(' ')) < 2:
            raise ValidationError({'nome': 'Nome e sobrenome são requeridos.'})
    super().clean()
```

Modificar cliente | Site de administração

localhost:8000/admin/atendimento/cliente/2/change/

120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes > Maraia

Modificar cliente

HISTÓRICO

Por favor corrija o erro abaixo

Nome e sobrenome são requeridos.

Nome: Maraia

CPF: 111.111.111-11

Estado Civil: Divorciado

[Apagar](#) [Salvar e adicionar outro\(a\)](#) [Salvar e continuar editando](#) [SALVAR](#)

<http://localhost:8000/admin/atendimento/cliente/2/change/>

Modificar cliente | Site de administração

localhost:8000/admin/atendimento/cliente/2/change/

120% BEM-VINDO(A), ADMIN. VER O SITE / ALTERAR SENHA / ENCERRAR SESSÃO

Administração do Django

Início > Atendimento > Clientes > Maraia Maraísa

O cliente "Maraia Maraísa" foi alterado com sucesso. Você pode modificar ele novamente abaixo.

Modificar cliente

HISTÓRICO

Nome: Maraia Maraísa

CPF: 111.111.111-11

Estado Civil: Divorciado

Apagar **Salvar e adicionar outro(a)** **Salvar e continuar editando** **SALVAR**

<http://localhost:8000/admin/atendimento/cliente/2/change/>

Customizando templates do "Django Admin"

A seguir segue um exemplo de customização a nível de templates. No caso vamos personalizar a tela de login do Django Admin. Para isso, basta:

- Definir uma pasta para armazenar os templates do Django Admin que serão sobreescritos.
 - Ex: mini_curso_django/templates/admin/
- Criar a devida entrada na configuração `settings.TEMPLATES ('DIRS')`:



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

Customizando templates do "Django Admin"

- Descobrir onde está instalado o Django, para poder realizar uma cópia dos templates que serão sobreescritos:
 - \$ python -c "import django; print(django.__path__)"
 - Ex: ['/Users/misael/.../python3.7/site-packages/django']
 - Arquivos copiados:
 - .../python3.7/site-packages/django/contrib/admin/templates/admin/base_site.html
 - .../python3.7/site-packages/django/contrib/admin/templates/admin/login.html

<https://docs.djangoproject.com/en/2.2/intro/tutorial07/#customize-the-admin-look-and-feel>

Customizando templates do "Django Admin"

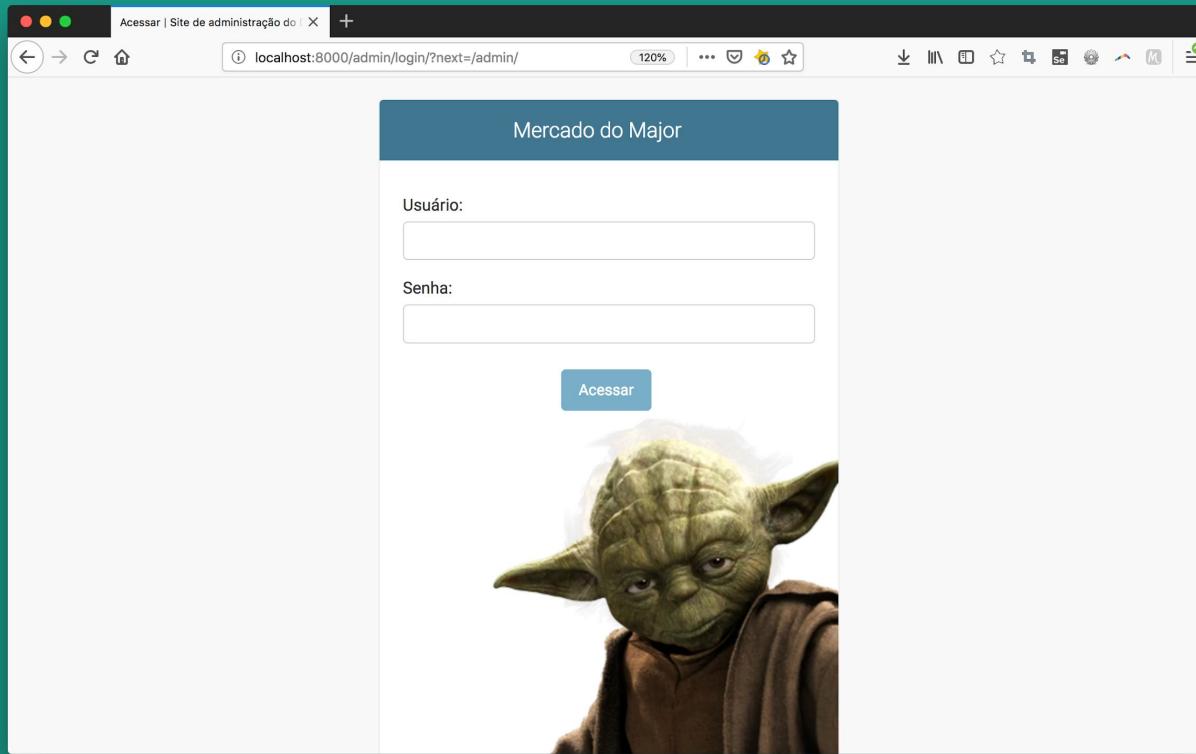
The image shows a file explorer on the left and two code editors on the right. The file explorer displays the project structure of 'mini_curso_django' with files like 'atendimento', 'LICENSE', and 'manage.py'. The 'templates/admin' directory contains 'base_site.html' and 'login.html'. The code editors show the content of these files.

base_site.html

```
1  {% extends "admin/base.html" %}  
2  
3  {% block title %}{{ title }} | {{ site_title|default:_('Django site admin') }}{% endblock %}  
4  
5  {% block branding %}  
6  {#<h1 id="site-name"><a href="{% url 'admin:index' %}">{{ site_header|default:_('Django admi  
7  <h1 id="site-name"><a href="{% url 'admin:index' %}">Mercado do Major</a></h1>  
8  {% endblock %}  
9  
10 {% block nav-global %}{% endblock %}
```

login.html

```
1  {% extends "admin/base_site.html" %}  
2  {% load i18n static %}  
3  
4  ...  
5  <div class="submit-row">  
6    <label> </label><input type="submit" value="{% trans 'Log in' %}">  
7  </div>  
8  </form>  
9  
10 <image src="https://vignette.wikia.nocookie.net/disney/images/9/95/Maste  
11 height="40%"/>  
12  
13  
14  </div>  
15  {% endblock %}
```



<http://localhost:8000/admin/>

Phuderozo major! Mermão dá pra fazer
"misera" com esse Django...



Dá sim "dezanos"! E olhe que tem coisa
que eu ainda que nem te mostrei!





Outros recursos interessantes do Django

O foco desta apresentação foi passar o essencial acerca dos pilares do Django. Devido ao tempo escasso, vários recursos muito interessantes não puderam abordados, mas merecem ser vistos em casa e acreditamos que serão alvo de futuras apresentações. Recomendamos que dêem uma estudada sobre:

- Forms;
- ModelForms;
- Class Based Views;
- Managers;

A página abaixo trata desses itens e muito mais:

<https://docs.djangoproject.com/en/2.2/>

Código fonte do Mini Curso Django

- Disponível em:
 - [https://github.com/misaelbarreto/mini curso django](https://github.com/misaelbarreto/mini_curso_django)
- Nele você vai encontrar o projeto completo, ou seja, com as três aplicações criadas e rodando.

OBRIGADO!

