

# Django Autenticação



Prof. Bruno Gomes



@profbrunogomes



# Aula de Hoje

2

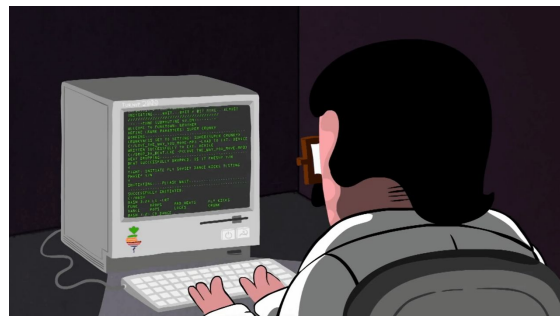
- » **Django**
  - » **Autenticação**
  - » **Customizando a Autenticação**
  - » **Permissão de Acesso**
  - » **Autenticação OAuth2**



# Antes de começar

3

- » Baixar **proj\_09\_auth\_1.zip** da pasta do Drive;
- » Extrair no mesmo diretório do Virtualenv.



## 1

# Autenticação

# Documentação

5

» <https://docs.djangoproject.com/en/4.1/topics/auth/default/>



# Aplicação Auth

6

- » O Django já vem com uma aplicação que fornece mecanismos de autenticação, chamada auth:

aula9/settings.py

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

- » Já trás as funções de autenticação (login, logout...)



# 1.1

## Autenticação

Login

# Configurando Login

8

- » Para informar ao Django a URL que contém o template com o formulário de autenticação, criar a variável LOGIN\_URL em **settings.py**:

```
STATIC_URL = '/static/'
```

```
LOGIN_URL = 'login'
```

aula9/settings.py



# Configurando Login

9

- » É necessário informar ao projeto qual será a URL acessada após a autenticação. Para isso, criar a variável LOGIN\_REDIRECT\_URL em **settings.py**:

```
LOGIN_URL = 'login'
```

```
LOGIN_REDIRECT_URL = 'perfil'
```

aula9/settings.py

# Configurando URL para Login

10

- » Em **urls.py**, é necessário criar a URL login, e apontar para o sistema de autenticação da app auth:

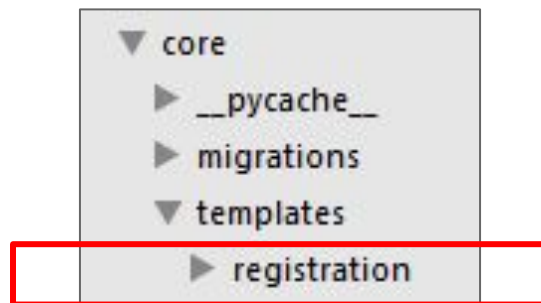
```
from django.contrib.auth.views import LoginView  
  
path('login/', LoginView.as_view(), name='login'),
```

aula9/urls.py

# Criando o Template para Login

11

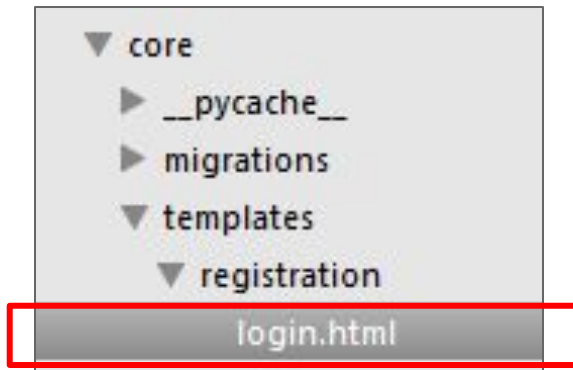
- » Por padrão, o django procura o template login dentro de uma pasta chamada **registration**. Então, criar a pasta dentro de templates:



## Criando o Template para Login

12

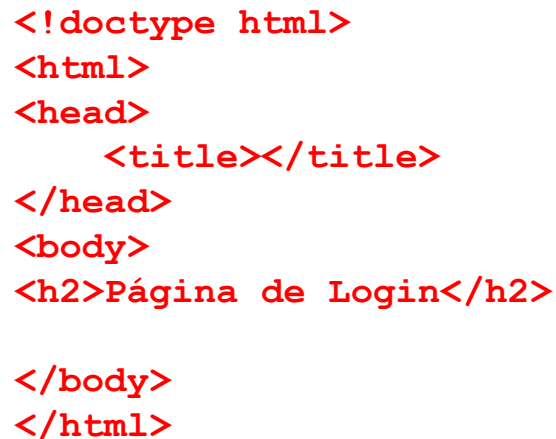
- » Dentro da pasta registration, criar o template login.html:



# Criando o Template para Login

13

- » Adicionar HTML básico no template **login.html**:



```
<!doctype html>
<html>
<head>
    <title></title>
</head>
<body>
<h2>Página de Login</h2>

</body>
</html>
```

aula9/templates/registration/login.html

# Criando o Template para Login

14

- » Ainda no template **index.html**, adicionar o formulário, imprimindo os campos através da variável form (padrão do sistema de autenticação):

```
<h2>Página de Login</h2>
<form method="post">
  {% csrf_token %}
  {{form}}
  <p><input type="submit" value="Autenticar" /></p>
</form>
```

[aula9/templates/registration/login.html](#)



# Testar no Navegador

15

- » Iniciar o servidor e Acessar:

```
http://localhost:8000/login
```

## Página de Login

Username:  Password:

# Ajustando o Template de Login

16

- » É possível gerar cada campo de forma individual:

```
{% csrf_token %}
```

```
<p>{{ form.username.label_tag }} {{ form.username }}</p>
```

```
<p>{{ form.password.label_tag }} {{ form.password }}</p>
```

```
<p><input type="submit" value="Autenticar" /></p>
```

[aula9/templates/registration/login.html](#)

# Testar no Navegador

17

- » Iniciar o servidor e Acessar:



`http://localhost:8000/login`

## Página de Login

Username:

Password:

Login

# Ajustando o Template de Login

18

- » Também é possível substituir os valores do label pelos valores que desejar:

```
{% csrf_token %}
```

```
<p>Usuário: {{ form.username }}</p>
```

```
<p>Senha: {{ form.password }}</p>
```

```
<p><input type="submit" value="Autenticar" /></p>
```

[aula9/templates/registration/login.html](#)

# Testar no Navegador

19

- » Iniciar o servidor e Acessar:



`http://localhost:8000/login`

## Página de Login

Usuário:

Senha:

Login

# Link para Login em Index

20

- » No template **index.html**, adicionar um link para login:

```
<h2>Página Inicial</h2>
```

```
<p><a href="{% url 'login' %}">Login</a></p>
```

[aula9/templates/index.html](#)



# Testar no Navegador

21

- » Iniciar o servidor e Acessar:



`http://localhost:8000`

## Página Inicial

[Login](#)

[Perfil](#)

## Página de Login

Usuário:

Senha:

Login

# 1.2

## Autenticação

Logout

## Configurando Logout

23

- » Para informar ao Django a URL que será acessada após realizar logout, criar a variável `LOGOUT_REDIRECT_URL` em **settings.py**:

```
LOGIN_REDIRECT_URL = 'perfil'
```

```
LOGOUT_REDIRECT_URL = 'home'
```

aula9/settings.py

# Logout

24

- » Em **urls.py**, é necessário criar a URL logout, e apontar para o sistema de autenticação da app auth:

```
from django.contrib.auth.views import LoginView, LogoutView

path("logout/", LogoutView.as_view(), name="logout"),
```

aula9/urls.py

# Logout

25

- » No template perfil.html, adicionar o link para a url LOGOUT:

```
<h2>Página de Perfil</h2>
```

```
<p><a href="{% url 'logout' %}">Sair</a></p>
```

core/templates/perfil.html

## Antes de Testar

26

- » Necessário adicionar um usuário no banco.
- » A aplicação django trabalha com o conceito de **superuser**, que veremos no próximo tópico.



# 1.3

## Autenticação

Superuser

# Documentação

28

» <https://docs.djangoproject.com/en/4.1/intro/tutorial02/>



# Criando o super usuário

29

- » É possível criar um super usuário para ter acesso total ao admin do sistema (assunto da próxima aula). O comando é:

```
python manage.py migrate  
python manage.py createsuperuser
```

Terminal

- » Sugestão:
  - » Usuário: **admin**
  - » Email: **admin@email.com**
  - » Senha: **admin12345**

```
Username (leave blank to use 'bruno'): admin  
Email address: admin@email.com  
Password:  
Password (again):  
Superuser created successfully.
```

# Criando o super usuário

30

## » Outra forma

```
python manage.py migrate
```

```
python manage.py createsuperuser --username=admin --email=admin@email.com
```

Terminal

## » Sugestão:

- » Usuário: **admin**
- » Email: **admin@email.com**
- » Senha: **admin12345**

```
(django) C:\projetos\ifrn\aulas_ifrn\django\proj_09_auth_3>python manage.py createsuperuser --username=admin --email=admin@email.com
Password:
Password (again):
Superuser created successfully.

(django) C:\projetos\ifrn\aulas_ifrn\django\proj_09_auth_3>
```

# Testar no Navegador

31

- » Iniciar o servidor e Acessar:



- » Clicar em **Login**, realizar autenticação com os dados cadastrados anteriormente;
- » Depois, clicar em **Sair**.



# 1.4

## Autenticação

Permissão de Acesso



# Bloqueando Acesso a Templates

33

- » Para que somente usuários autenticados acessem o template perfil, em views adicionar a anotação `@login_required` acima da função:

```
from django.contrib.auth.decorators import login_required

@login_required
def perfil(request):
    return render(request, 'perfil.html')
```

`core/views.py`

# Testar no Navegador

34

- » Iniciar o servidor e Acessar:



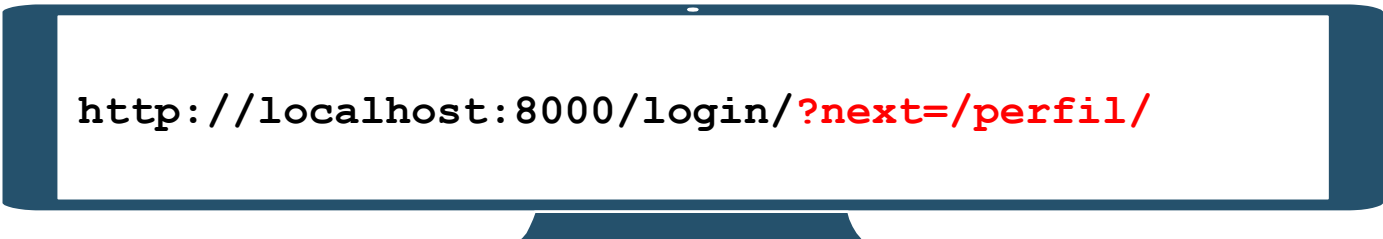
- » Com o usuário **desconectado**, tentar acessar a página de perfil. Será redirecionado à página de Login.

### Página de Login

Usuário:

Senha:

- » Ao tentar acessar perfil sem permissão, o usuário é redirecionado para a página de login, e a URL fica desta forma:

A dark blue computer monitor with a white screen. The screen displays the URL `http://localhost:8000/login/?next=/perfil/` in a monospaced font. The text `?next=/perfil/` is highlighted in red.

```
http://localhost:8000/login/?next=/perfil/
```

- » Significa que, após se autenticar, será redirecionado para a url que tentou acessar sem permissão (perfil).

## Exibindo dados no template se Autenticado

36

- » Se o usuário não estiver autenticado, o link para Perfil no template index não deverá aparecer. Para isso, basta acessar o atributo `is_authenticated` do usuário autenticado:

```
{% if user.is_authenticated %}  
<p><a href="{% url 'perfil' %}">Perfil</a></p>  
{% endif %}
```

`core/templates/index.html`

## Verificando se o usuário não está Autenticado

37

- » Para verificar se o usuário não está autenticado, utilizar a propriedade **not is\_authenticated** de user. Em index.html, adicionar:

```
{% if not user.is_authenticated %}  
<p><a href="{% url 'login' %}">Login</a></p>  
{% endif %}
```

core/templates/index.html

# Testar no Navegador

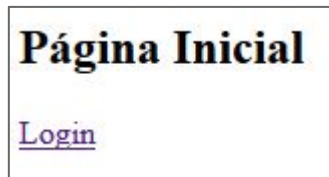
38

- » Iniciar o servidor e Acessar:

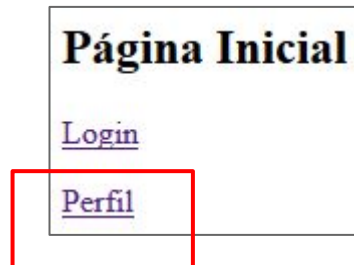


- » Acessar index sem estar autenticado;
- » Acessar index estando autenticado

Não autenticado



Autenticado



## Exibindo dados do Usuário Autenticado

39

- » Para exibir dados do usuário autenticado, basta imprimir qualquer atributo de usuário acessando através do objeto user (**perfil.html**):

```
<h2>Página de Perfil</h2>
```

```
<p>Olá {{ user.username }}</p>
```

[core/templates/perfil.html](#)



# Testar no Navegador

40

- » Iniciar o servidor e Acessar:

A stylized illustration of a web browser window with a dark blue border and a light blue base. Inside the window, the URL `http://localhost:8000/perfil` is displayed in a monospaced font.

`http://localhost:8000/perfil`

Navegador

## Página de Perfil

Olá admin

[Sair](#)

## Exibindo outros dados

41

- » É possível também exibir o e-mail

```
<h2>Página de Perfil</h2>  
<p>Olá {{ user.username }}</p>  
<p>E-mail: {{ user.email }}</p>
```

core/templates/perfil.html

# 2.

## Customizando a Autenticação

# Documentação

43

- » <https://docs.djangoproject.com/en/4.1/topics/auth/customizing/>



# Customização

44

- » O sistema de autenticação padrão do Django é muito simples;
- » É possível customizar o backend da biblioteca padrão:
  - » Mudar a forma de autenticação;
  - » Alterar permissões;
  - » Estender ou substituir o model User.

# 2.1

## Mudando a forma de Autenticação

# Modificando o template

46

- » Modificando os campos de usuário e senha:

```
<form method="post">
  {% csrf_token %}
  <p>Usuário: <input type="text" name="usuario" /></p>
  <p>Senha: <input type="password" name="senha" /></p>
  <p><input type="submit" value="Autenticar" /></p>
</form>
```

templates/registration/login.html



## Criando uma View

47

- » 1º passo: importar as duas funções responsáveis pela autenticação:
  - » **authenticate** - verifica o login e senha;
  - » **login** - realiza a autenticação no sistema.

```
from django.contrib.auth import authenticate, login
```

core/views.py

# Criando uma View

48

» 2º passo: criar uma view e implementar todo o processo:

```
def autenticacao(request):  
    if request.POST:  
        username = request.POST['usuario']  
        password = request.POST['senha']  
        user = authenticate(request, username=username, password=password)  
        if user is not None:  
            login(request, user)  
            return redirect('perfil')  
        else:  
            return render(request, 'registration\login.html')  
    else:  
        return render(request, 'registration\login.html')
```

core/views.py

## Modificando a URL

49

- » Importar a função `autenticacao` e modificar o path para login:

```
from core.views import home, perfil, autenticacao

urlpatterns = [
    path('login/', autenticacao, name='login'),
]
```

`proj_09_auth/urls.py`

# Testar no Navegador

50

- » Iniciar o servidor e Acessar:



## Página de Login

Usuário:

Senha:

Autenticar

- » Testar um usuário que não existe, posteriormente um cadastrado. Testar as páginas do projeto.

# 2.2

## Mudando a forma de Logout

## Criando uma View

52

- » 1º passo: importar a função responsável pelo logout:

```
from django.contrib.auth import logout
```

core/views.py

# Criando uma View

53

» 2º passo: criar uma view e implementar o processo:

```
def desconectar(request):  
    logout(request)  
    return redirect('home')
```

core/views.py



# Modificando a URL

54

- » Importar a função desconectar e modificar o path para logout:

```
from core.views import home, perfil, autenticacao, desconectar

urlpatterns = [
    path("logout/", desconectar, name="logout"),
]
```

proj\_09\_auth/urls.py

## Testar no Navegador

55

- » Iniciar o servidor e Acessar:



`http://localhost:8000/login`

Navegador

- » Realizar a autenticação e na página de Perfil clicar em Sair. Tentar acessar posteriormente a página de Perfil sem estar autenticado para comprovar que desconectou.

# 2.3

## Model User

# Documentação

57

- » <https://docs.djangoproject.com/en/4.1/ref/contrib/auth/#django.contrib.auth.models.User>



# Trabalhando com o Model User

58

## » Atributos primários:

- » username
- » password
- » email
- » first\_name
- » last\_name
- » is\_superuser

# Documentação

59

- » <https://docs.djangoproject.com/en/4.1/topics/auth/customizing/#extending-the-existing-user-model>
- » <https://docs.djangoproject.com/en/4.1/topics/auth/customizing/#extending-the-existing-user-model>





## Trabalhando com o Model User

60

- » **Situação:** o registro de um usuário muitas vezes necessita de mais campos além de usuário, senha e e-mail.
- » 1ª Solução:
  - » Criar um relacionamento 1 para 1 com o model User.



# Relacionamento com o Model User

61

- » Exemplo: criando um model perfil com os campos nome e idade:

```
from django.contrib.auth.models import User

class Perfil(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    nome = models.CharField('Nome Completo', max_length=100)
    idade = models.IntegerField('Idade')
```

core/models.py

## Observação

62

- » Caso deseje, pode definir que user será chave primária de Perfil:

```
user = models.OneToOneField(User,  
on_delete=models.CASCADE, primary_key=True)
```

core/models.py

## Observação

63

- » A desvantagem deste modelo de relacionamento é que a todo momento precisa ficar gerenciando este relacionamento, inclusive no momento do registro;
- » Muitos processos importantes precisam ser feitos de forma manual.

Quem desejar estudar a fundo esta forma deve consultar o manual, pois vamos utilizar nas aulas a próxima (estender).

# Trabalhando com o Model User

64

- » **Situação:** o registro de um usuário muitas vezes necessita de mais campos além de usuário, senha e e-mail.
- » 2ª Solução:
  - » Estendendo e substituindo o Model User

# Substituindo o Model User

65

- » Primeiro passo: criar um model que substituirá o model User, e estender o AbstractUser (neste exemplo usaremos Usuario):

```
from django.contrib.auth.models import AbstractUser

class Usuario(AbstractUser):
    nome = models.CharField('Nome', max_length=100)
    idade = models.IntegerField('Idade')
    cpf = models.CharField('CPF', max_length=11, unique=True)
```

core/models.py

## Observação

66

- » Ao estender `AbstractUser`, automaticamente o modelo passa a ter os campos padrões de `User` (`username`, `email`, `password`....).

```
from django.contrib.auth.models import AbstractUser

class Usuario(AbstractUser):
```

`core/models.py`

## Configurando settings.py

67

- » Configurar a variável para o novo model (neste exemplo estamos usando o modelo Usuario):

```
AUTH_USER_MODEL = 'core.Usuario'
```

[proj\\_09\\_auth/settings.py](#)



# Mudando o campo de Login

68

- » No próprio Model, adicionar a variável USERNAME\_FIELD e informar qual campo será o login (este campo deve ser unique):

```
class Usuario(AbstractUser):  
    nome = models.CharField('Nome', max_length=100)  
    idade = models.IntegerField('Idade')  
    cpf = models.CharField('CPF', max_length=11, unique=True)  
  
    USERNAME_FIELD = 'cpf'
```

proj\_09\_auth/core/models.py

## Testando - Gerando o Banco

69

- » Apagar a pasta migrate (localizada dentro de core) e o banco na raiz do projeto (db.sqlite3), e digitar novamente os comandos a seguir:

```
python manage.py makemigrations core  
python manage.py migrate
```

Terminal

## Testando - Cadastrando um Usuário

70

- » Vamos criar um cadastro **manual** de um usuário;
- » Em Views:

```
from .models import Usuario
```

`proj_09_auth/core/views.py`

## Testando

- » Criar a função do cadastro em Views:
- » Obs.: usuário sem ser administrador.

proj\_09\_auth/core/views.py

```
def cadastro_manual(request):  
    user = Usuario.objects.create_user(  
        username='admin',  
        email='admin@email.com',  
        cpf='11111111111',  
        nome='Administrador',  
        password='admin12345',  
        idade=30,  
        is_superuser=False)  
    user.save()  
    return redirect('home')
```

# Testando - Cadastrando um Usuário

72

- » Configurar uma URL temporária para o cadastro;
- » Em URLs:


```
from core.views import cadastro_manual  
  
path('cadastro_manual/', cadastro_manual),
```

proj\_09\_auth/urls.py

# Testar no Navegador

73

- » Iniciar o servidor e Acessar:



```
http://localhost:8000/cadastro_manual
```

Navegador

- » Após abrir a página inicial, tentar realizar o login através do CPF e senha.

# 2.3

## **Alterando Permissões**



# Documentação

75

- » <https://docs.djangoproject.com/en/4.1/topics/auth/customizing/#custom-permissions>
- » <https://docs.djangoproject.com/en/4.1/topics/auth/default/#permissions-and-authorization>



# Tipos de Permissões

76

- » O model User tem dois campos com relacionamento muitos-para-muitos:
  - » **user\_permissions** (permissões individuais);
  - » **groups** (permissões de grupos).

## Permissões Individuais - `user_permissions`

77

» Tipos de Operações:

- » `myuser.user_permissions.set([permission_list])`
- » `myuser.user_permissions.add(permission, permission, ...)`
- » `myuser.user_permissions.remove(permission, permission, ...)`
- » `myuser.user_permissions.clear()`

## Permissões - Sintaxe

78

- » Django automaticamente cria as permissões de **add**, **change**, **delete** e **view** para cada Model criado.
- » Sintaxe da permissão:
  - » `{app}.{action}_{model_name}`
- » Exemplo:
  - » `core.add_Curso`
  - » `core.change_Curso`

# Antes de começar

79

- » Criar uma página dentro da pasta template chamada pagina\_1.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Document</title>
</head>
<body>
  <p>Texto somente com Permissão 2</p>

  <p><a href="{% url 'logout' %}">Sair</a></p>
</body>
</html>
```

templates/pagina\_1.html

# Antes de começar

80

- » Criar uma View chamada Pagina\_Teste:

```
def pagina_1(request):  
    return render(request, 'pagina_1.html')
```

core/views.py

# Antes de começar

81

- » Criar a rota a seguir:

```
from core.views import pagina_1  
  
path('pagina_1', pagina_1),
```

proj\_09\_auth/urls.py



## Testar no Navegador

82

- » Iniciar o servidor e Acessar:



`http://localhost:8000/pagina_1`

Navegador

- » **A página irá abrir**, pois não é necessário nenhuma permissão de acesso.

# Permissões Individuais - Personalizadas

83

» Definindo as permissões que o Usuário poderá assumir:

```
class Usuario(AbstractUser):  
    nome = models.CharField('Nome', max_length=100)  
    idade = models.IntegerField('Idade')  
    cpf = models.CharField('CPF', max_length=11, unique=True)  
  
    USERNAME_FIELD = 'cpf'  
  
    class Meta:  
        permissions = [  
            ("permissao_01", "Poderá acessar a view da pagina_1.html"),  
            ("permissao_02", "Poderá visualizar um texto do template")  
        ]
```

core/models.py

# Atualizando as permissões no Banco

84

- » Executar os comandos a seguir no Terminal:

```
python manage.py makemigrations core  
python manage.py migrate
```

Terminal

# Protegendo a View com a Permissão Criada

85

- » Definindo que o acesso à View só será feito por usuários que tiverem a permissão **permissao\_01** definida:

```
from django.contrib.auth.decorators import permission_required

@login_required
@permission_required('core.permissao_01')
def pagina_1(request):
    return render(request, 'Teste.html')
```

core/views.py

# CURIOSIDADE

86

- » Se desejar implementar a verificação da permissão dentro da View, funciona através da função `has_perm`:


```
@login_required
def Pagina_Testes(request):
    if request.user.has_perm('core.permissao_01'):
        return render(request, 'Teste.html')
    else:
        return redirect('home')
```

core/views.py

## Testar no Navegador

87

- » Iniciar o servidor e Acessar:



```
http://localhost:8000/pagina_teste
```

Navegador

- » A página **não** irá abrir, mesmo realizando login, pois o usuário atual não tem permissão de acesso (admin).

# Adicionando Permissão a Usuário

88

- » Primeiro passo: Importar o objeto Permission em Views:

```
from django.contrib.auth.models import Permission
```

core/views.py



# Adicionando Permissão ao Usuário

89


```
def cadastro_manual(request):  
    user = Usuario.objects.create_user(  
        username='admin2',  
        email='admin2@email.com',  
        cpf='22222222222',  
        nome='Administrador',  
        password='admin12345',  
        idade=30,  
        is_superuser=False)  
  
    permission = Permission.objects.get(codename='permissao_01')  
    user.user_permissions.add(permission)  
  
    user.save()  
    return redirect('home')
```

core/views.py

# Testar no Navegador

90

- » Iniciar o servidor e Acessar:



```
http://localhost:8000/pagina_teste
```

Navegador

- » Realizar o login como admin2 (cpf 22222222222), a página irá abrir normalmente, pois este novo usuário possui permissão de acesso à view.

## Adicionando permissão ao Template

91

- » Na página Teste.html, adicionar a condição de acesso somente se tiver a permissão **permissao\_02**:

```
{% if perms.core.permissao_02 %}
```

```
<p>Texto somente com Permissão 2</p>
```


```
{% endif %}
```

core/views.py

## Testar no Navegador

92

- » Iniciar o servidor e Acessar:



```
http://localhost:8000/pagina_teste
```

Navegador

- » O usuário admin2 abrirá a página mas não conseguirá visualizar o texto, pois ele possui apenas a permissão `permissao_01`, e não a `permissao_02`.

## Adicionando Usuário com todas as permissões 93


`core/views.py`

```
def cadastro_manual(request):  
    user = Usuario.objects.create_user(  
        username='admin3',  
        email='admin3@email.com',  
        cpf='33333333333',  
        nome='Administrador',  
        password='admin12345',  
        idade=30,  
        is_superuser=False)  
    permission1 = Permission.objects.get(codename='permissao_01')  
    permission2 = Permission.objects.get(codename='permissao_02')  
    user.user_permissions.add(permission1, permission2)  
    user.save()  
    return redirect('home')
```

# Testar no Navegador

94

- » Iniciar o servidor e Acessar:



```
http://localhost:8000/pagina_teste
```

Navegador

- » Desconectar o usuário e acessar a página novamente. Realizar a autenticação como admin3 (cpf: 33333333333). Ele conseguirá acessar a página e o texto irá ser exibido.

# Permissões de Grupos

95

- » Tipos de Operações:
  - » myuser.**groups.set**([group\_list])
  - » myuser.**groups.add**(group, group, ...)
  - » myuser.**groups.remove**(group, group, ...)
  - » myuser.groups.**clear**()



# Permissões de Grupos

96

- » Django automaticamente cria as permissões de **add**, **change**, **delete** e **view** para cada Model criado.
- » Sintaxe da permissão:
  - » `{app}.{action}_{model_name}`
- » Exemplo:
  - » `core.add_Curso`
  - » `core.change_Curso`

# Observação

97

- » Não serão vistas agora.

# 3.

## Autenticação OAuth2

# Documentação

99

» <https://www.django-rest-framework.org/>



# Dúvidas?

