

## Django

Formulários, validação, autenticação e autorização

Ely – [elydasilvamiranda \[at\] gmail.com](mailto:elydasilvamiranda@gmail.com)

1

### Registrando usuários e criando perfis

- Já implemtamos a integração e perfis:
  - Convidar e aceitar convites;
  - Listar perfis, detalhar perfil e contatos;
- Iremos agora implementar:
  - Cadastrar novos usuários e consequentemente seus perfis;
  - Implementar a autenticação e autorização de usuários.

2

### Nova aplicação de usuários

- Haverá uma nova entidade no nosso sistema: usuário;
- Criaremos uma nova aplicação para gerenciar usuários;
- Ela funcionará como um novo módulo, integrada à aplicação de perfis;
- Os usuários serão mantidos com a ajuda do módulo administrativo do Django;

3

### Criando a aplicação de usuários

- Utilizamos o comando:
  - > python manage.py startapp usuarios
- Registramos a aplicação no settings.py:

```
# connectedin/connectedin/settings.py
# código anterior omitido
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'perfis',
    'usuarios'
)
# código posterior omitido
```

4

## Definindo a rota

- Para registrarmos usuários, usaremos a rota:
  - `http://localhost:8000/regarstrar`
- Criando a rota no arquivo de urls:

```
# connectedin/usuarios/urls.py
from django.conf.urls import url

urlpatterns = [
    url(r'^regarstrar/$', ???, name="regarstrar")
]
```

5

## Associando arquivos urls.py

- Devemos adicionar a referência do arquivo `urls.py` da aplicação usuários ao do projeto:

```
# connectedin/connectedin/urls.py

from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', include('perfis.urls')),
    url(r'^', include('usuarios.urls'))
]
```

6

## Criando uma class-based-view

- Class-based-views integram entradas de dados vindas de formulários a objetos;
- Possuem métodos para envios de dados via GET, POST, dentre outros;
- No nosso caso, criaremos uma página de chamada `regarstrar.html`:
  - Para exibir o formulário usaremos o método GET;
  - Ao realizarmos o envio dos dados do usuário, usaremos o método POST;

<https://docs.djangoproject.com/en/1.11/topics/class-based-views/>

7

## Criando uma class-based-view

```
# connectedin/usuarios/views.py

from django.shortcuts import render
from django.views.generic.base import View

class RegistarUsuarioView(View):

    template_name = 'regarstrar.html'

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

    def post(self, request):
        return render(request, self.template_name)
```

8

## Atualizando a rota

- Atualizaremos o urls.py para ter o parâmetro que faltava para chegar à visão;
- A visão será executada através do nome da classe, seguido do método as\_view():

```
# connectedin/usuarios/urls.py
from django.conf.urls import url
from views import RegistrarUsuarioView

urlpatterns = [
    url(r'^registrar/$', RegistrarUsuarioView.as_view(),
        name="registrar")
]
```

9

## Template base

- Criaremos um template base, assim como na aplicação perfis:

```
<!-- connectedin/usuarios/templates/base_usuario.html -->
{% load staticfiles %}
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>ConnectedIn</title>
    <link href="{% static 'styles/bootstrap.css' %}" rel="stylesheet">
    <link href="{% static 'styles/signin.css' %}" rel="stylesheet">
  </head>
  <body>
    <div class="container">
      {% block body %}
      {% endblock %}
    </div>
  </body>
</html>
```

10

## Página de registro

- A página de registro herdará do template definido no slide anterior;
- Deverá ter todos os campos do perfil e ainda o campo senha;
- A ideia principal é que ao cadastrar um usuário, seu perfil seja logo criado;
- Para a submissão dos dados usaremos o método POST para a url registrada anteriormente.

11

## Formulário para registro de usuários

- O Django permite que sejam gerados formulários de forma automática ou manual;
- Comentaremos brevemente sobre a forma automática mais adiante.

12

## Formulário para registro de usuários

```
<!-- connectedin/usuarios/templates/regarstrar.html -->
{% extends "base_usuario.html" %}
{% block body %}
<form class="form-signin" action="{% url 'registrar' %}"
      method="post">
  {% csrf_token %}
  <h2 class="form-signin-heading">Crie seu usuário</h2>
  <input type="text" id="id_email" name="email"
        class="form-control" placeholder="Email *" required autofocus>
  <input type="text" id="id_nome" name="nome"
        class="form-control" placeholder="Nome *" required >
  <input type="password" id="id_senha" name="senha"
        class="form-control" placeholder="Senha *" required>
  <hr />
  <input type="text" id="id_telefone" name="telefone"
        class="form-control" placeholder="Telefone">
  <input type="text" id="id_nome_empresa" name="nome_empresa"
        class="form-control" placeholder="Empresa">
  <hr />
  <button class="btn btn-lg btn-primary btn-block"
          type="submit" value="Login">Registrar</button>
</form>
{% endblock %}
```

13

## Sobre a tag {% csrf\_token %}

- Caso acessemos a página (GET), ela é exibida corretamente;
- Se subtermos o formulário sem essa tag, recebemos um erro:



- Cross-Site Request Forgery (CSRF):
  - É um ataque em que algum site use as credenciais já adquiridas do usuário;
  - Essas credenciais podem ser usadas para outras finalidades sem seu consentimento.

<https://docs.djangoproject.com/en/1.11/ref/csrf/>

14

## Sobre a tag {% csrf\_token %}

- O Django exige que protejamos o formulário contra o ataque CSRF;
- Assim, devemos incluir essa tag dentro de o formulário;
- Essa tag gera um token como o mostrado abaixo:

```
<input type="hidden" name='csrfmiddlewaretoken'
      value='T8sp41gEKJ9gC58xeoVbcAkfy9aJnot9' />
```
- A tag é do tipo hidden e é submetida ao enviar o formulário ao Django.

15

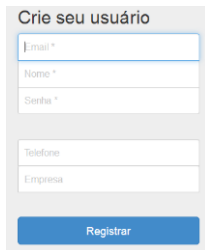
## Sobre a tag {% csrf\_token %}

- Este token evita:
  - que algum site use as credenciais já adquiridas do usuário para outras finalidades;
  - que terceiros submetam dados a partir de um formulário falso para nossa aplicação;
- Dessa forma, o formulário sempre terá que ser solicitado do nosso sistema;
- Apenas assim será possível fazer a submissão via POST.

16

## Testando nossa página

- No prompt, rode o servidor:  
> python manage.py runserver
- E no navegador, acesse:  
<http://localhost:8000/regarstrar>



17

## Classe django.Form

- A classe Form permite que mapeemos os campos presentes na página em atributos;
- É importante validarmos os dados submetidos antes de salvarmos no banco;
- Esses atributos, por sua vez, possuem definições de validação;
- Possui ainda um método específico para validação;

<https://docs.djangoproject.com/en/1.11/topics/forms/>

18

## Formulário de registro

- Todos os campos desse formulários são definidos como campos string, e-mail e obrigatórios:

```
# connectedin/usuarios/forms.py
from django import forms

class RegistrarUsuarioForm(forms.Form):
    nome = forms.CharField(required=True)
    email = forms.EmailField(required=True)
    senha = forms.CharField(required=True)
    telefone = forms.CharField(required=True)
    nome_empresa = forms.CharField(required=True)
```

- Além desses campos:
  - A classe form possui um campo chamado `cleaned_data`;
  - É uma lista onde podemos acessar os campos da seguinte forma: `self.cleaned_data['nome']`

19

## Validando um formulário

- A classe Form possui um método `is_valid` que diz se o formulário é válido ou não;
- Esse método deve ser sobrescrito;
- Via de regra:
  - Chamamos o método da superclasse para que validações básicas sejam executadas;
  - Depois fazemos nossas próprias validações;

<https://docs.djangoproject.com/en/1.11/ref/forms/validation/>

<https://docs.djangoproject.com/en/1.11/topics/forms/#rendering-form-error-messages>

20

## Validando um formulário

- Primeiro passo: verificar validações mais básicas, como: campos obrigatórios, campo de e-mail...
- Isso é feito chamando-se o método na super classe;
- Caso haja erros, deve-se enviar uma mensagem de erro pra ser exibido em uma página;
- Um função utilitária será criada para isso.

21

## Validações básicas

- Primeiro passo: verificar validações mais básicas, como: campos obrigatórios, campo de e-mail...

```
# connectedin/usuarios/forms.py
from django import forms

class RegistrarUsuarioForm(forms.Form):
    # código omitido
    def is_valid(self):
        valid = True
        if not super(RegistrarUsuarioForm, self).is_valid():
            self.adiciona_erro('Por favor, verifique os dados informados')
            valid = False

        return valid

    def adiciona_erro(self, message):
        errors =
            self._errors.setdefault(forms.forms.NON_FIELD_ERRORS,
                                     forms.utils.ErrorList())
        errors.append(message)
```

22

## Validando usuário

- Segundo passo: fazer validações específicas como se o usuário já existe;
- O Django já possui uma entidade chamada User;
- Podemos importá-la e adaptá-la ao nosso modelo:

```
# connectedin/usuarios/forms.py
from django import forms
from django.contrib.auth.models import User

class RegistrarUsuarioForm(forms.Form):

    #código omitido
```

23

## Validando usuário

- Verificando se um usuário já existe:

```
# connectedin/usuarios/forms.py
from django import forms
from django.contrib.auth.models import User

class RegistrarUsuarioForm(forms.Form):
    # código omitido
    def is_valid(self):
        valid = True
        if not super(RegistrarUsuarioForm, self).is_valid():
            self.adiciona_erro('Por favor, verifique os dados informados')
            valid = False

        user_exists = User.objects.filter(username=self.data['nome']).exists()
        if user_exists:
            self.adiciona_erro('Usuário já existente.')
            valid = False

        return valid

    # código omitido
```

24

## Relacionamento 1 para 1

- Um usuário tem necessariamente um perfil;
- As duas classes já existem:
  - podemos relacioná-las em nosso modelo em um relacionamento 1 para 1;
  - Um perfil teria um atributo User;
- Campo e-mail:
  - O campo e-mail de perfil deixaria de existir, pois User já possui esse campo;
  - Definiremos ele como uma propriedade vinda de User.

25

## Relacionamento 1 para 1

```
# connectedin/perfis/views.py
from django.db import models

class Perfil(models.Model):
    nome = models.CharField(max_length=255, null=False)
    #sem email
    telefone = models.CharField(max_length=15, null=False)
    nome_empresa =
        models.CharField(max_length=255, null=False)
    contatos = models.ManyToManyField('self')

    usuario =
        models.OneToOneField(User, related_name="perfil")

    @property
    def email(self):
        return self.usuario.email
# código posterior comentado
```

26

## Atualizando a view

- Para que a view funcione agora com o formulário e entidades atualizadas, devemos:
  - Dentro do método post:
    - Instanciar um novo formulário passando o objeto request como parâmetro;
    - Testar se o formulário é válido;
    - Criar um usuário;
    - Criar um perfil e setar seu atributo usuário;
    - Salvar o perfil e redirecioná-lo para a página index.html;
  - Nota: caso o formulário não seja validado, devolvemos o usuário a página de registro

27

## Atualizando a view

```
# connectedin/usuarios/view.py
from django.shortcuts import redirect, render
from django.contrib.auth.models import User
from django.views.generic.base import View
from perfis.models import Perfil
from usuarios.forms import RegistrarUsuarioForm

class RegistrarUsuarioView(View):
    # código omitido
    def post(self, request):
        form = RegistrarUsuarioForm(request.POST)
        if form.is_valid():
            dados_form = form.cleaned_data
            usuario = User.objects.create_user(dados_form['nome'],
                dados_form['email'], dados_form['senha'])

            perfil = Perfil(nome=dados_form['nome'],
                telefone=dados_form['telefone'],
                nome_empresa=dados_form['nome_empresa'],
                usuario=usuario)

            perfil.save()
            return redirect('index')

        return render(request, self.template_name, {'form': form})
```

28

## Mostrando os erros do formulário

- Exibiremos os erros na página de registro caso ocorram erros de validação:

```
<!-- connectedin/usuarios/templates/regarstrar.html -->
{% extends "base_usuario.html" %}

{% block body %}
<form class="form-signin" action="{% url 'registrar' %}" method="post">
  {% csrf_token %}
  <!-- código omitido -->

  {% if form.errors %}
    <div class="alert alert-danger">
      <button type="button" class="close"
        data-dismiss="alert" aria-hidden="true"></button>
      {{form.non_field_errors}}
    </div>
  {% endif %}
</form>
{% endblock %}
```

29

## Atualizando o esquema e banco

- Para garantir que todos os usuários tenham perfis, apague o banco atual;
- Em seguida, rode os comandos:
  - > *python manage.py makemigrations*
    - Neste ponto, forneça como resposta às perguntas o valor 1 por duas vezes;
- Agora execute as migrações:
  - > *python manage.py migrate*

30

## Testando o cadastro

- Acesse a url: <http://localhost:8000/registrar>;
- Realize dois testes de de registro de usuário:
  - Um com e-mail inválido. O sistema deve informar: "Por favor, verifique os dados informados";
  - Outro com os dados corretos e veja que será para a página index e a listagem do novo usuário;
  - Por fim, tente cadastrar o mesmo usuário. A mensagem "Usuário já existente" será apresentada;

31

## Autenticando e autorizando usuários

- A classe User pertence à API do Django;
- Podemos usar mais dessa API para gerenciar o login do sistema;
- O primeiro passo é mapear as urls login e logout;
- O Django já possui duas views prontas para tratar isso:
  - django.contrib.auth.views.LoginView;
  - django.contrib.auth.views.LogoutView;
  - Com essas views, podemos definir que templates deverão responder por login e logout no urls.py.

<https://docs.djangoproject.com/en/1.11/topics/auth/default/#django.contrib.auth.views.LoginView>

<https://docs.djangoproject.com/en/1.11/topics/auth/default/#django.contrib.auth.views.LogoutView>

32



## Definindo rotas de login e logout

```
#connectedin/usuarios/urls.py

from django.conf.urls import patterns, url
from views import RegistrarUsuarioView

urlpatterns = [
    url(r'^registrar/$', RegistrarUsuarioView.as_view(),
        name="registrar"),
    url(r'^login/$',
        auth_views.LoginView.as_view(template_name='login.html'),
        name = 'login'),
    url(r'^logout/$',
        auth_views.LogoutView.as_view(template_name='login.html'),
        name="logout")
]
```

33

## Página de login

- Para a página de login, o Django exige duas convenções:
  - O atributo action do formulário deve ser: `{% url 'django.contrib.auth.views.login' %}`;
  - Os atributos name das inputs de usuário e senha devem ser username e password;
- Deve-se adicionar nas últimas linhas do arquivo settings.py as URLs principais do sistema:

```
# connectedin/connectedin/settings.py
# código omitido
LOGIN_URL="/login/"
LOGOUT_URL="/logout/"
LOGIN_REDIRECT_URL="/"
```

34

## Página de login

```
<!-- connectedin/usuarios/templates/login.html -->
{% extends "base_usuario.html" %}
{% block body %}
<form class="form-signin" role="form"
    action="{% url 'django.contrib.auth.views.login' %}" method="post">
    {% csrf_token %}
    <h2 class="form-signin-heading">Login</h2>
    <input type="text" id="id_username" name="username"
        class="form-control" placeholder="Nome" required autofocus>
    <input type="password" id="id_password" name="password"
        class="form-control" placeholder="Senha" required>
    <button class="btn btn-lg btn-primary btn-block"
        type="submit" value="Login">Login</button>
    <input type="hidden" name="next" value="{{ next }}" />
    {% if form.errors %}
    <div class="alert alert-danger">
    <button type="button" class="close"
        data-dismiss="alert" aria-hidden="true"></button>
        Por favor, verifique o Email e/ou Senha informados
    </div>
    {% endif %}
    <a href="{% url 'registrar' %}">registre-se</a>
</form>
{% endblock %}
```

35

## Página de login

The image shows a visual mockup of the login page. It features a light gray background with a white box containing the form. The form has a title 'Login' in bold. Below the title are two input fields: 'Email' and 'Senha'. Below these fields is a blue button labeled 'Login'. At the bottom of the form box, there is a link that says 'registre-se'.

36

## Protegendo as views

- A autenticação consiste em realizar o login;
- A autorização significa exibir determinado recurso, se o usuário possuir permissões;
- Assim, devemos proteger algumas páginas de usuários não autenticados;
- Para proteger uma view há um "decorator" chamado `@login_required`;
- Ele deve ser colocado antes de em todas as views que precisem de autorização;

37

## Protegendo as views

```
# connectedin/perfis/views.py
# código omitido
from django.contrib.auth.decorators import login_required

@login_required
def index(request):
    # código omitido

@login_required
def exibir(request, perfil_id):
    # código omitido

@login_required
def convidar(request, perfil_id):
    # código omitido

@login_required
def aceitar(request, convite_id):
    # código omitido

@login_required
def get_perfil_logado(request):
    return request.user.perfil
```

← Usuário logado

38

## Django

Formulários, validação, autenticação e  
autorização

Ely – [elydasilvimiranda \[at\] gmail.com](mailto:elydasilvimiranda@gmail.com)

39