# Django Framework do Zero

## Como Criar um Aplicativo Web

*Canal **youtube.com/@VamosCodar** - Wanderson Reis*

## 06 - Registro e Login de Usuários

## Requisitos

1) Aula 01 - Introdução a Frameworks Web e Django: https://youtu.be/LpFNMn6Uw5s
2) Aula 02 - Configurações do Ambiente: https://youtu.be/7zxJi3_HRuw
3) Aula 03 - Primeira Aplicação - Olá Django: https://youtu.be/5MoZZttT6UA
4) Aula 04 - Introdução aos Templates: https://youtu.be/nYnZ-J8PrxA
5) Aula 05 - Introdução aos Models e ORM: https://youtu.be/-0PnRa29iuU

### Introdução

O Registro e Login de usuários é o recurso básico que praticamente todo sistema precisa implementar. O login de usuários habilita o controle de acesso aos recursos do sistema e personalizações por perfil ou papéis dos usuários.

O gerenciamento de usuários é uma funcionalidade que vai muito além do simples registro e login, pois é necessário garantir que usuários também consigam alterar dados pessoais ou sensíveis como senha. O Django já traz pronta a maior parte da estrutura para fazer a gestão dos usuários. Nesta aula vamos focar no registro e no login.

**Nesta aula:**

1) Definir o redirecionamento padrão após login e logout
2) Criar um novo django app para gestão de contas de usuários
3) Criar e personalizar um form Django para registro de usuários
4) Criar a view e o template para registro de usuário
5) Criar as rotas específicas para gestão de contas
6) Controle de acesso pela sessão logada

## Passo 1: Definir o redirecionamento padrão após login e logout

Edite o arquivo **django-do-zero/webapp/settings.py** e acrescente as novas chaves de configuração (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
# na parte inferior do arquivo => /ola é um exemplo, adeque conforme preferência
LOGIN_REDIRECT_URL = '/ola'
LOGOUT_REDIRECT_URL = '/index'
```

(**salve o arquivo**).

## Passo 2: Novo App Django para gestão de contas de usuários

Vamos criar uma novo app Django (módulo Python) para separar as responsabilidades de forma mais clara, o módulo polls lida apenas com equentes, sendo as contas de usuários um outro assunto.

Abra o terminal (ative o venv) e execute o seguinte comando:

django-admin startapp accounts **(ENTER)**

Uma nova pasta e respectivos arquivos serão gerados.

Para que o projeto Django utilize este novo app precisamos habilitá-lo, incluindo na lista de INSTALLED_APPS do arquivo **django-do-zero/webapp/settings.py,** edite o arquivo e faça a inclusão, assim:
(**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)

# localizar a chave de configuração e modificá-la
INSTALLED_APPS = [
  # (...manter tudo o que já existe…)
   'django.contrib.staticfiles',
   'blog.apps.BlogConfig',
   'accounts.apps.AccountsConfig',
]
```

(**salve o arquivo**).

## Passo 3: Criar um form Django para registro/cadastro de usuários

Criar um novo arquivo **django-do-zero/accounts/forms.py** que deve conter o seguinte conteúdo:

```
from django import forms
from django.contrib.auth import get_user_model
User = get_user_model() # obtém o model padrão para usuários do Django

class AccountSignupForm(forms.ModelForm): # define um formulário para registro
   password = forms.CharField(label="Senha", max_length=50,
                              widget=forms.PasswordInput())

   class Meta:
      model = User # conecta o form com o model padrão de usuário
      fields = ('username',  'email', 'password', ) # campos do model a exibir
```

(**salve o arquivo**).

Model User padrão do Django: https://docs.djangoproject.com/pt-br/4.2/ref/contrib/auth/

## Passo 4: Criar a view e o template para registro de usuário

Edite o arquivo **django-do-zero/accounts/views.py** e acrescente a nova view conforme o conteúdo a seguir:

```
# (...manter tudo o que já existe…)

from django.views.generic.edit import CreateView
from django.urls import reverse_lazy
from django.contrib.auth.hashers import make_password # para criptografar a senha
from django.contrib import messages

from django.contrib.auth import get_user_model
User = get_user_model() # obtém o model padrão para usuários do Django

from accounts.forms import AccountSignupForm # importa o form de registro

class AccountCreateView(CreateView):
    model = User # conecta o model a view
    template_name = 'registration/signup_form.html' # template para o form HTML
    form_class = AccountSignupForm # conecta o form a view
    success_url = reverse_lazy('login') # destino após a criação do novo usuário
    success_message = 'Usuário criado com sucesso!'

    def form_valid(self, form)  -> HttpResponse: # executa se os dados válidos
        form.instance.password = make_password(form.instance.password)
        form.save()
        messages.success(self.request, self.success_message)
        return super(AccountCreateView, self).form_valid(form)
```

(**salve o arquivo**).

Criar o arquivo de template para renderizar o formulário de registro de usuário. Dentro da pasta templates crie a  o arquivo **registration/signup_form.html** e o conteúdo essencial para este arquivo é a proteção contra CSRF  (Cross-Site Request Forgery, requisição falsa de outra origem) e a exibição do próprio form que chega na view como um objeto form. Então este mínimo deve ser:

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Enviar</button>
</form>
```

Contudo, uma referência mais completa para o template **registration/signup_form.html**, considerando a estrutura de templates que temos ficará assim (faça as adaptações conforme a estrutura do seu projeto):

```
{% extends '_layout1.html' %} {# herda tudo que já tem em _layout1.html #}

{% block content %} {# substitui  o bloco content em _layout1.html #}
<div class="card">
   <div class="card-header">
      <h5 class="card-title">Registro de Usuário</h5>
   </div>
   <div class="card-body">
      <form method="post">
         {% csrf_token %}
         {{ form.as_p }}
         <button type="submit" class="btn btn-success">Enviar</button>
      </form>
   </div>
</div>
{% endblock %} {# fechar o bloco content #}
```

(**salve o arquivo**).

Vamos precisar de um template para o login do usuário. Faça uma cópia do arquivo template **registration/signup_form.html** para **registration/login.html** e modifique o título para “Acessar” ou outro de sua preferência. O código será o mesmo, mas com o título adequado ao formulário de login (gerenciado automaticamente pelo Django).

<h5 class="card-title">Acessar</h5>

## Passo 5: Criar as rotas específicas para gestão de contas

Criar o novo arquivo de rotas para o App accounts em **django-do-zero/accounts/urls.py** contendo o seguinte conteúdo:

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

urlpatterns = [
   path('accounts/signup', # caminho que vai carregar a view com o formulário
       views.AccountCreateView.as_view(),
       name="signup"
   ),
]
```

(**salve o arquivo**).

Completando a configuração das rotas para gestão de contas de usuário, editar o arquivo principal de rotas do projeto localizado em **django-do-zero/webapp/urls.py** e acrescentar as novas rotas seguindo o exemplo abaixo (**atenção apenas ao conteúdo novo**):

```
# localizar e acrescentar às rotas principais
urlpatterns = [
    # (...manter tudo o que já existe…)
    path('', include('accounts.urls')), # rotas personalizadas como accounts/signup
    path('accounts/', include('django.contrib.auth.urls')), # rotas fornecidas pelo Django
]
```

(**salve o arquivo**).

As rotas padrão fornecidas pelo Django incluem automaticamente as views, os forms e as rotas necessárias à autenticação e a gestão básica das contas de usuários, são elas (**a parte "accounts" é o prefixo que definimos para rotas, mas pode ser alterado ou removido**) :

**accounts**/login/ [name='login']
**accounts**/logout/ [name='logout']
**accounts**/password_change/ [name='password_change']
**accounts**/password_change/done/ [name='password_change_done']
**accounts**/password_reset/ [name='password_reset']
**accounts**/password_reset/done/ [name='password_reset_done']
**accounts**/reset/<uidb64>/<token>/ [name='password_reset_confirm']
**accounts**/reset/done/ [name='password_reset_complete']

Todos os recursos do App **auth** (do Django) são personalizáveis (templates, forms, etc), mas já é possível usar a estrutura padrão para um início rápido.

Para completar a configuração básica de registro e login vamos incluir os links de acesso no menu montado pelo nosso template de layout. Editar o arquivo **templates/_layout1.html** e incluir na seção do menu de navegação o seguinte trecho de código:

```
# (...manter tudo o que já existe…)
<ul class="nav nav-pills">
 {% if user.is_authenticated %}  {# objeto user sempre está disponível nos templates #}
    <li class="nav-item">
       <a href="{% url 'logout' %}" class="nav-link px-2 link-dark">Sair</a>
    </li>
 {% else %}
    <li class="nav-item">
       <a href="{% url 'signup' %}" class="nav-link px-2 link-dark">Cadastrar</a>
    </li>
    <li class="nav-item">
       <a href="{% url 'login' %}" class="nav-link px-2 link-dark">Acessar</a>
    </li>
 {% endif %}
```

(**salve o arquivo**).

Nas linhas de código acima usamos uma estrutura para exibir os links de acesso conforme a situação da sessão do usuário.

Neste ponto já é possível testar.

Com o servidor em execução, acesse no navegador http://127.0.0.1:8000/accounts/signup

Também teremos no menu os links "**Cadastrar**", "**Acessar**" e "**Sair**", conforme o contexto da sessão.

## Passo 6: Controle de acesso pela sessão logada

Existem diversas formas de implementar o controle de acesso. Se for realizada diretamente nas views, temos dois formatos básicos: 1) view baseada em funções, 2) Views baseadas em classe (inclui as views genéricas que utilizamos no CRUD). Vamos implementar os dois exemplos:

Edite o arquivo **django-do-zero/blog/views.py** e inclua as modificações conforme o exemplo a seguir (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
# na parte superior do arquivo
from django.contrib.auth.decorators import login_required

# (...manter tudo o que já existe…)

# localizar as views e modificá-las
@login_required # controle de acesso usando o decorador de função
def ola(request):
    questions = Question.objects.all()
    context = {'all_questions': questions}
    return render(request, 'polls/questions.html', context)
```

(**salve o arquivo**).

Com as configurações acima ao acessar as rotas que respondem usando as views serão direcionadas para o login, caso não exista sessão ativa. Ou seja, apenas usuários logados poderão acessar.