Django Framework do Zero

Como Criar um Aplicativo Web

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

13 - Upload de Arquivos - Foto do Perfil

Requisitos

1) Aulas 01 a 12 - Django Framework do Zero - Playlist: https://www.youtube.com/playlist?list=PLFOqHo8oIjzewcT23HCxJV0xWO451CTJe

Introdução

Os Models no Django suportam nativamente dois tipos de arquivos: o **FileField** e o **ImageField**. Na prática o ImageField é uma especialização do FileField que valida se o arquivo sendo enviado é uma imagem mas requer a instalação do pacote pillow. Neste documento vamos usar o FileField. Além de adequar o Model desejado para trabalhar com vínculo de arquivos é necessário configurar o projeto Django para salvar e servir corretamente os arquivos.

Vamos implementar um exemplo que acrescenta uma imagem ao perfil usuário do nosso projeto django-do-zero.

Nesta aula:

1) Configurar o projeto para suportar arquivos de mídia
2) Acrescentar o FileField ao model CustomUser
3) Criar uma view e template para Editar o perfil do usuário
4) Criar uma rota para view AccountUpdateView

Passo 1: Configurar o projeto para suportar arquivos de mídia

Edite o arquivo **django-do-zero/webapp/settings.py** e acrescente as seguintes linhas no final (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
# na parte inferior do arquivo
MEDIA_URL = '/media/'  # Caminho para servir os arquivos de mídia
MEDIA_ROOT = os.path.join(BASE_DIR, 'public/') # Armazenamento dos arquivos
```

(**salve o arquivo**).

Será necessário configurar também o arquivo principal de rotas para incluir estas rotas para arquivos estáticos.

Edite o arquivo **django-do-zero/webapp/urls.py** e acrescente as seguintes alterações (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
# na parte superior do arquivo
from django.conf import settings
from django.conf.urls.static import static

# na parte inferior do arquivo
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

(**salve o arquivo**).

Em seguida, execute o servidor de desenvolvimento e observe se não há erros.

python manage.py runserver **(ENTER)**

Passo 2: Acrescentar o FileField ao model CustomUser

Edite o arquivo **django-do-zero/accounts/models.py** e acrescente o novo campo da seguinte forma (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
class CustomUser(AbstractUser):
    data_nascimento = models.DateField("Data de Nascimento", null=True, blank=True)
    cpf = models.CharField("CPF", max_length=11, null=True, blank=True)
    imagem = models.FileField(
        upload_to='images/user',
        default=None,
        null=True,
        blank=True # permite apagar o arquivo
    )
```

(**salve o arquivo**).

Siga o fluxo padrão de gerar a migração (makemigrations) e aplicar as alterações (migrate) ao banco de dados. Para isso execute (no terminal e dentro da pasta raiz do projeto):

python manage.py makemigrations **(ENTER)**

Em seguida aplique as migrações executando:

python manage.py migrate **(ENTER)**

Considerando que tudo esteja funcionando, agora basta incluir o novo campo nas respectivas views e templates desejados.

Passo 3: Criar uma view e template para Editar o perfil do usuário

Para habilitar o upload do arquivo (imagem) é necessário incluir o novo campo na listagem de campos, quando desejar editar o perfil. Vamos implementar uma view genérica **UpdateView** para editar o perfil do usuário.

Edite o arquivo **django-do-zero/accounts/views.py** e acrescente a view baseada em classe da seguinte forma (**atenção apenas ao conteúdo novo**):

```
# (...manter tudo o que já existe…)
# na parte superior do arquivo
from django.views.generic.edit import CreateView, UpdateView
from django.contrib.auth.mixins import LoginRequiredMixin

# na parte inferior do arquivo
class AccountUpdateView(LoginRequiredMixin, UpdateView):
    model = User
    template_name = 'accounts/user_form.html'
    fields = ('first_name', 'email', 'imagem',  ) # incluir os campos que deseja liberar a edição
    success_url = reverse_lazy('index') # rota para redirecionar após a edição
    success_message = 'Perfil atualizado com sucesso!'

    def get_queryset(self): # método que altera o objeto recuperado pela view
        user_id = self.kwargs.get('pk')  # recupera o argumento vindo na URL / rota
        user = self.request.user
        if user is None or not user.is_authenticated or user_id != user.id:
            return User.objects.none()

        return User.objects.filter(id=user.id) #  apenas o usuário do perfil logado pode editar

    def form_valid(self, form): # executa quando os dados estiverem válidos
        messages.success(self.request, self.success_message)
        return super(AccountUpdateView, self).form_valid(form)
```

(**salve o arquivo**).

Criar o template **accounts/user_form.html** dentro da pasta templates. Ele exibirá o formulário para editar o perfil. Deve conter um conteúdo similar ao exemplo abaixo:

```
{% extends '_layout1.html' %}

{% block content %}

<div class="card">
    <div class="card-header">
```

```
            <h5 class="card-title">Editar Perfil</h5>
        </div>
        <div class="card-body">
            <form method="post" enctype="multipart/form-data"> {# obrigatório no upload #}
                {% csrf_token %}
                {{ form.as_p }}
                <p>
                    {% if user.imagem.url is not None %}
                        <h5>Imagem atual:</h5>
                        <img src="{{ user.imagem.url }}" style="height: 150px;">
                    {% else %}
                        <span class="alert-warning">Não há imagem.</span>
                    {% endif %}
                </p>
                <button type="submit" class="btn btn-success">Salvar</button>
            </form>
        </div>
    </div>
    {% endblock %} {# fechar o bloco content #}
```

(**salve o arquivo**).

Passo 4: Criar uma rota para view AccountUpdateView

Edite o arquivo **django-do-zero/accounts/urls.py** e acrescente a nova rota seguindo o exemplo abaixo (**atenção apenas ao conteúdo novo**):

```
urlpatterns = [
    # (...manter tudo o que já existe…)
    path('account/<int:pk>/edit',
            views.AccountUpdateView.as_view(),
            name="account_edit"
    ),
]
```

(**salve o arquivo**).

Editar o seu template **_layout1.html** que contém o menu para acrescentar o link para a nova rota de edição de perfil quando o usuário estiver logado. Exemplo:

```
<a class="p-2 text-dark" href="{% url 'account_edit' pk=user.pk %}">Minha conta</a>
<a class="p-2 text-dark" href="{% url 'logout' %}">Sair</a>
```

Uma observação importante é que em todos os templates o objeto “user” fica automaticamente disponível. Se existir uma sessão logada os dados do usuário estarão acessíveis.

Em seguida suba o servidor de desenvolvimento e acesse a nova rota.

python manage.py runserver **(ENTER)**

No navegador acesse http://127.0.0.1:8000/account/<UM ID DE USUÁRIO>/edit

Mas lembre-se que pela implementação realizada somente o usuário logado poderá editar o próprio perfil. Alternativamente pode acessar pelo link "**Minha conta**", se inclui no menu.

Considerando que funcionou tudo corretamente, os próximos passos seriam criar uma página para exibir informações do perfil do usuário usando uma view genérica **DetailView**, por exemplo. Inclusive poderia ter duas views, uma exibindo um perfil público e outra um perfil privado onde somente o dono visualiza determinados campos.

Observe que o campo imagem que é do tipo **FileField** (do Django) é um objeto com suas próprias características e comportamentos.

O caminho web (URL completa) para o arquivo salvo no sistema é obtido pelo atributo **url**. Então para acessar fica **user.imagem.url** e conforme as configurações que fizemos no **Passo 1** estes arquivos serão salvos em **public/images/user.** É recomendado que estes arquivos fiquem separados por serem "dados" do sistema / usuário e devem ser tratados à parte no servidor (e não como base de código do sistema).

Mais detalhes sobre o campo FileField:
https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.fields.files.FieldFile