Django Framework do Zero

Como Criar um Aplicativo Web

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

11 - CRUD Básico - UPDATE

Requisitos

1) Aulas 01 a 10 - Django Framework do Zero - Playlist: https://www.youtube.com/playlist?list=PLFOqHo8oIjzewcT23HCxJV0xWO451CTJe

Nesta aula:

1) Implementar as views uma baseada em classe e outra em função
2) Definir as rotas para views
3) Adicionar um link/botão para “Editar Pergunta”
4) Personalizar o título do formulário de Criação / Edição

Passo 1: Views para edição de perguntas

Agora vamos implementar as views **QuestionUpdateView**() e **question_update**(), ambas terão a responsabilidade de editar perguntas existentes. O objetivo de criar duas é para você conhecer o formato de view baseada em classe (genérica, neste caso) e a view baseada em função.

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

```
# na parte superior do arquivo
from django.shortcuts import render, redirect, get_object_or_404
from django.views.generic.edit import CreateView, UpdateView

# na parte inferior do arquivo
# (...manter tudo o que já existe…)
def index(request): # adaptar para mostrar a lista de perguntas
    aviso = 'Aviso importante: esta página não exige login.'
    messages.warning(request, aviso)
    # return render(request, 'index.html', {'titulo': 'Últimas Enquetes'})
    questions = Question.objects.all()
    context = {'all_questions': questions, 'titulo': 'Últimas Enquetes' }
    return render(request, 'polls/questions.html', context) # renderiza e passa o contexto


class QuestionUpdateView(LoginRequiredMixin, UpdateView):
    model = Question
```

```
    template_name = 'polls/question_form.html'
    success_url = reverse_lazy('index')
    form_class = QuestionForm # compartilha o form criado anteriormente
    success_message = 'Pergunta atualizada com sucesso.'

    # implementa o método que conclui a ação com sucesso
    def form_valid(self, form):
        messages.success(self.request, self.success_message)
        return super(QuestionUpdateView, self).form_valid(form)

@login_required
def question_update(request, question_id):
    context = {} # dados para o contexto do form

    # recupera a pergunta pelo id (chave primária)
    question = get_object_or_404(Question, id=question_id)

    # cria o objeto form já preenchido com os dados da pergunta
    form = QuestionForm(request.POST or None, instance = question)
    context['form'] = form

    if request.method == "POST":
        if form.is_valid():
            form.save()
            messages.success(request, 'Pergunta atualizada com sucesso.')
            return redirect("index")

    return render(request, 'polls/question_form.html', context)
```

(**salve o arquivo**).

Está view compartilha o template **question_form.html** que foi criado na aula de implementação da operação **CREATE**. Portanto, não precisamos criar um novo template HTML, a diferença é que o formulário vem vazio no CREATE e já preenchido no UPDATE.

A adaptação da view **index(),** que passa a listar todas as perguntas, vai exigir a alteração do template polls/questions.html para incluir o botão " + Pergunta" igual existia no template index.html. A tag do link deve ficar fora do laço de repetição pois queremos que ele apareça somente uma vez no alto da página.

Editar o arquivo templates/index.html e acrescente:

```
<a href="{% url 'poll_add' %}" class="btn btn-lg btn-primary font-weight-bold mb-5">
    + Pergunta
</a>
```

Para concluir este **Passo 1** (o maior de todos), vamos realizar um pequeno ajuste na classe **QuestionForm** para lidar corretamente com a data.

Edite o arquivo **django-do-zero/polls/forms.py** e modifique o Form conforme indicado abaixo (**atenção apenas ao conteúdo novo em azul**):

```
 # (...manter tudo o que já existe…)
class Meta:
    # (...manter tudo o que já existe…)
    widgets = {
        'pub_date': forms.widgets.DateInput(
            format='%Y-%m-%d',
            attrs={'type': 'date'}
        ),
    }
```

(**salve o arquivo**).

Passo 2: Definir as rotas para views

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

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

# na parte superior do arquivo
from polls.views import (
  # (...manter tudo o que já existe…) - fizemos uma pequena reorganização
    index, ola,
    QuestionCreateView, question_create,
    QuestionUpdateView, question_update,
)
# na parte inferior do arquivo
urlpatterns = [
    # (...manter tudo o que já existe…)
    path('pergunta/create', question_create, name="poll_create"),
    path('enquete/<int:pk>/edit', QuestionUpdateView.as_view(), name="question_edit"),
    path('pergunta/<int:question_id>/update', question_update, name="question_update"),
]
```

(**salve o arquivo**).

Observe que na rota agora temos parâmetros que são passados à view. Os parâmetros são formados por <TIPO:NOME> e podem ter vários na mesma rota.

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

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

Substituir o ID-EXISTENTE por um ID de uma pergunta já cadastrada no seu ambiente, será exigido login para acesso.

No navegador acesse http://127.0.0.1:8000/enquete/ID-EXISTENTE/edit

Passo 3: Adicionar um link/botão para “Editar Pergunta”

Vamos editar o template **polls/questions.html** dentro da pasta **templates** e incluir uma tag similar ao exemplo abaixo:

Observe que o trecho de código abaixo fica dentro do laço FOR pois a cada nova pergunta tem seu próprio ID e link para editar.

```
<a href="{% url 'question_update' pk=question.id %}" class="link-secondary">
    Editar
</a>
```

(**salve o arquivo**).

Observe que o link da rota será gerada/traduzida automaticamente pelo comando de template **url** que recebe como argumento ‘**question_edit**’ mais o valor do parâmetro esperado pela rota (**int:pk**), um tipo inteiro que representa a chave primária da pergunta.

Suba o servidor (se já não estiver em execução) e acesse a página/rota onde estará o link criado.

Passo 4: Personalizar o título do formulário de Criação / Edição

Ao compartilhar o mesmo template para ambas operações CREATE e UPDATE ficamos com o título do formulário inconsistente.

No Django resolvemos isso tornando o título dinâmico passando para o contexto do template um variável conforme a rota / view sendo acessada.

Edite o arquivo **django-do-zerto/polls/views.py** e acrescente a view baseada em classe da seguinte forma (**atenção apenas ao conteúdo novo em azul, manter todo conteúdo existente!**):

```
# altera as classes e funções conforme a indicação abaixo
class QuestionCreateView(LoginRequiredMixin, CreateView):
    # (...manter tudo o que já existe…) e acrescente o novo método
    def get_context_data(self, **kwargs):
        context = super(QuestionCreateView, self).get_context_data(**kwargs)
        context['form_title'] = 'Criando uma pergunta'

        return context

@login_required
def question_create(request):
```

```
    # (...manter tudo o que já existe…)
    context = {'form_title': 'Criando uma pergunta'} # dados para o contexto do form

class QuestionUpdateView(LoginRequiredMixin, UpdateView):
    # (...manter tudo o que já existe…) e acrescente o novo método
    def get_context_data(self, **kwargs):
        context = super(QuestionUpdateView, self).get_context_data(**kwargs)
        context['form_title'] = 'Editando a pergunta'

        return context

@login_required
def question_update(request, question_id):
    # (...manter tudo o que já existe…)
    context = {'form_title': 'Editando a pergunta'} # dados para o contexto do form
```

(**salve o arquivo**).

No template **question_form.html** substituir o título (conteúdo dentro da tag h5) pela variável de template:

{{ form_title }}

(**salve o arquivo**).

Em seguida suba o servidor de desenvolvimento e acesse as rotas de criação e edição de perguntas para conferir se o título será atualizado conforme o contexto.

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