Django

Exibição de dados, iteração e exibição condicional, integração MTV

Ely – elydasilvamiranda [at] gmail.com

Projeto Connectedin

- Até agora:
  - Criação um projeto com a aplicação perfis;
  - Definição rotas e a estrutura do projeto;
  - Criação uma página dinâmica;
  - Banco de dados real;

Listando os perfis

- Nossa página index.html pode ser aproveitada para listar os perfis cadastrados;
- Para isso, precisamos de duas coisas:
  1. Resgatar da persistência todos os perfis;
  2. Preparar a página pra ser um template que faça uma interação gerando uma listagem;

Questão sobre performance: paginação e carregamento lazy: https://docs.djangoproject.com/en/1.11/topics/pagination/

Obtendo todos os perfis

- Utiliza-se o método all()

```
# connectedin/perfis/views.py
from django.shortcuts import render
from perfis.models import Perfil


def index(request):
    return
       render(request, 'index.html',
              {'perfis' : Perfil.objects.all()})

# código posterior omitido
```

Iterando elementos em templates

- Além das tags {{ }} são chamadas template tags;
- Elas exibem o conteúdo de variáveis;
- Há um outro tipo de template tags que representam estruturas de decisão e de repetição;
- Dentro das tags {% %} podemos ter qualquer expressão, incluindo for each e if/else;
- Assim, para exibir variáveis, usamos {{}} e para exibir instruções e outras expressões: {% %};

5

Iterando elementos em templates

- Para o caso do for, é preciso fazer um fechamento com {% endfor %}:

```
<!-- connectedin/perfis/templates/index.html -->
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>ConnectedIn</title>
    </head>
    <body>
        <h1>Index</h1>
        <ul>
            {% for perfil in perfis %}
                <li>{{ perfil.nome }} / {{ perfil.email }}</li>
            {% endfor %}
        </ul>
    </body>
</html>
```

6

Exibição condicional

- Com as template tags {% %}, também é possível fazer uma renderização condicional;
- Ou seja, exibir ou não um conteúdo dependendo de uma condição:
  - Caso existam perfis, exibimos os perfis;
  - No nosso exemplo, caso não existam perfis cadastrados, podemos exibir uma mensagem;

7

Exibição condicional

- Para o if, também é preciso fazer um fechamento com {% endif %}:

```
<!-- connectedin/perfis/templates/perfis/index.html -->
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>ConnectedIn</title>
    </head>
    <body>
        <h1>Index</h1>
        {% if perfis %}
            <ul>
                {% for perfil in perfis %}
                    <li>{{ perfil.nome }} / {{ perfil.email }}</li>
                {% endfor %}
            </ul>
        {% else %}
            <p>Nenhum Perfil encontrado</p>
        {% endif %}
    </body>
</html>
```

8

Acessando um perfil

- Uma outra possibilidade é a partir da listagem de um perfil, poder acessar um perfil específico;
- Criamos um link no template da iteração/exibição de cada perfil dinamicamente:

```
<!-- connectedin/perfis/templates/perfis/index.html -->
<!-- código omitido -->
        <h1>Index</h1>
        {% if perfis %}
            <ul>
                {% for perfil in perfis %}
                    <li>
                        <a href="/perfil/{{perfil.id}}">{{ perfil.nome }}
                        </a> / {{ perfil.email }}
                    </li>
                {% endfor %}
            </ul>
        {% else %}
            <p>Nenhum Perfil encontrado</p>
        {% endif %}
<!-- código omitido -->
```


URL nomeadas

- Observe o caminho do link "/perfis/{{perfil.id}}" ;
- Ele corresponde ao caminho que definimos no arquivo urls.py e para a função de view exibir;
- Há um possível problema que é a mudança desse caminho no urls.py;
- Caso esse link estivesse em várias páginas, teríamos que fazer alterações em vários pontos;
- Uma alternativa é nomear/apelidar os links no urls.py.

URL nomeadas

- O Django permite nomear uma view;
- No lugar do link, usamos o seu nome e o Django se encarrega de resolver a URL
- Dessa forma, mudanças de endereço ficam restritas a um único arquivo, o urls.py;
- Vamos alterar a função url com a regex que chama nossa view exibir e index
- Passaremos o parâmetro nomeado name, que será usado nas páginas;

URL nomeadas

- Onde tínhamos:

```
from django.conf.urls import include, url

urlpatterns = [
    url(r'^$', 'perfis.views.index'),
    url(r'^perfis/(?P<perfil_id>\d+)$', 'perfis.views.exibir')
]
```

- Teremos:

```
# connectedin/perfis/urls.py

from django.conf.urls import include, url

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^perfis/(?P<perfil_id>\d+)$', views.exibir, name='exibir')
]
```

URL nomeadas

- Utilizamos a template tag {% %} passando dentro dela o nome da URL e o id do perfil:

```
<!-- connectedin/perfis/templates/perfis/index.html -->
<!-- código omitido -->
        <h1>Index</h1>
        {% if perfis %}
            <ul>
                {% for perfil in perfis %}
                    <li>
                        <a href="{% url 'exibir' perfil.id %}">
                          {{ perfil.nome }}</a> / {{ perfil.email }}
                    </li>
                {% endfor %}
            </ul>
        {% else %}
            <p>Nenhum Perfil encontrado</p>
        {% endif %}
<!-- código omitido -->
```


Prática

- Altere sua aplicação da pools para exibir todos as questões;
- A partir de cada questão, deve ser possível clicar e exibir os detalhes em uma página específica.

Connectedin

- Em termos de funcionalidades, foram feitas as seguintes:
  - Listar perfis;
  - Exibir um perfil.
- Nova funcionalidade:
  - Possibilitar convidar um perfil;
  - Algo semelhante a seguir ou "add";

Convidando um perfil

- Primeiro passo: criar uma função no views.py;
- A função de convidar deve receber um perfil como parâmetro;
- Não detalharemos o código agora, então deixaremos ela "vazia" com o comando pass

```
# connectedin/perfis/views.py
# código anterior comentado
def convidar(request, perfil_id):
    pass
```

Definição da rota

- Usaremos um padrão de URL que indica que o "usuário logado" convida um perfil pelo ID.

  Ex: perfis/1/convidar

- Para isso, devemos editar o urls.py da aplicação:

```
# connectedin/perfis/urls.py
from django.conf.urls import url
from perfis import views
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^perfis/(?P<perfil_id>\d+)$',
          views.exibir,  name='exibir'),
    url(r'^perfis/(?P<perfil_id>\d+)/convidar$',
          views.convidar, name='convidar')
]
```


Adicionando o link Convidar

- Adicionaremos o link na página de detalhamento do perfil;
- Para adicionar um link, devemos usar a convenção usada anteriormente:

```
<!-- connectedin/perfis/templates/perfil.html -->
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>ConnectedIn</title>
    </head>
    <body>
         <h1>Detalhe Perfil: {{perfil.nome}}</h1>
         <a href="{% url 'convidar' perfil.id %}">convidar</a>
    </body>
</html>
```


Aplicando OO à classe Perfil

- A classe Perfil define os dados;
- Podemos enriquecê-la acrescentando métodos;
- Um possibilidade é criar um método *convidar*:
  - Terá a lógica para convidar um perfil;
  - Não confundir com o *convidar* da view;
  - Este último delegará o convite ao *convidar* da classe perfil;
  - A ideia principal é que um perfil seja responsável por convidar o outro;

Alterando a classe Perfil

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

class Perfil(models.Model):
    nome = models.CharField(max_length=255, null=False)
    email = models.CharField(max_length=255, null=False)
    telefone = models.CharField(max_length=15, null=False)
    nome_empresa = models.CharField(max_length=255, null=False)

    def convidar(self, perfil_convidado):
        pass
```

Simulando um usuário logado

- Na classe *Perfil*, o método *convidar* recebe um perfil a ser adicionado;
- Porém, quem adiciona o perfil é o usuário atual;
- Em aplicações web esse usuário corresponde ao usuário/perfil logado;
- Como ainda não há login/autenticação, podemos emular/fixar um perfil que fará os convites;
- Criaremos uma função que retornará sempre um mesmo perfil, por exemplo, o de ID igual a 1;

Simulando um usuário logado

```
# connectedin/perfis/views.py
from django.shortcuts import render
from models import Perfil
# código omitido

def convidar(request, perfil_id):
    pass

def get_perfil_logado(request):
   return Perfil.objects.get(id=1)
```

O request deve ser passado

Simulando um usuário logado

- O parâmetro request foi passado para a função get_perfil_logado;
- Trata-se de uma exigência do Django;
- Todas as funções declaradas nos arquivos views.py devem receber o request;
- Caso contrário irá gerar um erro;
- Essa solução do perfil/usuário logado será removido posteriormente;

Convidando um perfil

- Ao exibir um perfil, será exibido também um link para convidá-lo;
- Ao clicar nesse link, o navegador chamará a rota que executará a função *convidar* na view;
- Essa função, receberá o ID do perfil a ser convidado;
- Precisaremos então:
  - Buscar o perfil no banco consultando pelo ID;
  - Passá-lo como parâmetro para o método *convidar* do "usuário logado";
  - Retornar à página *index*, exibindo os perfis;

Convidando um perfil

```
# connectedin/perfis/views.py
from django.shortcuts import render
from perfis.models import Perfil
from django.shortcuts import redirect
# modificando a função convidar

def convidar(request, perfil_id):
    perfil_a_convidar = Perfil.objects.get(id=perfil_id)
    perfil_logado = get_perfil_logado(request)
    perfil_logado.convidar(perfil_a_convidar)
    return redirect('index')

def get_perfil_logado(request):
    return Perfil.objects.get(id=1)
```

25

Definindo uma classe Convite

- O método convidar desta classe precisa criar um Convite;
- Como trabalhamos com OO, podemos modelar isso através de uma classe;
- A classe *Convite* guarda duas referências de perfis: o que convidada e o convidado:

```
# connectedin/perfis/models.py
# logo abaixo da declaração da classe Perfil
class Convite(models.Model):
    solicitante
    convidado
```

26

Relacionando Perfil e Convite

- O Django criará uma tabela com os campos solicitante e convidado;
- Os tipos desses atributos, porém não foram definidos;
- Na verdade, eles serão chaves estrangeiras dos perfis relacionados:

```
# connectedin/perfis/models.py
# logo abaixo da declaração da classe Perfil
class Convite(models.Model):
    solicitante = models.ForeignKey(Perfil)
    convidado = models.ForeignKey(Perfil)
```

27

Filtros

- Filtrar por parte de uma string:

```
>>> perfis = Perfil.objects.filter(email__contains='s')
```

- Filtrar por parte de uma string independente de maiúscula ou minúscula:

```
>>> perfis = Perfil.objects.filter(email__icontains='s')
```

- Filtrar por string no início:

```
>>> perfis  = Perfil.objects.filter(email__startswith='s')
```

O estudo de filtros será aprofundado em outra aula

28

Filtrando convites

- Quando queremos filtrar itens em uma lista de objetos, usamos a função filter;

```
python manage.py shell
>>> from perfis.models import Convite
# retornam uma lista de convites
>>> convites_feitos = Convite.objects.filter(solicitante__id =1)
>>> convites_recebidos = Convite.objects.filter(convidado__id =1)
```

- Perceba o uso do __ (duplo sublinhado):
  - É uma convenção do Django para indicar que estamos navegando no relacionamento do modelo;
  - Por exemplo, queremos do atributo solicitante, que é um objeto, acessar seu id;

29

Relacionamento bidirecional

- Uma funcionalidade desejável é através de um perfil poder consultar rapidamente:
  - Os convites feitos;
  - Os convites recebidos;
  - Algo como:

```
>>> perfil = Perfil.objetcs.get(id=1)
>>> cf = perfil.convites_feitos
>>> cr = perfil.convites_recebidos
```

30

Relacionamento bidirecional

- Implementamos relacionamentos bidirecionais adicionando o parâmetro *related_name*:

```
# connectedin/perfis/models.py
# logo abaixo da declaração da classe Perfil
class Convite(models.Model):
    solicitante = models.ForeignKey(Perfil,
                          related_name='convites_feitos')
    convidado = models.ForeignKey(Perfil,
                          related_name='convites_recebidos')
```

- Os atributos convites feitos e recebidos serão criados automaticamente na classe perfil;
- Assim, um *Convite* conhece os perfis que o compõem e a classe Perfil conhece seus convites;

31

Concluindo a classe Perfil

```
# connectedin/perfis/models.py

from django.db import models

class Perfil(models.Model):
    nome = models.CharField(max_length=255, null=False)
    email = models.CharField(max_length=255, null=False)
    telefone = models.CharField(max_length=15, null=False)
    nome_empresa = models.CharField(max_length=255, null=False)

    def convidar(self, perfil_convidado):
        convite = Convite(solicitante=self,
                          convidado=perfil_convidado)
        convite.save()
```

32

Atualizando o banco

- Precisamos atualizar o esquema:

  >>> python manage.py makemigrations

- Atualizar também o banco:

  >>> python manage.py migrate

Testando

- Realize convites pela aplicação web;
- No prompt faça os seguintes testes:

  python manage.py shell

```
>>> from perfis.models import Perfil
>>> perfil = Perfil.objects.get(id=1)
>>> for convite in perfil.convites_feitos.all():
...         convite.convidado.nome
```


Django

Exibição de dados, iteração e exibição condicional, integração MTV

Ely – elydasilvamiranda [at] gmail.com