Django

URLs, templates, páginas dinâmicas, esquemas e persistência básica

Ely – elydasilvamiranda [at] gmail.com

Urls (relembrando...)

- Urls é um módulo python responsável por realizar o roteamento de URLs do projeto;
- Esse módulo utiliza o mapeamento das URLs utilizando expressões regulares (regex);
- Todas as urls podem ficar em um unico arquivo urls.py;
- É recomendável que cada aplicação contenha seu próprio arquivo urls.py;
- Posteriormente o arquivo urls.py do projeto deve importar os módulos urls.py de cada aplicação.

Definindo uma URL (relembrando...)

- As rotas do projeto são definidas no arquivo connectedin/connectedin/urls.py;
- Deve-se adicionar uma rota a mais ao final do arquivo:

```
# connectedin/connectedin/urls.py

from django.conf.urls import url
from django.contrib import admin
from perfis import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.index),
]
```

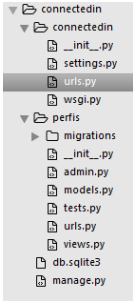

Aprimorando as rotas

- Supondo que um aplicação possuísse umas 20 rotas e que um projeto possuísse 20 aplicações...
- O arquivo urls.py teria 400 linhas de rotas...
- Problema: esse arquivo teria um tamanho que inviabilizaria sua manutenção;
- Solução:
  - cada aplicação ter seu próprio urls.py;
  - o arquivo urls.py do projeto importar os módulos urls.py de cada aplicação;
  - Dessa forma, modularizamos e facilitamos a manutenção de rotas.

Aprimorando as rotas

- Incluir no "urls" do projeto uma inclusão ao arquivo "urls" de cada aplicação;
- A extensão do arquivo é omitida;
- Arquivo connectedin.urls.py:

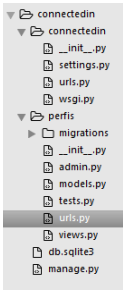
```
# arquivo connectedin/connectedin/urls.py
from django.conf.urls import include, url
from django.contrib import admin

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



Aprimorando as rotas

- Deve-se criar o arquivo connectedin/perfis/urls.py;

```
# connectedin/perfis/urls.py
from django.conf.urls import url
from perfis import views
urlpatterns = [
    url(r'^$', views.index)
]
```

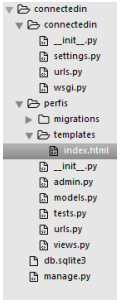


Templates

- Devolvemos anteriormente apenas um texto "solto";
- O ideal é devolver uma página como resposta ao usuário;
- O Django padroniza que as páginas devem ficar em uma pasta chamada "templates"
- Dessa forma, devemos criar a pasta e nela criar uma página chamada index.html;

Templates

```
<!-- connectedin/perfis/templates/index.html -->

<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>ConnectedIn</title>
    </head>
    <body>
        <h1>Bem-vindo ao Connectedin</h1>
    </body>
</html>
```



Templates

- Ao retornar um "template", deve-se alterar o métodos da view que o referencia;
- Deve-se "renderizar" uma página, passando como parâmetro o nome do arquivo:

```python
# connectedin/perfis/views.py

from django.shortcuts import render

def index(request):
    return render(request, 'index.html')
```

9

Páginas dinâmicas

- A página index.html é estática, ou seja, não muda entre requisições;
- Páginas como perfis de redes sociais são templates:
  - mudam de acordo com o usuário;
  - são chamadas dinâmicas;
- Definiremos mais adiante uma página de perfil:
  - Os dados de um usuário serão carregados de forma dinamicamente;
  - A partir de dados da requisição, a página será montada e devolvida ao usuário.

10

A página de Perfil

- Crie pasta templates uma página chamada perfil.html;

```html
<!-- connectedin/perfis/templates/perfil.html -->
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>ConnectedIn</title>
    </head>
    <body>
         <h1>Detalhe do Perfil</h1>
    </body>
</html>
```

11

Definindo a nova view

- Deve-se adicionar uma nova função ao arquivo views.py;
- Ao acessarmos "exibir" na urls, retornaremos a página perfil.html;

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

def index(request):
    return render(request, 'index.html')
def exibir(request):
    return render(request, 'perfil.html')
```

12

Registrando a nova rota

- Para definir a rota para a view criada, deve-se editar o arquivo urls.py da aplicação;
- A ideia é futuramente acessar um perfil pelo seu "id". Exemplo:
  - http://localhost:8000/perfis/1
  - O Django exibiria o perfil cujo id é 1;
  - Esse padrão de URL é fortemente recomendado (REST);
  - Define-se via expressões regulares;
  - o Django receberá esse "parâmetro" de um ou mais dígitos e permitirá buscar perfis pelo id.

13

Registrando a nova rota

- Para permitir que o id seja passado da url ao Django, usamos expressões regulares (regex);
- As regex são definidas como primeiro parâmetro da função url();
- Arquivo urls.py:

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

urlpatterns = [
    url(r'^$', views.index),
    url(r'^perfil/\d+$', views.exibir),
]
```

14

Testando a nova rota

- Acesso o navegador pelos seguintes endereços:
  - http://localhost:8000/perfis/10
  - http://localhost:8000/perfis/
  - http://localhost:8000/perfis/ab
- Apenas a primeira URL funciona, pois pela regex definida deve-se usar o padrão '/perfil/dígitos';

15

Repassando o id ao Django

- Define-se o nome do parâmetro a ser recuperado na função da view;
- Usa-se o padrão ?P<nome_parâmeto>;

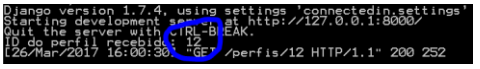
```
# connectedin/perfis/urls.py
from django.conf.urls import patterns, url

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

)
```

16

Recebendo o id no Django

- O nome do parâmetro id repassado deve ser acrescentado na função da view:

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

def index(request):
    return render(request, 'index.html')

def exibir(request, perfil_id):
    print ('ID do perfil recebido: %s' % (perfil_id))
    return render(request, 'perfil.html')
```



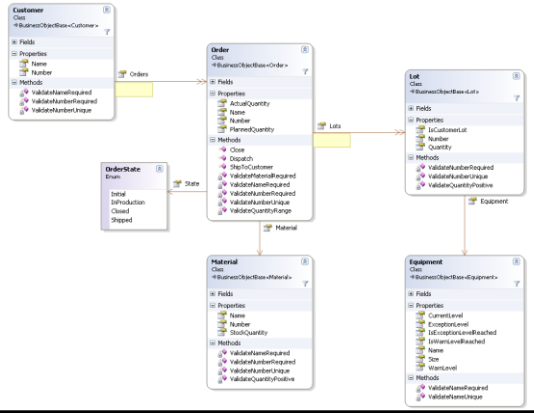
Montando uma página dinâmica

- Os dados de páginas dinâmicas tipicamente vêm de bancos de dados;
- Um alternativa por enquanto é criar dados em memória;
- Para isso, é preciso:
  - Obter o id passado por parâmetro (feito);
  - Criar um objeto Perfil e devolvê-lo à página;
  - Preencher a página em pontos definidos com os dados do perfil;

Models

- Modelo em Django é uma classe básica, uma entidade do sistema;
- Tipicamente são classes que possuem atributos a serem persistidos;
- Fazem parte do modelo de objetos do sistema.
- São classes que ficam armazenadas em meios de persistência como banco de dados;
- Exemplos: Conta, Pessoa, Perfil, Livro, Processo, Produto;
- No Django, colocamos essas classes no arquivo **models.py**.

Exemplo de classes de modelos



Classe Perfil

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


class Perfil(object):
    def __init__(self, nome='', email='',
                 telefone= '',
                 nome_empresa=''):
        self.nome = nome
        self.email = email
        self.telefone = telefone
        self.nome_empresa = nome_empresa
```


Retornando Perfis

- Deve-se instanciar alguns perfis;
- ... E retornar um dicionário para a página:

```
# connectedin/perfis/views.py
from django.shortcuts import render
from perfis.models import Perfil
# código omitido
def exibir(request, perfil_id):
    perfil = Perfil()
    if perfil_id == '1':
        perfil = Perfil('Elvis','elvis@gmail.com',
                        '99999-9999', 'IFPI')
    if perfil_id == '2':
        perfil = Perfil('Lucas', 'lucas@gmail.com',
                        '99987-4567', 'TCE')
    return render(request, 'perfil.html', { "perfil" : perfil})
```


Exibindo um perfil

- Para fazer acesso ao dicionário com o objeto perfil, são utilizadas chaves;
- Dentro das chaves, deve-se acessar o dicionário e os atributos:

```
<!-- connectedin/perfis/templates/perfil.html -->
<!DOCTYPE html>
    <!-- código omitido -->
    <body>
        <h1>Detalhe Perfil</h1>
        nome: {{perfil.nome}}, email: {{perfil.email}},
        telefone: {{perfil.telefone}},
        empresa: {{perfil.nome_empresa}}
    </body>
</html>
```


Prática

1. Crie o arquivo de rotas da aplicação pools e faça as alterações necessárias no arquivo urls.py do projeto mysite;
2. Crie uma página de boas vindas chamada index.html e exiba-a;
3. Crie uma página question.html, uma classe Question com os campos "question_text" e pub_date, representando a data da publicação da pergunta;
4. Altere os arquivos de views, rotas etc. para exibir essa página;
5. Faça uma alteração no na aplicação para que seja possível exibir 3 questões consultando pelo id a partir da barra de endereços do navegador.

Definindo um esquema

- No Django, podemos definir o esquema de banco através do modelo;
- Para isso, devemos ter as seguintes informações:
  - Nome da tabela utilizada;
  - Nomes dos campos e os tipos dos campos;
  - Método de leitura e gravação;
  - Meio de conexão com o banco de dados.
- O primeiro passo entretanto, é fazer com que nossa classe herde de models.Model.

Redefinindo 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)
```

https://docs.djangoproject.com/en/1.11/ref/models/fields/#field-types

Gerando o esquema

- A classe Perfil agora serve como fonte de informação para o esquema do banco;
- O processo de geração do esquema é feito através do comando makemigrations:

```
>> python manage.py makemigrations
Migrations for 'perfis':
  0001_initial.py:
    - Create model Perfil
```


Gerando o esquema

- makemigrations propaga as mudanças em nosso modelo no esquema do banco de dados;
- Quando houver qualquer alteração no modelo, precisaremos executar novamente o comando;
- Assim, um novo esquema é gerado;
- Esquemas ficam armazenados na pasta connectedin/perfis/migrations;
- Porém, o esquema ainda não é aplicado no banco nesta etapa.

Esquema gerado

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.db import models, migrations
class Migration(migrations.Migration):
    dependencies = [ ]
    operations = [
        migrations.CreateModel(
            name='Perfil',
            fields=[
                ('id', models.AutoField(verbose_name='ID', serialize=False,
                      auto_created=True, primary_key=True)),
                ('nome', models.CharField(max_length=255)),
                ('email', models.CharField(max_length=255)),
                ('telefone', models.CharField(max_length=15)),
                ('nome_empresa', models.CharField(max_length=255)),
            ],
            options={      },
            bases=(models.Model,),
        ),
    ]
```


Gerando o banco

- A partir do esquema, devemos gerar o banco com o comando migrate;
- O banco é criado no arquivo connectedin/db.sqlite3:

>> python manage.py migrate

Operations to perform:

Apply all migrations: admin, contenttypes, perfis, auth, sessions

Running migrations:

Applying perfis.0001_initial... OK

Salvando objetos

- Por enquanto, vamos salvar/persistir objetos pelo prompt;
- Podemos abrir o python como um terminal do django através do comando:

  >> python manage.py shell

- Dessa forma, o manage.py fará uma varredura nos arquivos do projeto simplificando imports;

Salvando objetos

- Ao extender a classe model.Models, vários métodos de manipulação de dados são herdados;
- Um deles é o método save();
- Um detalhe: a cada inclusão no banco, o campo id é auto-incrementado:

```
>>> from perfis.models import Perfil
>>> perfil =
   Perfil(nome='ely', email='ely@ely.com',
          telefone='n/a',
          nome_empresa=tce/pi')
>>> perfil.save()
```

Recuperando objetos

- Deve-se usar o método get do atributo objects, definido na classe models.Model;
- Ele aceita como parâmetro o nome de um campo e seu valor;
- Caso a consulta não retorne um resultado, uma exceção é lançada

```
>>> perfil = Perfil.objects.get(id=1)
```


Exibindo perfis

```
# connectedin/perfis/views.py
# código anterior omitido
def exibir(request, perfil_id):

    perfil =
       Perfil.objects.get(id=perfil_id)
    return render(request, 'perfil.html',
                  { "perfil" : perfil})
```


Alterando, excluindo e consultando

- O objeto retornado do banco pode ser alterado e salvado novamente:

```
>>> perfil.nome = 'Ely Miranda'
>>> perfil_encontrado.save()
>>>perfil = Perfil.objects.get(id=1)
>>> perfil.nome
'Ely Miranda'
```

- Excluir um perfil:

```
>>> perfil.delete()
```

- Pesquisar por mais de um campo:

```
>>> p = Perfil.objects.get(nome='Steve',
        email='steve@minecraft.com')
```


Prática

1. Altere a classe Question para ser persistida em banco;
2. Gere o esquema de migração;
3. Gere o banco;
4. Pelo prompt do python:
   - Crie objetos e salve;
   - Busque objetos;
   - Altere e salve objetos;
   - Busque novamente objetos alterados;
   - Exclua um objeto;
   - Consulte com alguma busca parcial por um campo string;
5. Altere a view que exibe as questões estáticas para resgatar as questões do banco.

Django

URLs, templates, páginas dinâmicas,
esquemas e persistência básica

Ely – elydasilvamiranda [at] gmail.com

37