Criando site em Django

Lucas Garcia de Araújo Martins

28 de Abril de 2016

1. Instalação
   - Python
   - Pip
   - VirtualEnvWrapper
2. Iniciando no Django
   - Preparando o ambiente
   - Criando projeto inicial
3. Criando a app de Movimentação
   - Criando uma Transação
   - Criando a migração
   - Django Contrib Admin

Python

Por padrão o Python já vem instalado no Linux e no Mac OS. Não aconselho tentar usar no Windows.

Gerenciador de pacote para Python

pip

A instalação de pacotes no Python podem ser realizadas através de um gerenciador chamado pip. Este funciona na linha de comando.
Através do pip podemos instalar as bibliotecas que iremos usar no nosso projeto.
Quando vai instalar qualquer pacote em Python é interessante que verifique a existência no pip para facilitar a instalação.

Como fazer a instalação do pip

A instalação do pip pode ser feita de acordo com o site oficial `https://pip.pypa.io/en/stable/installing/`. Então vamos fazer a instalação no próximo slide.

Instalando o pip

Para instalar o pip basta fazermos o download neste link `https://bootstrap.pypa.io/get-pip.py` de acordo com o site oficial de instalação `https://pip.pypa.io/en/stable/installing/`. Este arquivo nada mais é do que comandos em Python que irão baixar a última versão do pip e instalar para nós. Depois de baixar o arquivo basta apenas executar o comando com permissão administrativa. No Linux precisa ser root ou seu usuário ser um sudoers(colocar sudo antes do comando como no segundo exemplo abaixo):

```
# python get-pip.py
$ sudo python get-pip.py
```

Instalando o VirtualEnvWrapper

De acordo com o site oficial `http://virtualenvwrapper.readthedocs.org/en/latest/install.html` a instalação consiste em utilizar o pip instalado anteriormente com o seguinte comando.

```
# pip install virtualenvwrapper
```

A vantagem do virtualenvwrapper é que este cria um ambiente separado para instalação de bibliotecas no seu projeto então pode-se ter vários projetos com vários ambientes com bibliotecas diferentes do Python ou Django.

Instalando o VirtualEnvWrapper

Configurando o usuário

Para que os comandos do VirtualEnvWrapper possam funcionar basta que seja adicionado no .bashrc ou .profile os seguintes comandos:

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/Devel
export VIRTUALENVWRAPPER_SCRIPT=/usr/
   bin/virtualenvwrapper.sh
source /usr/bin/
   virtualenvwrapper_lazy.sh
```

Com isto já podemos criar nossos ambientes separados ou isolados.

Instalando o Django

Para começar a trabalhar com o Django é bom ter um ambiente isolado utilizando o virtualenvwrapper.

```
$ mkvirtualenv flisol
$ pip install django
```

No primeiro comando será criado um um ambiente chamado flisol onde as bibliotecas instaladas serão utilizadas somente neste ambiente enquanto estiver ativo. No segundo comando fazemos a instalação do Django neste ambiente isolado para não conflitar com outras versões de Django.

Comando de administração do Django

O Django tem o comando django-admin ou django-admin.py que auxilia nas tarefas comuns ou administrativas dentro de um projeto. Acessar a base de dados, atualizar a estrutura do banco de dados, criar uma nova app, criar um novo projeto, são exemplos de tarefas que podemos fazer com este comando.

Criando um projeto

Então vamos fazer nosso projeto? A criação de um projeto no Django começa com o comando:

```
django-admin.py startproject
    financeiro
```

Entendendo o comando acima:

- O termo django-admin.py é o próprio comando do Django;
- startproject indica que esta sendo criado um novo projeto;
- financeiro é o nome do projeto criado.

Com isto será criado um diretório ou pasta chamado financeiro com os arquivos mínimos de um projeto Django.

Entendendo os arquivos do Django

Dentro do diretório financeiro vai ter os seguintes arquivos e diretórios:

- manage.py é um arquivo que o Django usa para comandos tal como o django-admin.py. Sua diferença do django.admin.py é que apenas funciona no projeto atual;
- O diretório financeiro que contem informações sobre o site. Este costuma ser o nome do projeto em si.

Entendendo os arquivos do Django

Diretório financeiro ou nome do projeto

Arquivos dentro do diretório do projeto.

- __init__.py indica que este diretório tem um módulo Python;
- settings.py o arquivo com configuração do seu projeto. Neste tem informações do Banco de dados, suas apps que estão instaladas, o fuso-horário, idioma do site.
- urls.py Arquivo que contem as urls para o seu site.
- wsgi.py Arquivo de configuração para executar um servidor web.

Realizando as configurações básicas no projeto

Agora vamos realizar as configurações básicas para adequar o Django a nossa realidade. Primeiro vamos abrir o arquivo settings.py que esta dentro do diretório financeiro.

Realizando as configurações básicas no projeto

Linguagem

Por padrão o Django vem configurado para funcionar na linguagem inglês do Estados Unidos. Então agora vamos colocar em Português do Brasil alterando a variável LANGUAGE_CODE para 'pt-br'.

```
LANGUAGE_CODE = 'pt-br'
```

Realizando as configurações básicas no projeto

Fuso horário

Também precisamos colocar o horário do Django para o utilizado no Brasil. Neste caso devemos colocar 'America/Sao_Paulo' na variável TIME_ZONE.

```
TIME_ZONE = 'America/Sao_Paulo'
```

Banco de Dados

O Banco de Dados já vem configurado por padrão para usar o SQLite. Mas pode-se usar outros Banco de Dados Relacionais tais como o PostgreSQL, MySQL, etc.

```
DATABASES = {
    'default': {
    'ENGINE': 'django.db.backends.
      sqlite3',
    'NAME': os.path.join(BASE_DIR, '
      db.sqlite3'),
    }
}
```

No caso do SQLite não precisa de definir usuário e senha. Então fica apenas a Engine que é qual tipo de Banco de Dados está usando e o name é o arquivo que será salvo o Banco de Dados.

Executando o projeto

Agora sim podemos executar o projeto. No Django já existe por padrão um servidor de desenvolvimento. Para utilizar este servidor basta utilizar o comando:

```
$ python manage.py runserver
```

Este irá executar um servidor web na máquina local na porta 8000. Então agora basta abrirmos um navegador e acessar o link `http://localhost:8000`.
O Servidor que acabamos de executar é exclusivo para o desenvolvimento testar o funcionamento do site e não pode ser usado em produção.

Executando o projeto

Funcionamento pleno

Já temos nosso projeto inicial em pleno funcionamento. Porém falta colocarmos algumas páginas para poder ser um site de verdade.

Criando meu primeiro app

Agora iremos começar a fazer nossa aplicação. O Django trabalha com o conceito de app que nada mais é do que módulos no projeto.

Criando o módulo

O primeiro módulo ou app será a movimentação financeira realizada. Então para criar este app devemos digitar o comando:

```
$ python manage.py startapp
    movimentacao
```

Com este comando o Django irá criar os arquivos necessários para termos uma app em nosso projeto.

Instalando a app no projeto

Mesmo após criar nossa app ainda é necessário que seja informado no settings.py que esta app irá ser utilizada no projeto. Para isto basta adicionar a app na lista INSTALLED_APPS como segue abaixo:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'movimentacao',
]
```

Django Models

O Django trabalha com a ideia de modelos que representam os dados. O modelo é como uma tabela no banco de dados. Na verdade o Django cria uma tabela no banco de dados para cada modelo a princípio.

Django Models

models.py

O Django utiliza o arquivo models.py de cada aplicação para mapear os modelos e criar as tabelas. Cada modelo é uma classe Python que contem os atributos da entidade que estamos criando e alguns métodos que o Django utiliza para trabalhar com o modelo tal como a representação legível do modelo. Também tem a classe interna Meta que tem informações do modelo tal como a ordem que os campos serão listados numa consulta padrão, o nome do modelo legível tanto singular como plural.

Django Models

Atributo do modelo

Cada modelo tem alguns atributos que o Django mapeia para uma tabela do Banco de Dados. Então como no Banco de Dados cada atributo tem um tipo *inteiro*, *texto*, *data* no Django também tem esta diferenciação. No Django tem o atributo tipo CharField para texto simples, IntegerField para número inteiro, DecimalField para número decimal ou flutuante, FileField para arquivos, ImageField para imagem, etc.

Criando os modelos

Em nossa app movimentacao iremos ter 2 modelos:

- Categoria
- Transação

Criando os modelos

Categoria

A categoria na movimentação é constituída de um nome.
O nome pode ser alimentação, transporte, telefone, etc.

Criando os modelos

Transação

A transação na movimentação é constituída da data e hora que aconteceu o evento, uma descrição, a categoria e o valor.

Django Models

Exemplo de modelo

O modelo categoria no arquivo models.py:

```
from __future__ import
    unicode_literals
from django.db import models
class Categoria(models.Model):
    nome = models.CharField(
        max_length=255)
```

No exemplo acima temos a primeira linha com a importação do unicode_literals para Python 2.7 utilizar o formato de texto unicode.
Na segunda linha temos a importação do models do Django mais especificamente o db que é responsável pela ligação do Django com o banco de dados.

Django Models

Modelo categoria

O modelo categoria no arquivo models.py:

```
from __future__ import
    unicode_literals
from django.db import models
class Categoria(models.Model):
    nome = models.CharField(
        max_length=255)
```

Na terceira linha tem o modelo movimentacao que é uma classe filha de models.Model.
Na quarta linha tem um atributo chamado nome no qual foi definido um tipo texto que necessariamente terá o tamanho máximo de 255 caracteres.

Django Models

Modelo transação

O modelo transação no arquivo models.py:

```
...
class Transacao(models.Model):
    data_hora = models.DateTimeField
        ()
    descricao = models.CharField(
        max_length=255)
    categoria = models.ForeignKey(
        Categoria)
    valor = models.DecimalField(
        max_digits=11, decimal_places
        =2)
```

Django Models

Modelo transação

A data e hora foi representada pelo tipo DateTimeField. A descrição fica com um CharField tendo o tamanho máximo de 255. A categoria é uma chave estrangeira para a Categoria que no Django é um ForeignKey. O valor é um DecimalField que precisa ter o máximo 11 digitos que iremos usar e também como é valor monetário será aceito até 2 digitos decimais para os centavos. Com isto já é suficiente para o Django criar nosso banco de dados e as tabelas destes modelos.

A migração do banco de dados no Django

No Django utilizamos uma forma de controlar a evolução, criação ou as alterações no banco de dados chamada de migrations. Isto garante que a estrutura do banco de dados durante o decorrer do desenvolvimento tenha a estrutura evoluída e no momento de colocar o projeto em produção teremos um banco de dados consistente. É feito de forma automática e em raros casos que é preciso fazer alguma alteração manual.

Fazendo a migração do banco de dados

Para que o Django saiba que queremos atualizar a estrutura do modelo no banco de dados de uma app é necessário executar o comando makemigrations informando a app a ser atualizada. Nisto no Django será criado arquivos que tem informações da estrutura que queremos ter. Então vamos marcar a evolução da app movimentacao:

```
$ python manage.py makemigrations
    movimentacao
```

Fazendo a migração do banco de dados

Criando o banco de dados

Apos o makemigrations é necessário aplicar as alterações no banco de dados e isto é feito através do migrate. Então agora vamos utilizar o migrate e ver que ele já vai criar o banco de dados e vai ter a estrutura do modelo Transacao e Categoria.

```
$ python manage.py migrate
```

Visualizando o banco de dados

Já temos o banco de dados criado. Como foi utilizado o SQLite podemos usar o seguinte comando para visualizar o Banco de Dados:

```
$ python manage.py dbshell
```

Agora para visualizar as tabelas que existem basta utilizar:

```
sqlite> .tables
```

Django Admin

No Django já existe uma forma simples e rápida de ter a criação, a visualização, a edição e a remoção de um model com a utilização da app admin. Esta app já tem a parte administrativa pronta para ser utilizada com os modelos.

Acessando o Django Admin

O admin tem a url padrão configurada em 'admin/'.
Então para acessar o site administrativo basta que o servidor esteja executando (python manage.py runserver).
Então vamos no navegador acessar a url `http://localhost:8000/admin/`. Irá solicitar o usuário e senha que podemos criar com o comando:

```
python manage.py createsuperuser
```

Irá solicitar o username ou nome do usuário que por padrão podemos colocar admin. O e-mail address podemos deixar em branco. O password pode ser abc123456.
Agora é só voltar no site e entrar com este username e password.
Por padrão já temos a administração do usuário e grupo e permissão que no Django já vem pronto.

Conhecendo o admin

Vamos criar em Usuário e veremos uma lista de usuários que estão cadastrados e neste caso tem apenas o admin. Para editar o usuário basta clicar no link do nome do admin e irá aparecer a edição. Nesta pode-se alterar o nome do usuário, informações pessoais, as permissões e datas importantes.

As permissões padrão do admin

O admin já tem algumas permissões padrão para cada modelo que é gerenciado. São 3 permissões:

- add_nomemodel Criar registro do modelo;
- change_nomemodel Visualiza lista de registros e edita registro;
- delete_nomemodel Apaga registro.

Cada modelo criado já tem estas permissões que podem ser associadas ao usuário ou a um grupo. Um usuário pode ter vários grupos e consequentemente ter estas permissões.

Criando a administração do modelo Categoria

Para criar a administração do modelo Categoria basta editar o arquivo admin.py da app movimentacao. No caso para usar o básico do Admin basta usar o seguinte código:

```
from django.contrib import admin
from movimentacao.models import *

# Register your models here.
admin.site.register(Categoria)
```

Com isto já pode fazer o administrar a categoria e para isto vamos abrir novamente o site admin `http://localhost:8000/admin/` e vamos adicionar um categoria. Vamos clicar em Adicionar na linha de categoria e criar uma categoria com o nome

Nome legível para modelo

Como foi visto o Django utiliza por padrão o nome da classe modelo acompanhada de object para o nome do registro. Para ter outro nome mais legível podemos utilizar o método do python `__unicode__` que retorna um nome legível para um objeto:

```
class Categoria(models.Model):
    nome = models.CharField(
        max_length=255)

    def __unicode__(self):
        return self.nome
```

Agora podemos atualizar a url `http://localhost:8000/admin/movimentacao/categoria` que veremos o nome da categoria cadastrada.

Fazendo busca pelo nome da Categoria

Agora vamos colocar um campo de pesquisa para buscar pelo nome da categoria. Para isto basta editar o admin.py da app movimentacao. Neste caso já vamos utilizar uma classe personalizada do admin para ter esta opção. Primeiro cria-se uma classe filha de admin.ModelAdmin e no admin.site.register(Modelo) fica definido admin.site.register(Modelo, ModeloAdmin) para informar ao admin que tem um classe com as informações de gerenciamento do modelo.

Classe personalizada para o admin da Categoria

Então o arquivo admin.py vai ficar assim:

```
from django.contrib import admin
from movimentacao.models import *

class CategoriaAdmin(admin.ModelAdmin
    ):
      search_fields = ['nome']

# Register your models here.
admin.site.register(Categoria,
    CategoriaAdmin)
```

Na classe personalizada basta informar uma lista de campos que serão utilizados na pesquisa.

Admin da Categoria

Agora podemos voltar ao site `http://localhost:8000/admin/movimentacao/categoria/` e visualizar o campo de pesquisa.
Com isto já temos a administração da Categoria pronta.

Criando o admin para o modelo Transação

Para criar a administração da transação podemos fazer o mesmo que foi feito com a categoria no qual adiciona uma lista de campos para pesquisa:

```
...
class TransacaoAdmin(admin.ModelAdmin
    ):
      search_fields = ['descricao', '
          categoria__nome']

admin.site.register(Transacao,
    TransacaoAdmin)
```

Criando o admin para o modelo Transação

Então voltamos ao site administrativo `http://localhost:8000/admin/` e vemos a Transação para ser gerenciada. Vamos criar uma transação com a data e horário atual com a descrição de Arroz categoria alimentação e valor 11,00. Na pesquisa temos uma diferença que é a categoria no qual pode-se pesquisar por um campo da categoria e isto é permitido usando o thunder(__). Isto permite acessar campos de um uma tabela estrangeira.

Criando o admin para o modelo Transação

Nome em português do modelo transação

O Django admin cria um nome para cada modelo na sua página porém vemos que a Transação ficou com o nome sem acentuação e o plural errado. Isto corrigimos direto no modelo (models.py). É necessário no início da linha do arquivo models.py adicionar "# -*- coding: UTF-8 -*-".

Criando o admin para o modelo Transação

Nome em português do modelo transação

```
class Transacao(models.Model):
    ...
    class Meta:
    verbose_name = u'Transação'
    verbose_name_plural = u'
       Transações'
```

No modelo tem o classe interna Meta que tem meta-informações do modelo tais como nome e nome no plural, as permissões, etc. O verbose_name é para o nome do modelo no singular e o verbose_name_plural é para o nome do modelo no plural.

Criando o admin para o modelo Transação

Migração

Como foi feito uma alteração no modelo e queremos que esta seja salva na estrutura do banco de dado precisa-se fazer a marcação com:

```
$ python manage.py makemigrations
    movimentacao
```

E depois aplicar esta migração no banco com:

```
$ python manage.py migrate
```

Isto faz necessário devido toda alteração ser gerenciada pelo Django e facilita a vida do programador que não precisa ficar alterando a estrutura do banco.

Criando o admin para o modelo Transação

Listando campos

Como aconteceu com a Categoria na Transação na `http://localhost:8000/admin/movimentacao/transacao/` tem o nome de Transacao Object para cada transação criada. Só que neste caso temos mais de um campo que representa a transação então usar o mesmo artifício da Categoria não seria legal. Mas para isto também tem uma solução simples em uma linha no Django Admin bastando colocar o atributo list_display na classe TransacaoAdmin tendo uma lista de campos a serem exibidos.

Criando o admin para o modelo Transação

Listando campos

```
...
class TransacaoAdmin(admin.ModelAdmin
    ):
      search_fields = ['descricao']
      list_display = ['data_hora', '
         descricao','categoria','valor']
...
```

Então temos uma lista com os campos da transação sendo exibido na tela administrativa

Criando o admin para o modelo Transação

Nome dos campos sem acentuação

Agora observamos que a descrição da transação está com o nome sem acento e para resolver isto basta voltar no models.py e alterar a declaração deste campo(descricao) para:

```
class Transacao(models.Model):
    ...
    descricao = models.CharField(u'
       Descrição', max_length=255)
    ...
```

O primeiro parâmetro de um campo do modelo é sempre o nome que será exibido em formulário ou em telas que tenham este campo.

Criando o admin para o modelo Transação

Nome específico para a data e hora

Como também fizemos no nome da descrição, pode-se fazer na data e hora para ficar um nome mais legível.

```
class Transacao(models.Model):
    data_hora = models.DateTimeField(
        u'Data e horário')
```

Criando o admin para o modelo Transação

Nome do objeto

Mesmo usando o list_display o nome do objeto continua Transacao Object no admin e isto pode ser observado quando clica em algum item para editar e aparece no breadcrumb. Então vamos deixar no models.py na classe da Transação o metodo __unicode__ retornando um nome legível.

```
class Transacao(models.Model):
    def __unicode__(self):
        return '%s - %s - %s - %s' %
            (self.data_hora, self.
            descricao, self.categoria,
            self.valor)
```

Criando o admin para o modelo Transação

Nome do objeto

Nosso retorno tem uma formatação particular que o Python permite no qual informa-se uma string de formatação '%s - %s - %s - %s' substituindo o %s por cada item que fica depois de % dentro da string.

Criando o admin para o modelo Transação

Filtro

Vamos criar um filtro para poder escolher apenas visualizar a transação de uma categoria?

Criando o admin para o modelo Transação

Filtro

Então basta configurar no admin.py na classe Transacao um atributo list_filter com os campos que serão os filtros.

```
class TransacaoAdmin(admin.ModelAdmin
    ):
        ...
        list_filter = ['categoria', '
            data_hora', 'valor']
```

Criando o admin para o modelo Transação

Ordenação

Vamos agora adicionar uma nova transação para amanhã referente a um táxi que pegamos para viajar e a categoria é Transporte e foi cobrado 35,00. No caso na própria tela de cadastro de transação pode-se cadastrar uma nova categoria clicando no + ao lado do select de categoria. Isto é uma mão na roda para poder cadastrar um item que esta dentro de outro.

Criando o admin para o modelo Transação

Ordenação

Pode-se notar que as datas da transação ficou em ordem que é inserida e num extrato de banco por exemplo fica em ordem temporal. Para resolver temos de definir no model.py na classe Transação a ordem. Isto é feito utilizando ordering como atributo da classe Meta sendo uma lista de campos para ordenar. No nosso caso queremos que seja ordenado pela data_hora, depois categoria e valor.

Criando o admin para o modelo Transação

Ordenação

```
class Transacao(models.Model):
    ...
    class Meta:
        ordering = ['data_hora', '
            categoria', 'valor']
```