DANJGO

Exemplo de criação de um Blog

Seu primeiro projeto Django!

- Nós vamos criar um blog simples!
- Na console (em um diretório criado por você):
  - django-admin startproject mysite
- Django-admin é um script que irá criar os diretórios e arquivos que se parece com isso:
- ├── manage.py
- └── mysite
- settings.py
- urls.py
- wsgi.py
- __init__.py

Seu primeiro projeto Django!

- manage.py é um script que ajuda com a gestão do site. Com isso seremos capazes de iniciar um servidor de web no nosso computador sem instalar nada, entre outras coisas.
- O arquivo settings.py contém a configuração do seu site.

Configurando

- Vamos fazer algumas alterações no mysite/ settings.py. Abra o arquivo usando o editor de código que você instalou anteriormente.
- Seria bom ter a hora correta no nosso site.
- Em settings.py, localize a linha que contém TIME_ZONE e modifique para escolher seu próprio fuso horário:
  - TIME_ZONE = ‘America/Bahia'

Configurando

- Nós também precisaramos adicionar um caminho para arquivos estáticos (nós vamos descobrir tudo sobre arquivos estáticos e CSS mais tarde)
- Desça até o *final* do arquivo e logo abaixo da entrada STATIC_URL, adicione um novo um chamado STATIC_ROOT:
  - STATIC_URL = '/static/'
  - STATIC_ROOT = os.path.join(BASE_DIR, 'static')

Instalação de um banco de dados

- Há um monte de software de banco de dados diferente que pode armazenar dados para o seu site.
- Nós vamos usar o padrão, sqlite3.
- Isto já está configurado nesta parte do seu arquivo mysite/settings.py:
  - DATABASES = {
  - 'default': {
  - 'ENGINE': 'django.db.backends.sqlite3',
  - 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  - }
  - }

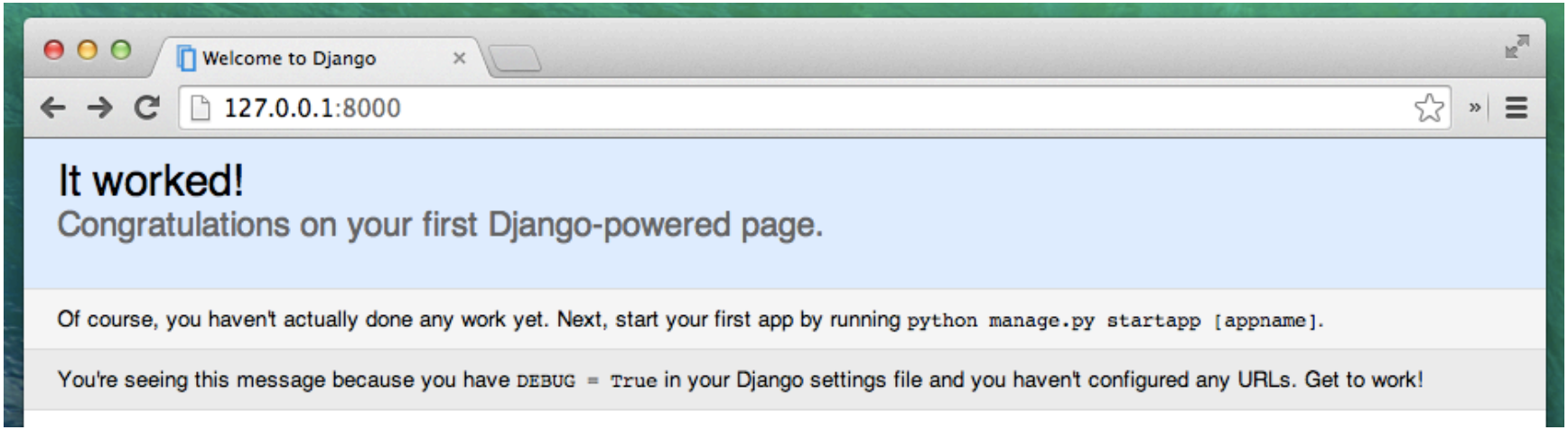
Instalação de um banco de dados

- Para criar um banco de dados para o nosso blog, vamos fazer o seguinte no console. Digite:
  - python manage.py migrate (precisamos estar no diretório que contém o arquivo manage.py)
  - Se isso der certo, você deve ver algo como isto:
    - Operations to perform:
    - Apply all migrations: admin, contenttypes, auth, sessions
    - Running migrations:
    - Applying contenttypes.0001_initial... OK
    - Applying auth.0001_initial... OK
    - Applying admin.0001_initial... OK
    - Applying sessions.0001_initial... OK

E está pronto!

- Hora de iniciar o servidor web e ver se nosso site está funcionando!
- Você precisa estar no diretório que contém o arquivo manage.py
- No console, nós podemos iniciar o servidor web executando o python manage.py runserver
- Abra seu navegador e digite: http://127.0.0.1:8000/

IT WORKED!!!!


Welcome to Django
127.0.0.1:8000
It worked!
Congratulations on your first Django-powered page.
Of course, you haven't actually done any work yet. Next, start your first app by running python manage.py startapp [appname].
You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

Modelos do Django

- Está na hora de criar algum conteúdo!
- Agora o que nós queremos criar é algo que armazene todos os posts no nosso blog
- Para fazer isso precisamos apenas criar objetos.
- Como nós iremos modelar as postagens do blog então? Queremos construir um blog, certo?

Modelos do Django

- Precisamos responder à pergunta: o que é uma postagem de blog? Que propriedades deve ter?
- Com certeza nosso blog precisa de alguma postagem com o seu conteúdo e um título, certo?
- Também seria bom saber quem a escreveu - então precisamos de um autor.
- Finalmente, queremos saber quando a postagem foi criada e publicada.

Modelos do Django

- Nosso Objeto seria assim:
  - Post
  - --------
    - title
    - text
    - author
    - created_date
    - published_date

Modelos do Django

- Que tipo de coisa pode ser feita com uma postagem?
- Seria legal ter algum método que publique a postagem, não é mesmo?
- Então precisamos de um método chamado publicar.
- Como já sabemos o que queremos alcançar, podemos começar a modelagem em Django!

Modelos do Django

- Um modelo no Django é um tipo especial de objeto - ele é salvo em um banco de dados
- Um banco de dados é uma coleção de dados.
- O banco de dados é um local em que você vai salvar dados sobre usuários, suas postagens, etc.
- Usaremos o SQLite como definido anteriomente.

Criando uma aplicação

- Antes de criar os modelos propriamente, é preciso criar uma aplicação que represente o nosso blog.
- Lembre que até agora nós apenas configuramos um ambiente que poderia ser usado para qualquer aplicação.
- Para criar um aplicativo precisamos executar o seguinte comando no console:
  - python manage.py startapp blog

Criando uma aplicação

- Você vai notar que um novo diretório blog é criado e que ele agora contém um número de arquivos.
- Nossos diretórios e arquivos no nosso projeto devem se parecer com este:
- mysite
- | __init__.py
- | settings.py
- | urls.py
- | wsgi.py
- ├── manage.py
- └── blog
- ├── migrations
- | __init__.py
- ├── __init__.py
- ├── admin.py
- ├── models.py
- ├── tests.py
- └── views.py

Criando uma aplicação

- Depois de criar um aplicativo também precisamos dizer ao Django que deve usá-lo.
- Fazemos isso no arquivo mysite/settings.py.
- Precisamos encontrar o INSTALLED_APPS e adicionar uma linha com 'blog', logo acima do). É assim que o produto final deve ficar assim:

Criando uma aplicação

- INSTALLED_APPS = (
- 'django.contrib.admin',
- 'django.contrib.auth',
- 'django.contrib.contenttypes',
- 'django.contrib.sessions',
- 'django.contrib.messages',
- 'django.contrib.staticfiles',
- 'blog',
-)

Criando o modelo Post do nosso blog

- No arquivo blog/models.py definimos todos os objetos chamados Modelos - este é um lugar em que vamos definir nossa postagem do blog.
- Vamos abrir blog/models.py, remova tudo dele e escreva o código como este:

Criando o modelo Post do nosso blog

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    author = models.ForeignKey('auth.User')
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(
            default=timezone.now)
    published_date = models.DateTimeField(
            blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

    def __str__(self):
        return self.title
```

Criando o modelo Post do nosso blog

- É assustador, não? Mas não se preocupe, vamos explicar o que estas linhas significam!
  - **class Post**(models.Model): - esta linha define o nosso modelo é um objeto do tipo Model.
  - **class** é uma palavra-chave especial que indica que estamos definindo um objeto.
  - **Post** é o nome do nosso modelo, podemos lhe dar um nome diferente (mas é preciso evitar os espaços em branco e caracteres especiais). Sempre comece um nome de classe com uma letra maiúscula.
  - **models.Model** significa que o **Post** é um modelo de Django, então o Django sabe ele que deve ser salvo no banco de dados.

Criando o modelo Post do nosso blog

- Agora podemos definir as propriedades que discutimos: titulo, texto, data_criacao, data_publicacao e autor.
- Para isso precisamos definir um tipo de campo (é um texto? É um número? Uma data? Uma relação com outro objeto, por exemplo, um usuário?).

Criando o modelo Post do nosso blog

- models.CharField - assim é como você define um texto com um número limitado de caracteres.
- models.TextField - este é para textos longos sem um limite. Será ideal para um conteúdo de post de blog, certo?
- models.DateTimeField - este é uma data e hora.
- models.ForeignKey - este é um link para outro modelo.

Criando o modelo Post do nosso blog

- Nós não vamos explicar cada pedaço de código aqui, pois isso levaria muito tempo. Você deve olhar a documentação do DJango se você quiser saber mais sobre campos do Model e como definir coisas além destas descritas anteriormente.
- Que tal **def publish(self):**? exatamente o nosso método de publish que falávamos antes

Criando o modelo Post do nosso blog

- **def**, significa que se trata de um função/ método.
- **publish** é o nome do método.
- Métodos muitas vezes **return** algo
- Há um exemplo disso, o método __str__. Nesse cenário, quando chamamos __str__() teremos um texto (**string**), com um título do Post.

Criando tabelas para nossos modelos no banco de dados

- O último passo é adicionar nosso novo modelo para nosso banco de dados.
- Primeiro temos que fazer o Django saber que nós temos algumas mudanças em nosso modelo (só criamos isso), digite:
  - python manage.py makemigrations blog
- Django prepara um arquivo de migração que temos de aplicar agora para nosso banco de dados, DIGITE:
  - python manage.py migrate blog
- PRONTO O NOSSO MODELO ESTÁ CRIADO!!!

Administração

- Para adicionar, editar e remover postagens (ADMINISTRAR O BLOG) nós criaremos usaremos o Django admin
- Vamos abrir o arquivo blog/admin.py e substituir seu conteúdo por:

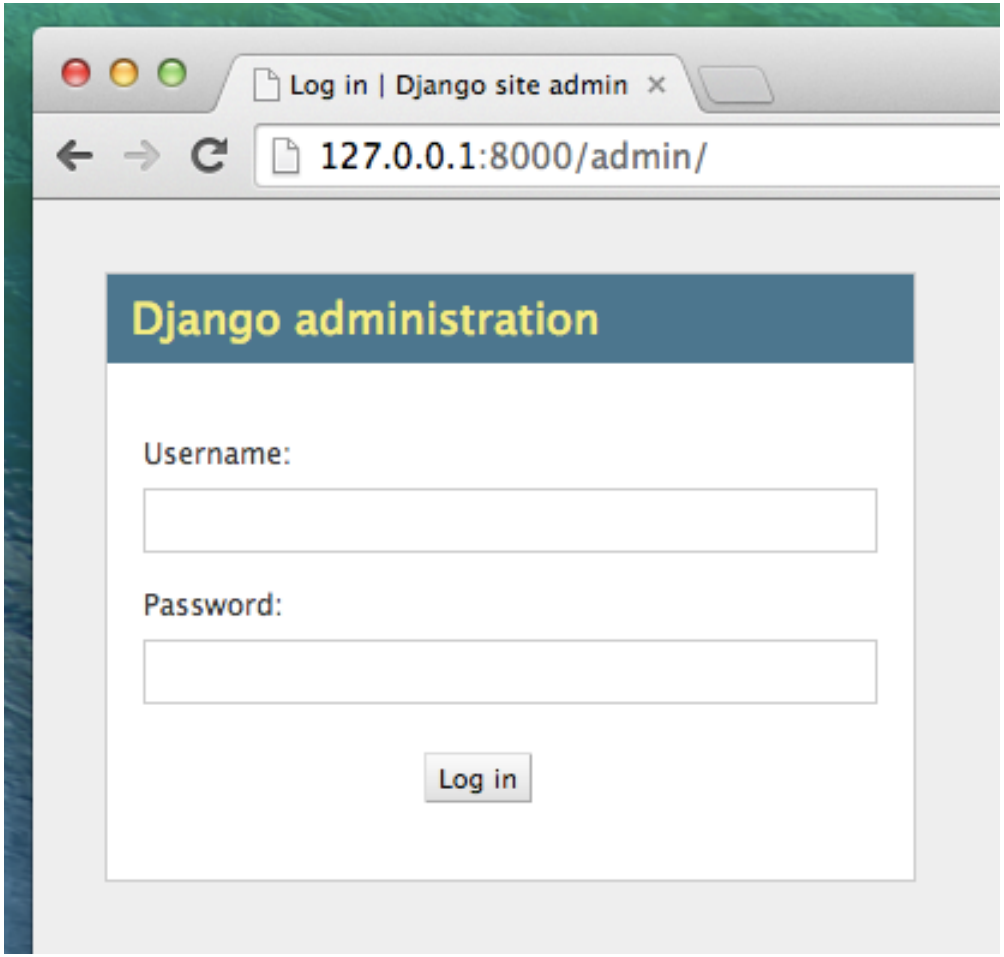
```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Administração

- Como você pode ver, nós importamos (incluímos) o modelo Post definido anteriormente
- Para tornar nosso modelo visível na página de administração, nós precisamos registrá-lo com:
  - admin.site.register(Post)
- OK, hora de olhar para o nosso modelo de Post.
- Lembre-se de executar python manage.py runserver
- Vá para o navegador e digite o endereço http://127.0.0.1:8000/admin/

Admistração



Administração

- Para fazer login você precisa criar um *superuser* - um usuário que possui controle sobre tudo do site.
- Volte para o terminal e digite:
  - python manage.py createsuperuser
  - pressione enter e digite seu nome de usuário (caixa baixa, sem espaço), endereço de e-mail e password quando eles forem requisitados.
  - Não se preocupe que você não pode ver a senha que você está digitando - é assim que deve ser
  - Só digitá-la e pressione 'Enter' para continuar.
  - Volte para a o navegador e faça login com as credenciais de superuser que você escolheu

Administração