Django Tutorial Parte 2: Criando o "esqueleto" de um site

O segundo artigo do tutorial de Django mostra uma forma de criar o "esqueleto" de um website, permitindo que você possa ampliá-lo com caracteristicas especificas do site, caminhos (patchs), modelos (models), visualizações (views) e templates.

Pré-requisitos:	Configurar um ambiente de desenvolvimento Django. Ter lido Tutorial Django: Website de uma Biblioteca Local.
Objetivo:	Ser capaz de usar as ferramentas do Django para começar seus próprios novos projetos de websites.

Visão Geral

Este artigo mostra como você pode criar o escopo de um website, permitindo popul-a-lo com características específicas do seu site, tais como configurações, paths, modelos, views e templates (nós os discutiremos em artigos que seguem à frente).

O processo é direto:

1. Use a ferramenta `django-admin` para criar a pasta do projeto, arquivos de template básicos, e o script de gestão do projeto (**manage.py**).
2. Use o script **manage.py** para criar um ou mais *aplicativos*.

> **Nota**: Um website pode consistir de uma ou mais áreas, como por exemplo, site, blog, wiki, área de download, etc. Django te encoraja a desenvolver esses componentes como aplicativos separados, que podem então ser reutilizados em diferentes projetos, caso seja necessário.

3. Registre os novos aplicativos para inclui-los no projeto.
4. Conecte o mapeador de url/path para cada aplicativo.

Para o website Biblioteca Local a pasta do website e a pasta do projeto terão, ambas, o nome *locallibrary*, e nós teremos apenas um aplicativo chamado *catalog*. O nível hierárquico mais alto da estrutura de pastas ficará assim:

```
locallibrary/          # Pasta do website
    manage.py         # Script para executara as ferramentas do Django pa
    locallibrary/      # Pasta do project folder (criado utilizando o dja
    catalog/           # Pasta do aplicativo (criado utilizando o django-
```

As próximas seções discutem esse processo em detalhes e mostram como você pode testar as mudanças. No final do artigo nós discutiremos algumas das outras configurações do site como um todo, você também pode fazer isso.

Criando o projeto

Primeiro abra o prompt de comando/terminal t(enha certeza que está em seu ambiente virtual), navegue até o diretório que deseja colocar seus aplicativos Django (coloque em um lugar fácil de achar, como dentro da pasta *documentos*), e crie uma pasta para seu novo website (nesse caso: *django_projects*). Acesse então a pasta usando o comando cd:

```
mkdir locallibrary
cd locallibrary
```

Crie um novo projeto usando o comando `django-admin startproject`, como mostrado abaixo, e entre nessa pasta.

```
django-admin startproject locallibrary
cd locallibrary
```

O comando `django-admin` cria uma estrutura com pastas e arquivos como a mostrada abaixo:

```
locallibrary/
    manage.py
    locallibrary/
```

```
locallibrary/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

Nosso diretório de trabalho atual deve parecer com isso:

```
../django_projects/locallibrary/
```

A sub-pasta do projeto *locallibrary* será a raíz para nosso site:

- **__init__.py** é um arquivo em branco que instrui o Python a tratar esse diretório como um pacote Python.
- **settings.py** contém todas as definições do website. É onde nós registramos qualquer aplicação que criarmos, a localização de nossos arquivos estáticos, configurações de banco de dados etc.
- **urls.py** define os mapeamentos de URL para visualização do site. Mesmo que esse arquivo possa conter *todo* o código para mapeamento de URL, é mais comum delegar apenas o mapeamento para aplicativos específicos, como será visto mais adiante.
- **wsgi.py** é usado para ajudar na comunicação entre seu aplicativo Django e o web server. Você pode tratar isso como um boilerplate.

O script **manage.py** é usado para criar aplicações, trabalhar com bancos de dados, e iniciar o webserver de desenvolvimento.

Criando o aplicativo de catálogo

Agora execute o seguinte comando para criar o *catálogo* da aplicação que fará parte de nosso projeto localibrary (o comando deve ser executado na mesma pasta que está o **manage.py** do seu projeto):

```
python3 manage.py startapp catalog
```

> **Nota**: O comando acima é para Linux/macOS X. No windows o comando deve ser: `py -3 manage.py startapp catalog`
>
> Se você está trabalhando com o Windows, substitua `python3` por `py -3` ao longo deste módulo.

> Se você está usando Python 3.7.0, use `py manage.py startapp catalog`

A ferramenta cria uma nova pasta e adiciona alguns arquivos para diferentes partes da aplicação (destacado em negrito abaixo). A maior parte dos arquivos é armazenada de acordo com seu propósito (e.g. views devem ser armazenadas em **views.py**, models em **models.py**, testes em **tests.py**, configurações de administração do site em **admin.py**, registro da aplicação em **apps.py**) e contém algum código mínimo para trabalhar com os objetos associados.

O diretório do projeto atualizado deve parecer com esse:

```
locallibrary/
    manage.py
    locallibrary/
    catalog/
        admin.py
        apps.py
        models.py
        tests.py
        views.py
        __init__.py
        migrations/
```

Além disso, nós temos:

- Uma pasta *migrations*, usada para guardar "*migrações*" — arquivos que permitem atualizar automaticamente seu banco de dados à medida que você modifica seus models.
- **__init__.py** — Um arquivo em branco criado de modo que Django/Python reconheça a pasta como um Python Package e permita que você use seus objetos dentro de outras partes do projeto.

> **Nota**: Você notou o que falta na lista de arquivos acima? Apesar de existir um lugar para suas views e seus models, não há nenhum lugar para colocar seus mapeamentos de URL, templates ou arquivos estáticos. Nós iremos te ensinar como criá-los mais adiante (isso não é necessário em todos websites, mas precisarem em nosso exemplo).

Registrando o aplicativo de catálogo

Agora que a aplicação foi criada, iremos registrá-la com o projeto para que ela seja incluída quando qualquer ferramenta for executada (por exemplo para adicionar models para o banco de

dados). Aplicações são registradas adicionando-as à lista `INSTALLED_APPS` que fica nas configurações do projeto.

Abra o arquivo de configurações do projeto **locallibrary/locallibrary/settings.py** e encontre a definição para a lista `INSTALLED_APPS`. Agora adicione uma nova linha no fim da lista, como a mostrada em negrito abaixo.

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

A nova linha especifica o objeto de configuração do aplicativo (`CatalogConfig`) que foi gerado em **/locallibrary/catalog/apps.py** onde a aplicação foi criada.

> **Nota**: Você deve ter notado que existem vários outros `INSTALLED_APPS` (e `MIDDLEWARE`, pelo final do arquivo de configuração). Eles permitem suporte para o site de administração do Django e, como resultado, várias funcionalidades que ele utiliza (incluindo seções, autenticação etc).

Especificando o Banco de Dados

Tipicamente, esse é o momento em que você também especifica o banco de dados que será usado no projeto— faz mais sentido usar o mesmo banco de dados tanto para desenvolvimento quanto para a produção (quando possível), a fim de evitar pequenas diferenças de comportamento. Você pode encontrar mais sobre as outras opções em Databases (Documentação Django).

Usaremos o banco de dados SQLite para este exemplo porque não esperamos ter muito acesso simultâneo em um banco de dados para demonstração, e também porque ele não requer trabalho adicional de configuração! Você pode ver como o banco de dados é configurado em **settings.py** (mais informações estão incluidas abaixo).

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

Já que nós estamos usando SQLite, nós não precisamos de nenhum outro passo aqui. Vamos ir em frente!

Outras configurações do projeto

O arquivo **settings.py** também é usado para configurar várias outras definições, mas por ora você provavelmente quer mudar apenas a TIME_ZONE — deve se utilizar uma string padrão da Lista de tz time zones (a coluna TZ na tabela contém os valores que você precisa). Mude seu valor de `TIME_ZONE` para uma string relativa ao seu fuso-horário, por exemplo:

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

Tem outras duas definições que você não vai mudar agora, mas que deve ficar ciente:

- `SECRET_KEY`. É uma chave secreta que é usada como parte da estratégia de segurança dos websites Django. Se você não está protegendo seu código durante o desenvolvimento, você precisará usar um código diferente (que talvez seja lido de uma variável de ambiente ou arquivo) quando colocar no ambiente de produção.
- `DEBUG`. Isto habilita a depuração de logs sejam exibidos em um erro ao invés de respostas de status de código HTTP. Isso deve ser definido como `False` na produção, já que informações de debug são úteis para invasores, mas por enquanto nós manteremos `True`.

Conectando o mapeador de URL

O website foi criado com um arquivo mapeador de URL (**urls.py**) na pasta do projeto. Embora você possa usar esse arquivo para gerenciar todos seus mapeamentos de URL, é mais comum fazer os mapeamentos diretamente no aplicativo associado.

faz os mapeamentos diretamente no aplicativo associado.

Abra **locallibrary/locallibrary/urls.py** e leia o texto que explica alguma formas de usar o mapeador de URL.

```
"""locallibrary URL Configuration

The `urlpatterns` list routes URLs to views. For more information please
    https://docs.djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, p
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Os mapeamentos de URL são gerenciados através da variável `urlpatterns` que é uma lista Python de funções `path()`. Cada função `path()` associa um padrão de URL para uma *view específica*, que será exibida quando o padrão for correspondido, ou com outra lista de testes de padrões de URL (no segundo caso, o padrão vem da "URL base" para padrões definidos no módulo target). A lista `urlpatterns` define inicialmente uma função única que mapeia todas URLs com o padrão admin para o módulo `admin.site.urls`, que contém as próprias definições de mapeamento de URL da área de administração do aplicativo.

> **Nota:** A rota em `path()` é uma string que define um padrão de URL para correspondência. Essa string pode incluir um nome de variável (entre tags), e.g. `'catalog/<id>/'`. Esse padrão corresponderá a uma URL como **/catalog/***any_chars***/** e passa *any_chars* para a view como uma string com paramêtros nome `id`). Nós discutiremos métodos de caminho e

> padrões de rota ainda mais em tópicos posteriores

Adicione as linhas abaixo no fim do arquivo a fim de adicionar um novo item à lista `urlpatterns`. Esse novo item inclui um `path()` que encaminha solicitações com o padrão `catalog/` para o módulo `catalog.urls` (o arquivo com a URL relativa **/catalog/urls.py**).

```
# Use include() to add paths from the catalog application
from django.conf.urls import include
from django.urls import path

urlpatterns += [
    path('catalog/', include('catalog.urls')),
]
```

Agora iremos mudar a URL raíz de nosso site (i.e. `127.0.0.1:8000`) para `127.0.0.1:8000/catalog/`; pois esse é o único app que iremos usar neste projeto. Para isso, usaremos uma função view especial (`RedirectView`), que leva como primeiro argumento a nova URL relativa para redirecionar para `/catalog/` quando o padrão URL especificado na função `path()` for chamado (a URL raíz nesse caso).

Adicione as linhas abaixo, novamente no fim do arquivo:

```
#Add URL maps to redirect the base URL to our application
from django.views.generic import RedirectView
urlpatterns += [
    path('', RedirectView.as_view(url='/catalog/')),
]
```

Deixe o primeiro parâmetro da função path vazio, implicando em '/'. Se você escrever o primeiro parâmetro como '/', Django irá te mostar o seguinte aviso assim que iniciar o servidor de desenvolvimento.

```
System check identified some issues:

WARNINGS:
?: (urls.W002) Your URL pattern '/' has a route beginning with a '/'.
Remove this slash as it is unnecessary.
If this pattern is targeted in an include(), ensure the include() patter
```

Por padrão, Django não "serve" arquivos estáticos como CSS, JavaScript e imagens, mas ele pode ser útil para o servidor web de desenvolvimento enquanto você cria seu site. Como

comentário final sobre o mapeador de URL, você pode habilitar a veiculação de arquivos estáticos durante o desenvolvimento adicionando as seguintes linhas.

Coloque o seguinte bloco no fim do arquivo:

```
# Use static() to add url mapping to serve static files during developme
from django.conf import settings
from django.conf.urls.static import static

urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC
```

> **Nota**: Existem várias maneiras de estender a lista `urlpatterns` (acima nós acrecentamos uma nova lista de itens usando o operador `+=` para separar claramente o velho do novo código). Poderiamos ter apenas incluído esse novo padrão de mapeamento na definição da lista original.
>
> ```
> urlpatterns = [
>     path('admin/', admin.site.urls),
>     path('catalog/', include('catalog.urls')),
>     path('', RedirectView.as_view(url='/catalog/', permanent=True)),
> ] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
> ```
>
> Além disso, incluimos a linha para importação (`from django.urls import include`) com o código que usa-o (que facilita ver o que nós adicionamos), porém, é mais comum incluir todas as linhas de import no topo do arquivo Python.

Finalmente, crie um arquivo dentro da pasta catalog e dê o nome **urls.py**, adicione então o seguinte texto para definir um `urlpatterns` importado (e vazio). É aqui onde você adicionará nossos padrões enquanto desenvolvemos o aplicativo.

```
from django.urls import path
from catalog import views
```

```
urlpatterns = [

]
```

Testando o framework do site

Você acabou de criar o escopo do site. Por enquanto o site ainda não faz nada, mas vale a pena testá-lo para garantir que nenhuma de nossas mudanças tenha criado algum problema.

Antes de começarmos, devemos primeiramente executar uma *migração de banco de dados*. Isso atualiza nosso banco de dados para incluir qualquer model em nossas aplicações instaladas (e remove avisos da build).

Migrando Bancos de Dados

Django usa um Object-Relational-Mapper (ORM) que mapeia as definições de Model no código Django para a estrutura do banco de dados subjacente. Como mudamos nossas definições de model, Django localiza as mudanças e cria scripts para migração de banco de dados (em **/locallibrary/catalog/migrations/**) para migrar automaticamente a estrutura de dados subjacente no banco de dados para manter a correnpondência com o model.

Quando criamos nosso website, Django adicionou automaticamente um número de models para serem usados na área admin do site (que nós veremos depois). Execute os comandos abaixo para definir as tabelas para aqueles models no banco de dados (verifique se você está no diretório que contém o arquivo **manage.py**):

```
python3 manage.py makemigrations
python3 manage.py migrate
```

> **Importante**: Você precisará executar os comandos acima sempre que alterar seus models de uma forma que afete a estrutura de dados que precisa ser armazenada (incluindo adição e remoção de todos models e campos individuais).

O comando `makemigrations` *cria* (mas não aplica) as migrações para todos aplicativos instalados em seu projeto (você pode especificar o nome do aplicativo para executar apenas uma migração para um único projeto). Isso te permite checar o código para essas migrações antes

migração para um único projeto). Isso te permite checar o código para essas migrações antes delas serem aplicadas — quando você é experiente em Django, você pode escolher ajustá-los um pouco!

O comando `migrate` aplica as migrações em seu banco de dados (Django rastreia quais foram adicionados ao banco de dados atual).

> **Nota**: Leia Migrations (Documentação Django) para informações adicionais sobre os comandos de migração menos usados.

Testando o website

Durante o desenvolvimento você pode testar o website usando o *webserver de desenvolvimento*, e vê-lo em seu navegador local.

> **Nota**: O web server de desenvolvimento não tem performance ou desempenho suficiente para uso em produção, mas é uma maneira bem fácil de atualizar seu website Django e utilizá-lo durante o desenvolvimento para conseguir um teste rápido e conveniente. Por padrão, o site é "hospedado" em seu computador local (`http://127.0.0.1:8000/`), mas você também pode especificar que outros computadores da rede acessem-o. Para mais informações acesse django-admin and manage.py: runserver (Documentação Django).

Execute o *web server de desenvolvimento* com o comando `runserver` (no mesmo diretório de **manage.py**):

```
python3 manage.py runserver

 Performing system checks...

 System check identified no issues (0 silenced).
 August 15, 2018 - 16:11:26
 Django version 2.1, using settings 'locallibrary.settings'
 Starting development server at http://127.0.0.1:8000/
 Quit the server with CTRL-BREAK.
```

Com o servidor funcionando, você pode ver seu site colocando o endereço

`http://127.0.0.1:8000/` em seu navegador local. Você deve ver uma página de erro como essa:

Não se assuste! Essa página de erro é esperada, pois nós não temos nehuma página ou url definida no módulo `catalogs.urls` (que é para onde somos redirecionados quando usamos a URL para a raíz do site).

> **Nota**: A página acima demontra um ótimo recurso do Django — o log de depuração automatizado. Uma tela de erro será exibida com informações referentes ao erro sempre que uma página não consiga ser encontrada, ou caso o código tenha algum erro. Nesse caso poderemos ver que a URL que nós fornecemos não corresponde a nenhum de nossos padrões de URL (como listado). O log será desativado durante a produção (quando colocamos nosso site online na WEB), nesse caso uma página menos informativa (porém, mais amigável ao usuário) será exibida.

No momento basta saber que o Django está funcionando!

> **Nota**: Você deve executar novamente as migrações e testar o site sempre que fizer alguma mudança signifcante. Não demora muito!

Desafio

O diretório **catalog/** contém arquivos para views, models, e outras partes da aplicação. Abra esses arquivos e inspecione o bolierplate (códigos incluídos em muitos lugares com pouca ou nenhuma alteração).

Como você viu acima, um mapeamento de URL para o site Admin já foi adicionado no arquivo **urls.py** do projeto. Vá à área do admin em seu navegador e veja o que acontece (você pode deduzir a URL correta para o mapeamento acima).

Sumário

Você acabou de criar um "esqueleto" para websties, agora você pode popular o site com URL's, models, views e templates.

models, views e templates.

Como o escopo para o website Local Library está completo e executando, é hora de começar a escrever códigos que farão o website realizar sua função.

Veja também

- Codificando seu primeiro app Django - parte 1 (Documentação Django)
- Aplicativos (Documentação Django). Contém informações de como configurar aplicativos.

Neste módulo

Last modified: 19 de ago. de 2020, by MDN contributors