

.

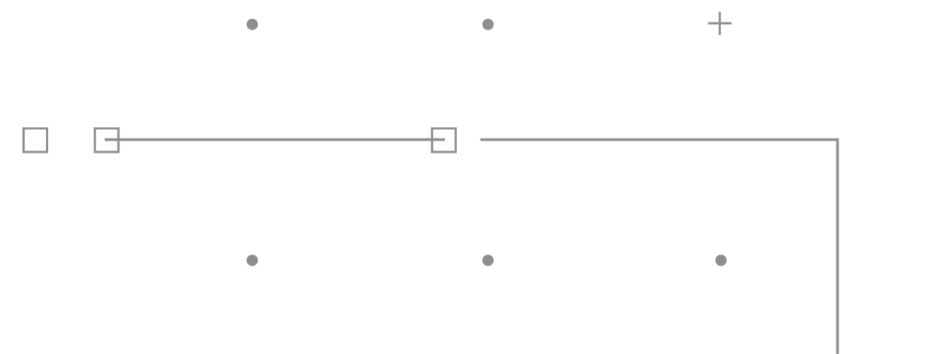
F

-

I

A

P



Trabalhando com Django

DJANGO - Banco de dados

O Django vem com recursos para manipular dados num banco de dados de forma simples e intuitiva.

Por padrão, ele trabalha com SQLite, mas pode-se utilizar outros bancos de dados nele.

Começaremos criando nossos modelos.

DJANGO - Banco de dados

Dentro da pasta da aplicação, existe um arquivo `models.py` que o Django cria automaticamente com o seguinte conteúdo:

```
from django.db import models

# Create your models here
```

DJANGO - Banco de dados

Vamos criar um primeiro modelo chamado `Livro`. Ele representará uma tabela no banco de dados. Deixe o conteúdo do seu arquivo `models.py` da seguinte forma:

```
from django.db import models

# Create your models here
class Livro(models.Model):
    titulo = models.CharField(max_length=256)
    paginas = models.IntegerField()
```

No modelo, teremos o campo `titulo` que é uma string e o campo `paginas` que é um número inteiro.

Automaticamente o Django inclui no modelo um campo `id` que será a chave primária do modelo.

DJANGO - Banco de dados

Para o Django entender os modelos criados, é preciso mexer no arquivo settings.py e incluir essa aplicação na sua devida seção. Dentro desse arquivo procure a variável `INSTALLED_APPS` e deixe ela da seguinte forma:

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

DJANGO - Banco de dados

Após estes procedimentos, agora iremos criar o arquivo que com a estrutura de a serem executados no banco de dados. Execute o seguinte comando:

```
python manage.py makemigrations gestao
```

Ao executar esse comando, o resultado será o seguinte:

```
(django) → livraria python manage.py makemigrations gestao
Migrations for 'gestao':
  gestao/migrations/0001_initial.py
    - Create model Livro
(django) → livraria
```

DJANGO - Banco de dados

Para aplicar todas as alterações no banco de dados, basta rodar o comando:

```
python manage.py migrate
```

```
(django) → livraria python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestao, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying gestao.0001_initial... OK
  Applying sessions.0001_initial... OK
(django) → livraria
```


DJANGO - Banco de dados

Toda vez que o comando citado no slide anterior é executado, ele verifica quais alterações ainda não foram aplicadas no banco de dados e as aplica.

Então se você altera seu arquivo `models.py` deve-se executar o comando `makemigrations` para detectar o que deve ser executado no banco de dados e efetivar essas alterações através do `migrate`

DJANGO - Views

Com a estrutura de banco de dados criada, agora iremos trabalhar com a manipulação desses dados.

Primeiro, vamos começar com um formulário para cadastrar o livro

Na pasta da aplicação `gestao`, crie uma nova pasta chamada `templates`.

Dentro desta pasta, crie uma nova pasta chamada `gestao` e dentro dela um arquivo chamado `form.html` com o seguinte conteúdo

```
<form method="post">
  {% csrf_token %}
  <fieldset>
    <legend>
      <h1>Cadastro de livros</h1>
    </legend>
    <label for="titulo">Título</label><br />
    <input type="text" name="titulo" id="titulo" value="">
    <br /><br />
    <label for="paginas">Páginas</label><br />
    <input type="text" name="paginas" id="paginas" value="">
    <br /><br />
    <input type="submit" value="Salvar">
  </fieldset>
</form>
```

DJANGO - Views

Edite o arquivo `views.py` para ficar da seguinte forma:

```
from django.shortcuts import render

# Create your views here
def form(request):
    return render(request, 'gestao/form.html')
```

É importado a função `render`, que é responsável por pegar o HTML criado e retorna a requisição com o seu conteúdo.

Ele busca dentro da pasta `templates` o arquivo chamado `gestao/form.html` para realizar esse processo

DJANGO - Views

Edite o arquivo `urls.py` para ficar da seguinte forma:

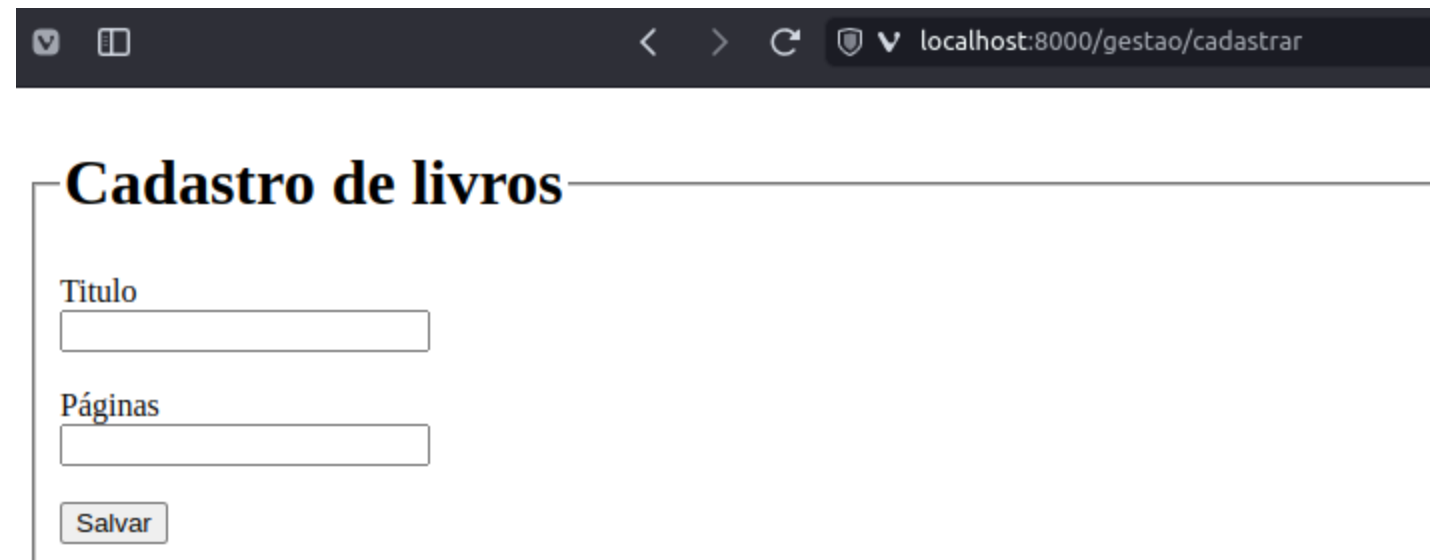
```
from django.urls import path

from . import views

urlpatterns = [
    path('cadastrar/', views.form, name='form')
]
```

DJANGO - Views

Ao rodar o projeto e acessar a url <http://localhost:8000/gestao/cadastrar>, deverá aparecer a seguinte tela:



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/gestao/cadastrar'. The page content is a form titled 'Cadastro de livros'. The form contains two input fields: 'Titulo' and 'Páginas'. Below these fields is a 'Salvar' button.

Cadastro de livros

Titulo

Páginas

Salvar

DJANGO - Views

A seguir, iremos salvar os dados e redirecionar para a listagem de livros. Primeiro, criaremos a lista de livros de uma forma bem simples. Dentro da pasta `templates/gestao`, crie um arquivo

`list.html` com o seguinte conteúdo:

```
{% if livros %}
  <ul>
    {% for livro in livros %}
      <li>
        <a href="/gestao/{{ livro.id }}">
          {{ livro.titulo }}
        </a>
      </li>
    {% endfor %}
  </ul>
{% else %}
  <p>Nenhum livro cadastrado.</p>
{% endif %}
```

DJANGO - Views

Edite o arquivo `views.py` para ficar da seguinte forma:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.urls import reverse

from .models import Livro

# Create your views here
def index(request):
    livros = Livro.objects.all()
    return render(request, 'gestao/list.html', {'livros': livros})

def form(request):
    return render(request, 'gestao/form.html')
```

DJANGO - Views

Edite o arquivo `urls.py` para ficar da seguinte forma:

```
from django.urls import path

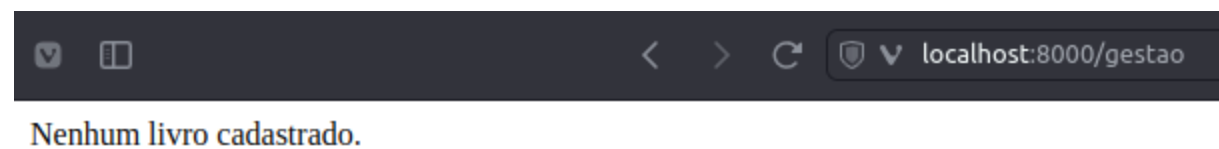
from . import views

app_name = "gestao"

urlpatterns = [
    path('cadastrar/', views.form, name='form')
    path('', views.index, name='index')
]
```


DJANGO - Views

Ao acessar <http://localhost:8000/gestao> deverá aparecer a seguinte página:



DJANGO - Views

Agora iremos cadastrar os livros. Adicione a seguinte função no arquivo `views.py`

```
def save(request):  
    titulo = request.POST['titulo']  
    paginas = int(request.POST['paginas'])  
    livro = Livro(titulo=titulo, paginas=paginas)  
    livro.save()  
  
    return HttpResponseRedirect(reverse("gestao:index"))
```

DJANGO - Views

Edite o arquivo `urls.py` para ficar da seguinte forma:

```
from django.urls import path

from . import views

app_name = "gestao"

urlpatterns = [
    path('salvar/', views.save, name='save')
    path('cadastrar/', views.form, name='form')
    path('', views.index, name='index')
]
```

DJANGO - Views

E finalmente, deixe seu arquivo `form.html` da seguinte forma:

```
<form action="{% url 'gestao:save' %}" method="post">
  {% csrf_token %}
  <fieldset>
    <legend>
      <h1>Cadastro de livros</h1>
    </legend>
    <label for="titulo">Título</label><br />
    <input type="text" name="titulo" id="titulo" value="">
    <br /><br />
    <label for="paginas">Páginas</label><br />
    <input type="text" name="paginas" id="paginas" value="">
    <br /><br />
    <input type="submit" value="Salvar">
  </fieldset>
</form>
```

DJANGO - Views

Ao final desse processo todo, ao criar um livro, ele será redirecionado para a página de listagem de livros!

DJANGO - Explicações

Mas o que aconteceu até agora? Vamos as explicações!

Primeiro, o arquivo `views.py`.

- Na linha 1, é importada a classe `HttpResponseRedirect`. Ela é responsável por dar uma resposta com o código de status 302 e redireciona o tráfego para a url desejada
- Na função `save`, pegamos os dados requisição feita através de um método **POST**, que no Django virá um dicionário onde as chaves são os campos enviados do formulário. Eles são baseados na propriedade `name` dos inputs que existem dentro da tag `form`
- Cria-se uma instância da classe Livro, que é a representação de uma tabela no banco de dados, e ao chamar o método `save`, salva os dados no banco
- Ao final, redireciona a aplicação para a url que tem o nome `gestao:index`
- Na função `index`, é utilizada a classe `Livro` para fazer uma busca de todos os livros cadastrados no banco de dados. A função render recebe dois argumentos obrigatórios que é a requisição feita e qual html deseja renderizar. Ele aceita um terceiro argumento, que é um dicionário onde é possível passar qualquer conteúdo a ser utilizado dentro do arquivo html, o que é chamado de contexto

DJANGO - Explicações

O arquivo `urls.py`.

- Na linha 5 é definido a variável `app_name`, para assim conseguir utilizar namespaces no nome das rotas
- A variável `urlpatterns` é uma lista que contém todas as rotas que serão utilizadas pela sua aplicação. Para utilizar na aplicação, é definido o name, que é a forma que o Django vai se basear para achar uma rota. Quando definimos o `app_name`, podemos utilizar da `{app_name}:{name}`

DJANGO - Explicações

O arquivo `list.html`.

- Na primeira linha, é verificada a variável chamada `livros`, que foi passada através do contexto.
- Django oferece um recurso chamado **template language**. Com ele, é possível escrever código muito parecido com Python dentro de um arquivo html. Sua sintaxe é sempre denotada por `{% %}`
- Caso a variável `livros` esteja vazia ou seja nula, mostra uma mensagem. Do contrário, faz uma iteração nela e mostra a lista de livros

LEMBRE-SE: TEMPLATE LANGUAGE NÃO É A MESMA COISA QUE ESCREVER CÓDIGO PYTHON

DJANGO - Explicações

O arquivo `form.html`.

- É definido o action para onde o formulário vai enviar os dados. É utilizado uma **template tag** chamada `url` que procura através de uma string a url a ser utilizada no action. Ou seja, conforme citado anteriormente, foi utilizado o `{app_name}:{name}` de uma url definida no `urls.py` para colocar a url no action do formulário.
- Na linha 2, é utilizada mais uma template tag chamada `csrf_token`. Sua função é garantir que a requisição seja feita apenas através desse formulário. É uma forma segura de garantir que sites de terceiros não tentem fazer requisições na sua aplicação. CSRF é a sigla de **Cross-Site Request Forgery**

DJANGO - Views

Até aqui foi feito a listagem e criação de livros. Agora criaremos uma url para visualizar os dados de um livro. Dentro da pasta `templates/gestao`, crie um arquivo chamado `view.html` com o seguinte conteúdo:

```
<p>Titulo: {{ livro.titulo }}</p>
<p>Páginas: {{ livro.paginas }}</p>
```

DJANGO - Views

Agora criaremos a rota para renderizar o `view.html`. Edite o arquivo `urls.py` e deixe ele da seguinte forma:

```
from django.urls import path

from . import views

app_name = "gestao"

urlpatterns = [
    path('livro/<int:livro_id>', views.view, name='view')
    path('salvar/', views.save, name='save')
    path('cadastrar/', views.form, name='form')
    path('', views.index, name='index')
]
```

Foi criada a rota `livro/<int:livro_id>`. Essa sintaxe define que será recebido um parâmetro que será passado um argumento para a função chamado `livro_id`. Veremos a seguir como fica a implementação dessa rota

DJANGO - Views

Adicione no arquivo `views.py` a seguinte função:

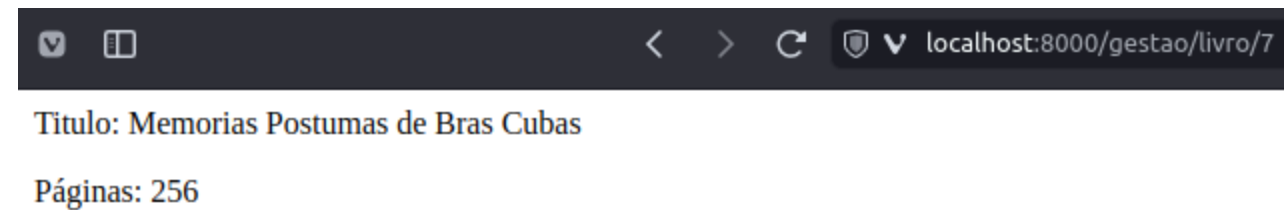
```
def view(request, livro_id):  
    livro = get_objet_or_404(Livro, pk=livro_id)  
    return render(request, 'gestao/view.html', {'livro': livro})
```

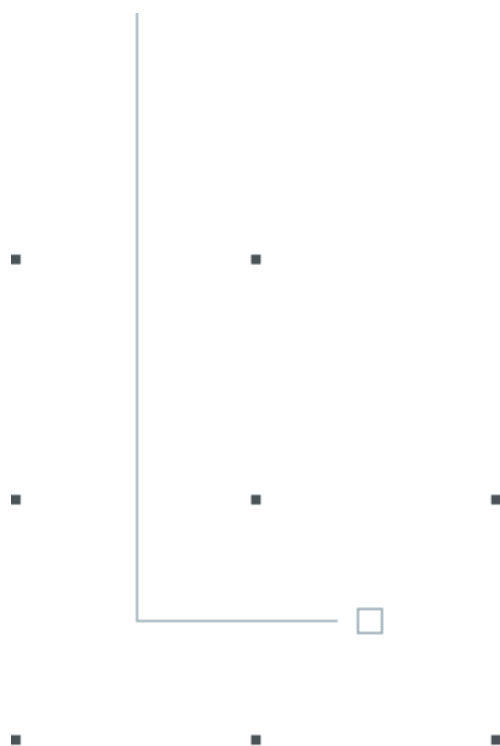
Como visto no arquivo `urls.py`, é criada a rota passando um argumento chamado `livro_id`. Aqui na função, basta colocar ele como argumento da sua função que o Django vai fazer toda a mágica para isso acontecer.

Também é utilizada a função `get_object_or_404`, que deve ser importada do pacote `django.shortcuts`. Essa função automaticamente vai retornar um status 404 para o cliente caso ele tente acessar um livro que não existe

DJANGO - Views

Pronto! Ao final disso tudo, ao acessar a listagem de livros e clicar em algum livro, deve aparecer a seguinte página:





.

F

-

I

A

P

