



Tutorial Básico de Django Framework

Django é um framework para desenvolvimento rápido para web, escrito em Python, que utiliza o padrão model-template-view. Foi criado originalmente como sistema para gerenciar um site jornalístico na cidade de Lawrence, no Kansas. Tornou-se um projeto de código aberto e foi publicado sob a licença BSD em 2005.

1- Requisitos:

Como o Django é um framework baseado em Python o único requisito básico dele é ter instalado o Python (2.3 ou superior.) ([Download Python](#))
E para executar o Django usamos um ambiente virtual do Python, [saiba mais](#)

2- Instalação:

Antes de tudo, verifique se já possui o Django instalado com o seguinte comando

```
[ $ python -m django --version ]
```

(pode ser necessário que ao invés de 'python' se utilize 'python3', dependendo da versão instalada).

Todo o guia de instalação pode ser consultado neste [Link](#)

3- Documentação:

Como o objetivo do tutorial é ser simples e dinâmico, muitas coisas podem ficar de fora ou serem explicadas de forma rápida, para isto você pode acessar toda a documentação oficial do Django por [Aqui](#)

Aqui irei seguir o padrão model-template-view do Django e mostrar como se pode fazer um CRUD usando apenas django.

Iniciando um projeto: Com o ambiente virtual ativado, abra a sua pasta para começar.

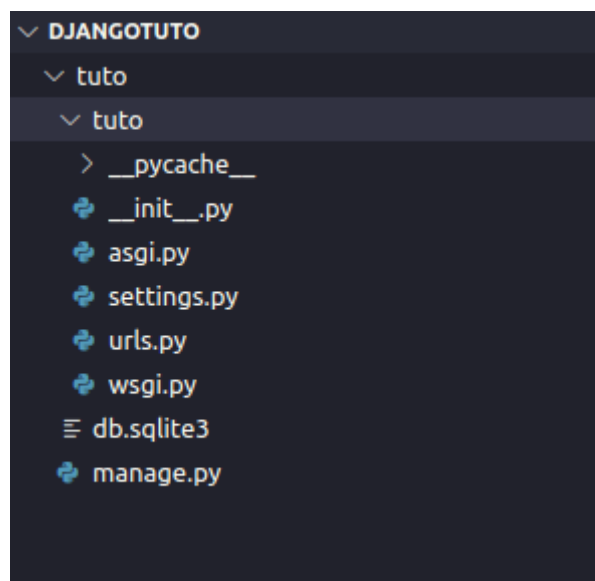
Comando para iniciar um projeto:

```
[ $ django-admin startproject "nome-do-projeto" ]
```

(No lugar de "nome-do-projeto" irei usar "tuto")

Os arquivos necessários para começar já são criados no diretório onde o comando foi executado.

Visão Geral do oque temos:



- O diretório "nome-do-projeto"/ exterior é um contêiner para o seu projeto. Seu nome não importa para o Django; você pode renomeá-lo para qualquer nome que você quiser.
- **manage.py**: um utilitário de linha de comando que permite a você interagir com esse projeto Django de várias maneiras.
- O diretório "nome-do-projeto"/ interior é o pacote Python para o seu projeto. Seu nome é o nome do pacote Python que você vai precisar usar para importar coisas do seu interior (por exemplo, "nome-do-projeto".urls).
- "nome-do-projeto"/**__init__.py**: um arquivo vazio que diz ao Python que este diretório deve ser considerado um pacote Python.
- "nome-do-projeto"/**settings.py**: configurações para este projeto Django. Configurações do Django irá revelar para você tudo sobre o funcionamento

dos settings.

- "nome-do-projeto"/**urls.py**: as declarações de URLs para este projeto Django; um "índice" de seu site movido a Django.
- "nome-do-projeto"/**asgi.py**: um ponto de integração para servidores web compatíveis com ASGI usado para servir seu projeto.
- "nome-do-projeto"/**wsgi.py**: um ponto de integração para servidores web compatíveis com WSGI usado para servir seu projeto.

Para verificar se tudo foi instalado de forma correta, execute o esse comando:

```
[ $ python manage.py runserver ]
```

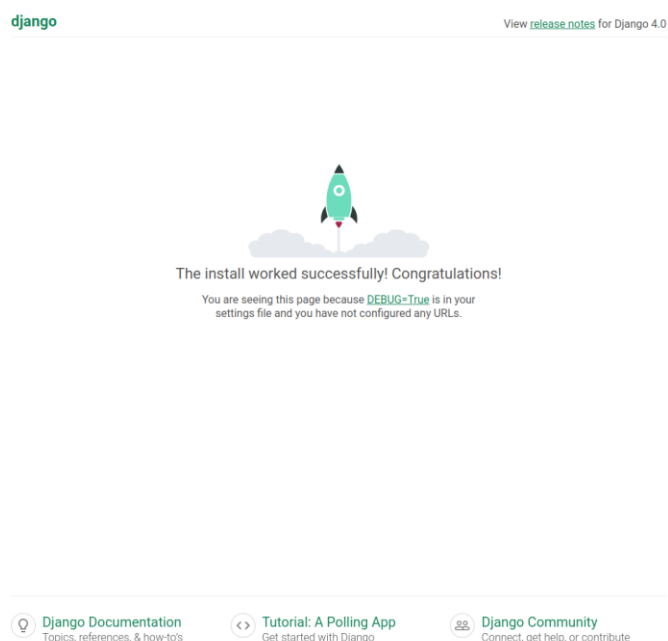
Caso você receba o seguinte resultando:

```
PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL
(env) guilherme.souza@pr7008251:~/djangotuto/tuto$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
August 19, 2022 - 19:54:22
Django version 4.0.6, using settings 'tuto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Quer dizer que tudo ocorreu como deveria. Abrindo o <http://127.0.0.1:8000/>, você verá uma tela como esta:



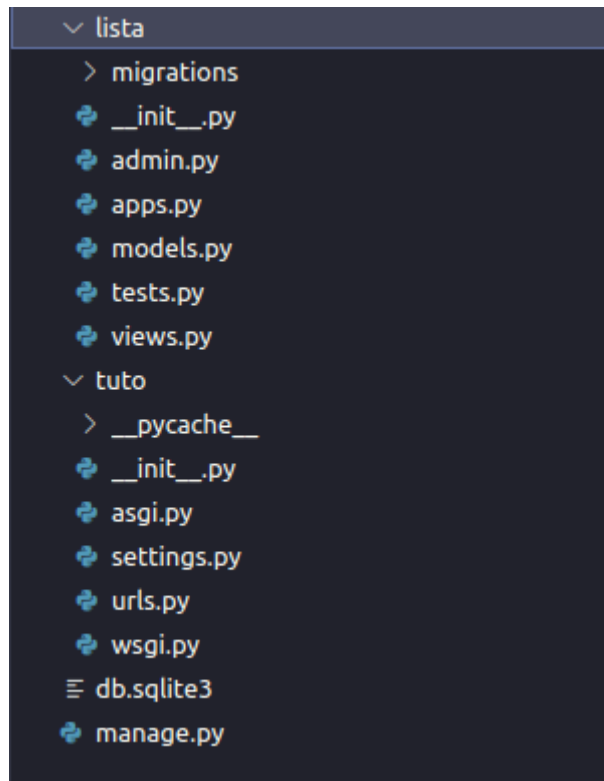
Isso quer dizer que tudo está funcionando certinho!

Agora precisamos criar nossa aplicação. Certifique-se de que esteja no mesmo diretório do **manage.py** e execute o seguinte comando:

```
[ $ python manage.py startapp "nome-da-aplicacao" ]
```

(No lugar de “nome-da-aplicacao” irei usar “lista”)

Este comando serve para criar uma pasta com os arquivos necessários para o funcionamento de nossa aplicação, será algo assim:



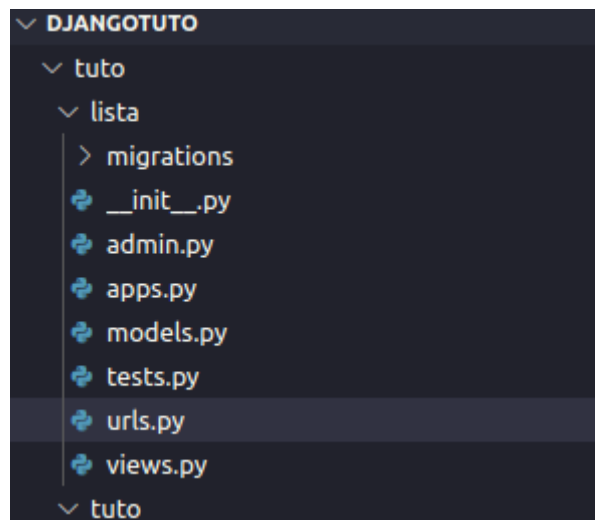
Você entenderá o que cada arquivo faz mais para frente, por agora, vamos criar nossa primeira view. Abra o arquivo **lista/views.py** e cole o seguinte código.

```
[  
  
from django.http import HttpResponse  
def index(request):  
    return HttpResponse('Olá Mundo!')  
  
]
```

Está é a view mais simples possível no Django, agora precisamos chamar ela, para isso temos que “mapear” na urls.py.

Como uma boa prática, dividimos os arquivos “urls.py” para cada aplicação/projeto e, quando necessário, apenas as referenciamos.

Crie um arquivo chamado urls.py no diretório da aplicação.
Que agora deverá estar assim:



Agora aponte a raiz da URLconf para o módulo “**lista.urls**”

No arquivo **tuto/urls.py** adicione o seguinte comando:

```
[  
from django.contrib import admin  
from django.urls import include, path  
  
urlpatterns = [  
    path(' ', include('lista.urls')),  
    path('admin/', admin.site.urls),  
]
```

A função `include()` permite referenciar outras URLconfs. Qualquer lugar que o Django encontrar `include()`, irá recortar todas as partes da URL encontrada até aquele ponto e enviar a string restante para URLconf incluído para processamento posterior.

```
]
```

No arquivo **lista/urls.py** adicione o seguinte comando:

```
[  
from django.urls import path
```

```
from lista.views import index
```

```
urlpatterns = [  
    path('index/', index, name='index'),  
]  
]
```

Agora que a view index está ligada no URLconf, verifique o que foi feito com o seguinte comando:

```
[ $ python manage.py runserver ]
```

Acesse <http://localhost:8000/index/> no seu navegador, e deverá ver o texto “Olá Mundo! ”, o qual foi definido na view “index”.

Agora precisamos configurar o banco de dados do Django, ele já possui, por padrão, um banco de dados chamado SQLite, que pode ser substituído por outras aplicações, mas para experimentar ele já é o suficiente.

Abra o arquivo **tuto/settings.py**, vamos fazer algumas alterações. Primeiro em “INSTALLED_APPS”, vamos adicionar nossa aplicação, basta colocar o nome dela entre aspas e uma vírgula, de modo que fique assim.

```
33 INSTALLED_APPS = [  
34     'django.contrib.admin',  
35     'django.contrib.auth',  
36     'django.contrib.contenttypes',  
37     'django.contrib.sessions',  
38     'django.contrib.messages',  
39     'django.contrib.staticfiles',  
40     'lista',  
41 ]
```

Agora em “LANGUAGE_CODE” e “TIME_ZONE”, vamos mudar para “pt-br” e “America/Sao_Paulo”, respectivamente.

De modo que fique:

```
107 LANGUAGE_CODE = 'pt-br'
108
109 TIME_ZONE = 'America/Sao_Paulo'
110
```

Aqui definimos o fuso horário e o idioma padrão da nossa aplicação.

Para carregar/criar todo o banco de dados das aplicações vamos executar alguns comandos.

Dentro da sua venv, no terminal, execute:

```
[ $ python manage.py migrate ]
```

Que retornara algo assim:

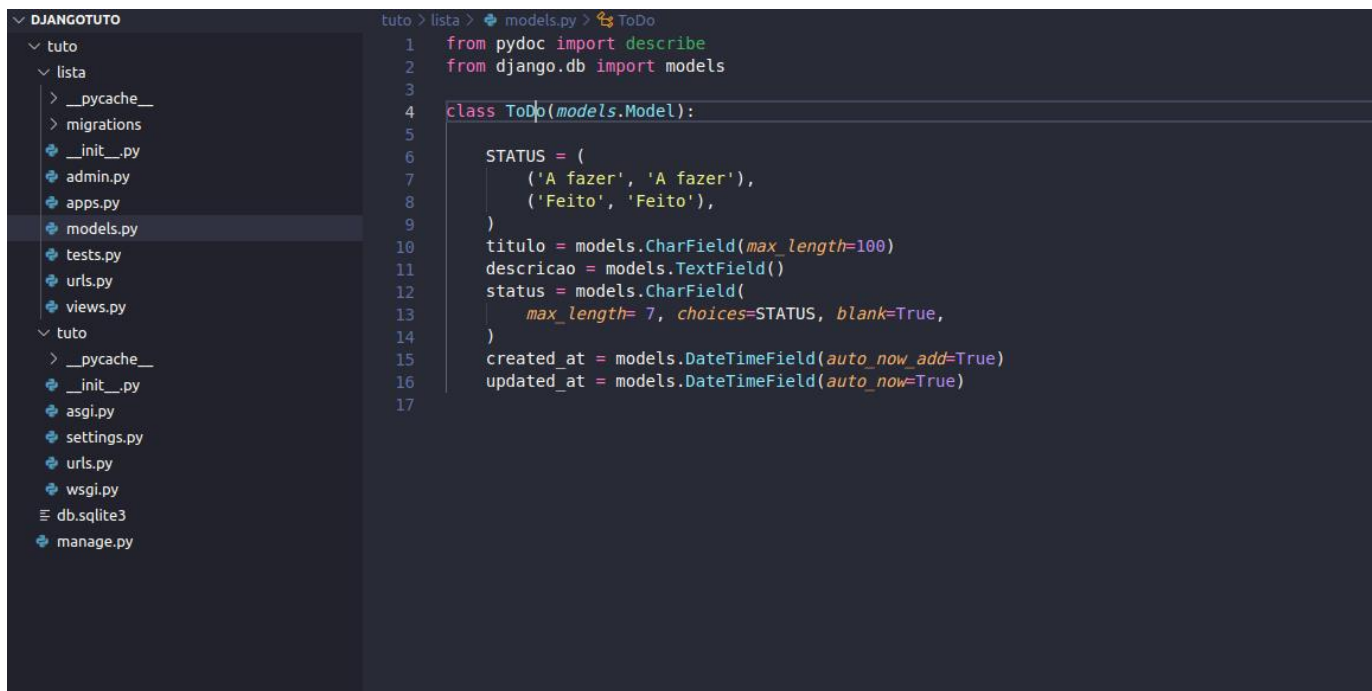
```
(env) guilherme.souza@pr7008251:~/djangotuto/tuto$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying 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 sessions.0001_initial... OK
(env) guilherme.souza@pr7008251:~/djangotuto/tuto$
```

O comando migrate, passa por todos os “INSTALLED_APPS” e cria todas as tabelas de banco de dados necessárias.

Vamos criar uma aplicação básica de lista de afazeres. Para começar vamos definir o modelo, o layout do banco de dados.

Por enquanto teremos apenas uma classe de modelo chamada “ToDo” onde irá receber o título da lista, descrição, status, e, como boa prática, dois status de tempo para dizer quando foi criado e atualizado

Edite o arquivo **lista/models.py** de acordo com o seu modelo, o que usaremos aqui será este:

The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'DJANGOTUTO' with a subdirectory 'tuto'. Inside 'tuto', there is a subdirectory 'lista' containing files like '__pycache__', 'migrations', '__init__.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is selected. The code editor shows the following Python code:

```
1 from pydoc import describe
2 from django.db import models
3
4 class ToDo(models.Model):
5
6     STATUS = (
7         ('A fazer', 'A fazer'),
8         ('Feito', 'Feito'),
9     )
10    titulo = models.CharField(max_length=100)
11    descricao = models.TextField()
12    status = models.CharField(
13        max_length= 7, choices=STATUS, blank=True,
14    )
15    created_at = models.DateTimeField(auto_now_add=True)
16    updated_at = models.DateTimeField(auto_now=True)
17
```

O modelo é representado por uma classe com base no `django.db.models.Model`.

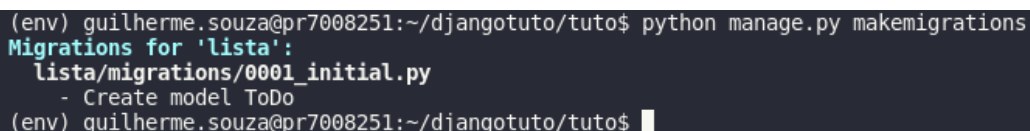
Onde podemos criar atributos e classe que representam um campo no banco de dados no modelo.

Cada Field tem um tipo, como o `CharField` para campo de caracteres e o `DateTimeField` para campo de data/hora, que por sua vez recebem parâmetros quando necessário, como o “`max_length`” que além de dizer o máximo de caracteres também serve para validação.

Agora vamos carregar este modelo para o banco de dados. Outro comando:

```
[ $ python manage.py makemigrations ]
```

Que deverá retornar algo similar a isso:

The image shows a terminal window with the following output:

```
(env) guilherme.souza@pr7008251:~/djangotuto/tuto$ python manage.py makemigrations
Migrations for 'lista':
  lista/migrations/0001_initial.py
    - Create model ToDo
(env) guilherme.souza@pr7008251:~/djangotuto/tuto$
```

O `makemigrations` basicamente diz ao Django que fez algumas alterações nos seus modelos, e que deve ser armazenado como migrate.

É possível ver todas as migrations que serão criadas em **lista/migrations** que, no momento, é o “0001_initial.py”, onde você pode alterar manualmente caso seja necessário.

Rode o migrate novamente para criar essas tabelas dos modelos:

```
[ $ python manage.py migrate ]
```

Que retornará:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, lista, sessions
Running migrations:
  Applying lista.0001_initial... OK
```

Ou seja, temos três pontos principais para criar um Modelo:

- Mude/Crie seu modelos (models.py)
- Rode **python manage.py makemigrations** para criar as migrações com as modificações
- Rode **python manage.py migrate** para aplicar as modificações no banco de dados.

O Django já cria um site de administração automaticamente quando iniciado, porém, precisamos de um usuário admin para acessar esse site. Vamos criá-lo.

Execute:

```
[ $ python manage.py createsuperuser ]
```

Insira o nome de usuário, pressione enter.

Em seguida o e-mail.

Por último, a senha duas vezes para confirmação.

Vamos verificar as mudanças feitas, inicie o servidor.

```
[ $ python manage.py runserver ]
```

Agora vamos para o <http://127.0.0.1:8000/admin/>.

Onde você deverá ver uma tela como esta:

A screenshot of the Django Admin login page. It features a dark blue header with the text "Administração do Django". Below the header is a white box containing two input fields: "Usuário:" and "Senha:". At the bottom of the box is a blue button labeled "Acessar".

Administração do Django

Usuário:

Senha:

Acessar

A tradução da página estará de acordo com a configuração feita no "LANGUAGE_CODE"

Após o login teremos uma tela como esta:

A screenshot of the Django Admin dashboard. The top header is dark blue with the text "Administração do Django" and a navigation bar with links: "BEM-VINDO(A), ADMIN.", "VER O SITE", "ALTERAR SENHA", and "ENCERRAR SESSÃO". Below the header, the main content area is titled "Administração do Site". It contains a table with two rows: "Grupos" and "Usuários", each with "+ Adicionar" and "Modificar" links. To the right, there are two sections: "Ações recentes" and "Minhas Ações", both showing "Nenhum disponível".

Administração do Django

BEM-VINDO(A), ADMIN. [VER O SITE](#) / [ALTERAR SENHA](#) / [ENCERRAR SESSÃO](#)

Administração do Site

AUTENTICAÇÃO E AUTORIZAÇÃO	
Grupos	+ Adicionar Modificar
Usuários	+ Adicionar Modificar

Ações recentes

Minhas Ações

Nenhum disponível

O nosso modelo ainda não está disponível aqui. Vamos registrar ele nesta página de administração.

Edite o arquivo **lista/admin.py** adicionando as seguintes linhas:

```
[  
from django.contrib import admin  
from lista.models import ToDo  
  
admin.site.register(ToDo)
```

]

Também poderíamos usar “from .model import ToDo” para importar nosso model, porém eu recomendo que utilize o caminho inteiro até que se aprenda exatamente o caminho do qual está sendo importado.

E agora teremos isto:

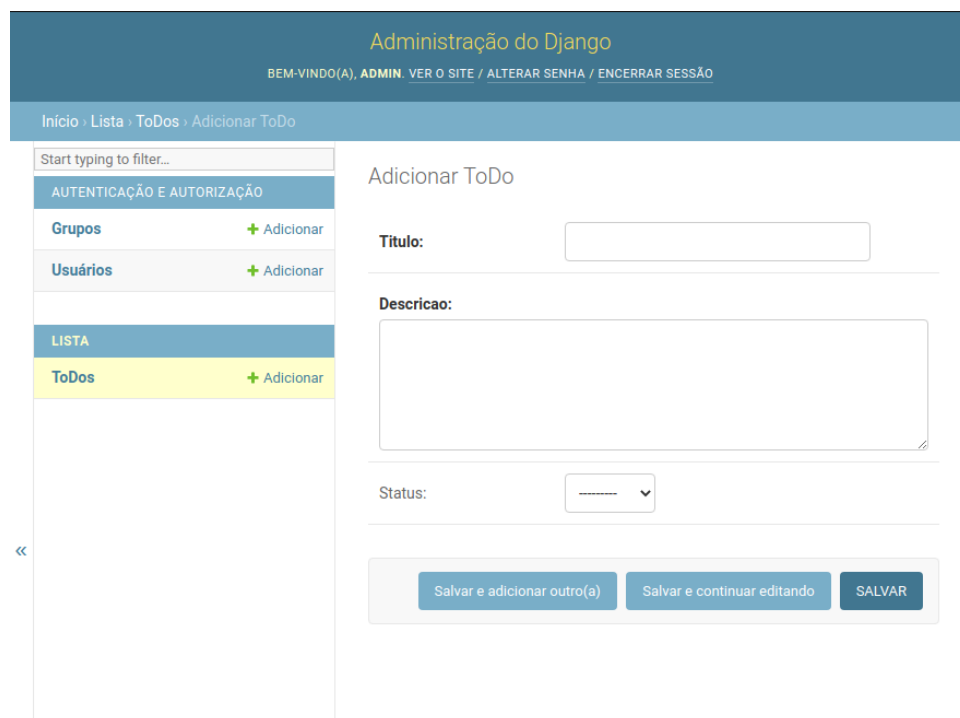


Note que o nome ficou um pouco esquisito, vamos arrumar isso em **models.py**.


Basta adicionar uma class Meta e passar a função verbose_name com o nome que desejamos, de modo que fique assim:

```
18
19
20 class Meta:
21     verbose_name = 'ToDo'
```

Vamos adicionar uma tarefa como teste





Note que aqui temos todos os campos no qual adicionamos no nosso modelo, menos o **created_at** e o **updated_at**, pois neles passamos parâmetros que são adicionados de forma automática.

 O ToDo "ToDo object (1)" foi adicionado com sucesso.

Selecione ToDo para modificar

ADICIONAR TODO +

Ação:   0 de 1 selecionados

☐ TODO

☐ ToDo object (1)

1 ToDo

ToDo object(1)? Vamos arrumar isto, novamente no **models.py**.

Adicione as seguintes linhas:

```
[  
    def __str__(self):  
        return self.titulo  
]
```

☐ TODO

☐ Lavar o carro

1 ToDo

Agora sim, e clicando no título ainda podemos alterar os campos já preenchidos da nossa lista.

Modificar ToDo

HISTÓRICO

Lavar o carro

Título:

Lavar o carro

Descrição:

Preciso lavar o carro antes de ir ao aniversário que ocorrerá no sábado.

Status:

A fazer ▾

Apagar

Salvar e adicionar outro(a)

Salvar e continuar editando

SALVAR

Onde temos também um histórico de todas as modificações feitas, por qual usuário, e horário das mudanças.

Histórico de modificações: Lavar o carro

DATA/HORA	USUÁRIO	AÇÃO
22 de Agosto de 2022 às 14:19	admin	Adicionado.

Um CRUD inteirinho sem precisarmos de muito esforço. Porém esta é apenas o começo do nosso padrão model-template-view.

Agora vamos criar uma view que faça algo, vamos modificar a que já temos, em **lista/views.py**, para que ela apresente nossos modelos já criados, ou melhor, nossa lista de afazeres.

Usaremos o `ToDo.objects.all()` para pegar todos os dados dentro do `ToDo`.

Esta view irá retornar um **render**, que precisa do **request** (o que foi pedido), o **template** (uma página em HTML para ser renderizada) e **context** (onde estão nossos

objetos que serão enviados para o template, neste caso o “listas”).

Ao final se parecerá com algo assim:

```
tuto > lista > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  from lista.models import ToDo
5
6  def index(request):
7      listas = ToDo.objects.all()
8      return render(request, 'index.html', {'listas' : listas} )
9
10
```

O código carrega o template chamado “index.html” e passa o context para ele.

Note que o template não foi criado ainda, vamos fazer isto agora!

Primeiro crie uma pasta chamada “templates” no diretório lista, o Django irá procurar templates lá.

Agora, dentro de templates, crie o arquivo index.html (em resumo o caminho se parecerá com isso: lista/templates/index.html)

Escreva o seguinte código no template:

```
[
<h1>Olá mundo, essas são as nossas tarefas </h1>
{% for lista in listas %}
<li>{{ lista }}</li>
{% endfor %}
]
```

Acesse <http://127.0.0.1:8000/index/> e veja como ficou o template.

Vamos adicionar uma forma de ver os detalhes da tarefa:

Primeiro vamos criar um caminho para ver os detalhes, edite o arquivo **lista/urls.py** para que fique assim

```
tuto > lista > urls.py > ...
1  from django.urls import path
2
3  from lista.views import index, indexdetails
4
5  urlpatterns = [
6      path('index/', index, name='index'),
7      path('details/<int:id>', indexdetails, name='details')
8  ]
```

Sempre que adicionamos novos modelos eles já possuem, por padrão, um ID, e aqui está sendo requisitado para acessar cada item pelo seu ID.

Agora criaremos a view em **lista/views.py** com uma função nova, de forma que fique assim:

```
tuto > lista > views.py > indexdetails
1  from django.shortcuts import get_object_or_404, render
2  from django.http import HttpResponse
3
4  from lista.models import ToDo
5
6  def index(request):
7      listas = ToDo.objects.all()
8      return render(request, 'index.html', {'listas' : listas} )
9
10 def indexdetails(request, id):
11     lista = get_object_or_404(ToDo, id=id)
12     return render(request, 'detail.html', {'lista': lista })
```

Agora usaremos o “get_object_or_404”, que é mais um atalho do Django para pegar objetos e caso ele não exista, retornará um erro 404. Novamente ele renderiza um template HTML e passa o context para ele.

Em **lista/templates** crie um arquivo chamado **detail.html** e coloque as seguintes linhas:

```
[
<h1>{{lista.titulo}}</h1>

<ul>

    <li>{{lista.descricao}}</li>

    <li>{{lista.status}}</li>

    <li>{{lista.created_at}}</li>

</ul>
```


]

(entenda o `{{}}`, `{% %}` [Aqui](#))

Tudo pronto, agora falta uma forma de chegarmos a este caminho, vamos adicionar um link no `index.html`, de modo que fique parecido com isso:

```
tuto > lista > templates > <> index.html > ...
1  <h1>Olá mundo, essas são as nossas tarefas </h1>
2  {% for lista in listas %}
3  <li>{{ lista }}
4      <a href="/details/{{lista.id}}">ver</a>
5  </li>
6  {% endfor %}
```

E pronto, agora podemos ver alguns detalhes dos nossos afazeres!

Agora criamos nosso modelo (**ToDo**) que está sendo apresentado no Template (`index.html`), com os detalhes em `detail.html`, que são conectados por meio das nossas views e acessados por caminhos no arquivo `urls.py`

Mas nisso contemplamos apenas o R do CRUD. Vamos continuar!

No momento conseguimos adicionar itens a nossa lista apenas acessando o ambiente de administração, vamos fazer isso de forma “menos interna”.

Para isso o django já dá um grande suporte, no diretório `lista` crie o arquivo `forms.py`

Dentro do arquivo `forms.py` adicione as seguintes linhas:

[

```
from django.forms import ModelForm
```

```
from lista.models import ToDo
```

```
class ToDoForm(ModelForm):
```

```
    class Meta:
```

```
    model = ToDo
    exclude = ['created_at', 'updated_at']
]
```

Agora vamos fazer uma view, em **lista/views.py**, para ligarmos esse form.
Algo assim:

```
[
def add(request):
    form = ToDoForm(request.POST or None)
    if form.is_valid():
        form.save()
        return redirect('index')
    return render(request, 'add.html', {'form': form})
]
```

(Não esqueça das importações:

```
from lista.forms import ToDoForm
from django.shortcuts import get_object_or_404, redirect, render )
```

O “request.POST” é a requisição HTTP para pegar os dados inseridos dentro do form

Dentro do diretório **lista/templates** adicione o arquivo add.html, onde o nosso form será renderizado, com os seguintes comandos:

```
[
<h1>Novo afazer </h1>
<form method="post">
    {% csrf_token %}
    {{form}}
    <button type="submit">Adicionar</button>
]
```

</form>

]

O “{% csrf_token %}” é obrigatório sempre que passamos o método POST no django, é um requisito de proteção contra entradas mal-intencionadas

Agora precisamos de um caminho para acessar esse formulário, vamos adicionar um no **lista/urls.py**, faça de modo que fique assim:

```
tuto > lista > urls.py > ...
1  from django.urls import path
2
3  from lista.views import index, indexdetails, add
4
5  urlpatterns = [
6      path('index/', index, name='index'),
7      path('details/<int:id>', indexdetails, name='details'),
8      path('add/', add, name='add')
9  ]
```

(A view criada, add, precisa ser importada, como foi feito na imagem acima)

Pronto, só falta uma forma de chegarmos nessa url, vamos colocar um button no index.html para acessarmos nosso template, algo parecido com isto:

```
tuto > lista > templates > index.html > ...
1  <h1>Olá mundo, essas são as nossas tarefas </h1>
2  <a href="/add"><button>Adicionar</button></a>
3  {% for lista in listas %}
4  <li>{{ lista }}
5      <a href="/details/{{lista.id}}">ver</a>
6  </li>
7  {% endfor %}
```

E voilà, o C do CRUD, não de forma muito bela, nosso foco aqui é apenas as funcionalidades e não a beleza.

Ok, precisamos atualizar, afinal de contas, os afazeres devem ser concluídos.

Vamos fazer um view para mudar isto.

Adicione as seguintes linhas em **lista/views.py**.

```
[
def update(request, id):
    lista = ToDo.objects.get(pk=id)
    form = ToDoForm(request.POST or None, instance=lista)
    if form.is_valid():
        form.save()
        return redirect('index')
    return render(request, 'add.html', {'lista': lista, 'form': form})
]
```

Em **lista/urls.py** importe a view e adicione o caminho de modo que fique assim:

```
tuto > lista >  urls.py > ...
1  from django.urls import path
2
3  from lista.views import index, indexdetails, add, update
4
5  urlpatterns = [
6      path('index/', index, name='index'),
7      path('details/<int:id>', indexdetails, name='details'),
8      path('add/', add, name='add'),
9      path('update/<int:id>', update, name='update'),
10 ]
```

Novamente pedindo o ID para acessar o item em específico.

E novamente, vamos adicionar uma forma de acessar cada item no index.html.

```
[
<h1>Olá mundo, essas são as nossas tarefas </h1>
<a href="/add"><button>Adicionar</button></a>
{% for lista in listas %}
<ul>
    <li>{{ lista }}
        <a href="/details/{{lista.id}}"><button> ver </button></a>
    
```

```
        <a href="/update/{{lista.id}}"><button> Editar </button></a>
    </li>
</ul>
{% endfor %}
]
```

(Apenas coloquei os links em buttons para melhor visualização)

Prontinho, mais uma etapa concluída, agora fizemos o U do CRUD.

Vamos para o último passo, uma forma de deletar itens.

Primeiro, edite o arquivo **lista/view.py**.

```
[
def delete(request, id):
    lista = ToDo.objects.get(pk = id)
    lista.delete()
    return redirect('index')
]
```

Segundo, agora é a vez do arquivo **lista/urls.py**.

```
[
    path('delete/<int:id>', delete, name='delete'),
]
```

É para ficar assim:

```
tuto > lista >  urls.py > ...
1  from django.urls import path
2
3  from lista.views import delete, index, indexdetails, add, update
4
5  urlpatterns = [
6      path('index/', index, name='index'),
7      path('details/<int:id>', indexdetails, name='details'),
8      path('add/', add, name='add'),
9      path('update/<int:id>', update, name='update'),
10     path('delete/<int:id>', delete, name='delete'),
11 ]
```

Por último, uma forma de acessar no **templates/index.html**

```
[
<h1>Olá mundo, essas são as nossas tarefas </h1>
<a href="/add"><button>Adicionar</button></a>
{% for lista in listas %}
<ul>
    <li>{{ lista }}
        <a href="/details/{{lista.id}}"><button> ver </button></a>

        <a href="/update/{{lista.id}}"><button> Editar </button></a>

        <a href="/delete/{{lista.id}}"onclick="return confirm('Tem certeza de que deseja
deletar este item?')"><button> deletar </button></a>

    </li>
</ul>
{% endfor %}
]
```

(Aqui eu coloquei uma pequena verificação antes de deletar o item para que ela não faça “puff” e suma sem mais nem menos)

E assim passamos por todo o CRUD com o django, simples e rápido, não é?

Toda a aplicação feita está disponível neste GitHub [Aqui](#)