



# A importância dos testes automatizados do Django

Raíssa Azevedo

## A IMPORTÂNCIA DE REALIZAR TESTES AUTOMATIZADOS

[Raíssa Azevedo](#)

Após mais um dia desvendando as funcionalidades do *Django Framework* me deparei com o arquivo **tests.py**. Estudando e me aprofundando a respeito cheguei aos seguintes passos de aprendizado:

Eu tinha um projeto de back-end realizado e publicado chamado [Fusion](#). Já tinha dado o projeto como finalizado. Mas, em todo o processo de criação, todos os testes que fiz buscando erros no código ou de lógica de programação foram manuais. Ou seja, eu precisava simular um servidor, adicionar os dados, e verificar o que me impedia de publicar.

Então, segue passos de realização de testes automatizados no modo de produção do Django Framework:

## REALIZANDO TESTES DE USABILIDADE NO DJANGO

No Django cada aplicação criada irá conter um arquivo “tests.py”

O recomendado é apagar esse arquivo e criar um diretório dentro da pasta de aplicação denominado “Tests”.

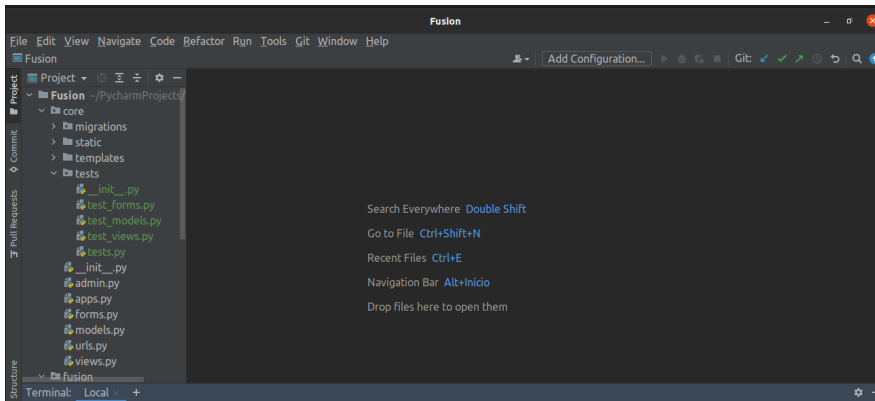
Lá irá colocar todos os arquivos py que irá realizar os testes de usabilidade.

Nesse caso, a aplicação do meu projeto se chama core. E criei o diretório *tests* dentro dessa aplicação.

**Obs:** Todo arquivo de teste irá começar com **test\_**

No arquivo **\_\_init\_\_.py**:

Foi realizado um teste do teste, que não está relacionado ao projeto, é somente para verificar se não há falha no Framework (99% de certeza que não haverá):



**Instale as seguintes bibliotecas:**

- **pip3 install model\_mommy coverage**

```
from django.test import TestCase

# Teste de números

def add_num(num):
    return num + 1

class SimplesTestCase(TestCase):

    # Roda toda vez
    def setUp(self):
        self.numero = 41

    # Testa a unidade de código
    def test_add_num(self):
        valor = add_num(self.numero)
        self.assertTrue(valor == 42 )
```

Após escrever o seguinte código:

```
coverage run manage.py test
```

O teste unitário só irá retornar um valor **True** que indica que o teste automatizado está funcionando.

## INICIANDO OS TESTES REAIS DO PROJETO

Na raiz do Projeto crie um arquivo **.coveragerc**

Esse arquivo irá indicar onde irá ocorrer os testes dentro do sistema criado

### DENTRO DO ARQUIVO .COVERAGERC

```
[run]
source = .
```

Este “.” no código irá indicar que os testes irão ocorrer somente nas partes indicadas dentro do Projeto.

## IGNORANDO ARQUIVOS QUE NÃO PRECISAM SER TESTADOS

O Framework Django por já testa boa parte do conteúdo presente no código. Sua responsabilidade é testar somente as criações novas do back-end como: Models (banco de dados), Formulários e Views.

Então, indique os arquivos a serem ignorados dessa forma:

```
[run]
source = .

omit =
    */__init__.py
    */settings.py
    */manage.py
    */wsgi.py
    */apps.py
    */urls.py
    */admin.py
    */migrations/*
    */tests/*
    */asgi.py
```

Todos esses arquivos e diretórios já são testados pelo Django, portanto não há necessidade de fazer um novo teste.

## EXEMPLIFICANDO A EXECUÇÃO DE UM TESTE DE USABILIDADE

Testando o models.py:

Crie um arquivo no diretório de testes chamado **test\_models.py**

Antes de tudo verifique tudo que precisa ser testado no seu models, executando:

```
coverage run manage.py test
```

Seguido de:

```
coverage html
```

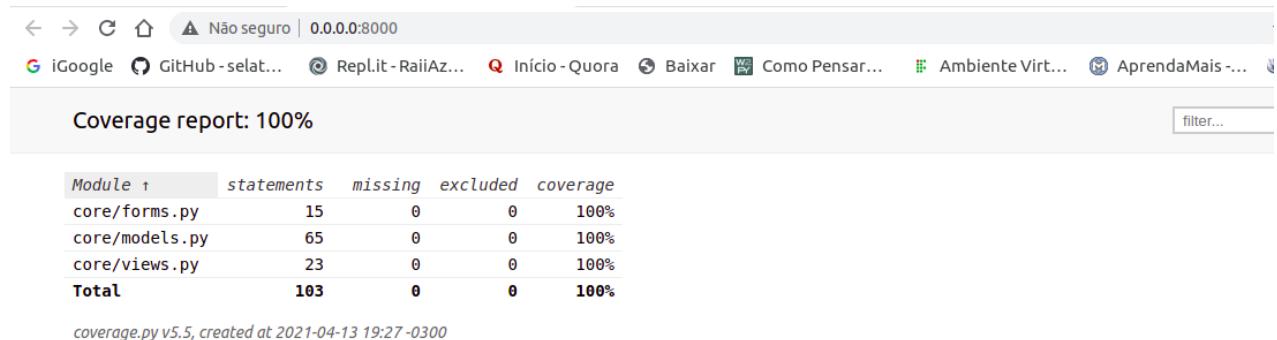
Isso irá gerar um diretório chamado **Htmlcov**

Através do terminal abra esse diretório e execute o seguinte código:

```
python3 -m http.server
```

Isso irá gerar um servidor de relatório de testes que irá indicar tudo que precisa ser testado no seu models.

O servidor irá parecer mais ou menos como esse:



The screenshot shows a web browser window with the address bar at 0.0.0.0:8000. The page title is "Coverage report: 100%". Below the title is a table with the following data:

Module	statements	missing	excluded	coverage
core/forms.py	15	0	0	100%
core/models.py	65	0	0	100%
core/views.py	23	0	0	100%
<b>Total</b>	<b>103</b>	<b>0</b>	<b>0</b>	<b>100%</b>

Below the table, it says "coverage.py v5.5, created at 2021-04-13 19:27 -0300".

E irá indicar o que precisa ser testado, o que falta ser testado e a porcentagem de cobertura de testes. Tudo descrito em “Module” é um link que você pode abrir para encontrar mais dados específicos, como aqui no models.py:



The screenshot shows a web browser window displaying a detailed coverage report for **core/models.py**. The report shows 100% coverage for 65 statements, with 65 run, 0 missing, and 0 excluded. Below the summary, the code for **models.py** is displayed with line numbers 1 through 28. The code is as follows:

```
1 import uuid
2 from django.db import models
3 from stdimage.models import StdImageField
4
5
6 def get_file_path(instance, filename):
7     ext = filename.split('.')[-1]
8     filename = f'{uuid.uuid4()}.{ext}'
9     return filename
10
11
12 class Base(models.Model):
13     criados = models.DateTimeField('Criação', auto_now_add=True)
14     modificado = models.DateTimeField('Atualização', auto_now=True)
15     ativo = models.BooleanField('Ativo?', default=True)
16
17     class Meta:
18         abstract = True
19
20
21 class Servico(Base):
22     ICONE_CHOICES = (
23         ('lni-cog', 'Engrenagem'),
24         ('lni-stats-up', 'Gráfico'),
25         ('lni-users', 'Usuários'),
26         ('lni-layers', 'Design'),
27         ('lni-mobile', 'Mobile'),
28         ('lni-rocket', 'Foguete'),
```

Se houver algo que precisa ser testado, irá estar com coloração vermelha. E o que está verde, é tudo que está testado e sem presença de falhas. Esse relatório serve para indicar ao cliente a que pé anda seu projeto também.

## COMO SABER SE HÁ ERROS NA EXECUÇÃO DE UM TESTE DE USABILIDADE

Ao executar um comando de testes, e se não houver falhas, o sistema irá te retornar uma resposta como esta:

```
raissa@raissa-Inspiron-15-3567:~/PycharmProjects/Fusion$ coverage run manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 11 tests in 0.275s

OK
Destroying test database for alias 'default'...
raissa@raissa-Inspiron-15-3567:~/PycharmProjects/Fusion$
```

Mas se houver erros, a resposta do sistema será assim:

```
raissa@raissa-Inspiron-15-3567:~/PycharmProjects/Fusion$ coverage run manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....E.....
=====
ERROR: test_str (core.tests.test_models.CargoTestCase)
-----
Traceback (most recent call last):
  File "/home/raissa/PycharmProjects/Fusion/core/tests/test_models.py", line 30, in test_str
    self.assertEqual(str(self.cargo), self.cargo.nome)
AttributeError: 'Cargo' object has no attribute 'nome'
-----
Ran 11 tests in 0.127s

FAILED (errors=1)
Destroying test database for alias 'default'...
raissa@raissa-Inspiron-15-3567:~/PycharmProjects/Fusion$
```

O sistema irá indicar a quantidade de falhas encontradas, qual arquivo está a falha e a linha exata onde está o erro.

**Obs:** Toda vez que for executar um novo comando de teste, lembre-se primeiro de apagar o diretório **htmlcov/** com o seguinte código:

```
rm -rf htmlcov
```

Para não haver conflito de dados de testes.

## PORQUE EXECUTAR TESTES DE USABILIDADE É IMPORTANTE ?

Como citado anteriormente no texto, ao criar um projeto, você pode optar por testes manuais, mas só irá descobrir onde estão as falhas de programação no momento de inserir os dados e executar. O que torna o processo mais demorado. Uma vez o teste é executado, e indicando a presença de zero falhas, você poderá seguir para a próxima etapa do processo com a consciência tranquila de não haver erro no seu código. Além disso, um projeto com o comprovante de cobertura de testes 100% é um ponto positivo na carreira de qualquer programador.