Shell

```
nte> python .\manage.py shell
```

```
>>> from services.models import Service
>>> servicios = Service.objects.all()
>>> servicios[0]
<Service: asd>
>>> servicios[0].created
datetime.datetime(2022, 10, 24, 19, 7, 7, 433931, tzinfo=datetime.timezone.utc)
>>>
```

Tests

Entre más crece una aplicación más difícil se vuelve probar correctamente todos los escenarios manualmente.

Las pruebas automatizadas ofrecen:

- Ejecución rápida
- Nivel de detalle más preciso
- Probar exactamente la misma funcionalidad
- Actuan como el “primer usuario”

Tipos de pruebas

- Unitarias
  - Verifica comportamiento functional de un un componente individual
- De regresión
  - Pruebas que reproducen errores históricos.
  - Inicialmente se ejecuta para verificar que el error ha sido corregido
  - Se ejecutan para el aseguramiento que dichos errores no fueron reintroducidos con cambios posteriores
- De integración
  - Verifican ejecución de grupos de componentes.
- Otras
  - Pruebas de seguridad (caja negra, blanca), automatizadas, de humo, funcionales, de rendimiento, etc.

Pruebas unitaras

Una de las clases base es django.test.TestCase, esta clase de prueba crea una base de datos limpia antes de la ejecución de las pruebas. También posee una prueba Client que se puede utilizer para simular la interacción de un usuario con el Código en el nivel de vista.

Ejecución de las pruebas

Django encuentra todos los nombres “test*.py”

Buena práctica: Cada módulo deberá contener archivos separados para modelos, vistas, formularios y cualquier otro tipo de código que se requiera probar.

Ejemplo de estructura

- Product

- /tests/
  - __init__.py #indica que el directorio es un paquete
  - test_models.py
  - test_forms.py
  - test_views.py

Métodos de TestCase

TestCase define 2 métodos que se pueden utilizer para la configuración de la prueba

- setUpTestData()
  - Se llama al comienzo de la ejecución de la prueba
  - Generalmente se usa para crear objetos que no se modificarán ni cambiarán en nunguno de los métodos de prueba
- setup()
  - Se llama antes de cada función de prueba para configurar cualquier objeto que pueda ser modificado por la prueba

Para realizar las verificaciones se utilizan los “assertTrue, assertFalse, assertEqual, assertRedirects, assertTemplateUsed) etc.

En el archivo tests.py de la aplicación services

```
1   from django.test import TestCase
2   from services.models import Order
3
    Francisco Vazquez, 5 hours ago | 1 author (Francisco Vazquez)
4   class OrderModelTest(TestCase):
5       def test_not_empty(self):
6           order = Order(
7               nombre = 'Francisco Vázquez',
8               direccion = '4 Pte 3421',
9               total = 345.23,
10              correo = 'franvazgom@gmail.com'
11          )
12          Order.save(order)
13          orders = Order.objects.all()
14          self.assertIs(len(orders) > 0, True)
15          self.assertEquals(orders[0].nombre, 'Francisco Vázquez')
```

Observe que se crea una base de datos “temporal” para las pruebas.

Para correr una prueba

```
> python .\manage.py test services
```

Para mayor detalle --verbosity 2

Prueba para un modelo

code > webrestaurante > services > tests > test_models.py > ...

```
You, 1 second ago | 2 authors (Francisco Vazquez and one other)
from django.test import TestCase
from services.models import Service

You, 1 second ago | 2 authors (Francisco Vazquez and one other)
class OrderModelTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        Service.objects.create(
            title = 'Servicio 1',
            subtitle = 'Subtitulo de pruebas',
            content = 'Contenido del servicio 1',
            image = 'ruta de la imagen'
        )

    def test_title_name_label(self):
        service = Service.objects.get(id=1)
        field_label = service._meta.get_field('title').verbose_name
        self.assertEquals(field_label, 'Título')

    def test_title_max_length(self):
        service = Service.objects.get(id=1)
        max_length = service._meta.get_field('title').max_length
        self.assertEquals(max_length, 100)
```

Pruebas para un formulario

code > webrestaurante > services > tests > test_forms.py > ...

```
You, 1 second ago | 2 authors (Francisco Vazquez and one other)
from django.test import TestCase
from services.forms import OrderForm

Francisco Vazquez, 5 hours ago | 1 author (Francisco Vazquez)
class OrderModelTest(TestCase):
    def test_title_label(self):
        form = OrderForm()
        self.assertTrue(form.fields['nombre'].label == 'Nombre')

    def test_incomplete_fields(self):
        data = {
            'direccion':'4 Pte 3421',
            'total':345.23,
            'correo':'franvazgom@gmail.com'}
        form = OrderForm(data)
        self.assertFalse(form.is_valid())
```

Verifique las pruebas.

Test_views

Realizar el siguiente cambio en la vista

```
ode > webrestaurante > services > views.py > ServiceCreateOrder
    You, 1 second ago | 2 authors (You and one other)
1   from django.shortcuts import render, HttpResponseRedirect
2   from django.views.generic.edit import CreateView
3   from django.views.generic import TemplateView, ListView
4   from services.models import Service, Order
5   from .forms import ServiceForm, OrderForm
6   from django.urls import reverse_lazy
7   from django.contrib.admin.views.decorators import staff_mem
8
9   # def service_list(request):
10  #     services = Service.objects.all()
11  #     return render(request, 'services/service_list.html',
12
    You, 1 second ago | 1 author (You)
13  class ServiceListView(ListView):
14      model = Service
15      paginate_by = 2
```

En el html cambiar a:

```
4   {% include 'services/menu_services.h
5   {% for service in object_list  %}
6     <section class="page-section">
```

En los urls:

```
3   from .views import ServiceListView
4
5   services_urlspatterns = ([
6       # path('', views.service_list, name='service_list'),
7       path('', ServiceListView.as_view(), name='service_list'),
8       path('create/', views.create, name='create'),
```

```
code > webrestaurante > services > tests > test_views.py > ServiceListViewTest
    Francisco Vazquez, 5 hours ago | 1 author (Francisco Vazquez)
1   from django.test import TestCase
2   from services.models import Service
3   from django.urls import reverse
4
    Francisco Vazquez, 5 hours ago | 1 author (Francisco Vazquez)
5   class ServiceListViewTest(TestCase):
6       @classmethod
7       def setUpTestData(cls):
8           n_services = 10
9           for id in range(n_services):
10              Service.objects.create(title= 'Servicio ' + str(id),
11              subtitle = 'Subtitulo del servicio ' + str(id),
12              content = 'Contenido del servicio ' + str(id),
13              image = 'ruta de imagen del servicio ' + str(id))
14
15      def test_view_url_desired_location(self):
16          resp = self.client.get('/services/')
17          self.assertEquals(resp.status_code, 200)
```

```python
    def test_correct_template(self):
        resp = self.client.get('/services/')
        self.assertEquals(resp.status_code, 200)
        self.assertTemplateUsed(resp, 'services/service_list.html')

    def test_pagination_is_two(self):
        resp = self.client.get(reverse('services:service_list'))
        self.assertEquals(resp.status_code, 200)
        self.assertTrue('is_paginated' in resp.context)
        self.assertTrue(resp.context['is_paginated'] == True)
        self.assertTrue(len(resp.context['service_list']) == 2)

    def test_pagination_remain(self):
        resp = self.client.get(reverse('services:service_list')+'?page=5')
        self.assertEquals(resp.status_code, 200)

    def test_pagination_not_found(self):
        resp = self.client.get(reverse('services:service_list')+'?page=6')
        self.assertEquals(resp.status_code, 404)
```

Django provee un cliente para simular la interacción con las vistas.

Desde el Shell, verificar lo siguiente

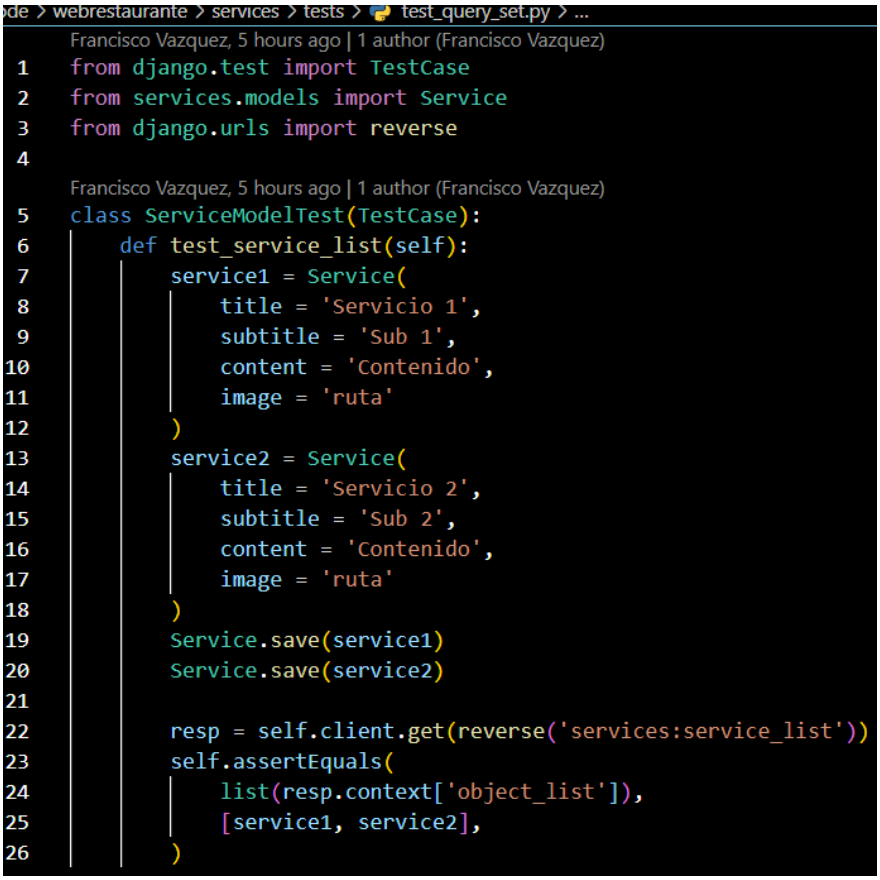
```
====================================================================
(venv) PS D:\devenv\MisProyectos\Git\clases\Anahuac\DAW0822\Practicas\solutions\django\webrestaurante> python .\manage.py shell
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.test.utils import setup_test_environment
>>> setup_test_environment()
>>> from django.test import Client
>>> client = Client()
>>> response = client.get('/')
>>> response.status_code
200
>>> from django.urls import reverse
>>> response = client.get(reverse('services:service_list'))
>>> response.status_code
200
>>> response.content
b'<!DOCTYPE html>\n<html lang="es">\n<!-- digital color -->\n\n<head>\n  <!-- Required meta tags -->\n  <meta charset="utf-8">\n
```

En el archivo service_list.html, se recibe el querySet ‘object_list’, mismo que puede ser obtenido desde el contexto.

```
>>> response.context['object_list']
<QuerySet [<Service: asd>, <Service: rrrrrrrrrrrrrrrrrrrrrr>]>
>>> response.context['object_list'][0].title
'asd'
>>> response.context['object_list'][0].content
'<p>dasds</p>'
```

Ejercicio

Crear la siguiente prueba para una vista tipo ListView



Verificar la prueba.