

Урок №9

Тестирование веб-приложений

на котором расскажут, зачем нужно тестирование, какие есть виды тестов, какие инструменты для тестирования программ и Django-проектов существуют.

16 ноября 2020 года

Антон Кухтичев

Содержание занятия

1. Цели и виды тестирования
2. unittest
3. mock
4. factory_boy
5. freezegun



Тестирование

Цели и виды тестирования

Цели тестирования



- Проверка правильности реализации;
- Проверка обработки внештатных ситуаций и граничных условий;
- Минимизация последствий.

Виды тестирования



- **Unit-тестирование**

Проверяют функциональное поведение для отдельных компонентов, часто классов и функций.

- **Интеграционное тестирование**

Проверка совместной работы групп компонентов. Данные тесты отвечают за совместную работу между компонентами, не обращая внимания на внутренние процессы в компонентах.



- **Функциональное тестирование (несколько функций);**
- **Тестирование производительности (performance testing)**
 - Нагрузочное тестирование (load testing)
 - Стресс тестирование (stress testing)
- **Регрессионное тестирование (regression testing)**

Тесты которые воспроизводят исторические ошибки (баги). Каждый тест вначале запускается для проверки того, что баг был исправлен, а затем перезапускается для того, чтобы убедиться, что он не был внесен снова с появлением новых изменений в коде.

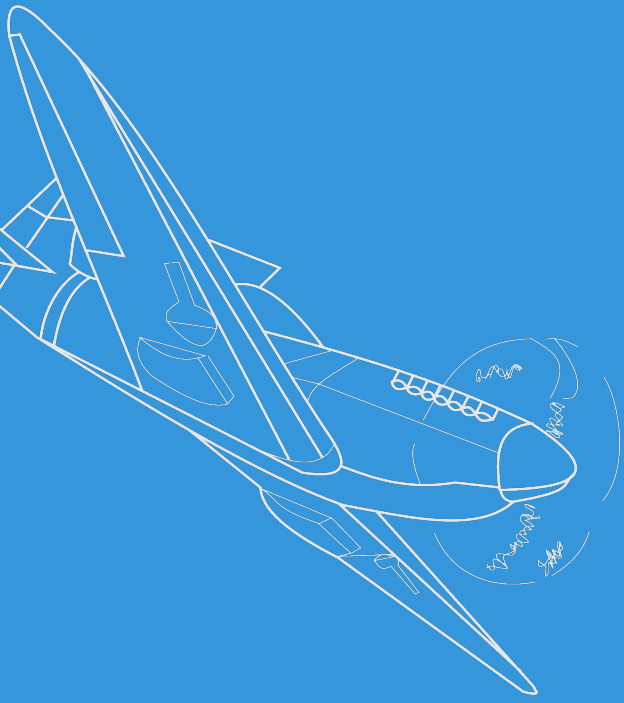
Test driven development



TDD – test driven development – техника разработки ПО, основывается на повторении коротких циклов разработки: пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода.



- Пишем тест, в котором проверяем, что функция `someCode()` возвращает нужные значения в разных ситуациях;
- Проверяем, что тесты упали (кода еще нет);
- Пишем код функции очень просто — так чтобы тесты прошли;
- Проверяем, что тесты прошли;
- На этом шаге можем задуматься о качестве кода. Можем спокойно рефакторить и изменять код как угодно;
- Повторяем все вышеуказанные шаги еще раз.



Selenium



Selenium WebDriver – это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

Установка

```
pip install selenium
```

Selenium



- Требует конкретного драйвера для конкретного браузера (Chrome, Firefox и т.д.);
- Автоматическое управление браузером;
- Поддержка Ajax;
- Автоматические скриншоты;

Виды тестирования. Selenium



```
import unittest

from selenium import webdriver

from selenium.webdriver.common.keys import Keys
```

```
class PythonOrgSearch(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()

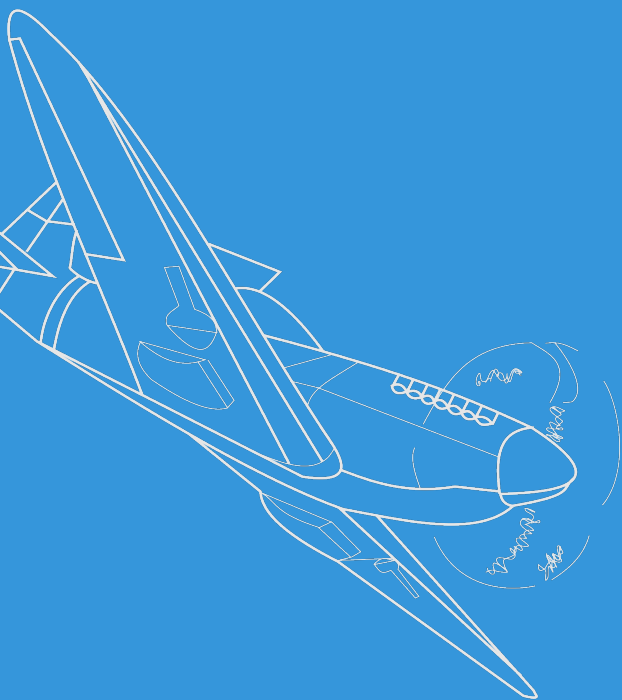
    def test_search_in_python_org(self):
        driver = self.driver
        driver.get("http://www.python.org")
        self.assertIn("Python", driver.title)
```

```
...
elem = driver.find_element_by_name("q")
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)

assert "No results found." not in
driver.page_source

def tearDown(self):
    self.driver.close()

if __name__ == "__main__":
    unittest.main()
```



Покрытие тестов

Степень покрытия тестами (test coverage)



`coverage` - библиотека для проверки покрытия тестами

```
pip install coverage
```

общий случай

```
coverage run tests.py
```

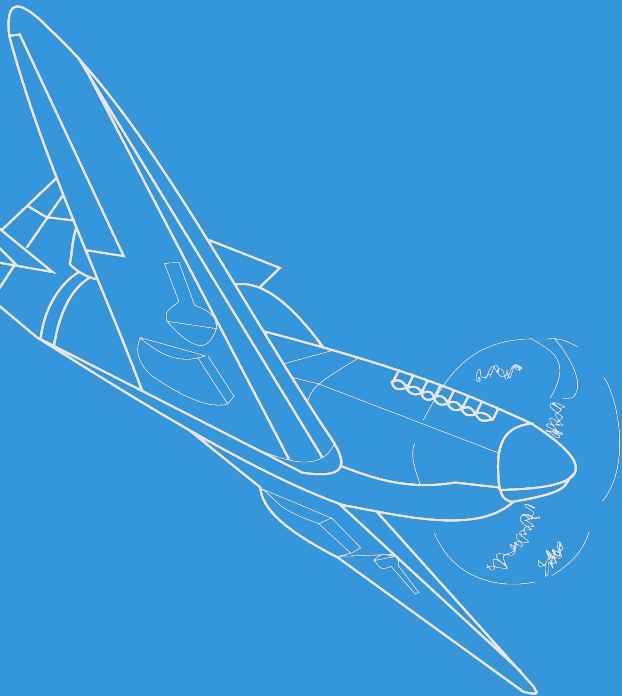
```
coverage report -m
```

```
coverage html
```

django

```
coverage run --source='.' manage.py test myapp
```

```
coverage report
```



Инструменты для тестирования в Python

Инструменты для тестирования в Python



- doctest
- pytest
- hypothesis
- unittest



```
def multiply(a, b):  
    """  
    >>> multiply(4, 3)  
    12  
    >>> multiply('a', 3)  
    'aaa'  
    """  
    return a * b
```

Запуск

```
python -m doctest <file>
```



```
class TestClass(object):
```

```
    def test_one(self):
```

```
        x = 'this'
```

```
        assert 'h' in x
```

```
    def test_two(self):
```

```
        x = 'hello'
```

```
        assert hasattr(x, 'check')
```

Заныск

pytest <file>

hypothesis



```
from hypothesis import given

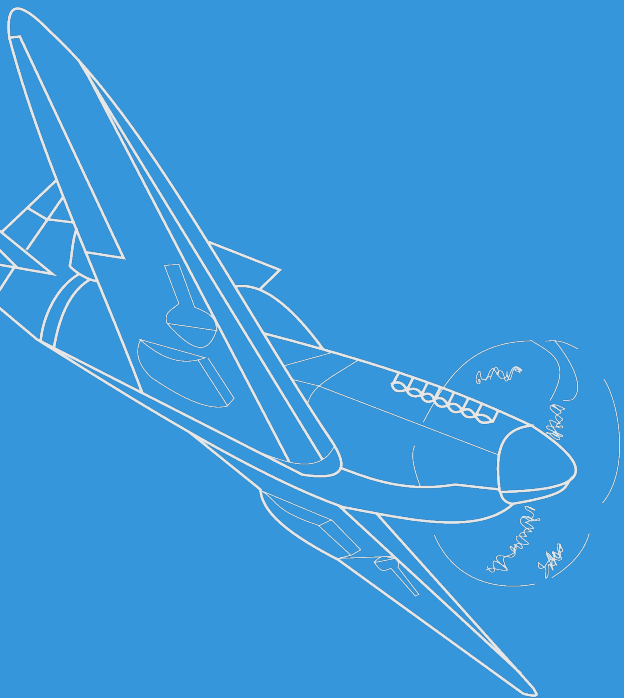
from hypothesis.strategies import text

@given(text())

def test_decode_inverts_encode(s):

    assert decode(encode(s)) == s
```

<https://hypothesis.readthedocs.io/en/latest/quickstart.html>



unittest



- `def setUp(self)` — Установки запускаются перед каждым тестом
- `def tearDown(self)` — Очистка после каждого метода
- `def test_<название теста>(self)` — Код теста;

unittest



```
import unittest

class FirstTestClass(unittest.TestCase):

    def test_upper(self):

        self.assertEqual('text'.upper(), 'TEXT')

if __name__ == '__main__':

    unittest.main()
```

Запуск

```
python -m unittest <file>
```

unittest. Расширенный набор проверок assert



- `assertEqual(a, b)`
- `assertNotEqual(a, b)`
- `assertTrue(x)`
- `assertFalse(x)`
- `assertIs(a, b)`
- `assertIsNot(a, b)`
- `assertIsNone(x)`
- `assertIn(a, b)`
- `assertIsInstance(a, b)`
- `assertLessEqual(a, b)`
- `assertListEqual(a, b)`
- `assertDictEqual(a, b)`
- `assertRaises(exc, fun, *args, **kwargs)`
- `assertJSONEqual(a, b)`

unittest и Django



```
from django.test import Client, TestCase
class SomeTest(TestCase):
    def setUp(self):
        self.client = Client()
    def test_something(self):
        response_1 = self.client.post('/url1/', {'id': 123})
        response_2 = self.client.get('/url2/')
```

Тестовый http-клиент. Имитирует работу браузера, может отправлять http-запросы

Запрашивает относительный путь, например /login/

Тестирование в Django. unittest



```
import json

from django.test import TestCase, Client
from video.models import Category

class VideoAPITest(TestCase):
    def setUp(self):
        self.client = Client()
        self.category =
        Category.objects.create(name='test
        category')

        self.response_map = {
            'name': 'test category',
            'video_count': 0
        }
```

```
    def test_category_values(self):
        response =
        self.client.get('/api/video_category/')
        content = json.loads(response.data)
        self.assertEqual(content['data']['category']['
        title'], 'test_category')

    def tearDown(self):
        print('I am done')
```



Mock — это просто объект, который создает пустой тест для определенной части программы.

- Высокая скорость
- Избежание нежелательных побочных эффектов во время тестирования

```
from unittest.mock import patch

class TestUserSubscription(TestCase):
    @patch('users.views.get_subscription_status')
    def test_subscription(self, get_subscription_status_mock):
        get_subscription_status_mock.return_value = True
        ...
```

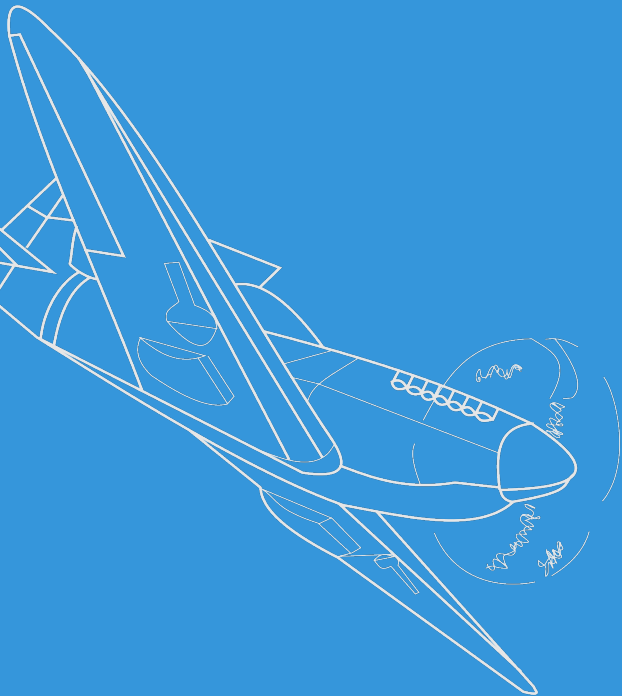


Атрибуты объекта Mock с информацией о вызовах

- `called` — вызывался ли объект вообще
- `call_count` — количество вызовов
- `call_args` — аргументы последнего вызова
- `call_args_list` — список всех аргументов
- `method_calls` — аргументы обращений к вложенным методам и атрибутам
- `mock_calls` — то же самое, но в целом и для самого объекта, и для вложенных

Пример

```
self.assertEqual(get_subscription_status_mock.call_count, 1)
```



Factory boy



Библиотека `factory_boy` служит для генерации тестовых объектов (в т.ч. связанных) по заданным параметрам.

Объявляем фабрику

```
class RandomUserFactory(factory.Factory):  
    class Meta:  
        model = models.User  
  
        first_name = factory.Faker('first_name', locale='ru_RU')  
        last_name = factory.Faker('last_name')  
        email = factory.Sequence(lambda n: f'person{n}@example.com')
```



- `paragraph(nb_sentences=3, variable_nb_sentences=True, ext_word_list=None)`
- `sentence(nb_words=6, variable_nb_words=True, ext_word_list=None)`
- `text(max_nb_chars=200, ext_word_list=None)`
- `word(ext_word_list=None)`
- `first_name()`, `last_name()`, `name()`

Подробнее [тут](#)



```
# Создаёт объект User, которые не сохранён.
```

```
user = RandomUserFactory.build()
```

```
# Возвращает сохранённый объект User.
```

```
user = RandomUserFactory.create()
```

```
# Returns a stub object (just a bunch of attributes)
```

```
obj = RandomUserFactory.stub()
```

```
users = RandomUserFactory.build_batch(10, first_name="Joe")
```

Запуск тестов. Общий случай



Найти и выполнить все тесты

```
python -m unittest discover
```

Тесты нескольких модулей

```
python -m unittest test_module1 test_module2
```

Тестирование одного кейса - набора тестов

```
python -m unittest tests.SomeTestCase
```

Тестирование одного метода

```
python -m unittest tests.SomeTestCase.test_some_method
```


Запуск тестов. Django



Тестируем все приложения.

```
./manage.py test
```

Тестируем конкретное приложение

```
./manage.py test app_name
```

Тестируем конкретный набор тестов в приложении

```
./manage.py test app_name.tests.TestCaseName
```

Тестируем конкретный тест в конкретном наборе в конкретном
приложении

```
./manage.py test app_name.tests.TestCaseName.test_function
```

Оптимизация. Тестовое окружение



Настроить verbose.

```
python -m unittest --verbose tests
```

Запустить тесты для быстрого обнаружения ошибок.

```
python -m unittest --failfast tests
```

Не запускать миграции при каждом запуске тестов (для Django).

```
./manage.py test --keepdb
```

Распараллелить запуск на несколько процессов (для Django).

```
./manage.py test --parallel=2
```

Оптимизация. Тестовое окружение



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'USER': 'quack',
        'NAME': 'quack_db',
        'TEST': {
            'NAME': 'mytestdatabase',
        },
    },
}

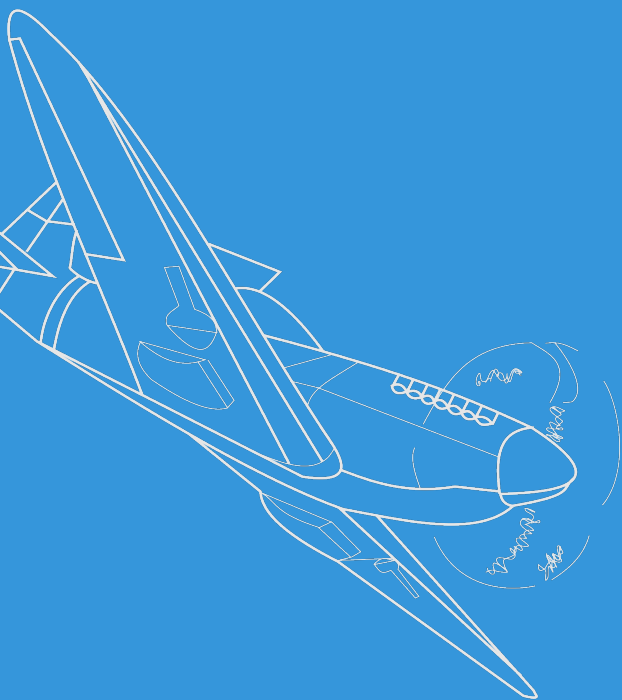
# settings.py
TESTING = 'test' in sys.argv or 'jenkins' in sys.argv

# psql
postgres=# ALTER USER quack CREATEDB;
```

Code time!



Пишем тесты для нашего Django-проекта.



Фикстуры



Фикстуры (fixtures) содержат набор данных, которые Django может импортировать в базу данных.



```
[ {  
  "model": "myapp.person",  
  "pk": 1,  
  "fields": {  
    "first_name": "John",  
    "last_name": "Lennon"  
  }  
},
```

```
{  
  "model": "myapp.person",  
  "pk": 2,  
  "fields": {  
    "first_name": "Paul",  
    "last_name": "McCartney"  
  }  
} ]
```



Скачать данные из приложения app_label в файл file_name.json

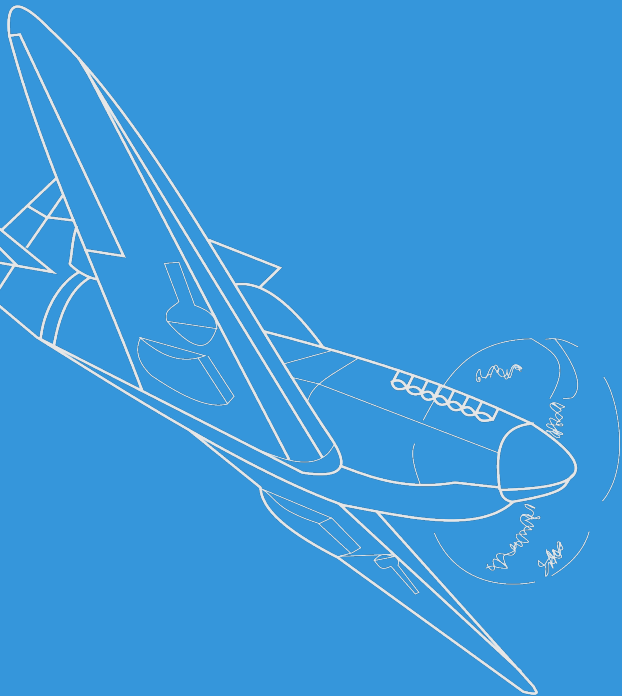
```
django-admin dumpdata app_label > file_name.json
```

Загрузить из файла mydata.json

```
django-admin loaddata mydata.json
```




```
from django.test import TestCase
from myapp.models import Animal
class AnimalTestCase(TestCase):
    fixtures = ['mammals.json', 'birds.json']
    def setUp(self):
        # Test definitions as before.
        call_setup_methods()
    def test_fluffy_animals(self):
        # A test that uses the fixtures.
        call_some_test_code()
```



freezegun

freezegun



```
from freezegun import freeze_time
from datetime import datetime
from django.test import TestCase

class TestWithDatetime(TestCase):
    def setUp(self):
        self.dt = datetime(
            year=2018, month=8,
            day=1, hour=12, minute=00
        )
```

```
def test_something(self):
    meeting_id = 1
    data = {'data': 'some_data'}
    with freeze_time(self.dt):
        update_meeting(meeting_id, data)
```

freezegun



```
from freezegun import freeze_time
import datetime
import unittest

@freeze_time("2012-01-14")
def test():
    assert datetime.datetime.now() == datetime.datetime(2012, 01, 14)

@freeze_time("1955-11-12")
class MyTests(unittest.TestCase):
    def test_the_class(self):
        self.assertEqual(datetime.datetime.now(), datetime.datetime(1955,
11, 12))
```

Домашнее задание № 9



1. Покрыть тестами все вьюхи и по желанию другие функции;
2. Написать selenium-тест (найти элемент + клик на элемент);
3. Использовать mock-объект при тестировании;
4. Использовать factory boy;
5. Узнать степень покрытия тестами с помощью библиотеки coverage.

Рекомендуемая литература

Экстремальное программирование.
Разработка через тестирование |

Бек Кент

Selenium

unittest

Hypothesis

freezegun

factory_boy



Для саморазвития (опционально)

Чтобы не набирать двумя
пальчиками

Спасибо за
внимание!

Антон Кухтичев



a.kukhtichev@mail.ru



[@toshunster](https://www.instagram.com/toshunster)