Автотесты.

Сайты, в процессе развития и разработки, становится все сложнее тестировать вручную. Кроме такого тестирования, сложными становятся внутренние взаимодействия между компонентами - внесение небольшого изменения в одной части приложения влияет на другие.

При этом, чтобы все продолжало работать нужно вносить все больше и больше изменений и, желательно так, чтобы не добавлялись новые ошибки.

Одним из способов который позволяет смягчить последствия добавления изменений, является внедрение в разработку автоматического тестирования - оно должно просто и надёжно запускаться каждый раз, когда вы вносите изменения в свой код.

Данное руководство рассматривает вопросы автоматизации

юнит-тестирования вашего сайта при помощи фреймворка Django для тестов.

urls.py

**urlpatterns = patterns(None,**

**url(r'^registration/$', registration, name='registration'),**

**url(r'^login/$', ..., name='login'),**

**url(r'^logout/$', logout\_then\_login, name='logout'),**

**)**

tests.py

**from django import test**

**from django.core.urlresolvers import reverse**

**\_\_test\_\_ = {"urls": """**

**>>> c = test.Client()**

**>>> c.get(reverse('registration')).status\_code**

**200**

**>>> c.get(reverse('login')).status\_code**

**200**

**>>> c.get(reverse('logout')).status\_code**

**302**

**"""}**

**class YourTestClass(TestCase):**

**@classmethod**

**def setUpTestData(cls):**

**print("setUpTestData: Run once to set up non-modified data for all class methods.")**

**pass**

**def setUp(self):**

**print("setUp: Run once for every test method to setup clean data.")**

**pass**

**def test\_false\_is\_false(self):**

**print("Method: test\_false\_is\_false.")**

**self.assertFalse(False)**

**def test\_false\_is\_true(self):**

**print("Method: test\_false\_is\_true.")**

**self.assertTrue(False)**

**def test\_one\_plus\_one\_equals\_two(self):**

**print("Method: test\_one\_plus\_one\_equals\_two.")**

**self.assertEqual(1 + 1, 2)**

Простейшим способом запуска всех тестов является применение следующей команды:

**python3 manage.py test --verbosity 2**

**python3 manage.py test catalog.tests**

Как было отмечено ранее, мы должны тестировать все то, что является частью нашего кода, а не библиотеки/код, которые уже были протестированы командами разработчиков Django, или Python.

Рассмотрим модель Author. Мы должны провести тесты текстовых меток всех полей, поскольку, даже несмотря на то, что не все они определены, у нас есть проект, в котором сказано, что все их значения должны быть заданы. Если мы не проведём их тестирование, тогда мы не будем знать, что данные метки действительно содержат необходимые значения. Мы уверены в том, что Django создаст поле заданной длины, таким образом наши тесты будут проверять нужный нам размер поля, а заодно и его содержимое.

**class Author(models.Model):**

**first\_name = models.CharField(max\_length=100)**

**last\_name = models.CharField(max\_length=100)**

**date\_of\_birth = models.DateField(null=True, blank=True)**

**date\_of\_death = models.DateField('Died', null=True, blank=True)**

**def get\_absolute\_url(self):**

**return reverse('author-detail', args=[str(self.id)])**

**def \_\_str\_\_(self):**

**return '%s, %s' % (self.last\_name, self.first\_name)**

Откройте файл /catalog/tests/test\_models.py и замените все его содержимое кодом, приведённом во фрагменте для тестирования модели Author (фрагмент представлен ниже).

**from django.test import TestCase**

**# Create your tests here.**

**from catalog.models import Author**

**class AuthorModelTest(TestCase):**

**@classmethod**

**def setUpTestData(cls):**

**#Set up non-modified objects used by all test methods**

**Author.objects.create(first\_name='Big', last\_name='Bob')**

**def test\_first\_name\_label(self):**

**author=Author.objects.get(id=1)**

**field\_label = author.\_meta.get\_field('first\_name').verbose\_name**

**self.assertEquals(field\_label,'first name')**

**def test\_date\_of\_death\_label(self):**

**author=Author.objects.get(id=1)**

**field\_label = author.\_meta.get\_field('date\_of\_death').verbose\_name**

**self.assertEquals(field\_label,'died')**

**def test\_first\_name\_max\_length(self):**

**author=Author.objects.get(id=1)**

**max\_length = author.\_meta.get\_field('first\_name').max\_length**

**self.assertEquals(max\_length,100)**

**def test\_object\_name\_is\_last\_name\_comma\_first\_name(self):**

**author=Author.objects.get(id=1)**

**expected\_object\_name = '%s, %s' % (author.last\_name, author.first\_name)**

**self.assertEquals(expected\_object\_name,str(author))**

**def test\_get\_absolute\_url(self):**

**author=Author.objects.get(id=1)**

**#This will also fail if the urlconf is not defined.**

**self.assertEquals(author.get\_absolute\_url(),'/catalog/author/1')**

Тесты полей проверяют значения текстовых меток (verbose\_name), включая их ожидаемую длину. Все методы имеют описательные имена, а их логика придерживается одной и той же структуры:

**# Получение объекта для тестирования**

**author=Author.objects.get(id=1)**

**# Получение метаданных поля для получения необходимых значений**

**field\_label = author.\_meta.get\_field('first\_name').verbose\_name**

**# Сравнить значение с ожидаемым результатом**

**self.assertEquals(field\_label,'first name')**

Кроме того, нам надо провести тесты наших собственных методов. Они просто проверяют, что имена объектов имеют следующие значения "Last Name, First Name" и что URL-адрес, по которому мы получаем экземпляр Author, такой как ожидается.

**def test\_object\_name\_is\_last\_name\_comma\_first\_name(self):**

**author=Author.objects.get(id=1)**

**expected\_object\_name = '%s, %s' % (author.last\_name, author.first\_name)**

**self.assertEquals(expected\_object\_name,str(author))**

**def test\_get\_absolute\_url(self):**

**author=Author.objects.get(id=1)**

**#This will also fail if the urlconf is not defined.**

**self.assertEquals(author.get\_absolute\_url(),'/catalog/author/1')**

Теперь запустите тесты. Если вы создали модель Author, в соответствии с разделом о моделях данного руководства, то весьма вероятно, что вы получите сообщение об ошибке для метки date\_of\_death, как показано ниже. Тест провалился потому что, в соответствии с соглашением Django, первый символ имени метки должен быть в верхнем регистре (Django делает это автоматически).