Metodika na tvorbu testov

Obsah

Typy testov

Unit tests

- Izolované testy, ktoré testujú jednu konkrétnu funkciu.
- Jednoduchšie sa píšu.
- Čím viac bude jednotkových testov, tým menej treba integračných testov.

Integration tests

- Väčšie testy, ktoré sa sústreďujú na správanie a testujú celú aplikáciu.
- Testujú, ako spolu jednotlivé komponenty pracujú.

Čo sa má testovať

Všetky aspekty kódu, ktoré sa môžu pokaziť alebo nemusia fungovať správne. Čím viac, tým lepšie. Netestujú sa žiadne knižnice ani funkcie poskytnuté od Python alebo Django, pretože tie si oni testujú sami. Testuje sa teda iba vlastný kód.

Štruktúra

Testy sa nachádzajú v adresári ‘tests‘. V prípade, že je mnoho testov v jednej z úrovní tohto adresára, vytvoria sa podadresáre. Súbory s testami majú tvar ‘test_*.py‘. Jednotlivé testy sú oddelené do súborov na základe toho, čo testujú (napr. ‘test_model_control.py‘, ‘test_view_scheme_detail.py‘, ...). Je vhodné oddeliť jednotkové testy od integračných. Príklad štruktúri je nasledovný:

```
/tests/
    __init__.py
    /integration_tests/
        __init__.py
        test_html_control_detail.py
        test_html_scheme_detail.py
    /unit_tests/
        __init__.py
        /model/
            __init__.py
            test_model_control.py
            test_model_scheme.py
        /view/
            __init__.py
            test_view_control_detail.py
            test_view_scheme_detail.py
```

Tvorba testov

Na tvorbu testovacích tried sa využíva trieda ‘TestCase‘ z knižnice django.test. Táto knižnica vychádza zo štandardnej knižnice unittest pre python.

Triedy na testovanie majú tvar '*TestCase(TestCase)'. Metódy na testovanie majú tvar 'test_*()'.
Na konfiguráciu testov sa môžu využiť metódy triedy TestCase (ak nie sú potrebné, netreba ich písať), najznámejšie sú

- setUp()
  - zavolá sa pred každou testovacou metódou
  - slúži na vytvorenie objektov alebo inú prípravu pre ostatné testy v triede
  - každá metóda dostane novú verziu objektu
- tearDown()
  - zavolá sa po každej testovacej metóde
  - slúži na "upratanie" po teste
- setUpClass()
  - zavolá sa iba raz za celú triedu, pred zavolaním ostatných metód
  - slúži na vytvorenie objektov, ktoré sa nebudú v žiadnych testovacích metódach meniť
  - argumentom metódy je trieda podľa konvencií nazývaná 'cls'
  - metóda musí byť označená ako 'classmethod':
    - ```
      @classmethod
      def setUpClass(cls):
          ...
      ```

- tearDownClass()
  - zavolá sa iba raz za celú triedu, po zavolaní ostatných metód
  - slúži na "upratanie" po testovaní celej triedy
  - argumentom metódy je trieda podľa konvencií nazývaná 'cls'
  - metóda musí byť označená ako 'classmethod':
    - ```
      @classmethod
      def tearDownClass(cls):
          ...
      ```

- o ďalších metódach je možné sa dozvedieť viac v dokumentácií: https://docs.python.org/3/library/unittest.html

Obsahom testu je zavolanie metódy 'assert*()'. Zoznam týchto metód je pomerne rozsiahli, preferuje sa nepoužívať iba 'assertFalse()' a 'assertTrue()', ale používať čo najvhodnejšiu metódu vzhľadom na test. Zoznam všetkých metód je možné nájsť v dokumentácií: https://docs.python.org/3/library/unittest.html

Preferuje sa čo najmenej "Assets per Test", ideálne "One Asser per Test". Jednotlivé testy majú byť od seba nezávislé. Poradie vykonania testov nie je v réžií autora testov.

Príklad testovacej triedy:

```
from django.test import TestCase
```

```
class YourTestCase(TestCase):
    def setUp(self):
        #Setup run before every test method.
        pass

    def tearDown(self):
        #Clean up run after every test method.
        pass

    def test_something_that_will_pass(self):
        self.assertFalse(False)
```

Testovacia databáza

Nie je potrebné vytvárať novú databázu na testovanie. Django si vytvorí na začiatku testovania vlastnú (z existujúcej databázy definovanej zo súboru settings.py s prístupom do nej zo súboru settings.ini). Táto databáza bude obsahovať iba čisté tabuľky bez údajov a počas testovania sa štandardne používať. Po testovaní ju následne sám odstráni.

Do databázy je možné ukladať záznamy štandartnými metódami ako pri písaní netestového kódu.

Majme triedu:

```
from django.db import models

# Create your models here.


class CertificationScheme(models.Model):
    """Model for certification scheme"""
    scheme_name = models.CharField(max_length=100)
    identifier = models.CharField(max_length=50)
    version = models.CharField(max_length=10)
    publisher = models.CharField(max_length=50)
```

Vytváranie tejto triedy je napríklad nasledovné:

```
CertificationScheme.objects.create(scheme_name='test_name',
                                   identifier='test_identifier',
                                   version='test_ver',
                                   publisher='test_publisher')
```

V tomto prípade si však treba dať pozor na atribúty, ktoré zvyčajne databáza vytvára sama (napr. pk alebo id). Autor testu sa nemôže spoliehať, že budú vychádzať vždy z prednastavených hodnôt (napr. pk=1), pretože databáza svoje záznamy priebežne odstraňuje po každej testovacej triede, ale neresetuje svoje interné počítadlá (napr. v rámci jednej testovacej triedy bude vytvorený prvý záznam s pk=1, ale v rámci ďalšej testovacej triedy bude vytvorený záznam s pk=2, pritom záznam s pk=1 nebude existovať).

Lepším, prehľadnejším a odporúčaným spôsobom je využívať fixtures. Sú to súbory vo formáte Json, ktorými sa naplní databáza v rámci testovacej triedy pred spustením ľubovoľnej testovacej metódy. Testovacie fixtures vytvárame v adresári ‘/fixtures/tests/‘. Súbory s testovacími fixtures majú tvar ‘test_*.json‘.

Príklad fixtures zo súboru ‘/fixtures/tests/test_schemes.json‘:

```
[
  {
    "model": "my_app.CertificationScheme",
    "pk": 1,
    "fields": {
      "scheme_name": "Profi",
      "identifier": "PF",
      "version": "1.0",
      "publisher": "FIIT"
    }
  }
]
```

Pre načítanie fixtures do testovacej triedy stačí využiť preddefinované pole ‘fixtures‘ a zadať názov súboru (‘fixtures’ je pole, preto je možné zadávať ľubovoľný počet súborov). Príklad načítania fixtures:

```
"""This module holds unit tests for certification scheme."""
from django.test import TestCase
from my_app.models import CertificationScheme

# Create your tests for certification scheme model here.


class CertificationSchemeTestCase(TestCase):
    """Test case for certification scheme in models."""

    # This will load fixtures from file 'fixtures/tests/test_schemes.json'
    fixtures = ['tests/test_schemes']

    def test_fixture(self):
        """Test method for fixture test."""
        scheme = CertificationScheme.objects.get(pk='1')
        self.assertIsInstance(scheme, CertificationScheme)
```

Testy pre jednotlivé časti aplikácie

Ako testovať jednotlivé časti aplikácie je možné nájsť napríklad v tejto dokumentácií: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing

Každá časť aplikácie sa testuje inak. V tejto metodike budú uvedené tie základné.

Testovanie models

Testovanie je zvyčajne jednotkové a závisí od toho, čo je v testovanej triede uvedené. Zvyčajne sa však testuje každá metóda v testovanej triede, a to aspoň raz (ak existujú nejaké špeciálne prípady alebo scenáre, tak všetky). Jedna testovacia metóda by nemala testovať viac ako jednu testovanú metódu.

Majme nasledujúcu triedu:

```
from django.db import models

# Create your models here.


class CertificationScheme(models.Model):
    """Model for certification scheme"""
```

```
    scheme_name = models.CharField(max_length=100)
    identifier = models.CharField(max_length=50)
    version = models.CharField(max_length=10)
    publisher = models.CharField(max_length=50)

    def __str__(self):
        return self.scheme_name

    def temp(self):
        return self.identifier
```

Príklad testu môže vyzerať takto:

```
"""This module holds unit tests for certification scheme."""
from django.test import TestCase
from my_app.models import CertificationScheme

# Create your tests for certification scheme model here.


class CertificationSchemeTestCase(TestCase):
    """Test case for certification scheme in models."""
    fixtures = ['tests/test_schemes']

    def test_creation(self):
        """Test method for object creation."""
        scheme = CertificationScheme.objects.get(pk='1')
        self.assertIsInstance(scheme, CertificationScheme)

    def test_str(self):
        """Test method for str method."""
        scheme = CertificationScheme.objects.get(pk='1')
        self.assertIsInstance(scheme, CertificationScheme)
        self.assertEqual(scheme.__str__(), scheme.scheme_name)

    def test_temp(self):
        """Test method for temp method."""
        scheme = CertificationScheme.objects.get(pk='1')
        self.assertIsInstance(scheme, CertificationScheme)
        self.assertEqual(scheme.temp(), scheme.identifier)
```

Metóda ‘test_creation(self)‘ testuje, či pri vytváraní a získaní objektu nedošlo k chybe (nie je to testovanie funkcie Pythonu, neimportujeme na to žiadnu knižnicu a my sme túto triedu vytvorili, takže to môžeme tiež overiť).

Posledné dve metódy ‘test_str(self)‘ a ‘test_temp(self)‘ testujú metódy v testovanej triede ‘__str__(self)‘ a ‘temp(self)‘. Pred tým ale skontrolujú, či pri vrátení objektu nedošlo k chybe (nevadí, že toto testuje aj prvá testovacia metóda, pretože testovacie metódy majú byť od seba nezávislé – pri písaní týchto dvoch testov treba predpokladať, že metóda ‘test_creation(self)‘ nemusí existovať).

Testovanie views

Testovanie views je možné jednotkovými testami. Vytvorili sme pomocnú testovaciu triedu pre prihlásenie používateľa, pretože ju budú potrebovať viaceré testovacie triedy:

```
"""This module holds setup method for creating user.
Also offers login method for user.
"""
from django.test import TestCase
from django.contrib.auth.models import User
```

```
# Helper class for user login.


class SetUpTestCase(TestCase):
    """Test case for creating user.
    Also offers login method for user.
    """

    @classmethod
    def setUpClass(cls):
        """setUpClass method for creating user."""
        super(SetUpTestCase, cls).setUpClass()
        test_user = User.objects.create_user(username='testuser',
                                             password='12345')

    def login(self):
        """Login method for user."""
        self.client.login(username='testuser', password='12345')
```

Majme nasledujúcu triedu pre view:

```
class SchemeDetailView(generic.DetailView):
    @method_decorator(login_required)
    def get(self, request, **kwargs):
        scheme = CertificationScheme.objects.get(pk=kwargs['pk'])
        info = {'scheme': scheme}
        return render(request, 'scheme_detail.html', context=info)
```

Príklad testu môže byť nasledovný:

```
"""This module holds unit tests for scheme detail view."""
from django.core.urlresolvers import reverse
from my_app.tests.unit_tests.test_setup import SetUpTestCase

# Create your tests for scheme detail view here.


class SchemeDetailTestCase(SetUpTestCase):
    """Test case for scheme detail in views."""
    fixtures = ['tests/test_schemes']

    def test_get_successful(self):
        """Test method for successful get method."""
        self.login()
        resp = self.client.get(reverse('scheme-detail', args=['1']))
        self.assertEqual(resp.status_code, 200)

    def test_get_login_required(self):
        """Test method for failed get method because login is required"""
        resp = self.client.get(reverse('scheme-detail', args=['1']))
        self.assertEqual(resp.status_code, 302)

    def test_get_uses_correct_template(self):
        """Test method for correct template usage."""
        self.login()
        resp = self.client.get(reverse('scheme-detail', args=['1']))
        self.assertEqual(resp.status_code, 200)
        self.assertTemplateUsed(resp, 'scheme_detail.html')
```

Trieda ‘SetUpTestCase‘ nám zabezpečí vytvorenie používateľa a poskytuje metódu na jedno prihlásenie.

Metóda ‘test_get_successful(self)‘ testuje, či je stránka správne načítaná po prihlásení (status_code=200).

Metóda ‘test_get_login_required(self)‘ testuje, či stránka správne vyžaduje prihlásenie používateľa (status_code=302).

Metóda ‘test_get_uses_correct_template(self)‘ testuje, či stránka využíva správny template po prihlásení. Pred tým ale skontroluje, či je status_code=200 (nevadí, že toto testuje aj prvá testovacia metóda, pretože testovacie metódy majú byť od seba nezávislé – pri písaní týchto dvoch testov treba predpokladať, že metóda ‘test_get_successful(self)‘ nemusí existovať).

Testovanie htmls

Tieto testy štrukturujeme na základe html súborov, ktoré predstavujú obrazovky pre používateľa. Preto sa jedná o integračné testy. Vytvorili sme pomocnú testovaciu triedu pre zapnutie brosera a prihlásenie používateľa, pretože ich budú potrebovať viaceré testovacie triedy:

```
"""
This module holds setup methods for integration testing.
"""
import os

from django.contrib.staticfiles.testing import StaticLiveServerTestCase
from django.core.urlresolvers import reverse
from django.test import override_settings
from django.contrib.auth.models import User
from pyvirtualdisplay import Display
from selenium import webdriver


@override_settings(LANGUAGE_CODE="en", LANGUAGES=(("en", "English"),))
class SetUpTestCase(StaticLiveServerTestCase):
    """Test case for setup methods for integration testing."""
    fixtures = ["tests/test_schemes"]
    user_username = "testuser"
    user_email = "testuser@test.com"
    user_password = "12345"

    @classmethod
    def setUpClass(cls):
        """setUpClass method for opening browser."""
        super(SetUpTestCase, cls).setUpClass()
        if os.name == "posix":
            cls.display = Display(visible=0, size=(1366, 720))
            cls.display.start()
        cls.driver = webdriver.Chrome()
        cls.driver.set_window_size(1366, 720)

    @classmethod
    def tearDownClass(cls):
        """setUpClass method for closing browser."""
        cls.driver.quit()
        if os.name == "posix":
            cls.display.popen.kill()
        super(SetUpTestCase, cls).tearDownClass()

    def setUp(self):
        """setUp method for creating user. """
        super(SetUpTestCase, self).setUp()
        User.objects.create_user(username=self.user_username,
                                 email=self.user_email,
                                 password=self.user_password)

    def insert_login_credentials(self, username, password):
```

```
        """Insert login credentials, click and test session cookie."""
        self.driver.find_element_by_id("id_username").clear()
        self.driver.find_element_by_id("id_username").send_keys(username)

        self.driver.find_element_by_id("id_password").clear()
        self.driver.find_element_by_id("id_password").send_keys(password)

        self.driver.find_element_by_xpath(
            '//button[contains(., "Login")]').click()
        self.assertIsNotNone(self.driver.get_cookie("sessionid"))

    def login_user(self, url):
        """Login method for basic user."""
        self.driver.get("%s%s" % (self.live_server_url, url))
        self.insert_login_credentials(self.user_username,
                                      self.user_password)
```

Majme nasledujúcu časť urls:

```
urlpatterns = [
    url(r'^$', views.HomePageView.as_view(), name="index"),
    url(r'^page/$', views.TestPageView.as_view(), name="page"),
    url(r'^accounts/', include('django.contrib.auth.urls')
]
```

Podstránka ‘page‘ využíva html s časťou:

```
<a href="{% url 'index' %}">{% trans "Homepage" %}</a>
```

Príklad integračného testovanie by mohol byť takýto:

```
"""This module holds integration tests for html."""
from django.core.urlresolvers import reverse
from .test_setup import SetUpTestCase

# Create your tests here.


class MyTestCase(SetUpTestCase):
    """Test case."""

    def test_to_homepage(self):
        """Test method for redirect to homepage"""
        self.login_user(reverse('login'))
        self.assertEqual("%s%s" % (self.live_server_url,
                                   reverse("page")
        self.driver.find_element_by_xpath(
            '//a[contains(., "Homepage")]').click()
        self.assertEqual('%s%s' % (self.live_server_url,
                                   reverse('index')),
                         self.driver.current_url)
```

Trieda ‘SetUpTestCase‘ nám zabezpečí správne otvorenie/zatvorenie browsera aj vytvorenie používateľa a poskytuje nám metódy pre prihlásenie.

Metóda ‘test_to_homepage(self)‘ testuje odkaz v html. Je vhodné sa vyhýbať písaniu konkrétnej URL vo forme stringu, preto sa využíva pomocná premenná ‘self.live_server_url’ a metóda ‘reverse()‘. Táto metóda z názvu v jej argumente vytvorí URL príponou (argumentom metódy je názov zadefinovaný v ’urls.py‘). Najskôr prebehne prihlásenie používateľa na adrese s názvom ‘login‘. Následne sa dostaneme na stránku s názvom ‘page’. Vyhľadá sa element pomocou xpath výrazu a vykoná sa kliknutie. Nakoniec sa overí, či prebehlo presmetovanie url.

Spúšťanie testov (na Windowse)

Je nutné mať nainštalované:

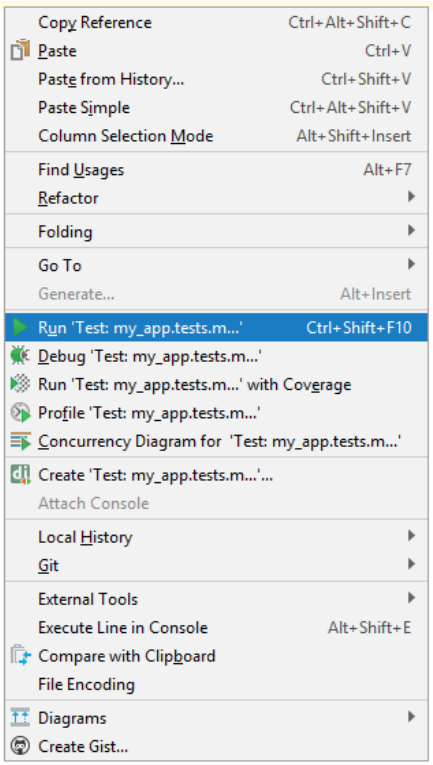
- pyvirtualdisplay – cez command line príkazom ‘pip install pyvirtualdisplay‘
- selenium – cez command line príkazom ‘pip install selenium‘
- Google Chrome – stiahnuť zo stránky (https://www.google.com/chrome/browser/desktop/index.html)
- Chromedriver – zo stránky stiahnuť verziu na win32, v zip súbore sa nachádza chromedriver.exe (https://chromedriver.storage.googleapis.com/index.html?path=2.33/)

Následne je potrebné pridať cestu k priečinku so súborom chromedriver.exe medzi systémové premenné prostredia (do ‘Path’) a reštartovať počítač.

Testy celej aplikácie je možné spúšťať príkazom ’./manage.py test‘ nad adresárom so zdrojovým kódom z command line alebo z konzoly PyCharm. Pre konzolu v PyCharm po stlačení ctrl+alt+R stačí napísať iba “test‘:

```
manage.py@OFCSA > test
```

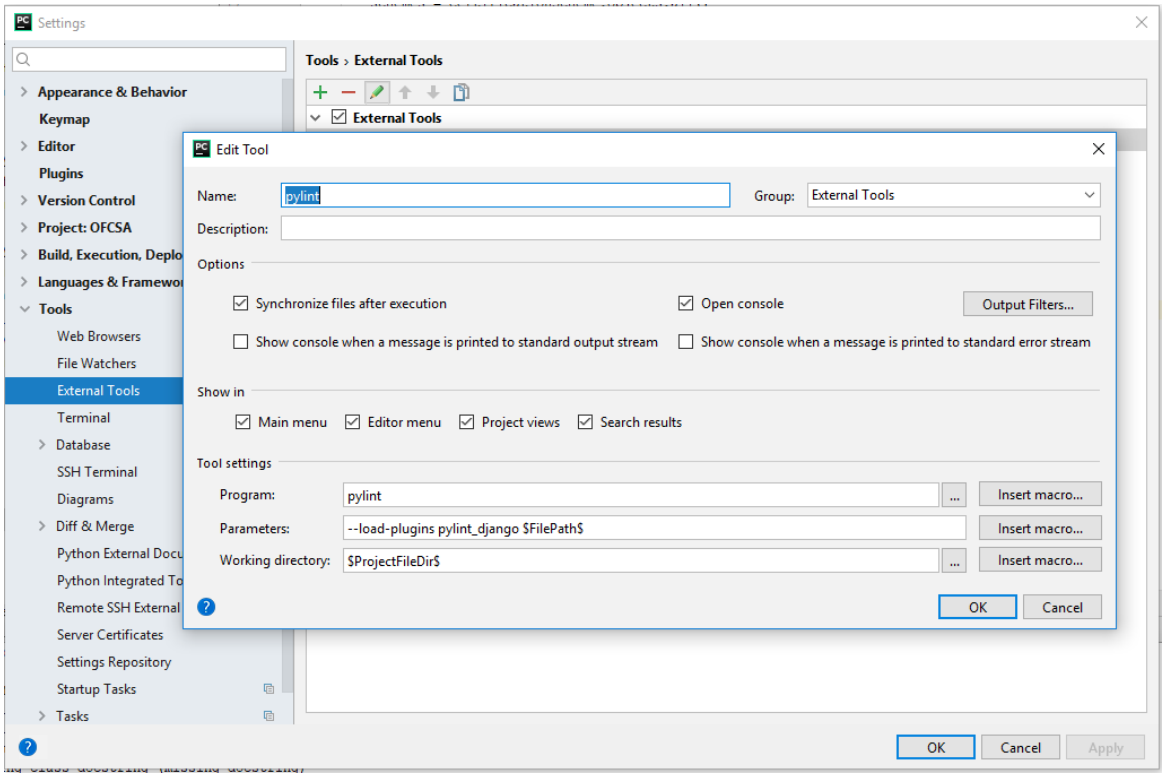
Jednotlivé testy z PyCharm je možné spúšť aj pravým kliknutím na konkrétnu triedu alebo metódu, spustia sa iba testy na základe miesta kliknutia:



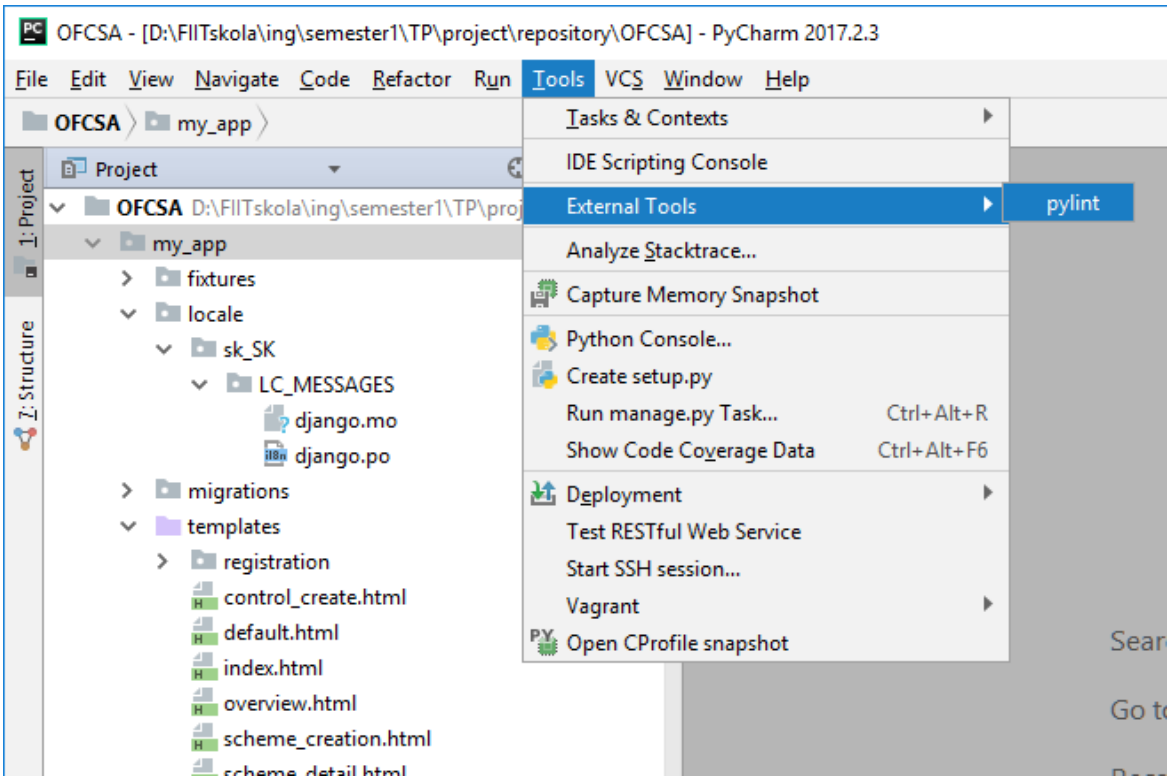
Statická analýza kódu

Na statickú analýzu kódu sa používa pylint s pluginom django. Stiahnuť je to možné cez command line príkazom ‘pip install pylint-django‘. Na konfiguráciu využívame súbor ‘.pylintrc’.

Do PyCharm je možné pylint pridať cez File->Settings->Tools->External Tools->Add(zelený “+“) a takto nakonfigurovať:



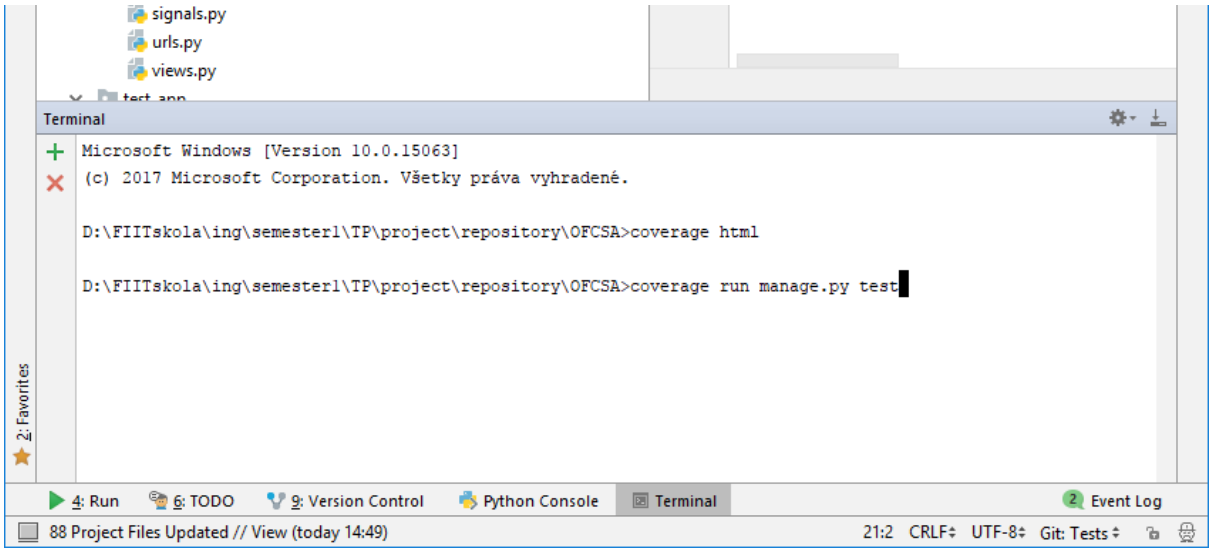
Ak bol pylint takto nakonfigurovaný, spúšťanie je nasledovné (otestujú sa všetky '*.py' súbory v adresári a podadresároch – závisí od označeného adresára, v tomto príklade 'my_app'):



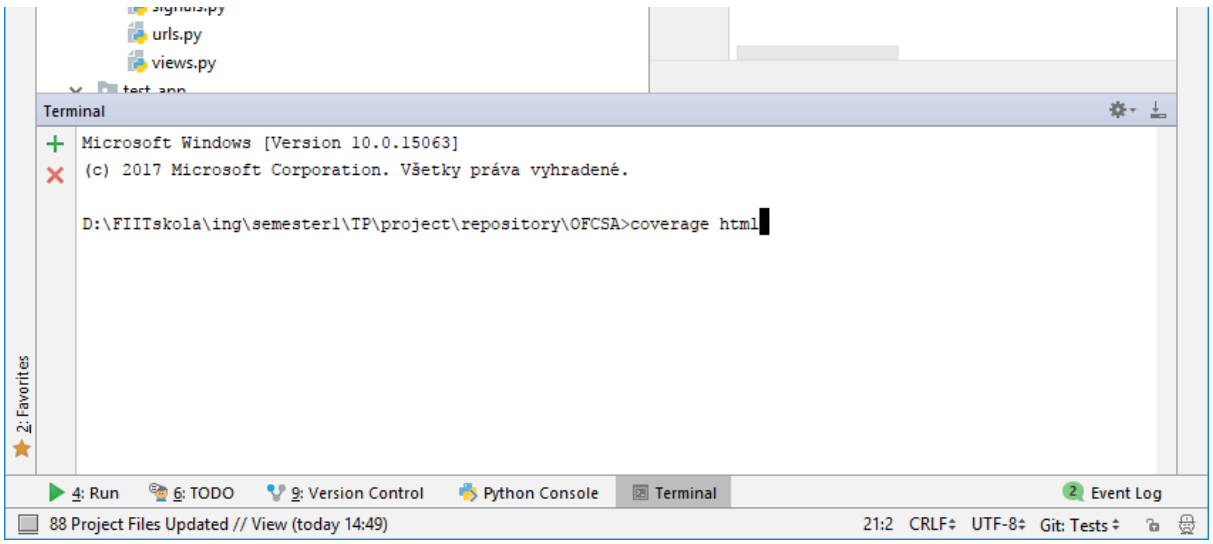
Pokrytie testami

Pri písaní kódu sa môže stať, že niektoré časti autor zabudne otestovať. Preto ako pomôcku využívame ’coverage’. Stiahnuť je to možné cez command line príkazom ‘pip install coverage‘. Na konfiguráciu využívame súbor ‘.coveragerc’.

Spúšťanie nie je možné cez django priamo cez ‘manage.py’. Je nutné využiť command line alebo Terminal z PyCharm. Zapnutie coverage prebieha nad adresárom so zdrojovým kódom príkazom ’coverage run manage.py test‘. Pre Terminal v PyCharm:



Následne prebehnú všetky testy. Následne je možné vygenerovať výstup, najlepšie ako html príkazom ’coverage html‘:



Výstupom je priečinok htmlcov v adresári so zdrojovými súbormi. Po otvorení základného vygenerovaného súboru ‘htmlcov/index.html’ je možné sa preklikávať v prehliadači a vidieť konkrétne zdrojové súbory s vyznačenými neotestovanými riadkami.

Primárne zdroje

[1] https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing
[2] https://docs.djangoproject.com/en/1.11/topics/testing/
[3] https://realpython.com/blog/python/testing-in-django-part-1-best-practices-and-examples/
[4] https://docs.python.org/3/library/unittest.html
[5] http://django-testing-docs.readthedocs.io/en/latest/fixtures.html