Painless testing

Max Kharandziuk

djangostars

Åsa Lundberg
2009
solkatt.deviantart.com

assert is a way to guard you code

- check program invariants
- checking contracts (e.g. pre-conditions and post-conditions)
- checked documentation
- runtime checks on program logic

Invariant check example:

```
def some_function(arg):
    assert not DB.closed()
    ... # code goes here
    assert not DB.closed()
    return result
```

Contract check example:

```python
def first_upper(astring):
    assert isinstance(astring, str) and len(astring) > 0
    result = astring[0].upper()
    assert isinstance(result, str) and len(result) == 1
    assert result == result.upper()
    return result
```

Runtime check example:

```python
assert target in (x, y, z)
if target == x:
    run_x_code()
elif target == y:
    run_y_code()
else:
    assert target == z
    run_z_code()
```

Don’t do that

```
try:
    assert False
except AssertionException:
    ...
```

A little bit of practice:

```
assert sqrt(4) == 2
# should we provide a test for sqrt(9)?
# sqrt(4.5)?
# sqrt(-1)?
```

Concepts:

- black box
- possitive/negative logic
- domains of input

Guidelines:

- Don't start with Selenium
- Don't test UI
- Think about maintance
- Grow your test suite carefully
- Don't test everything

No shared state: don’t use fixtures

cons:

- hard to maintain
- it`s not descriptive
- it`s JSON

Better: factory_boy – factories for data

pros:

- it's just Python code
- explicit input data for tests
- easy to create complicated data(1-n or n-m relations)
- easy to extend for something other then RDBMS

```python
class AuthorFactory(factory.DjangoModelFactory):
  FACTORY_FOR = models.Author
  name = factory.Sequence(lambda i: "name{}".format(i))

class PostFactory(factory.DjangoModelFactory):
  FACTORY_FOR = models.Post
  title = factory.Sequence(lambda i: "title{}".format(i))
  author = factory.SubFactory(AuthorFactory)


post = PostFactory(title='specific')
assert(post.author != None)
```

Proper level of abstraction: think request-response

WebTest

A test framework for functional testing

Django-WebTest

Set of helpers intended to simpify integration with Django

Overview:

- faster then Selenium
- convenient form handling
- convenient authentication

but:

- no JS/Ajax testing

```python
from django_webtest import WebTest

class AuthTest(WebTest):
  fixtures = ['users.json']

  def test_login(self):
      form = self.app.get(reverse('auth_login')).form
      form['username'] = 'foo'
      form['password'] = 'bar'
      response = form.submit().follow()
      self.assertEqual(
        response.context['user'].username,
        'foo'
      )
```

```python
def test_login(self):
    user = UserFactory(username='foo', password=PASSWORD)
    form = self.app.get(reverse('auth_login')).form
    form['username'] = user.username
    form['password'] = PASSWORD
    response = form.submit().follow()
    self.assertEqual(
      response.context['user'].username,
      user.username
    )
```

```python
def test_can_perform_search__with_tag(self):
    me = UserFactory()
    response = self.app.get(
      reverse('search:query',),
      params={
        'q': 'some query',
        'tag': 'super'
      },
      user=me.username,
    )
    self.assertTemplateUsed(response, 'search/search.html')
```

```python
def test_can_sign_up(self):
    _user = factories.UserFactory.build()
    assert _user.pk == None
    response = self.app.post(
        reverse('api-v1:users'),
        params={
            'username': _user.username,
            'password': factories.USER_PASSWORD,
        },
        xhr=True,
    )
    self.assertEqual(201, response.status_code)
    user = models.User.objects.get(username=_user.username)
    self.assertIsNotNone(user)
```

```python
def test_can_perform_search_by_post_title(self):
    me = UserFactory()
    searched_posts = []
    searched_post = [
        PostFactory(title='searched1'),
        PostFactory(title='searched2')
    ]
    response = self.app.get(
        reverse('search:query',),
        params={
            'q': 'search',
        }, user=me.username,
    )
    self.assertEqual(response.status_int, 200)
    result = response.json
    self.assertEqual(len(result['result']), 2)
```

it's really easy to test REST API

```python
def test_user_can_get_and_use_token(self):
    user = factories.UserFactory()
    response = self.app.post(
        reverse('api-v1:login'),
        params={ 'username': user.username, 'password': factories.USER_PAS
        xhr=True,
    )
    token = response.json['token']
    response = self.app.get(
        reverse('api-v1:users'),
        headers={ u'Authorization': 'JWT {}'.format(token) },
    )
    self.assertEqual(response.status_code, 200)
```

```python
def test_user_creates_post(self):
    user = factories.UserFactory()
    assert user.posts.count() == 0
    response = self.app.post(
        reverse('api-v1:login'), xhr=True,
        params={
            'username': user.username, 'password': factories.USER_PASSWORD,
        },
    )
    token = response.json['token']
    response = self.app.post(
        reverse('api-v1:posts'), user=user.username,
        params={ 'title': 'some text', 'body': 'title', },
        headers={u'Authorization': 'JWT {}'.format(token)},
    )
    self.assertEqual(user.posts.count(), 1)
```

Automate boring things!

Consider a simple text search implementation

```
Post(body="foo foo bar")
Post(body="foo bar")
Post(body="bar bar bar")

query("foo bar")
```

no mocks(except third-patry API, e.g.: Twitter)

```
@patch('project.hub.LoginHandler.onBitesRegistrationComplete', on_bites_regist
@patch('project.project_aws_sns.AWSPushNotification.exec_in_pool')
@patch('project.hub.LoginHandler.get_gigya_user_info')
@patch('project.hub.LoginHandler.registration', registration)
@patch('project.session.RedisSessionStore.get_session')
@patch('project.hub.LoginHandler.session', new_callable=PropertyMock)
@patch('project.hub.LoginHandler.check_user_password')
def test_login_api(...
```

```python
class UsersEndpointTestCase(WebTest):
  @patch('users.backends.requests')
  def test_can_sign_up_and_get_token(self, mock_requests):
    data_for_mock = {
      "id": "737522959636397", "first_name": "Max",
      "last_name": "Kharandziuk", "locale": "en_GB",
    }
    mock_requests.get.return_value.status_code = 200
    mock_requests.get.return_value.json = lambda: data_for_mock
    response = self.app.post(
      reverse('api-v1:login'),
      params={ 'username': '#token', 'password': factories.USER_PASSWORD, },
      xhr=True,
    )
    assert response.json['token']
    user = models.User.objects.get(fb_id='737522959636397')
    self.assertIsNotNone(user)
```

no logic in tests(don’t use loops or conditions!)

one class of data input -- one test method

BONAPARTE

Q&A