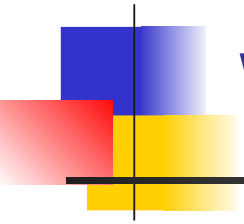


Web Application Testing in Python

James Brucker

Guidance

What to Test?

- Logic
- Flow Control
- Application Flow, e.g. Page Flow
- Configuration

"Don't test Constants" (e.g. HTML template text)

Test your code (not the framework)

What is Being Tested?

```
import django.test
from polls.models import Question

class QuestionTest(django.test.TestCase):
    def setUp(self):
        Question.objects.create(
                question_text="Question One")
        Question.objects.create(
                question_text="Question Two")

    def test_create_questions(self):
        self.assertEqual(2, Questions.objects.count())
```

What is Being Tested? (cont'd)

```
    def test_question_text(self):
        self.assertTrue(
            any("Question One" in q.question_text
                for q in Questions.objects.all() )
```

This is testing Django's model persistence.

OK to do it occasionally while learning Django.

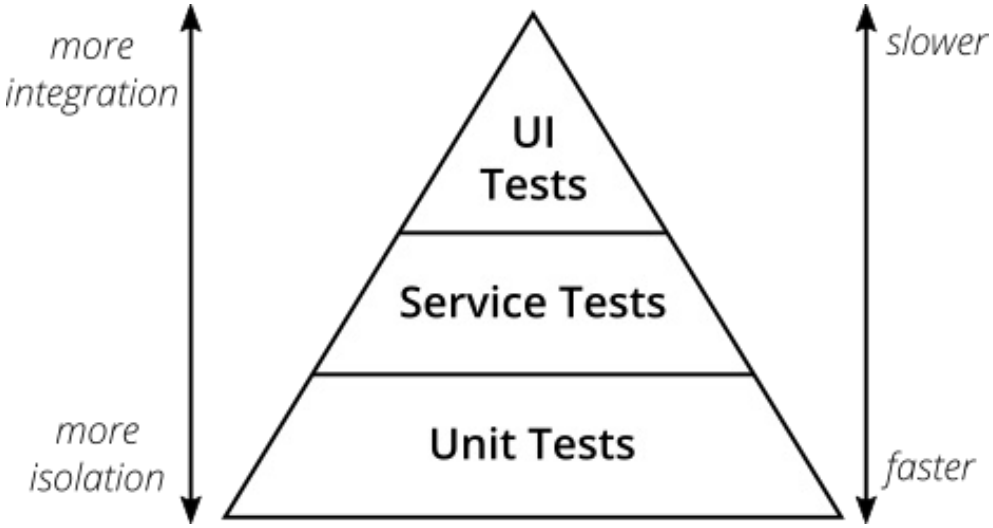
But its not a useful test of your code.

Are Unit Tests Enough?

No.

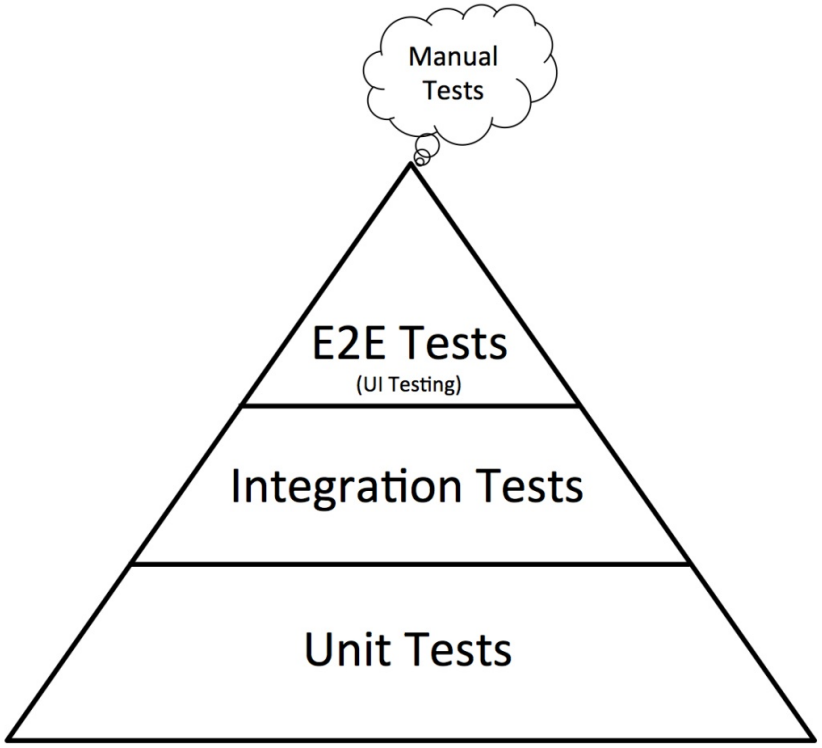
Unit tests don't test whether the application works.

The Testing Pyramid



Mike Cohen's original Pyramid

"Practical" pyramid



Integration Testing

Test the interaction between components.

- Components belonging to your app.
- Back-end services called by front-end
- External components and web services.

Integration tests often access a "service layer", or your standard URLs.

Django Test Client

`django.test.Client` is useful for testing URLs

```
$ python manage.py shell
>>> from django.test import Client
>>> c = Client()
# Get the /polls/ page. Should contain some polls
>>> response = c.get('/polls/')
# Did it succeed?
>>> response.status_code
200
# Print the html content
>>> response.content
'<html>\n<head>\n<style>...\n<h1>Active Polls</h1>...
```

Django Client, again

Can test for redirects, templates, response codes

```
# The root url / should redirect to polls
>>> response = c.get('/')
>>> testcase.assertRedirects(response, '/polls/')
# or:
>>> assert response.status_code == 302
# "Location" header field is the redirect url
>>> assert response.get('Location') == '/polls/'
# The polls list page should use index template
>>> resp = c.get('/polls/')
>>> assert 'polls/index.html' in resp.template_name
# The polls list contains "best prog lang" question
>>> testcase.assertInHTML(
    'best programming language', str(response.body))
```

Test POST, too

```
# create a new poll (new feature!)
client.post('/polls/', {'text': 'Where is God?'})
```

Functional or "End-to-End" Tests

Test the "development" or "production" app while its running! -- not a 'test' server.

Run tests through an actual web browser.

Test the application as a whole.

E2E Testing Tools

Selenium - control an actual web browser using code.

- Interface in many languages, incl. Python & Java
- Django has built-in support
- Selenium IDE for creating tests in a web browser

Cypress.io - Javascript testing tool. Natively interacts with pages in your application.

- uses Mocha and Chai for writing tests
- tests written in Javascript

Puppeteer - library for controlling a "headless" Chrome browser. Uses Javascript and node.js.

- many uses: page scraping, web crawling, testing

Selenium Example

Use duckduckgo.com to find links to Kasertsart U.

Requires:

- Selenium WebDriver (`pip install selenium`)
- driver for Firefox browser (called "geckodriver")

  https://github.com/mozilla/geckodriver/releases

  You can use Chrome or Safari, too.

Selenium: get a web page

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
browser = webdriver.Firefox()
browser.implicitly_wait(10) # seconds
# get the duckduckgo search page
url = "https://duckduckgo.com"
browser.get( url )
```

Selenium: find on page & send data

```
# find the search box on page
# there are many find_by_* commands

field_id = 'search_form_input_homepage'

input_field =
     browser.find_element_by_id(field_id)

input_field.send_keys("Kasetsart Univer")

input_field.send_keys(Keys.RETURN)

# now the browser should display results
```

Page Scraping

```
# get the links from results page
# hacky way: use known CSS formatting

link_divs =
  browser.find_elements_by_css_selector(
  '#links > div')

print(f"Found {len(link_divs)} matches.")

# Each result is a WebElement object
# we can search them. Look for <a href=...

element = link_divs[0]
            .find_element_by_tag_name('a')
```

Page Scraping (2)

```
# "element" refers to another WebElement:
# <a href=...>some text</a>
# Get the 'href' value

url = element.get_attribute('href')

print("First link on page is", url)

# What the heck! Let's go visit...

element.click()

# OK, enough. Go back to search results.

browser.back( )
```

Another Way to Find Links

```
# The Hyperlinks all use class 'result__a'

links = browser.
  find_elements_by_class_name('result__a')

for link in links:

    if link.tag_name == 'a':

        url = link.get_attribute('href')

        print(url)
```

Composite Design Pattern

WebElement may contain other WebElements.

WebElement is the primary object for interacting with a web page using Selenium.

References

The Practical Test Pyramid

`https://martinfowler.com/articles/practical-test-pyramid.html`

Good Selenium Tutorial in Python (7 parts)

https://blog.testproject.io/2019/07/16/set-your-test-automation-goals/

The same author has other good testing tutorials:

https://blog.testproject.io/2019/07/16/

Django E2E Tests with Selenium

TDD in Python (online book)

Several chapters use Selenium for E2E testing of the Django project used in book.

Testing the Github Public API

developer.github.com/v3/users/