Effectively test your webapp with python and selenium

tools and best practices

what’s this talk about

our experience with Selenium and python tools so far

lessons learned

what we are currently trying out

hi !

Andrei Coman

Software Developer @ 

github.com/comandrei

twitter.com/festerc

take 1

```
class ResourceTestCase(TestCase):

def test_create_resource(self):
    page = Page()
    page.create_resource(title='Foo')

def test_view_resource(self):
    page = Page()
    self.assertEqual(page.title, 'Foo')
```

take 1

```
class Page(BasePage):

  def create_resource(self, title):
      title = self.get_element_by_xpath("//div
[@id='primary_asset_display']/../div/input
[@class='title']")
      ...

  def view_resource(self):
      self.title = self.get_element_by_xpath("/div
[@id='view-resource']//h3[@class='title']")
   ...
```

take 1

```
class ResourceTestCase(TestCase):

def test_create_resource(self): < create state on server
    page = Page()
    page.create_resource(title='Foo')

def test_view_resource(self):
    page = Page()
    self.assertEqual(page.title, 'Foo')
```

take 1

```
class Page(BasePage):

  def create_resource(self, title):
      title = self.get_element_by_xpath("//div
[@id='primary_asset_display']/../div/input
[@class='title']")
      # tighly coupled with HTML structure

  def view_resource(self):
      self.title = self.get_element_by_xpath("/div
[@id='view-resource']//h3[@class='title']")
  ...
```

So we want something better

- some of our tests were fixture generators
  - an easy way to create/cleanup test data
- element identification tightly coupled with HTML structure
  - try to move to something more robust

so...

enter take 2

take 2

```
class ResourceTest(LiveServerTestCase):

def setUp(self):
      Post.objects.create(title='Foo')

def test_view_resource(self):
      ….
      page = Page()
      self.assertEqual(page.title, 'Foo')
```

take 2

```
class Page(BasePage):

    def base_page_elements(self):
         self.title = self.get_element_by_id('title')
         self.play_button  = self.get_element_by_id('play')

    def play_media(self):
      self.play_button.click()
```

take 2

```
class ResourceTest(LiveServerTestCase): # not a real env

def setUp(self):
      Post.objects.create(title='Foo')

def test_view_resource(self):
      ….
      page = Page()
      self.assertEqual(page.title, 'Foo')
```

take 2

```
class Page(BasePage):

    def base_page_elements(self):
         self.title = self.get_element_by_id('title')
         self.play_button  = self.get_element_by_id('play')
    # started using css selector instead of xPath


    def play_media(self):
      self.play_button.click()
```

So we want something better

- separate short/long running tests
  - google does this!
- tests must be able to run independently : opens up posibility of running them in parallel
- tests must run on any environment: production or dev environment

what’s next ?

run tests daily and on demand

- must run in under 5-10 minutes to give fast feedback

run on any environment (including local dev env)

decouple tests from HTML structure of the page

take 3

```
class Page(BasePage):

    def base_page_elements(self):
      self.title = self.get_element_by_selector('.sel-resource-title')


    def login(self, username, password):
       username_input = self.get_element_by_selector('.sel-login-username')
       password_input = self.get_element_by_selector('.sel-login-password')
       login_button = self.get_element_by_selector('.sel-login-button')
       username_input.send_keys(username)
       password_input.send_keys(password)
       login_button.click()
```

take 3

```
@pytest.mark.large

def test_view_resource(base_url, selenium, variables):
      page = Page(selenium, base_url)
      expected_resource = variables['resources']['video']
      page.open(expected_resource['url'])
      self.assertEqual(page.title, resource['title'])
```

py.test --base-url http://production.org/

py.test variables

Pass a JSON file and use it as a fixture

py.test -- variables foo.json

```
def test_foo(self, variables):
    assert variables['foo'] == 'bar'
```

foo.json:

```
{
    “foo” : “bar”
}
```

pytest-html

- generate a HTML report of your test run
- include timings
- for failed tests: screenshots and traceback listing

pytest-selenium

- support for multiple webdrivers (Chrome, Firefox)
- support for cloud base services (Sauce Labs, BrowserStack)
- integration with pytest-html for screenshot on failure

take away

leverage application APIs for fixture setup/teardown

add metadata in your application HTML to enable easy element retrieval for tests

define test classes and timebox each class (small/ large/ xlarge)

thank you!

?