Django 5: Automation Testing

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session’s Content

- Template inheritance
- Static files

Today’s Content

- Automation testing
  - Model testing
  - View testing
- LiveServerTestCase

Automation Testing

Model Testing

- `polls/urls.py`
- What is happening in this `TestCase`?
  - Creates a special database for testing
  - Looks for test methods whose name begins with `test`
  - Creates a `Question` instance whose `pub_date` field is 30 days in the future
  - Expected output: `was_published_recently()` returns `False` for questions whose `pub_date` is in the future
- Run `python manage.py test polls`
- What is the output?
  - `AssertionError: True is not False`
- `was_published_recently()` appears to have a small bug

```
from django.test import TestCase
from django.utils import timezone
from datetime import timedelta
from .models import Question

class TestQuestionModel(TestCase):
    def test_was_published_recently_with_future_question(self):
        time = timezone.now() + timedelta(days=30)
        future_question = Question(pub_date=time)
        self.assertFalse(future_question.was_published_recently())
```

Model Testing

- `polls/models.py`
- Refactor `was_published_recently()`
- Run `python manage.py test polls`
- What is the output?

```
def was_published_recently(self):
    now = timezone.now()
    return now - timedelta(days=1) <= self.pub_date <= now
```

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
Destroying test database for alias 'default'...
```

Model Testing

- `polls/tests.py`
- `was_published_recently()` returns `False` for questions whose `pub_date` is older than 1 day
- `was_published_recently()`returns `True` for questions whose `pub_date` is within the last day

```
def test_was_published_recently_with_old_question(self):
    time = timezone.now() - timedelta(days=1, seconds=1)
    old_question = Question(pub_date=time)
    self.assertFalse(old_question.was_published_recently())

def test_was_published_recently_with_recent_question(self):
    time = timezone.now() - timedelta(hours=23, minutes=59, seconds=59)
    recent_question = Question(pub_date=time)
    self.assertTrue(recent_question.was_published_recently())
```

View Testing

- `polls/views.py`
- Refactor `get_queryset()`
- `from django.utils import timezone`
- Returns the last five published questions not including those set to be published in the future

```
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        return Question.objects.filter(
            pub_date__lte=timezone.now()
        ).order_by('-pub_date')[:5]
```

View Testing

- `polls/tests.py`
- `from django.urls import reverse`
- Testing `IndexView`
- `create_question()` - creates a question with the given `question_text` & given number of `days` offset to now

```
def create_question(question_text, days):
    time = timezone.now() + timedelta(days=days)
    return Question.objects.create(question_text=question_text, pub_date=time)

class TestQuestionIndexView(TestCase):
    def test_no_questions(self):
        pass

    def test_past_question(self):
        pass

    def test_future_question(self):
        pass

    def test_future_question_and_past_question(self):
        pass

    def test_two_past_questions(self):
        pass
```

View Testing

- `polls/tests.py`
- `test_no_questions()` - if no questions exist, an appropriate message is displayed
- `test_past_questions()` - questions with a `pub_date` in the past are displayed
- `test_future_questions()` - questions with a `pub_date` in the future are not displayed

```
def test_no_questions(self):
    response = self.client.get(reverse('polls:index'))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'No polls are available.')
    self.assertQuerysetEqual(response.context['latest_question_list'], [])

def test_past_question(self):
    create_question(question_text='Did you enjoy class today?', days=-30)
    response = self.client.get(reverse('polls:index'))
    self.assertQuerysetEqual(
        response.context['latest_question_list'],
        ['<Question: Did you enjoy class today?>']
    )

def test_future_question(self):
    create_question(question_text='What is your favourite course?', days=30)
    response = self.client.get(reverse('polls:index'))
    self.assertContains(response, 'No polls are available.')
    self.assertQuerysetEqual(response.context['latest_question_list'], [])
```

View Testing

- `polls/tests.py`
- `test_future_question_and_past_question()` - only past questions are display
- `test_two_past_questions()` - multiple questions are displayed

```
def test_future_question_and_past_question(self):
    create_question(question_text='Did you enjoy class today?', days=-30)
    create_question(question_text='What is your favourite course?', days=30)
    response = self.client.get(reverse('polls:index'))
    self.assertQuerysetEqual(
        response.context['latest_question_list'],
        ['<Question: Did you enjoy class today?>']
    )

def test_two_past_questions(self):
    create_question(question_text='Did you enjoy class today?', days=-30)
    create_question(question_text='Who is your favourite teacher?', days=-5)
    response = self.client.get(reverse('polls:index'))
    self.assertQuerysetEqual(
        response.context['latest_question_list'],
        ['<Question: Who is your favourite teacher?>', '<Question: Did you enjoy class today?>']
    )
```

View Testing

- `polls/tests.py`
- Testing `DetailView`
  - Testing `ResultsView` will be very similar
- `test_future_question()` - only past questions are display
- `test_two_past_questions()` - multiple questions are displayed

```
class TestQuestionDetailView(TestCase):
    def test_future_question(self):
        future_question = create_question(question_text='What is your favourite course?', days=30)
        url = reverse('polls:detail', args=(future_question.id,))
        response = self.client.get(url)
        self.assertEqual(response.status_code, 404)

    def test_past_question(self):
        past_question = create_question(question_text='Did you enjoy class today?', days=-30)
        url = reverse('polls:detail', args=(past_question.id,))
        response = self.client.get(url)
        self.assertContains(response, past_question.question_text)
```

LiveServerTestCase

LiveServerTestCase

- Install Selenium

```
# Windows
...\> pipenv install selenium

# Linux or macOS
$ pipenv install selenium
```

Installation

- View `Pipfile` & `Pipfile.lock`

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]
django = "*"
selenium = "*"

[requires]
python_version = "3.7"
```

LiveServerTestCase

- Allows us to use automated test clients, i.e., Selenium
  - Executes a series of functional tests
  - Simulates real user's actions
- Resource: https://docs.djangoproject.com/en/3.0/topics/testing/tools/#liveservertestcase

LiveServerTestCase

- `polls/tests.py`
- `setUpClass()` - launches a Django server in the background
- `tearDownClass()` - shuts the Django server down
- Run `python manage.py test polls.tests.TestSelenium.test_python_org_search`
- What is the output?

```
class TestSelenium(StaticLiveServerTestCase):
    @classmethod
    def setUpClass(cls):
        super().setUpClass()
        cls.driver = webdriver.Chrome('polls/chromedriver')

    @classmethod
    def tearDownClass(cls):
        cls.driver.quit()
        super().tearDownClass()

    def test_python_org_search(self):
        self.driver.get('https://www.python.org')
        self.assertTrue('Python' in self.driver.title)
        elem = self.driver.find_element_by_name('q')
        elem.clear()
        elem.send_keys('pycon')
        elem.send_keys(Keys.RETURN)
        self.assertTrue('No results found.' not in self.driver.page_source)
```

Practical

Programming Activity

- Checkout to master - `git checkout master`
- Create a new branch called 11-practical - `git checkout -b 11-practical`
- Copy 11-practical.ipynb from the course materials repository into your practicals repository
- Open up the Anaconda Prompt (it should be install on all lab computers) & `cd` to your practicals repository
- Run the following command: `jupyter notebook`