FASTER END-TO-END TESTS WITH FIXTURE FACTORIES

Stephan Jaensch
@s_jaensch
YELP'S MISSION

To connect people to great local businesses
WHAT I'LL TALK ABOUT

• What are end-to-end tests?
• The problem with setting up end-to-end tests
• How do fixture (or data) factories help solve the problem?
• How can we make tests faster?
END-TO-END TESTS - THE WHAT AND WHY

OUR SERVICE TECHNOLOGY STACK

pyramid

OPEN API

SQLAlchemy
WHAT'S HARD ABOUT E2E TESTS?

Source: https://www.slideshare.net/danveloper/microservices-the-right-way
CREATING E2E TEST STATE

```
-- Some businesses used in promo tests
INSERT INTO `business` (`id`, `name`, `address1`,
INSERT INTO `business_alias` (id, alias, business_
INSERT INTO `business_payment_account_entity` (id,
INSERT INTO `biz_user_business` (id, biz_user_id,
```

```
-- Data for test_get_reasons_for_flagging: one non
INSERT INTO `business` (`id`, `name`, `address1`,
INSERT INTO `business_alias` (id, alias, business_
INSERT INTO `biz_user_business` (id, biz_user_id,
```

Lots of SQL statements...
from myapp.models import Animal

class AnimalTestCase(TestCase):
    def setUp(self):
        Animal.objects.create(name="lion", sound="roar")
        Animal.objects.create(name="cat", sound="meow")

...but for downstream services as well
...and without code duplication
TESTING FACTORY LIBRARIES

- **python_packages-biz_claims_models**
- **python_packages-ee_metrics_models**

- Contain SQLAlchemy models as well as factory functions
- Factories provide a slightly higher level of abstraction
- Take care of common default values
- Make sure data is logically correct
def test_get_answer_list_empty_list(testapp):
    response = testapp.get(
        '/business/{business_id}/question/'
        '{question_id}/answer/v1'.format(
            business_id=encrypt_id(3),
            question_id=encrypt_id(2),
        ),
        params={
            'limit': LIMIT,
            'offset': OFFSET,
        },
    )

    assert response.json == {...}
def test_get_answer_list_empty_list(testapp, question, business_id):
    response = testapp.get(
        '/business/{business_id}/question/
        {question_id}/answer/v1'.format(
            business_id=encrypt_id(business_id),
            question_id=encrypt_id(question.id),
        ),
        params={
            'limit': LIMIT,
            'offset': OFFSET,
        },
    )

    assert response.json == {...}
@pytest.fixture
def business_id(db_session):
    return business_factory.create(db_session)

def create(
    session,
    name='Levchins',
    ...
):
    business = Business(
        name=name,
        ...
    )
    session.add(business)
    session.commit()
    return business.id
WHY NOT USE MODELS DIRECTLY?

<table>
<thead>
<tr>
<th>Table</th>
<th>PK/FK</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>biz_user_private</td>
<td>PK/FK</td>
<td>biz_user_id, last_name, email</td>
</tr>
<tr>
<td>biz_user</td>
<td>PK id</td>
<td>id, first_name, language_preference</td>
</tr>
<tr>
<td>biz_user_business</td>
<td>PK id</td>
<td>id, biz_user_id, business_id</td>
</tr>
</tbody>
</table>
def biz_user_create(session, business_id=None, email=None, ...):
    """Creates a biz_user with password 'password'""
    biz_user = BizUser(
        ...
    )
    session.add(biz_user)
    session.flush()

    session.add(
        BizUserPrivate(
            biz_user_id=biz_user.id,
            email=email,
        )
    )

    if business_id:
        add_biz_user_to_business(session, biz_user.id, business_id)
    
    session.commit()

    return biz_user.id
DOWNSTREAM SERVICES WITH DATA CREATION APIS

@pytest.fixture
def question(question_answer_client, business_id, confirmed_user_id):
    return question_answer_client.business.create_question(
        body=
        {'text': 'Test question...?',
         'platform_source': 'other',
         'subscription': False,}
    ).result().question
PROS AND CONS

- Natural fit for pytest fixtures
- Much easier data creation
- People create separate data entries for each test "automatically"
- Need to maintain data creation factories
- Potentially slower (not sharing data across tests)
HOW TO MAKE TESTS FASTER

Source: http://www.methodsandtools.com/tools/selendroid.html
EXECUTE TESTS IN PARALLEL WITH PYTEST

$ pip install pytest-xdist
$ python -m pytest -s -vvv -n 4 tests/acceptance

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIZCORE-490</td>
<td>Fix high number of failures when running tests in parallel</td>
</tr>
<tr>
<td>BIZCORE-491</td>
<td>Fix end_campaign test</td>
</tr>
<tr>
<td>BIZCORE-492</td>
<td>Fix special hours related tests</td>
</tr>
<tr>
<td>BIZCORE-493</td>
<td>Fix issue with test collection because of invalid token tests</td>
</tr>
<tr>
<td>BIZCORE-499</td>
<td>Fix question and answer parallel test execution</td>
</tr>
<tr>
<td>BIZCORE-500</td>
<td>Fix notification settings parallel test execution</td>
</tr>
<tr>
<td>BIZCORE-520</td>
<td>Enable parallel test execution for all acceptance tests</td>
</tr>
</tbody>
</table>
EXECUTE TESTS IN PARALLEL WITH PYTEST

<table>
<thead>
<tr>
<th>Issue ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIZCORE-529</td>
<td>Investigate tests failure</td>
</tr>
<tr>
<td>BIZCORE-530</td>
<td>Refactor all editing tests in detail acceptance tests</td>
</tr>
<tr>
<td>BIZCORE-669</td>
<td>Fix flaky detail acceptance tests related to service areas</td>
</tr>
<tr>
<td>BIZCORE-711</td>
<td>Fix flaky detail acceptance test due to bio photo</td>
</tr>
<tr>
<td>BIZACT-1058</td>
<td>Fix flaky acceptance tests related to temporary closures</td>
</tr>
</tbody>
</table>
TAKEAWAYS

- Use fixture factories for faster development and more correct test data
- Convert tests for test isolation and repeatability
- Take advantage of it by executing tests in parallel
OTHER TALKS BY YELPERS

"Write more decorators (and fewer classes)"
by Antonio Verardi; Tuesday, 10:30, Anfiteatro 2

"Teeing up Python: Code Golf"
by Lee Sheng; Wednesday, 12:10, PyCharm Room
QUESTIONS?

sjaensch@yelp.com / @s_jaensch

https://github.com/sjaensch/faster_end_to_end_tests_talk.git