Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по модульному тестированию

по дисциплине «Технология разработки качественного программного продукта»

Выполнили студенты гр. 3530904/80105: Пинаев Н. Д.
Распереза А. Д.

Руководитель: Маслаков А. П.

Санкт-Петербург
2022 г.

Задание

Необходимо выполнить модульное тестирование разработанного программного продукта. Кодовая база всего продукта должна быть покрыта тестами на 80% и более, но не менее 20 тестов на каждого члена команды.

Обязательные требования:

1. Тестирование должно производиться автоматически при сборке проекта тем сборщиком, который используется для формирования исполняемого файла (cmake, gradle, maven, ant, etc.) Фреймворки для написания юнит-тестов: junit, testNG, XUnit.Net, Nunit – любые, подходящие для языка вашего проекта и формирующие необходимую отчётность.
2. Применение нескольких техник тест-дизайна: классы эквивалентности, граничные условия, попарное тестирование, etc.

Отчёт по модульному тестированию должен содержать:

1. Описание выполненной работы, использованных инструментах, применённых техниках тест-дизайна.
2. Отчёт о прохождении тестов с результатами и оценкой покрытия кода тестами.
3. Описание процедуры расширения тестового набора на примере добавления нового блока кода, алгоритма, метода.

Описание выполненной работы, использованных инструментов, применённых техниках тест-дизайна.

Для написания модульных тестов мы использовали фрэймворк pytest, а для измерения покрытия pytest-cov и coverage. Для того, чтобы хранить креды мы использовали pytest-config, позволяющий импортировать в тест-проект чувствительные данные, такие как пароль и логин, например.

Были созданы базовые фикстуры для отсутствия переиспользования кода.

Описание работы

Мы протестировали unit-тестами все базовые классы и методы AppStore, StoreClient для работы с Apple Store маркетом, все базовые методы класса GooglePlayAPI и DeviceBuilder, а также вспомогательные функции для работы инструмента.

Отчёт о прохождении тестов с результатами и оценкой покрытия кода тестами.

Общее покрытие составляет 94 %:

Все 45 тестов проходят успешно:

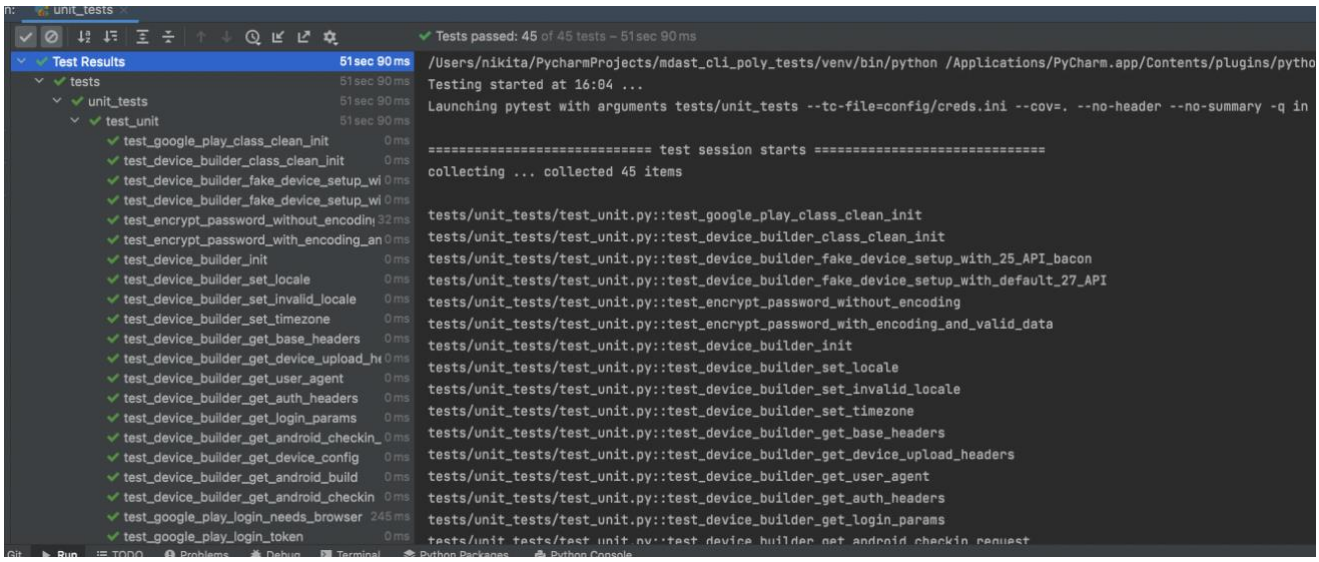
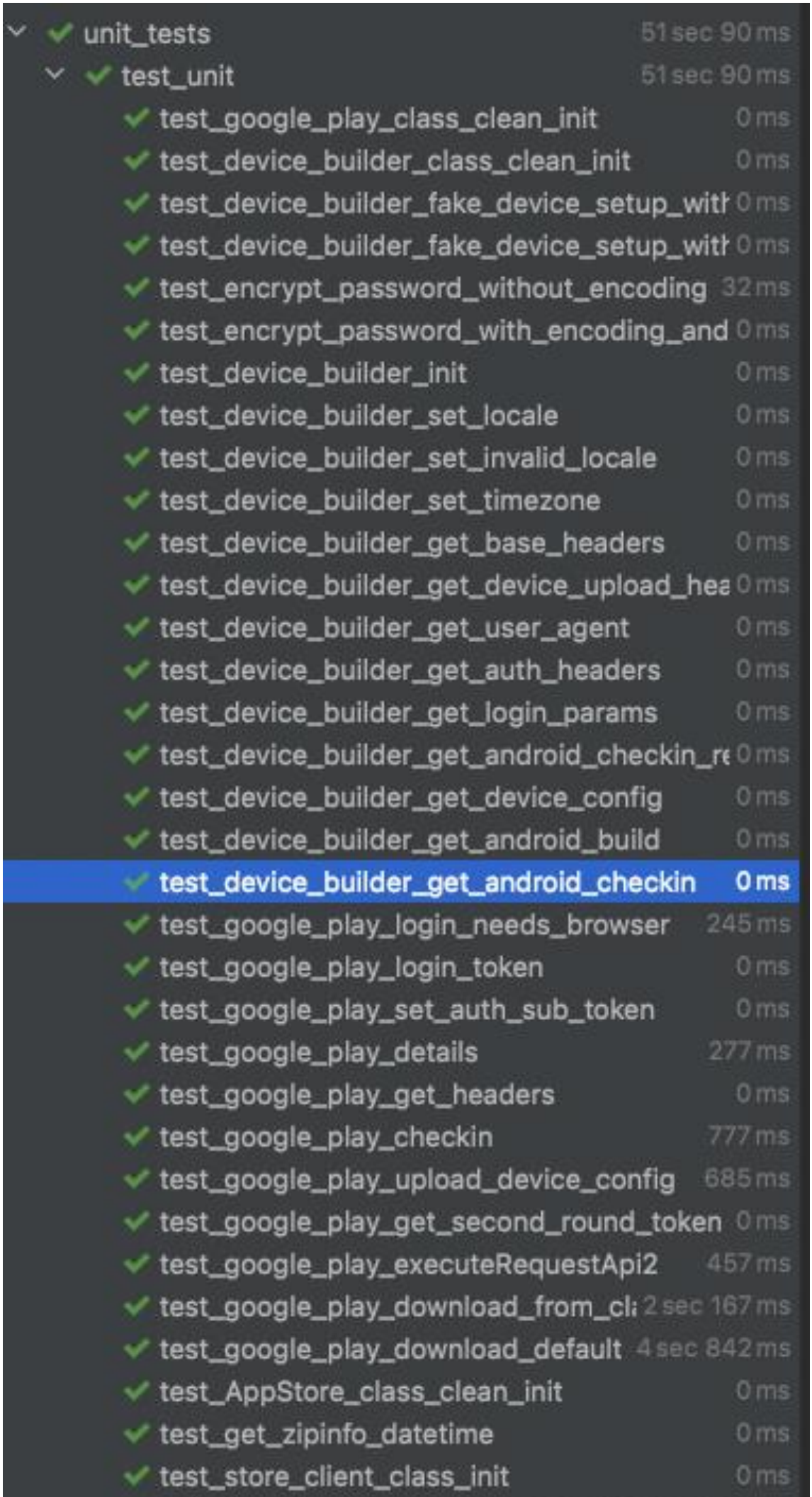


Рисунок 1 Скрин прохождения всех тестов

```
(venv) (base) ➜  mdast_cli_poly_tests git:(unit_tests) ✗ coverage report
Name                                                                    Stmts   Miss  Cover
-------------------------------------------------------------------------------------------
mdast_cli/__init__.py                                                       2      0   100%
mdast_cli/distribution_systems/__init__.py                                  0      0   100%
mdast_cli/distribution_systems/appstore.py                                111     19    83%
mdast_cli/distribution_systems/appstore_client/__init__.py                  0      0   100%
mdast_cli/distribution_systems/appstore_client/store.py                    52      9    83%
mdast_cli/distribution_systems/base.py                                      8      1    88%
mdast_cli/distribution_systems/google_play.py                              37     11    70%
mdast_cli/distribution_systems/gpapi/__init__.py                            0      0   100%
mdast_cli/distribution_systems/gpapi/config.py                            110      1    99%
mdast_cli/distribution_systems/gpapi/googleplay.py                        235     73    69%
mdast_cli/distribution_systems/gpapi/googleplay_pb2.py                    918      0   100%
mdast_cli/distribution_systems/gpapi/utils.py                              13      0   100%
mdast_cli/helpers/__init__.py                                               0      0   100%
mdast_cli/helpers/const.py                                                 17      0   100%
mdast_cli/helpers/helpers.py                                                5      0   100%
mdast_cli/helpers/logging.py                                               19      0   100%
tests/conftest.py                                                          19      0   100%
tests/unit_tests/__init__.py                                                0      0   100%
tests/unit_tests/test_unit.py                                             278      2    99%
-------------------------------------------------------------------------------------------
TOTAL                                                                    1824    116    94%
(venv) (base) ➜  mdast_cli_poly_tests git:(unit_tests) ✗
```

Рисунок 2 Отчет по покрытию кода юнит тестами

Список всех тестов, которые были реализованы:


unit_tests 51 sec 90 ms
test_unit 51 sec 90 ms
test_google_play_class_clean_init 0 ms
test_device_builder_class_clean_init 0 ms
test_device_builder_fake_device_setup_witl 0 ms
test_device_builder_fake_device_setup_witl 0 ms
test_encrypt_password_without_encoding 32 ms
test_encrypt_password_with_encoding_and 0 ms
test_device_builder_init 0 ms
test_device_builder_set_locale 0 ms
test_device_builder_set_invalid_locale 0 ms
test_device_builder_set_timezone 0 ms
test_device_builder_get_base_headers 0 ms
test_device_builder_get_device_upload_hea 0 ms
test_device_builder_get_user_agent 0 ms
test_device_builder_get_auth_headers 0 ms
test_device_builder_get_login_params 0 ms
test_device_builder_get_android_checkin_re 0 ms
test_device_builder_get_device_config 0 ms
test_device_builder_get_android_build 0 ms
test_device_builder_get_android_checkin 0 ms
test_google_play_login_needs_browser 245 ms
test_google_play_login_token 0 ms
test_google_play_set_auth_sub_token 0 ms
test_google_play_details 277 ms
test_google_play_get_headers 0 ms
test_google_play_checkin 777 ms
test_google_play_upload_device_config 685 ms
test_google_play_get_second_round_token 0 ms
test_google_play_executeRequestApi2 457 ms
test_google_play_download_from_cl 2 sec 167 ms
test_google_play_download_default 4 sec 842 ms
test_AppStore_class_clean_init 0 ms
test_get_zipinfo_datetime 0 ms
test_store_client_class_init 0 ms

С кодом можно ознакомиться в репо:

https://github.com/npinaev/mdast-cli

ветка: unit_tests

Примененные техники тест дизайна

Также в ходе выполнения тест дизайна мы реализовали некоторые техники тест-дизайна в соответствии с заданием, а также изучили теорию про них.

Вот что получилось:

1) Эквивалентное тестирование (Equivalence Partitioning)

Для исполнения такой техники тест дизайна надо взять все возможные варианты ввода текста и разделить их на валидные и невалидные. Для примера возьмем:

Для проверки работы метода задания локали для фэйкового девайса, который впоследствии может быть зарегистрирован, отправим пустую строку в это поле,

Сработает обработчик ошибок и поймается ошибка ValueError.

```
def test_device_builder_set_invalid_locale(gp_api):
    current_locale = gp_api.deviceBuilder.locale
    try:
        gp_api.deviceBuilder.setLocale('')
    except ValueError as e:
        assert True
        assert e.args[0] == 'locale is not defined'
    assert gp_api.deviceBuilder.locale != ''
    assert gp_api.deviceBuilder.locale == current_locale
```

Для проверки задания корректной локали попробуем сделать то же самое, но с валидным значением:

```
def test_device_builder_set_locale(gp_api):
    gp_api.deviceBuilder.setLocale('test_locale')
    assert gp_api.deviceBuilder.locale == 'test_locale'
```

Готово!

2) Угадывание ошибок (error guessing)

Нужно отлавливать сбои прогаммы, которые будут ожидаемы, например заданием некорректной последовательности операций. Для использования такой техники тест дизайна необходимо хорошо знать продукт.

Попробуем найти по бандлу приложение, введя bundle=app1337

```
def test_download_app_appstore_invalid_data(appstore_logged_in):
    try:
        appstore_logged_in.bundle_id = 'app1337'
        appstore_logged_in.download_app()
    except SystemExit:

        assert True
```

Системный выход с кодом 4 будет произведен на 2 уровня вложенности ниже, но чтоб добиться именно его, а не Value Error надо хорошо знать продукт

Остальные приемы применить в рамках юнит-тестирования тяжело, так как везде написаны обработчики исключений, которые возвращают ошибки или завершают работу, а изменение состояний не реализуемо, так как в этих юнит тестах созданы фикстуры для сетапа и в каждом тест-кейсе реализовано тестирование определенной функциональности без перехода состояний.

Описание процедуры расширения тестового набора на примере добавления нового блока кода, алгоритма, метода.

Для добавления нового блока кода/алгоритма/метода и расширения тестового набора надо учитывать такие факторы как:

- Изменение нынешнего кода – если в процессе рефакторинга поменялась старая функциональность, то необходимо исправить все тесты, которые это затронуло
- Тесты должны быть написаны в одном стиле, используются те же фикстуры, те же методы, которые были реализованы, нельзя реализовывать одну и ту же функциональность дважды
- Для каждого автоматизированного теста должен быть написан тест-кейс, описывающий необходимые требования, предусловия, ожидаемый результат и шаги выполнения
- Тесты не должны падать без изменения кода, то есть должны быть выполнены критерии по стабильности