Министерство образования и науки РФ
Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ
Telegram-бот для мониторинга цен

Выполнили
студенты гр. 5130904/00103

Солодовников С. Ф.
Мухамадиева Э. В.
Плетнева А. Д.
Нефедова Т. В.

Преподаватель

Маслаков А. П.

Санкт-Петербург
2024

1) Описание выполненной работы, использованных инструментах, применённых техниках тест-дизайна.

В ходе тестирования были задействованы не только инструменты pytest и unittest для написания и запуска тестов, но и дополнительные средства, способствующие эффективной проверке функциональности и корректности работы кода. Одним из ключевых аспектов было обеспечение возможности асинхронного тестирования, что позволило убедиться в правильной обработке асинхронных операций и событий в программе. Для этой цели были использованы специальные декораторы @pytest.mark.asyncio, которые позволяют запускать асинхронные тесты в среде asyncio.

Кроме того, для удобства организации тестов и определения фикстур были применены маркеры @pytest.mark.fixture. Это позволило создавать и использовать различные фикстуры для подготовки данных перед выполнением тестов или для общего использования в различных частях тестового набора. Такой подход способствует улучшению структуры тестов и повышению их эффективности.

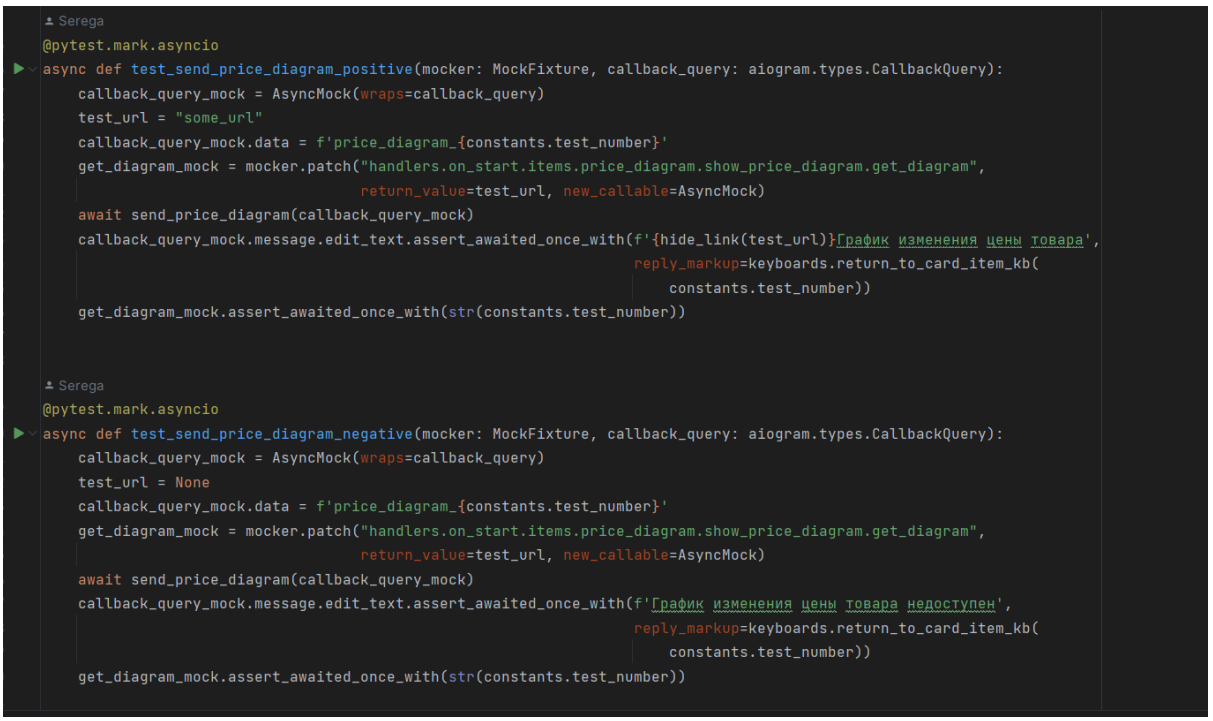
Для оценки уровня покрытия кода тестами был задействован инструмент pytest coverage, который предоставил детальную информацию о том, какие части кода были протестированы, а какие требуют дополнительного внимания. Анализ покрытия кода тестами является важным этапом при модульном тестировании, поскольку позволяет выявить потенциальные проблемные места и обеспечить более полное покрытие функциональности программы.

Модульное тестирование является ключевым аспектом разработки надежного программного обеспечения, особенно когда речь идет о разработке телеграм-ботов с использованием библиотеки aiogram и фреймворка тестирования pytest. Ваш подход к тестированию может включать различные техники дизайна тестов, чтобы обеспечить глубокое и всестороннее покрытие тестами. Вот несколько техник, которые могут быть полезны в вашем контексте:

1. Тестирование на основе эквивалентного разбиения

Эквивалентное разбиение позволяет сократить количество тестов, необходимых для полного покрытия, путем разделения входных данных на классы эквивалентности, так что тестирование одного значения из класса эквивалентности предполагается равносильным тестированию всех остальных значений этого класса.

Пример: При тестировании команд телеграм-бота (например, /start, /help) можно считать, что все валидные команды являются одним классом эквивалентности, а все невалидные входные данные (например, случайный текст) — другим. Таким образом, можно написать один тест для проверки ответа бота на валидную команду и еще один тест для проверки его реакции на невалидный ввод.



```
Serega
@pytest.mark.asyncio
async def test_send_price_diagram_positive(mocker: MockFixture, callback_query: aiogram.types.CallbackQuery):
    callback_query_mock = AsyncMock(wraps=callback_query)
    test_url = "some_url"
    callback_query_mock.data = f'price_diagram_{constants.test_number}'
    get_diagram_mock = mocker.patch("handlers.on_start.items.price_diagram.show_price_diagram.get_diagram",
                                    return_value=test_url, new_callable=AsyncMock)
    await send_price_diagram(callback_query_mock)
    callback_query_mock.message.edit_text.assert_awaited_once_with(f'{hide_link(test_url)}График изменения цены товара',
                                                                   reply_markup=keyboards.return_to_card_item_kb(
                                                                       constants.test_number))
    get_diagram_mock.assert_awaited_once_with(str(constants.test_number))


Serega
@pytest.mark.asyncio
async def test_send_price_diagram_negative(mocker: MockFixture, callback_query: aiogram.types.CallbackQuery):
    callback_query_mock = AsyncMock(wraps=callback_query)
    test_url = None
    callback_query_mock.data = f'price_diagram_{constants.test_number}'
    get_diagram_mock = mocker.patch("handlers.on_start.items.price_diagram.show_price_diagram.get_diagram",
                                    return_value=test_url, new_callable=AsyncMock)
    await send_price_diagram(callback_query_mock)
    callback_query_mock.message.edit_text.assert_awaited_once_with(f'График изменения цены товара недоступен',
                                                                   reply_markup=keyboards.return_to_card_item_kb(
                                                                       constants.test_number))
    get_diagram_mock.assert_awaited_once_with(str(constants.test_number))
```

2. Анализ Граничных Значений (Boundary Value Analysis - BVA)

Техника граничного значения анализа фокусируется на тестировании крайних значений диапазонов входных данных, предполагая, что ошибки чаще всего возникают на границах этих диапазонов.

Пример: Если ваш бот принимает числовые значения для некоторых команд (например, количество элементов для отображения), стоит протестировать значения на границах допустимого диапазона (например, 0, 1, максимально-допустимое количество) и за его пределами (например, -1, максимальное количество +1), чтобы убедиться, что бот корректно обрабатывает эти ситуации.

```python
@pytest.mark.parametrize("number, result",
                         [('123', True), (12345678901234567890, False), (12345678901234567890, True), ('abc', False)])
def test_validate_articul(number, result):
    assert validate_articul(number) == result
```

3. Branch Testing (Тестирование ветвлений) является техникой дизайна тестов, которая используется в контексте тестирования программного обеспечения. Эта техника фокусируется на проверке всех возможных путей выполнения через код программы, включая каждую ветку управляющих структур, таких как условные операторы (if/else) и циклы (for, while).

Цель Branch Testing состоит в том, чтобы убедиться, что каждая ветка кода была исполнена хотя бы один раз, что помогает обнаруживать ошибки в логике ветвления и гарантировать, что все условные операторы были тщательно проверены. Это один из способов достижения высокого уровня покрытия кода тестами и повышения качества программного продукта.

Предугадывание ошибки (Error Guessing - EG). Это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы "предугадать" при каких входных условиях система может выдать ошибку. Например, спецификация говорит: "пользователь должен ввести код". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание ошибки.

```python
@pytest.mark.asyncio
async def test_process_name_articul_incorrect(state: FSMContext, product_service, user_service):
    incorrect_articul = 'qwwe222'
    message = AsyncMock(text=incorrect_articul)
    with patch('utils.validate_articul') as mock_validate_articul:
        mock_validate_articul.return_value = False
        await process_name(message, state, product_service, user_service)
    assert await state.get_data() == {'articul': incorrect_articul}
    assert await state.get_state() == Form.articul
    message.answer.assert_called_once_with(f"Артикул некорректен, попробуйте снова!",
                                           reply_markup=keyboards.return_to_menu_kb)


 pletn
@pytest.mark.asyncio
async def test_process_name_articul_correct_not_exists(state: FSMContext, product_service, user_service):
    correct_articul = '123456789'
    message = AsyncMock(text=correct_articul)
    with patch('utils.validate_articul') as mock_validate_articul:
        mock_validate_articul.return_value = True
        with patch('utils.exist_in_api') as mock_exist_in_api:
            mock_exist_in_api.return_value = None
            await process_name(message, state, product_service, user_service)
    assert await state.get_data() == {'articul': correct_articul}
    assert await state.get_state() == Form.articul
    message.answer.assert_called_once_with(f"Товара не существует попробуйте снова!",
                                           reply_markup=keyboards.return_to_menu_kb)
```

2) Отчёт о прохождении тестов с результатами и оценкой покрытия кода тестами:

```
111 passed,
```

```
(.venv) PS C:\Users\eliel\PycharmProjects\wb_bot1> coverage report
Name                                                                    Stmts   Miss  Cover
-------------------------------------------------------------------------------------------
api\api_service.py                                                         68     23    66%
api\models\item_info.py                                                     9      0   100%
config.py                                                                   4      0   100%
db\models\__init__.py                                                       4      0   100%
db\models\base.py                                                           2      0   100%
db\models\product.py                                                       12      0   100%
db\models\user.py                                                           8      0   100%
db\models\user_product.py                                                  10      0   100%
db\product_service.py                                                      67      0   100%
db\user_service.py                                                         71      0   100%
db\utils.py                                                                29      5    83%
form.py                                                                     8      0   100%
handlers\__init__.py                                                        2      0   100%
handlers\on_start\__init__.py                                               6      0   100%
handlers\on_start\add_item\__init__.py                                      0      0   100%
handlers\on_start\add_item\back.py                                         10      0   100%
handlers\on_start\add_item\handle_input.py                                 46      0   100%
handlers\on_start\add_item\input_item.py                                   10      0   100%
handlers\on_start\help.py                                                   9      0   100%
handlers\on_start\items\__init__.py                                         5      0   100%
handlers\on_start\items\change_notify\__init__.py                           3      0   100%
```

```
handlers\on_start\items\change_notify\back.py                    14      0   100%
handlers\on_start\items\change_notify\change_notify.py            9      0   100%
handlers\on_start\items\change_notify\change_notify_options.py   22      0   100%
handlers\on_start\items\change_price_to_last.py                  15      0   100%
handlers\on_start\items\my_items.py                              14      0   100%
handlers\on_start\items\price_diagram\__init__.py                 2      0   100%
handlers\on_start\items\price_diagram\back.py                    12      4    67%
handlers\on_start\items\price_diagram\show_price_diagram.py      54     12    78%
handlers\on_start\items\stop_tracking.py                         12      0   100%
handlers\on_start\support.py                                     12      0   100%
handlers\on_start\support_button.py                              11      0   100%
handlers\router.py                                                2      0   100%
handlers\start.py                                                13      0   100%
keyboards.py                                                     11      0   100%
main.py                                                          31      8    74%
middleware\service_middleware.py                                 20      0   100%
services\items_checker.py                                        39      0   100%
services\notifier.py                                             61     51    16%
utils.py                                                          6      1    83%
-------------------------------------------------------------------------------
TOTAL                                                           743    104    86%
```

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

1. Собираемся добавить кнопку “Написать в поддержку” в телеграм боте
2. Пишем метод для кнопки

```python
from aiogram import Router, F
from aiogram.fsm.context import FSMContext
from aiogram.types import Message

import keyboards
from form import Form

from handlers.router import router
from main import bot


2 usages  pletn
@router.message(F.text.casefold() == '📝 написать в поддержку')
async def support(message: Message, state: FSMContext) -> None:
    await state.set_state(Form.support)

    await message.answer(
        f'Привет! Оставьте свои пожелание разработчикам)',
        reply_markup=keyboards.return_to_menu_kb
    )
```

3. Пишем обработчик для пользовательского ввода

```python
from aiogram import Router, F
from aiogram.fsm.context import FSMContext
from aiogram.types import Message

import keyboards
from form import Form

from handlers.router import router
from main import bot


2 usages  pletn
@router.message(Form.support)
async def process_msg_to_support(message: Message, state: FSMContext) -> None:
    # await state.update_data(support=message.text)
    await bot.send_message(491198715, message.text)
    await message.answer('Спасибо! Комментарий был отправлен разработчикам.', reply_markup=keyboards.menu_kb)
    await state.clear()
```

4. Для расширения тестового набора для тестирования нового функционала, пишем тесты для кнопки
   Чтобы написать тест, мы можем переиспользовать уже имеющийся код, например, в виде фикстур, которые можно импортировать из conftest.

```python
from unittest.mock import AsyncMock, create_autospec
from tests.conftest import *

import aiogram
import pytest
from aiogram.fsm.context import FSMContext
from pytest_mock import MockFixture

import keyboards
from form import Form
from handlers import support
from tests.conftest import input_message


Serega
@pytest.mark.asyncio
async def test_support_button(input_message: aiogram.types.Message, mocker: MockFixture):
    fsm_mock: FSMContext = create_autospec(FSMContext, instance=True, new_callable=AsyncMock)
    message_answer_mock: AsyncMock = mocker.patch("aiogram.types.Message.answer", new_callable=AsyncMock)

    await support(input_message, fsm_mock)

    fsm_mock.set_state.assert_awaited_once_with(Form.support)
    message_answer_mock.assert_awaited_once_with(f'Привет! Оставьте свои пожелание разработчикам)',
                                                 reply_markup=keyboards.return_to_menu_kb)
```

```python
from unittest.mock import Mock, AsyncMock, create_autospec
from tests.conftest import *

import aiogram
import pytest
from aiogram.fsm.context import FSMContext
from pytest_mock import MockFixture

import keyboards
import utils
from tests.conftest import input_message
from handlers.on_start.support import process_msg_to_support


Serega
@pytest.mark.asyncio
async def test_support(input_message: aiogram.types.Message, mocker: MockFixture):
    admin_id = 491198715
    send_message_mock = mocker.patch("aiogram.Bot.send_message", new_callable=AsyncMock)
    fsm_mock: FSMContext = create_autospec(FSMContext, instance=True, new_callable=AsyncMock)
    message_answer_mock: AsyncMock = mocker.patch("aiogram.types.Message.answer", new_callable=AsyncMock)

    await process_msg_to_support(input_message, fsm_mock)

    send_message_mock.assert_awaited_once_with(admin_id, input_message.text)
    message_answer_mock.assert_awaited_once_with("Спасибо! Комментарий был отправлен разработчикам.",
                                                 reply_markup=keyboards.menu_kb)
    fsm_mock.clear.assert_awaited_once()
```