

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

ИНТЕГРАЦИОННОЕ 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

Сценарий 1 *(серега)

Сценарий: Проверка реакции бота на команду /start

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Получение ответа от бота.

Ожидаемый результат:

- Бот должен отправить приветственное сообщение, в объекте сообщения должен содержаться объект ReplyMarkup клавиатуры.

Сценарий 2 *(элина)

Сценарий: проверка UserService сервиса: метод patch_start_price

Шаги:

1. Создать объект UserProduct
2. Выполнить patch_start_price для UserProduct
3. Убедиться, что UserProduct была изменена start_price

Постусловия:

1. Удалить из БД тестовый UserProduct

Ожидаемый результат:

- метод UserService .patch_start_price() работает корректно

Сценарий 3 *(таня)

Сценарий: Проверка реакции бота на кнопку /Прекратить отслеживание

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения /Мои товары
4. Нажатие на кнопку /Прекратить отслеживание у тестового товара
5. Получение ответа от бота

Ожидаемый результат:

- Бот должен отредактировать сообщение с карточкой товара.

Сценарий 4 *(серега)

Сценарий: Проверка реакции бота на кнопку /Помощь

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения /Помощь
4. Получение ответа от бота

Ожидаемый результат:

- Бот должен отправить сообщение, содержащее информацию о боте и его возможностях

Сценарий 5 * (серега)

Сценарий: Проверка реакции бота на кнопку /Обратиться в поддержку

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения /Обратиться в поддержку
4. Получение ответа от бота, после которого возможен ввод сообщения от пользователя
5. Отправка сообщения, содержащего обращение
6. Получение ответа от бота о том, что сообщение доставлено разработчикам
7. Проверка, что сообщение доставлено разработчикам

Ожидаемый результат:

- после пункт 3:.. сообщение с предложением ввести текст обращения пришло пользователю
- после пункт 5: сообщение о доставке сообщения пришло пользователю
- сообщение с текстом обращение пришло разработчику

Сценарий 6*(серега)

Сценарий: Проверка работы бота в случае появления товара в наличии

Шаги:

1. Запуск телеграм бота.
2. Замокнуть get_changed_items - вернуть “измененные товары”
3. Получение ответа от бота о том, что товар появился в наличии
4. Получение ответа от бота с карточкой товара, который появился в наличии

Ожидаемый результат:

- Бот должен отправить сообщение о том, что товар появился в наличии
- Бот должен отправить сообщение с карточкой товара, который появился в наличии

Сценарий 7*(серега)

Сценарий: Проверка работы бота в случае, если товар больше не в наличии

Шаги:

1. Запуск телеграм бота.
2. Замокнуть get_changed_items - вернуть “измененные товары”
3. Получение ответа от бота о том, что товара нет в наличии
4. Получение ответа от бота с карточкой товара, которого нет в наличии

Ожидаемый результат:

- Бот должен отправить сообщение о том, что товар появился в наличии
- Бот должен отправить сообщение с карточкой товара, который появился в наличии

Сценарий 8*(таня)

Сценарий: Проверка реакции бота на кнопку /Посмотреть динамику цен

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Переход на /Мои товары
4. Выбрать товар и у него перейти на /Посмотреть динамику цен
5. Получение отредактированного сообщения от бота, сообщение должно содержать hide_link, а также изменения встроенной клавиатуры

Ожидаемый результат:

- Сообщение должно измениться соответствующим образом

Сценарий 9*(саша)

Сценарий: Проверка реакции бота на кнопку /Отслеживать от последней цены

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Переход на /Мои товары
4. Выбрать товар и у него перейти на /Отслеживать от последней цены
5. Получение всплывающего сообщения “Цена успешно изменена”, а также редактирование карточки товара с изменением начальной цены

Ожидаемый результат:

- Всплывающее сообщение “Цена успешно изменена” и редактирование карточки товара

Сценарий 10*(саша)

Сценарий: Проверка реакции бота на кнопку “мои товары”, если у тестируемого пользователя уже есть товары

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения “Мои товары”
4. Получение ответа бота

Ожидаемый результат:

- Бот отправляет несколько сообщений, каждое сообщение содержит объект клавиатуры, в котором представлена карточка товара

Сценарий 11*(саша)

Сценарий: Проверка реакции бота на кнопку “мои товары”, если пользователь еще не добавил товары

Шаги:

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения “Мои товары”
4. Получение ответа бота

Ожидаемый результат:

- Бот должен отправить сообщение с информацией о том, что товары еще не были добавлены.

Сценарий 12*(саша)

Сценарий: Проверка реакции бота на попытку добавить **существующего товара в наличии, если у пользователя его нет**

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения “Добавить товар”
4. Отправка сообщения с артикулом товара
5. Получение ответа бота

Ожидаемый результат:

- Бот должен отправить сообщение “Товар успешно добавлен”
- следующим сообщением бот должен отправить сообщение с карточкой товара

Сценарий 13*(саша)

Сценарий: Проверка реакции бота на попытку добавить **существующего товара в наличии, если у пользователя уже есть этот товар**

1. Запуск телеграм бота.
2. Отправка сообщения с командой /start от пользователя.
3. Отправка сообщения “Добавить товар”
4. Отправка сообщения с артикулом товара
5. Получение ответа бота

Ожидаемый результат:

- Бот должен отправить сообщение “Вы уже отслеживаете этот товар!”

Сценарий 14*(элина)

Сценарий: проверка Product сервиса: метод patch_product

Шаги:

1. Добавить в БД тестовый товар
2. Сохранить все поля товара
3. Выполнить patch тестового товара в бд на новые значения полей
4. Убедиться, что изменились только требуемые значения полей

Постусловия:

5. Удалить из БД тестовый товар

Ожидаемый результат:

- метод Product.patch_product() работает корректно

Сценарий 15*(тан)

Сценарий: проверка Product сервиса: метод add_product если такого товара не существует

Шаги:

1. Создать тестовый товар с артикулом art, добавить его в БД
2. Создать еще один тестовый товар с таким же артикулом art
3. Выполнить add_product с товар из пункта 2.
4. Убедиться, что в БД не был добавлен товар из пункта 2.

Постусловия:

5. Удалить из БД товар 1.

Ожидаемый результат:

- метод Product.add_product() работает корректно для товара с дубликатом по артикулу.

Сценарий 16*(таня)

Сценарий: проверка UserProduct сервиса: метод add_user_product

Шаги:

1. Добавить в БД тестовый товар
2. Добавить в БД тестового юзера
3. Выполнить add_user_product для тестового юзера и товара
4. Убедиться, что связь юзера и товара была успешно создана

Постусловия:

5. Удалить из БД тестовые данные

Ожидаемый результат:

- метод UserProduct.add_user_product() работает корректно

Сценарий 17*(таня)

Сценарий: проверка UserProduct сервиса: метод delete_user_product

Шаги:

1. Добавить в БД тестовый товар
2. Добавить в БД тестового юзера
3. Связать юзера с товаром
4. Выполнить delete_user_product для тестового юзера и товара
5. Убедиться, что связь была удалена из БД

Постусловия:

6. Удалить из БД тестовые данные

Ожидаемый результат:

- метод UserProduct.delete_user_product() работает корректно

Сценарий 18*(элина)

Сценарий: проверка UserService сервиса: метод add_user

Шаги:

1. Создать объект тестового юзера
2. Выполнить add_user
3. Убедиться, что юзера был добавлен в БД

Постусловия:

4. Удалить из БД тестовые данные

Ожидаемый результат:

- метод UserProduct.add_user() работает корректно

Сценарий 19*(элина)

Сценарий: проверка ApiService сервиса: метод get_product, проверка совпадение формата данных с ожидаемым

Шаги.

1. Найти товар, который точно будет долго жить на маркетплейсе, его артикул : art
2. Выполнить get_product
3. Проверить ответ

Ожидаемый результат

- Полученный объект должен иметь следующие поля:
- id == art
- sizes: not Null

Сценарий 20*(элина)

Сценарий: проверка ApiService сервиса: метод get_price_history, проверка совпадения формата данных с ожидаемым

Шаги.

1. Найти товар, который точно будет долго жить на маркетплейсе, его артикул : art
2. Выполнить get_price_history
3. Проверить ответ

Ожидаемый результат

- Полученный массив либо пустой [], либо объект массива содержит следующие поля
- dt: валидный timestamp
- price: словарь вида: {'RUB': ЗНАЧЕНИЕ}

Техники тест дизайна

```
@pytest.mark.asyncio(scope="module")
async def test_start(start_bot, conv):
    await conv.send_message("/start")
    conv.get_response()
    resp: Message = await conv.get_response()
    reply_markup: ReplyKeyboardMarkup = resp.reply_markup
    all_buttons: list[KeyboardButton] = []
    for row in reply_markup.rows:
        all_buttons.extend(row.buttons)
    for button, title in zip(all_buttons, constants.expected_start_kb_texts):
        assert button.text == title

    assert resp.text == utils.info
```

В приведенном примере теста на Python мы проверяем, что после отправки команды /start пользователю отображается корректное приветственное сообщение и предлагается ожидаемый набор кнопок для дальнейших действий. Этот тест включает в себя элементы **тестирования по контракту**(Contract Testing): (проверка соответствия текста кнопок ожидаемым значениям) и **тестирования сценариев использования**(Use Case Testing) (имитация действий пользователя при начале работы с ботом).

Тестирование последовательности (Sequence Testing): Основывается на тестировании последовательностей операций или процессов, которые должны выполняться в определенном порядке.

```
@pytest.mark.asyncio
async def test_support(connection, conv, start_bot):
    await conv.send_message("/start")
    await conv.get_response()
    await conv.send_message(constants.write_support_text)
    resp: Message = await conv.get_response()
    assert resp.text == constants.support_hello_text
    reply_markup: ReplyInlineMarkup = resp.reply_markup
    all_buttons: list[KeyboardButtonCallback] = []
    for row in reply_markup.rows:
        all_buttons.extend(row.buttons)
    cb_button = all_buttons[0]
    assert cb_button.text == 'Назад'
    assert cb_button.data == b'to_menu'
    test_support_msg = "there are huge problems in bot"
    await conv.send_message(test_support_msg)
    resp = await conv.get_response()
    assert resp.text == test_support_msg
    resp = await conv.get_response()
    assert resp.text == constants.support_end_text
```

Взаимодействие Компонентов: Тест проверяет, как бот реагирует на команды и сообщения пользователя, отображая правильные кнопки и тексты. Это демонстрирует интеграцию и взаимодействие между разными частями системы.

Проверяется, что система корректно переходит от одного шага к другому и правильно обрабатывает пользовательский ввод, подтверждая функциональность интеграции между компонентами.

Инкрементное тестирование (Incremental Testing):

"снизу вверх" (Bottom-Up Testing). В рамках этого подхода мы начали с тестирования сервисов, обращающихся к базе данных, которые находятся на более низком уровне в иерархии системы. Затем, после того как эти компоненты были протестированы и верифицированы, мы перешли к тестированию обработчиков телеграм-бота, которые используют уже проверенные сервисы и находятся на более высоком уровне архитектуры.

```

@pytest.mark.asyncio(scope="module")
async def test_my_items_empty(start_bot, conv, mocker: MockerFixture):
    await conv.send_message("/start")
    await conv.get_response()
    mocker.patch(target="db.user_service.UserService.get_user_products", return_value=None)
    await conv.send_message("👋 Мои товары")
    resp: Message = await conv.get_response()
    print(resp)
    assert resp.text == 'Вы еще ничего не добавили'

```

Отчёт о прохождении тестов с результатами на сервере непрерывной интеграции

```

1  ▶ Run pytest ./tests/integration
14 ===== test session starts =====
15 platform linux -- Python 3.10.13, pytest-8.1.1, pluggy-1.4.0
16 rootdir: /home/runner/work/wb_bot/wb_bot
17 plugins: asyncio-0.21.0, mock-3.12.0
18 asyncio: mode=strict
19 collected 21 items
20
21 tests/integration/add_item_test.py .. [ 9%]
22 tests/integration/change_price_test.py . [ 14%]
23 tests/integration/items_test.py .. [ 23%]
24 tests/integration/test_api_service.py ... [ 38%]
25 tests/integration/test_help.py . [ 42%]
26 tests/integration/test_notify.py .. [ 52%]
27 tests/integration/test_price_diagram.py . [ 57%]
28 tests/integration/test_product_service.py .. [ 66%]
29 tests/integration/test_start.py . [ 71%]
30 tests/integration/test_stop_tracking.py . [ 76%]
31 tests/integration/test_support.py . [ 80%]
32 tests/integration/test_user_service.py .... [100%]
33

```

✓ Update README.md #1

Re-run all jobs ...

Summary

Jobs

✓ build

Run details

Usage

Workflow file

build

succeeded 6 hours ago in 1m 40s

Beta

Give feedback

Search logs

Refresh

Settings

Step	Duration
> ✓ Set up job	1s
> ✓ Run actions/checkout@v2	1s
> ✓ Set up Python 3.10	0s
> ✓ Install dependencies	31s
> ✓ Run integration tests	1m 3s
> ✓ Post Set up Python 3.10	0s
> ✓ Post Run actions/checkout@v2	0s
> ✓ Complete job	0s

Настройка CI

Workflow file for this run
.github/workflows/main.yml at 6123827

```
1 name: Python application test
2
3 on: [ push, pull_request ]
4
5 jobs:
6   build:
7
8     runs-on: ubuntu-latest
9
10    steps:
11      - uses: actions/checkout@v2
12      - name: Set up Python 3.10
13        uses: actions/setup-python@v3
14        with:
15          python-version: '3.10'
16      - name: Install dependencies
17        run: |
18          python -m pip install --upgrade pip
19          pip install -r requirements.txt
20      - name: Run integration tests
21        env:
22          TOKEN: ${ secrets.TOKEN }
23          bd_pass: ${ secrets.bd_pass }
24          test_bd_pass: ${ secrets.test_bd_pass }
25          api_id: ${ secrets.api_id }
26          api_hash: ${ secrets.api_hash }
27          session_str: ${ secrets.session_str }
28          api_key: ${ secrets.api_key }
29        run: |
30          pytest ./tests/integration
```

Описание процедуры расширения тестового набора на примере добавления новой функциональной части (или модуля)

Процедура расширения тестового набора при добавлении новой функциональной части в интеграционном тестировании включает несколько шагов для обеспечения полного покрытия нового модуля тестами.

1. Анализ требований: В первую очередь необходимо изучить требования к новой функциональной части и понять, как она взаимодействует с уже существующими модулями системы.
2. Разработка тестовых сценариев: На основе анализа требований разрабатываются новые тестовые сценарии, которые покрывают все возможные варианты использования новой функциональности.
3. Интеграция тестов: Новые тестовые сценарии интегрируются в общий набор тестов для системы, чтобы обеспечить целостное покрытие всех модулей.
4. Запуск и отладка тестов: После интеграции новых тестов их необходимо запустить и протестировать на корректность работы. В случае обнаружения ошибок производится их отладка.

5. Автоматизация тестов: Для повторного использования тестовых сценариев и обеспечения их стабильности рекомендуется автоматизировать тесты, чтобы упростить процесс тестирования при дальнейших изменениях в системе.