КУРСОВАЯ РАБОТА

по теме

Телеграм бот “Агрегатор цен на wildberries”

по дисциплине «Технологии разработки качественного программного обеспечения»

|  |  |
| --- | --- |
| Выполнили  студенты гр. 5130940/00105 | Квасков С.С.  Ганенко Л.Н.  Степанов Г.Ж. |
| Преподаватель | Маслаков А.П. |

# Юнит-Тесты

**1. Описание выполненной работы, использованных инструментах,**

**применённых техниках тест-дизайна.**

Для проведения unit-тестирования была использована библиотека pytest в связке c библиотекой unittest языка программирования python, которые предоставляют такую функциональность как удобный рекурсивный запуск тестовых сценариев, создание и настройка моков, подсчёт покрытия unit-тестами проекта, и так далее.

Запуск тестов был налажен и автоматизирован при помощи библиотеки python doit, благодаря чему запускать unit-тесты на весь проект можно одной консольной командой

**Классы эквивалентности**

Смысл этого подхода заключается в выборе значений, представляющих различные классы тестовых данных, чтобы мы могли проверить требования к продукту

пример:

./src/provider/database/unit\_test.py/304 (тесты про list)

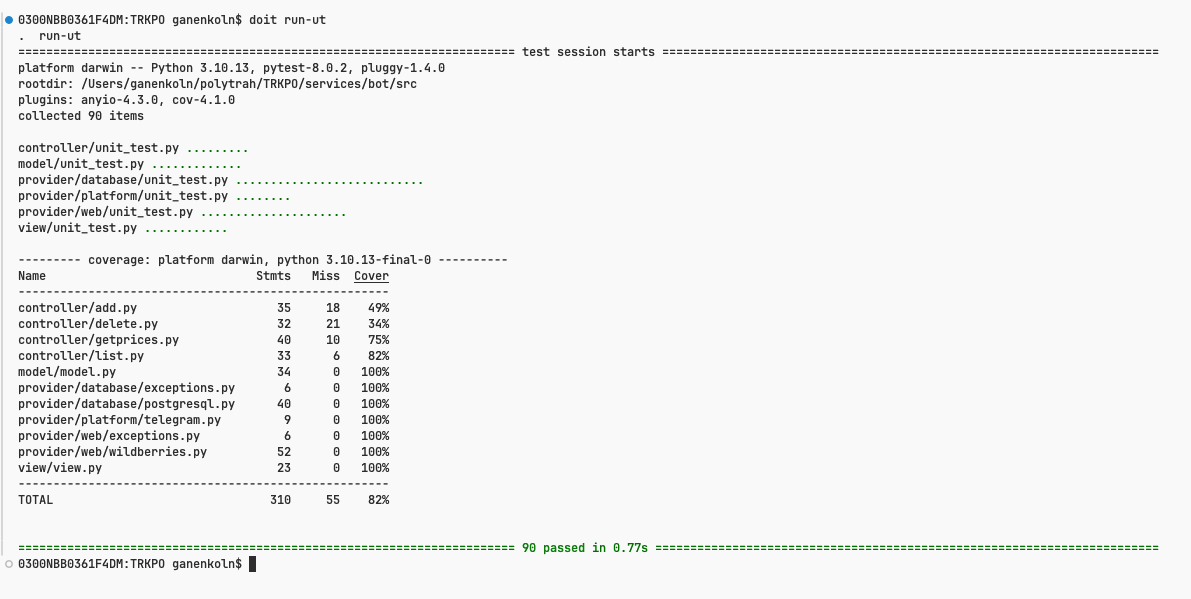
**Граничные условия**

Это техника проверки поведения продукта на крайних (граничных) значениях входных данных

пример:

./src/controller/unit\_test.py/227 (тесты про getprices)

**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.**

Для добавления тестов на вновь написанный код необходимо: создать unit\_test.py файл в новой директории, в этом файле объявить стандартную структуру unit-тестов для pytest/unittest, написать в корне проекта команду `doit run-ut`, которая автоматически подхватит вновь созданный тестовый файл. В случае, если новый код добавляется в директорию, в которой уже присутствует unit\_test.py файл, то необходимо просто по примеру добавить новые тест-кейсы.

# Интеграционные тесты

**Отчёт о выполненной работе, использованных инструментах,**

**применённых техниках тест-дизайна**

Смотря на общую схему приложения, было принято решение тестировать интеграцию PYTHON-BOT и базы данных POSTGRES.

Для начала приведём сравнение общих схем приложения и тестового стенда.

|  |  |
| --- | --- |
| Упрощённая схема приложения | Упрощённая схема тестового стенда |
|  |  |

Как мы видим, чтобы сконцентрироваться на тестировании интеграции PYTHON-BOT и базы данных POSTGRES, в отличии от приложения, в тестовом стенде «мокнуты» WILDBERRIES-API и TELEGRAM (клиент и платформа).

WILDBERRIES-API «мокнуто» в целях повышения скорости и стабильности интеграционного тестирования (отвечает долго, иногда во время интеграционного тестирования может быть недоступно из-за проблем на стороне WILDBERRIES)  
  
TELEGRAM клиент и платформа «мокнуты» (заменены кастомным простым HTTP-клиентом) для увеличения стабильности и простоты интеграционных тестов (нас не касаются возможные возникающие проблемы на стороне TELEGRAM, плюс нет необходимости решать проблемы аутентификации)

В ходе выполнения работы был написан 21 интеграционный тест. Тестирование производилось по методу чёрного ящика, то есть использовалось API PYTHON-BOT, тестировалось поведение сервиса согласно спецификации (соответсвующие ссылки указаны для каждого тест-кейса). Для тестирования использовалась библиотека python pytest, для автоматизации запуска тестов использовалась библиотека python doit, запуск тестов триггерится автоматически при создании пулл-реквеста или создания нового коммита, но интеграционные тесты можно запускать локально посредством таск-раннера doit.

**Тест-план со словесным описанием тестовых сценариев, которые**

**планируется реализовать в интеграционном тестировании**

Зелёным цветом выделены позитивные тест-кейсы (sunny cases)

Красным цветом выделены негативные тест-кейсы (rainy cases)

\*Валидным считается товар, URL которого начинается на «https://www.wb.ru»

\*\*Мок WILDBERRIES-API (при условии добавления валидного\*) товара всегда отдаёт название товара равное «mock\_title», название продавца равное «mock\_vendor», цену товара равную «999»

\*\*\*Мок WILDBERRIES-API (при условии добавления товара с title = «wildberries\_exception») инициирует эмуляцию исключения на стороне WILDBERRIES-API

Сценарии интеграционного тестирования

1. **Добавление нового товара**
   1. **Добавление валидного\* товара**

Описание: обычное добавление валидного\* товара на отслеживание

Спецификация: **F\_Add\_cmd\_2** + **F\_Add\_cmd\_7** + **F\_Add\_cmd\_9**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта (валидный\*) * ID пользователя, добавляющего продукт |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот сообщает об успешном добавлении товара, и прилагает URL добавленного товара, а так же ID пользователя для которого товар был добавлен |
| 5 | Проверка что после получение успешного HTTP-ответа:   * товар действительно был добавлен в базу данных и содержит корректные URL товара и ID пользователя (те, которые мы задавали в шаге 3). * остальные поля согласно \*\* |
| 6 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Добавление уже существующего товара тем же пользователем**

Описание: один и тот же пользователь добавляет два раза один и тот же товар (ошибка дубликата)

Спецификация**: F\_Add\_cmd\_2** + **F\_Add\_cmd\_8** + **F\_Add\_cmd\_10**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка **первого** HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://www.wb.ru * ID пользователя, добавляющего продукт = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот сообщает об успешном добавлении товара, и прилагает URL добавленного товара, а так же ID пользователя для которого товар был добавлен |
| 5 | Проверка что после получение успешного HTTP-ответа:   * товар действительно был добавлен в базу данных и содержит корректные URL товара и ID пользователя (те, которые мы задавали в шаге 3). * остальные поля согласно \*\* |
| 6 | Отправка **второго** HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://www.wb.ru * ID пользователя, добавляющего продукт = 55555 |
| 7 | На запрос в шаге 6 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 400 BAD REQUEST * JSON-тело ответа: бот сообщает об ошибке добавления дубликата товара «already exists», и прилагает URL недобавленного товара, а так же ID пользователя для которого была произведена попытка добавления |
| 8 | Проверка что после получения неудачного HTTP-ответа дубликат товара не был добавлен в базу данных |
| 9 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Добавление уже существующего товара другим пользователем**

Описание: два пользователя добавляют товар с одинаковым URL

Спецификация**: F\_DopReq\_2** + **F\_Add\_cmd\_2** + **F\_Add\_cmd\_7** + **F\_Add\_cmd\_9**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка **первого** HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://www.wb.ru * ID пользователя, добавляющего продукт = 11111 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот сообщает об успешном добавлении товара, и прилагает URL добавленного товара, а так же ID пользователя для которого товар был добавлен |
| 5 | Проверка что после получение успешного HTTP-ответа:   * Первый товар действительно был добавлен в базу данных и содержит корректные URL товара и ID пользователя (те, которые мы задавали в шаге 3). * остальные поля согласно \*\* |
| 6 | Отправка **второго** HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://www.wb.ru * ID пользователя, добавляющего продукт = 22222 |
| 7 | На запрос в шаге 6 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот сообщает об успешном добавлении товара, и прилагает URL добавленного товара, а так же ID пользователя для которого товар был добавлен |
| 8 | Проверка что после получение успешного HTTP-ответа:   * Второй товар действительно был добавлен в базу данных и содержит корректные URL товара и ID пользователя (те, которые мы задавали в шаге 7). * остальные поля согласно \*\* |
| 9 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Добавление товара с URL не WILDBERRIES**

Описание: пользователь пытается добавить товар, URL которого не является URL WILDBERRIES

Спецификация**: F\_Add\_cmd\_2** + **F\_Add\_cmd\_4** + **F\_Add\_cmd\_10**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://vk.com (не WILDBERRIES) * ID пользователя, добавляющего продукт = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 400 BAD REQUEST * JSON-тело ответа: бот сообщает об ошибке добавления товара «invalid URL», и прилагает URL недобавленного товара, а так же ID пользователя для которого была произведена попытка добавления |
| 5 | Проверка что после получения неудачного HTTP-ответа товар не был добавлен в базу данных |
| 6 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Добавление товара с несуществующим URL**

Описание: пользователь пытается добавить товар, URL которого не ведёт ни на какой ресурс (несуществующий URL)

Спецификация**: F\_Add\_cmd\_2** + **F\_Add\_cmd\_3** + **F\_Add\_cmd\_10**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://some\_unknown\_url\_i\_just\_imagine.com (не существует) * ID пользователя, добавляющего продукт = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 400 BAD REQUEST * JSON-тело ответа: бот сообщает об ошибке добавления товара «unexisting URL», и прилагает URL недобавленного товара, а так же ID пользователя для которого была произведена попытка добавления |
| 5 | Проверка что после получения неудачного HTTP-ответа товар не был добавлен в базу данных |
| 6 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Добавление товара без базы данных**

Описание: пользователь пытается добавить валидный\* товар, но во время добавления оказывается что база данных недоступна

Спецификация**: F\_Add\_cmd\_2** + **F\_Add\_cmd\_11** + **F\_Add\_cmd\_10**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск только контейнера с PYTHON-BOT |
| 2 | Проверка отсутствия возможности подключиться к базе данных |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – добавление товара) * URL добавляемого продукта = https://www.wb.ru * ID пользователя, добавляющего продукт = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 500 INTERNAL SERVER ERROR * JSON-тело ответа: бот сообщает об ошибке добавления товара «database error», и прилагает URL недобавленного товара, а так же ID пользователя для которого была произведена попытка добавления |
| 5 | Остановка контейнера с PYTHON-BOT |

1. **Инициация парсинга цен на товары**
   1. **Парсинг товаров при пустой базе данных**

Описание: пользователь пытается получить список цен на товары, при том, что не было добавлено ни одного товара

Спецификация**: F\_GetPrices\_cmd\_1** + **F\_GetPrices\_cmd\_2** + **F\_GetPrices\_cmd\_3**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – парсинг цен) * ID пользователя, получающего список цен = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает об ошибке получения цен на товары «not found», и прилагает ID пользователя для которого была произведена попытка получения цен |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Парсинг товаров при НЕ пустой базе данных для текущего пользователя**

Описание: пользователь пытается получить список цен на товары, товары для данного пользователя есть в базе данных

Спецификация**: F\_GetPrices\_cmd\_1** + **F\_GetPrices\_cmd\_4**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление 3-х валидных\* товаров в базу данных для пользователя с ID = 55555 |
| 3 | Проверка что в базу действительно было добавлено 3 товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – парсинг цен) * ID пользователя, получающего список цен = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот возвращает сообщение с 3-мя товарами (которые были добавлены в шаге 2), и ценами для них\*\* |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Парсинг товаров при НЕ пустой базе данных для НЕ текущего пользователя**

Описание: пользователь пытается получить список цен на товары, товары есть в базе данных, но они принадлежат другому пользователю, для текущего пользователя нет товаров

Спецификация**: F\_GetPrices\_cmd\_1** + **F\_GetPrices\_cmd\_2** + **F\_GetPrices\_cmd\_3** + **F\_DopReq\_2**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление 3-х валидных\* товаров в базу данных для пользователей с ID = 22222, ID = 33333, ID = 44444 |
| 3 | Проверка что в базу действительно было добавлено 3 товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – парсинг цен) * ID пользователя, получающего список цен = 11111 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает ошибке получения цен на товары «not found», и прилагает ID пользователя для которого была произведена попытка получения цен |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Парсинг товаров при ошибке от wildberries**

Описание: пользователь пытается получить список цен на товары, но при обработке одного из товаров WILDBERRIES-API\*\*\* падает с ошибкой

Спецификация**: F\_GetPrices\_cmd\_1** + **F\_GetPrices\_cmd\_3**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление 3-х товаров в базу данных для пользователя с ID = 55555 (два валидных\* товара, и один вызовет исключение\*\*\*) |
| 3 | Проверка что в базу действительно было добавлено 3 товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – парсинг цен) * ID пользователя, получающего список цен = 11111 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 500 INTERNAL SERVER ERROR * JSON-тело ответа: бот сообщает ошибке получения цен на товары «unexpected exception on wildberries side», и прилагает ID пользователя для которого была произведена попытка получения цен |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Парсинг товаров без базы данных**

Описание: пользователь пытается получить список цен на товары, но во время получения списка оказывается что база данных недоступна

Спецификация**: F\_GetPrices\_cmd\_1** + **F\_GetPrices\_cmd\_5** + **F\_GetPrices\_cmd\_3**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск только контейнера с PYTHON-BOT |
| 2 | Проверка отсутствия возможности подключиться к базе данных |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – парсинг цен) * ID пользователя, добавляющего продукт = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 500 INTERNAL SERVER ERROR * JSON-тело ответа: бот сообщает ошибке получения цен на товары «database error», и прилагает ID пользователя для которого была произведена попытка получения цен |
| 5 | Остановка контейнера с PYTHON-BOT |

1. **Удаление товара**
   1. **Удаление при пустой базе данных**

Описание: пользователь пытается удалить добавленный товар, при том, что не было добавлено ни одного товара

Спецификация**: F\_Delete\_cmd\_2** + **F\_Delete\_cmd\_3** + **F\_Delete\_cmd\_4**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – удаление товара) * URL удаляемого товара = https://www.wb.ru/1 * ID пользователя, удаляющего товар = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает об ошибке удаления товара «not found», и прилагает URL товара который пытались удалить, а так же ID пользователя который пытался удалить товар |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Удаление существующего товара пользователем-владельцем**

Описание: пользователь пытается получить список цен на товары, товары для данного пользователя есть в базе данных

Спецификация**: F\_Delete\_cmd\_2** + **F\_Delete\_cmd\_5**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление одного валидного\* товара c URL = https://www.wb.ru/1 в базу данных для пользователя с ID = 55555 |
| 3 | Проверка что в базу действительно было добавлен один товар |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – удаление товара) * URL удаляемого товара = https://www.wb.ru/1 * ID пользователя, удаляющего товар = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот сообщает об успешном удалении товара и прилагает URL удалённого товара, а так же ID пользователя который удалял товар |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Удаление существующего товара НЕ пользователем-владельцем**

Описание: в базе данных есть товары, но они не принадлежат текущему пользователю, и при попытке удалить какой-либо товар, текущий пользователь получит сообщение об ошибке

Спецификация**: F\_Delete\_cmd\_2** + **F\_Delete\_cmd\_3** + **F\_Delete\_cmd\_4** + **F\_DopReq\_2**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление одного валидного\* товара c URL = https://www.wb.ru/1 в базу данных для пользователя с ID = 22222 |
| 3 | Проверка что в базу действительно было добавлен один товар |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – удаление товара) * URL удаляемого товара = https://www.wb.ru/1 * ID пользователя, удаляющего товар = 11111 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает об ошибке удаления товара «not found», и прилагает URL товара который пытались удалить, а так же ID пользователя который пытался удалить товар |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Удаление несуществующего товара**

Описание: в базе данных есть товары, принадлежащие текущему пользователю, но пользователь пытается удалить товары, которых нет в списке

Спецификация**: F\_Delete\_cmd\_2** + **F\_Delete\_cmd\_3** + **F\_Delete\_cmd\_4**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление трёх валидных\* товаров в базу данных для пользователя с ID = 55555   * URL\_1 = https://www.wb.ru/1 * URL\_2 = https://www.wb.ru/2 * URL\_3 = https://www.wb.ru/3 |
| 3 | Проверка что в базу действительно было добавлено три товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – удаление товара) * URL удаляемого товара = https://www.wb.ru/4 * ID пользователя, удаляющего товар = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает об ошибке удаления товара «not found», и прилагает URL товара который пытались удалить, а так же ID пользователя который пытался удалить товар |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Удаление товара без базы данных**

Описание: пользователь пытается удалить товар, но во время удаления оказывается, что база данных недоступна

Спецификация**: F\_Delete\_cmd\_2** + **F\_Delete\_cmd\_6** + **F\_Delete\_cmd\_4**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск только контейнера с PYTHON-BOT |
| 2 | Проверка отсутствия возможности подключиться к базе данных |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – удаление товара) * URL удаляемого товара = https://www.wb.ru/1 * ID пользователя, удаляющего товар = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 500 INTERNAL SERVER ERROR * JSON-тело ответа: бот сообщает ошибке удаления товара «database error», прилагает URL товара который пытались удалить и ID пользователя для которого была произведена попытка удалить товар |
| 5 | Остановка контейнера с PYTHON-BOT |

1. **Получение списка товаров**
   1. **При пустой базе данных**

Описание: пользователь пытается получить список товаров, при том, что не было добавлено ни одного товара

Спецификация**: F\_List\_cmd\_2**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Проверка пустоты базы данных перед тестом |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – получение списка товаров) * ID пользователя, получающего список товаров = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает об ошибке получения списка товаров «not found», и прилагает ID пользователя для которого была произведена попытка получения списка |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **При НЕ пустой базе данных для текущего пользователя**

Описание: пользователь пытается получить список товаров, товары для данного пользователя есть в базе данных

Спецификация**: F\_List\_cmd\_1**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление 3-х валидных\* товаров в базу данных для пользователя с ID = 55555 |
| 3 | Проверка что в базу действительно было добавлено 3 товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – получение списка товаров) * ID пользователя, получающего список товаров = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: успешный 200 OK * JSON-тело ответа: бот возвращает сообщение с 3-мя товарами (которые были добавлены в шаге 2 |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **При НЕ пустой базе данных для НЕ текущего пользователя**

Описание: пользователь пытается получить список товаров, товары есть в базе данных, но они принадлежат другому пользователю, для текущего пользователя нет товаров

Спецификация**: F\_List\_cmd\_2** + **F\_DopReq\_2**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Добавление 3-х валидных\* товаров в базу данных для пользователей с ID = 22222, ID = 33333, ID = 44444 |
| 3 | Проверка что в базу действительно было добавлено 3 товара |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – получение списка товаров) * ID пользователя, получающего список товаров = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает ошибке получения списка товаров «not found», и прилагает ID пользователя для которого была произведена попытка получения списка |
| 5 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

* 1. **Получение списка товаров без базы данных**

Описание: пользователь пытается получить список товаров, но во время получения списка оказывается, что база данных недоступна

Спецификация**: F\_List\_cmd\_3** + **F\_List\_cmd\_2**

Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск только контейнера с PYTHON-BOT |
| 2 | Проверка отсутствия возможности подключиться к базе данных |
| 3 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – получение списка товаров) * ID пользователя, получающего список товаров = 55555 |
| 4 | На запрос в шаге 3 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 500 INTERNAL SERVER ERROR * JSON-тело ответа: бот сообщает ошибке получения списка товаров «database error», и прилагает ID пользователя для которого была произведена попытка получения списка |
| 5 | Остановка контейнера с PYTHON-BOT |

1. Инициация неизвестной команды
   1. **Инициация любой неизвестной команды**

Описание: пользователь пытается вызвать неизвестную команду в PYTHON-BOT

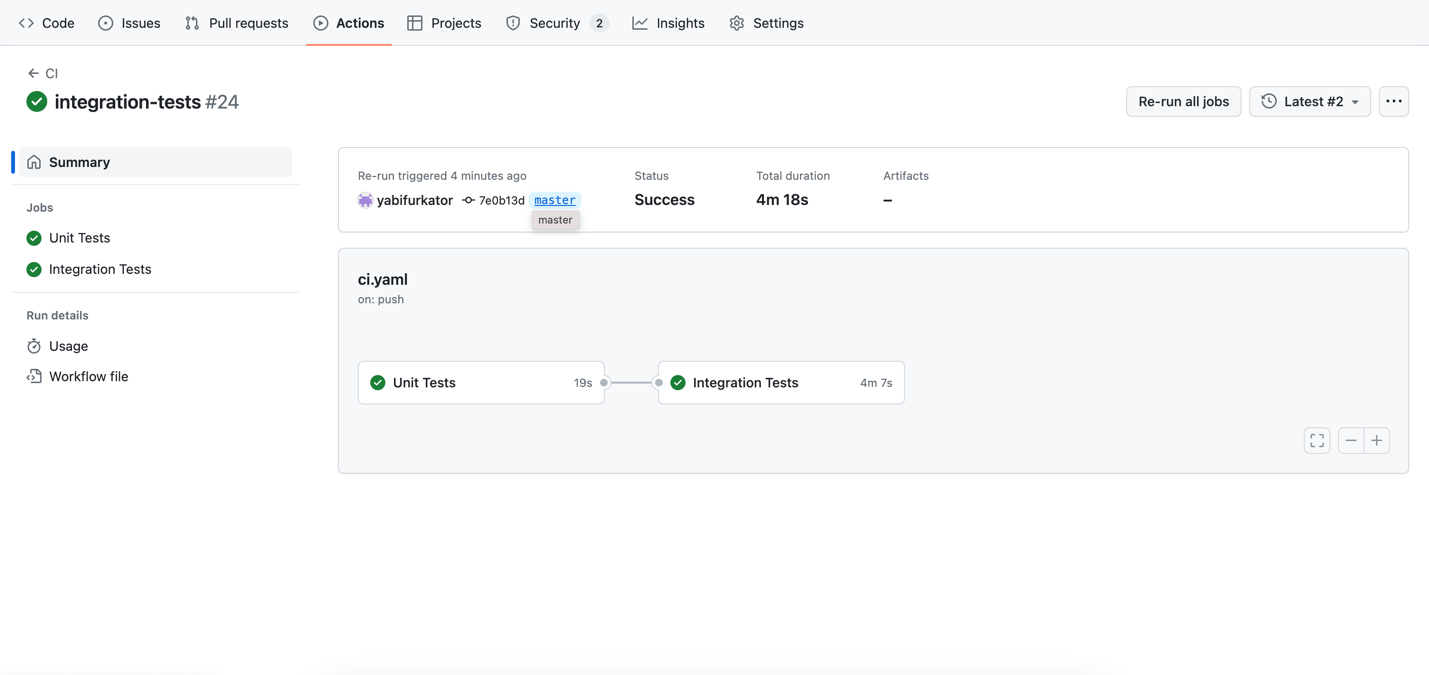
Спецификация**: F\_DopReq\_8**

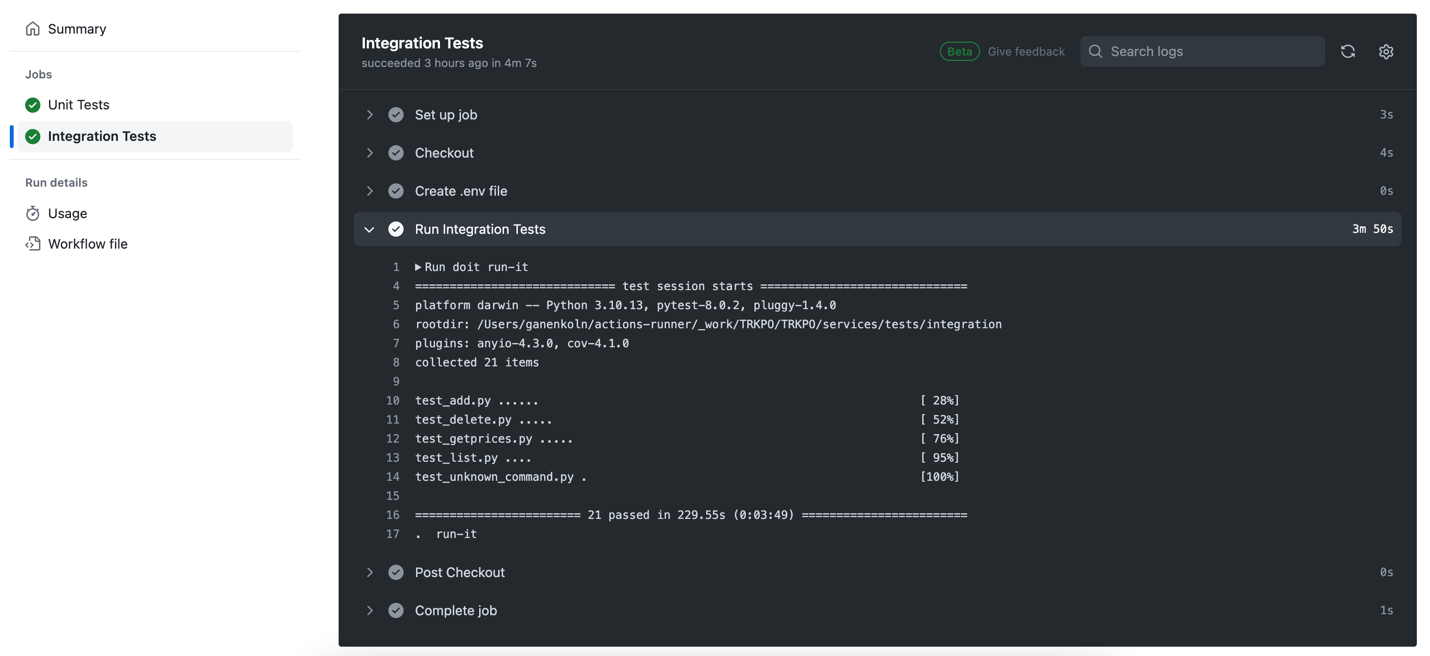
Сценарий:

|  |  |
| --- | --- |
| шаг | описание |
| 1 | Запуск контейнеров с PYTHON-BOT и POSTGRES |
| 2 | Отправка HTTP-запроса при помощи кастомного HTTP-клиента. Запрос содержит следующие данные:   * Вызываемую команду (эндпоинт – любой случайный не существующий) |
| 3 | На запрос в шаге 2 получается HTTP-ответ, в котором проверяется:   * Код ответа: негативный 404 NOT FOUND * JSON-тело ответа: бот сообщает о том, что команда не найдена |
| 4 | Остановка контейнеров с PYTHON-BOT и POSTGRES |

**Отчёт о прохождении тестов с результатами на сервере непрерывной**

**интеграции.**



Логи стейджа интеграционных тестов:  


Логи локального запуска:



**Описание процедуры расширения тестового набора на примере**

**добавления новой функциональной части (или модуля)**

1. Так как тестирование проходит по методу чёрного ящика, то для тестирования новой функциональности, эта новая функциональность ложна быть доступна извне (посредством какого-либо API)
2. В папку services/tests/integration необходимо добавить новый питон-файл, называющийся на test\_\*.py
3. Написанные в нём тесты посредством pytest должны взаимодействовать с API сервиса, проверяя ответы и состояние базы данных