

Министерство образования и науки
федеральное государственное автономное образовательное учреждение
высшего образования

«Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

Отчет по дисциплине: **«Тестирование программного обеспечения»**

Лабораторная работа 2

Выполнила: Полтавец
Елена Андреевна

Группа: К3322

Проверил: Кочубеев Николай
Сергеевич

Санкт-Петербург

2024

Цель: нужно научиться разрабатывать и применять интеграционные тесты для проверки взаимодействия между компонентами в существующем проекте.

Задачи:

- Выбрать репозиторий для тестирования с GitHub;
- Провести анализ взаимодействий;
- Написать интеграционные тесты;
- Подготовить отчет о проделанной работе.

Ход работы

1. Выбор репозитория с GitHub

Для выполнения лабораторной работы по разработке интеграционных тестов был выбран проект [wttr.in](https://github.com/wttr-in/wttr.in), который представляет собой консольный сервис для получения прогноза погоды. Этот проект содержит несколько компонентов, которые взаимодействуют между собой, и предоставляет API для извлечения данных.

На рисунке 1 представлен пример использования данного проекта.

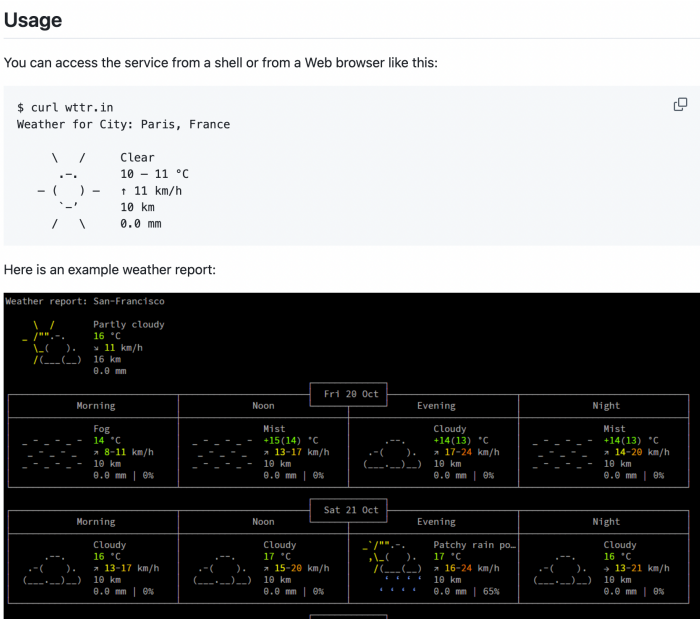


Рисунок 1 – Пример использования проекта

2. Анализ взаимодействий между модулями

wttr.in — это сервис, который предоставляет информацию о погоде в текстовом и графическом форматах. Он поддерживает запросы через HTTP, а также различные форматы вывода. Основные компоненты проекта:

- Серверная часть (на основе Python и Go) для обработки запросов на получение данных о погоде;
- Клиенты (curl, браузеры), которые делают запросы к серверу;
- База данных или сторонние API для получения актуальных данных о погоде.

Ключевые точки интеграции:

- Получение данных о погоде (API делает запросы к сторонним погодным сервисам, например, WorldWeatherOnline);
- Форматирование и вывод данных (после получения данных о погоде происходит форматирование и вывод в ответ на запрос_.

Критические части системы:

- Логика обработки запросов в серверной части (обработка URL, работа с параметрами);
- Взаимодействие с внешними API для получения данных о погоде.

3. Написание интеграционных тестов

Для проверки интеграций были написаны тесты на Python с использованием библиотеки unittest и requests для имитации HTTP-запросов:

- Проверка получения данных о погоде для конкретного города ;

Выполняется HTTP GET запрос к API wttr.in для получения информации о погоде в Лондоне с указанием формата ответа в JSON (*format=json*).

Проверяется, что статус ответа равен 200 (успешное выполнение запроса). Ожидается, что в полученных данных присутствует ключ *current_condition*, который содержит информацию о текущих погодных условиях (Рисунок 2)

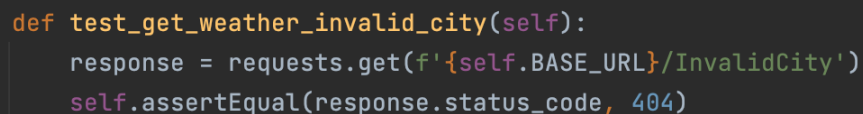


```
1 import unittest
2 import requests
3
4 class TestWeatherAPIIntegration(unittest.TestCase):
5
6     BASE_URL = 'http://wttr.in'
7
8     def test_get_weather_for_city(self):
9         response = requests.get(f'{self.BASE_URL}/London?format=json')
10        self.assertEqual(response.status_code, 200)
11        data = response.json()
12        self.assertIn('current_condition', data)
```

Рисунок 2 – Проверка получения данных о погоде для города

- Проверка обработки некорректного запроса (город не найден) ✓;

Выполняется HTTP GET запрос к API wttr.in с некорректным названием города (например, *InvalidCity*). Проверяется, что статус ответа равен 404 (не найдено), что указывает на корректное поведение API в случае, если запрашиваемый город отсутствует (Рисунок 3).



```
def test_get_weather_invalid_city(self):
    response = requests.get(f'{self.BASE_URL}/InvalidCity')
    self.assertEqual(response.status_code, 404)
```


Рисунок 3 – Проверка обработки некорректного запроса

- Проверка правильного формата ответа API в JSON ✓;

Выполняется HTTP GET запрос к API для получения погоды в Москве с указанием формата JSON. Проверяется, что статус ответа равен 200. Ожидается, что заголовок ответа Content-Type равен *application/json*, что подтверждает, что данные были возвращены в нужном формате (Рисунок 4).

```
def test_get_weather_format_json(self):
    response = requests.get(f'{self.BASE_URL}/Moscow?format=json')
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.headers['Content-Type'], 'application/json')
```

Рисунок 4 – Проверка правильности формата ответа API

- Проверка, что API возвращает данные для нескольких городов ;

Выполняется HTTP GET запрос к API wttr.in для двух городов: Лондон и Нью-Йорк. Проверяется, что статус ответа для каждого города равен 200 (Рисунок 5).

```
def test_get_weather_multiple_cities(self):
    cities = ["London", "New+York"]
    responses = [requests.get(f'{self.BASE_URL}/{city}?format=json') for city in cities]
    for response in responses:
        self.assertEqual(response.status_code, 200)
```

Рисунок 5 – Проверка правильности возвращения данных для нескольких городов

- Проверка получения текущего времени для города .

Выполняется HTTP GET запрос к API для получения данных о погоде в Лондоне. Проверяется, что статус ответа равен 200. Проверяется что в ответе присутствует ключ *observation_time*, который содержит текущее время (Рисунок 6).

```
def test_get_current_time_for_city(self):
    response = requests.get(f'{self.BASE_URL}/London?format=json')
    self.assertEqual(response.status_code, 200)
    data = response.json()

    self.assertIn('current_condition', data)
    self.assertIn('observation_time', data['current_condition'][0])
```

Рисунок 5 – Проверка правильности получение текущего времени для города

4. Результаты тестирования

Все тесты прошли успешно. API корректно обрабатывает запросы и возвращает данные в нужном формате. Соответственно интеграция между модулями работает без очевидных недочётов. Однако при недоступности сервиса API или при ошибочных вводах возможны проблемы, которые требуют дополнительной обработки.

Вывод: был проведен анализ взаимодействий в проекте, написаны интеграционные тесты, на основании этого подготовлен отчет о проделанной работе.

Ссылка на репозиторий с тестами: https://github.com/Malenago/Testing_PO.