УДК 004.415

Сіклічук А. С., здобувачка 3 курсу спеціальності 122 Комп'ютерні науки,
Січко Т. В., канд. техн. наук, доцент, доцент кафедри інформаційних і прикладних технологій

ТЕСТУВАННЯ API ЯК ІНСТРУМЕНТ ВИЯВЛЕННЯ ПОМИЛОК У РОБОТІ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ

Донецький національний університет імені Василя Стуса, м. Вінниця

Вступ. У сучасному програмуванні й розробці програмного забезпечення важливим етапом є не тільки створення функціональності, але й гарантування її надійності та стабільності через проведення належного тестування. Зокрема, тестування API (Application Programming Interface) є важливим складником у забезпеченні правильної взаємодії між різними компонентами програм та сервісів. Найбільш поширеним інструментом для розв'язання цієї задачі підходить автоматизоване тестування, яке за допомогою різних мов програмування, фреймворків та бібліотек дає змогу зробити цей процес набагато швидшим та більш організованим.

Актуальність. API є невід'ємною частиною клієнт-серверної взаємодії. Він являє собою набір правил та інструкцій, які визначають, як програми або компоненти програмного забезпечення можуть взаємодіяти один з одним. Цей термін використовується для опису інтерфейсу, який дає змогу взаємодіяти з вебсерверами та отримувати або передавати дані через мережу Інтернет. Для цього використовується протокол HTTP, що забезпечує доступ до різних послуг чи ресурсів через мережу. Саме тому тестування коректної поведінки інтерфейсів є надважливим в сучасному циклі тестування вебдодатків.

Важливість тестування API можна продемонструвати на прикладі роботи з відкритою API-документацією. Ця документація знаходиться на сервісі Swagger [1], на якому безпосередньо можна здійснювати саме тестування. Вона має назву PetStore і являє собою набір інформації, яка описує функціональність та використання самого інтерфейсу. Загалом це зразок додатка, що можна використовувати для тестування. Додаток імітує онлайн-зоомагазин, і користувачі можуть додавати та отримувати інформацію про своїх вихованців. Уявимо, що у нас є версія додатка Pet Store, але з графічним інтерфейсом. Відповідно на сайті має відображатися інформація про нашого улюбленця, яку ми заздалегідь внесли в особистому кабінеті. Але ця інформація відсутня або відображається частково. Є два варіанти того, що може слугувати причиною: помилка на клієнті, тобто front-end не відображає правильно дані, або помилка на сервері, тобто back-end має певні проблеми в роботі логіки і відправляє на клієнт неправильні дані. Тут і стає в нагоді API-документація. За її допомогою можна створити тест, що перевірить,

чи коректно відпрацьовує логіка додатка. Реалізовувати його ми будемо за допомогою мови програмування Python та фреймворка для тестування Pytest. Це все можна було б реалізувати і за допомогою самого сервісу Swagger, але написані власноруч тести зручніше зберігати та оптимізувати, якщо з часом логіка додатка буде змінюватися. Запуск у вигляді так званих тест-ранів (набір тестів) і обробка помилок буде здійснюватися швидше, і ми будемо отримувати чіткі та зрозумілі результати. Також за такого підходу легше зберігати тестові дані, які будуть використовуватися під час кожного запуску програми.

Перед тим, як написати код для перевірки правильності отримання даних про домашнього улюбленця, потрібно його створити. Розглянемо наявний для цього запит в API-документації від СВАГЕР (рис. 1).

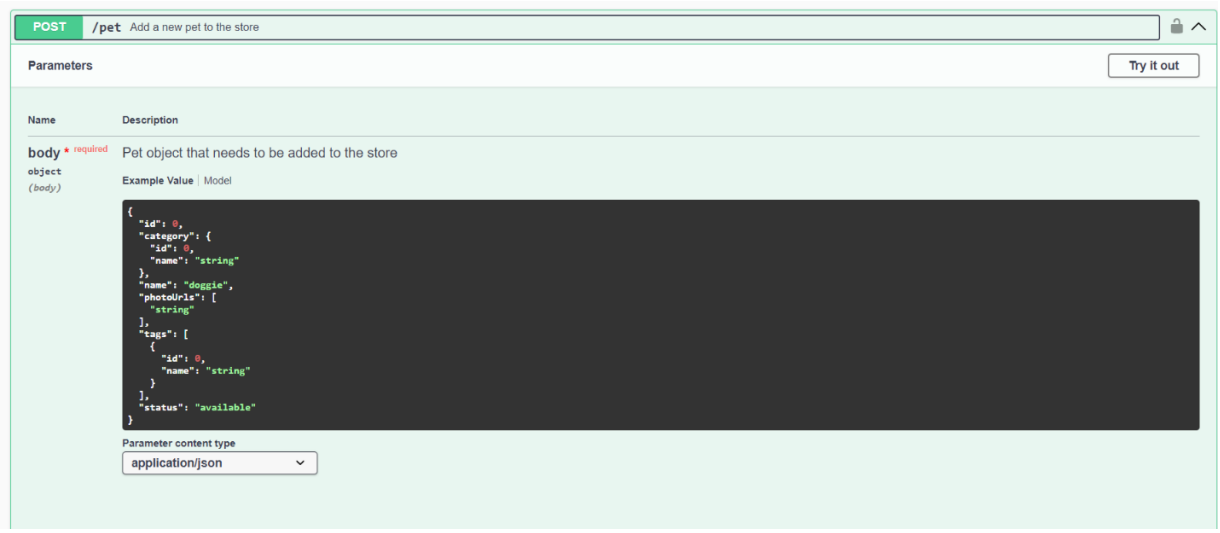

Рисунок 1. Структура запиту на створення інформації про тварину

Структура запиту складається з методу POST, URL та тіла запиту. В тілі у нас передається на сервер вся інформація в форматі JSON.

Для отримання даних про улюбленця використовується запит, що складається з методу GET та URL (рис. 2). Пошук тваринки відбувається за ID, який попередньо призначається під час створення.

Рисунок 2. Структура запиту на отримання даних про тварину

Перед тим, як писати тести, потрібно зробити set-up та підготувати тестові дані. Налаштування середовища можна зробити за допомогою вбудованого в Python pip або за допомогою Poetry [3]. Це інструменти для управління залежностями, як-от встановлені бібліотеки, пакети та ін. Різниця лише в тому, що pip зберігає інформацію про залежності, а poetry зберігає інформацію про сам проєкт загалом, і зберігає це в одному місці, що є дуже зручним.

У Pytest є так звані фікстури – функції, що дають змогу виконувати певні підготовчі дії перед тим, як запускати тести, і позначаються декоратором @pytest.fixture. У цьому проєкті в нас буде реалізовано дві фікстури: одна зчитує BASE URL та з'єднує його з нашим PATH (частинка URL, яка визначає конкретний ресурс або місце на вебсервері), а інші фікстура реалізує логування, що дає змогу переглянути вміст відповіді від сервера після надсилання запиту:

```
import logging
import pytest
from requests_toolbelt import sessions

URL = "https://petstore.swagger.io/v2/"

@pytest.fixture(scope="function")
def logger():
  logger = logging.getLogger("api")
   return logger

@pytest.fixture(scope="function")
def api_client():
   api_client = sessions.BaseUrlSession(base_url=URL)
   return api_client
Нехай в нас є тварина, інформація про яку зберігається у змінній post_body:
post_body = {
   "id": 1,
   "category": {
      "id": 0,
      "name": "category1"
   },
   "name": "Bob",
   "photoUrls": [
      "string"
   ],
   "tags": [
      {
         "id": 0,
         "name": "string"
      }
```

```
    ],
    "status": "available"
}
```

Структура тесту складається з надсилання запиту на інформацію про тварину, отримання статус-коду відповідно до того, чи була операція успішна, та валідацію отриманих даних з тими даними, які ми вказували під час створення тварини (змінна post_body).

```
def test_find_by_valid_id(api_client, logger):
    petID = 1
    response = api_client.get(url=f"pet/{petID}")
    logger.info(response.text)
    response_body = response.json()
    assert response.status_code == 200
    assert post_body['id'] == response_body['id']
```

Запускаємо тест та перевіряємо результати. Якщо логіка програми відпрацьовує коректно, то на виході сервер поверне нам статус-код 200 та інформацію про нашу тварину у форматі JSON, що відповідає тим даним, які ми вносили під час створення вихованця [рис. 3].

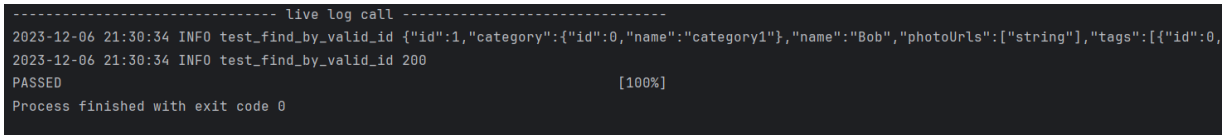

Рисунок 3. Результат виконання тесту

Отже, ми маємо коректні дані та статус-код 200, що значить правильне відпрацювання логіки програми. Тому якщо вважати, що ми маємо такий вебдодаток, який реалізовано для загального користування, і він має графічний користувацький інтерфейс, а інформація про нашу тваринку не відображається, хоча вона була попередньо занесена до програми, то проблему потрібно шукати саме на стороні front-end розробників.

Висновки. Ми визначили, що проблема у некоректному відображені даних зовсім не стосується логіки програми, сервер правильно обробляє та надсилає дані на клієнт, що полегшить роботу розробникам та збереже їх час.

Список використаних джерел

1. Swagger. URL: https://petstore.swagger.io/#/ (дата звернення: 04.12.2023).
2. Pytest. URL: https://docs.pytest.org/en/7.4.x//#/ (дата звернення: 04.12.2023).
3. Poetry. URL: https://python-poetry.org/ (дата звернення: 04.12.2023).