ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ СТРЕС-ТЕСТІВ НА ОСНОВІ OPENAPI МОВОЮ C#»**

Виконав: студент 4 курсу, групи ПД-41 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Прокопчук К.К.
(прізвище та ініціали)
Керівник Шевченко С.М.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність – 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
інженерії програмного забезпечення

Негоденко О.В.

“____” _____2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

_____Прокопчуку Кирилу Костянтиновичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для OpenAPI автоматичної генерації стрес-тестів на основі мовою C#»_____
Керівник роботи ____Шевченко С.М., к.п.н., доцент_____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “24” лютого 2023 року №26
2. Строк подання студентом роботи 1.06.2023 року
3. Вихідні дані до роботи:
Наукові джерела з питань тестування програмного продукту.
Технологія OpenAPI, мова програмування – C#.
Програмне забезпечення для автоматичної генерації стрес-тестів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно зробити)_____

1. Аналіз предметної області щодо методик існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків
2. Теоретичні основи для розробки програмного засобу, вибір та обґрунтування технологій та засобів розробки.
3. Розробка програмного забезпечення.
4. Тестування розробленого додатку

5. Перелік графічного матеріалу
   5.1. Аналіз аналогів.
   5.2. Вимоги до програмного забезпечення.
   5.3. Програмні засоби реалізації.
   5.4. Діаграма варіантів використання.
   5.5. Діаграма класів.
   5.6. Діаграма діяльності.
   5.7. Формат файлу конфігурації.
   5.8. Список команд для запуску додатку.
   5.9. Робота додатку з логуванням та запуском цільової програми стрес-тестування.

6. Дата видачі завдання 24.02.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури	24.02 – 12.03	
2	Опис предметної області	13.03 – 20.03	
3	Постановка задачі	21.03 – 28.03	
4	Програмна реалізація	29.03 – 30.04	
5	Тестування розробленого ПО	01.05 – 10.05	
6	Вступ, висновки, реферат	11.05 – 20.05	
7	Попередній захист роботи	22.05 – 26.05	
8	Перевірка на плагіат	27.05 – 31.05	
9	Здача роботи в деканат	01.06.	

Студент _____ Прокопчук К.К.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шевченко С.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 43 с., 11 рис., 1 табл., 21 джерело.

ТЕСТУВАННЯ, СТРЕС-ТЕСТ, ВЕБ-ЗАСТОСУНОК, АВТОМАТИЗАЦІЯ, C#, OPENAPI, .NET CORE, API, JSON

Об'єкт дослідження – процес створення стрес-тестів.

Предмет дослідження – програмне забезпечення для автоматичної генерації стрес-тестів.

Мета роботи – підтримка процесу створення стрес-тестів в автоматичному режимі за допомогою додатку мовою C# та використання OpenAPI.

Методи дослідження: системно-структурні методи, порівняльний аналіз, методи тестування програмного забезпечення.

Дане дослідження присвячене проблемі тестування програмного забезпечення, а саме, розробці застосунку для автоматичної генерації стресового тестування. Тестування на кожному етапі розробки програмного забезпечення відіграє вирішальну роль, що підтверджує актуальність та важливість даної теми. У роботі проаналізовані технології тестування веб-додатків, окреслені їх характеристики та сфери застосування. Обґрунтовано вибір інструментальних засобів автоматичної генерації стрес-тестів та представлено їх реалізацію у процесі розробки на основі стандарту OpenAPI мовою C#. Тестування розробленого додатку показало задовільні результати: додаток дозволяє спростити написання та зменшення потреб ресурсів для покриття стрес-тестами API веб-застосунку.

Фінальний продукт може бути корисним розробникам, тестувальникам, замовникам веб-додатків для перевірки системи на швидкість роботи без витрати великих зусиль для ручного створення усіх необхідних тестових сценаріїв, а також у процесі вивчення дисципліни «Якість програмного забезпечення та тестування» студентами спеціальності 121 Інженерія програмного забезпечення.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UI – (англ. User interface) Інтерфейс користувача

Продакшн – середовище, яке розгорнуто для реальних клієнтах

MVP -(англ. Minimum viable product) – мінімальна робоча версія застосунку

ООП – Об’єктно-орієнтоване програмування

IL -(англ. Intermidiate language) – посередня мова

CLR -(англ. common language runtime) – спільне середовище виконання

CLS – (англ. common language specification) – спільна мовна специфікація

JSON – (англ. JavaScript Object Notation) – формат файлу з використанням нотації об’єктів мови JavaScript

API – (англ. Application Programming Interface) – набір методів для взаємодії з різними компонентами.

ВСТУП

Тестування і забезпечення якості – невід'ємний етап у розробці будь-якого програмного забезпечення. Помилок можна допуститися на будь-якому етапі розробки програмного забезпечення, незважаючи на рівень складності завдання. Також кожна нова зміна старого коду буде потребувати більше часу для ручного тестування та перевірки того, що старий функціонал не змінився. З цієї точки зору написання автоматичних тестів під час розробки є дуже економічно вигідним, оскільки розробник в контексті їх написання буде витрачати набагато менше часу, аніж коли людина вперше бачить частину коду.

Наразі існує багато застосунків, які автоматично під час розробки показують помилки компіляції або навіть можливі помилки під час роботи застосунку. Але це не виключає логічних проблем чи проблем під навантаженням. У зв'язку зі зростанням кількості користувачів та обсягів даних, вимоги до надійності та продуктивності програмного забезпечення стають все вищими, а часу на реалізацію все менше, чим і визначається актуальність і важливість даного дослідження. Саме стрес-тестування дозволяє визначити межі можливостей системи, виявити слабкі місця та помилки, що можуть виникнути при високому навантаженні.

Метою дослідження є підтримка процесу створення стрес-тестів в автоматичному режимі за допомогою додатку мовою C# та використання OpenAPI.

Для досягнення цієї мети в роботі необхідно вирішити такі **завдання:**

1. Провести огляд та аналіз літературних джерел, методик існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків.

2. Проаналізувати програмні рішення, що використовуються на поточний момент для автоматичного тестування та/або автоматичного генерування тестових сценаріїв для автоматизації тестування веб-додатків.

3. Дослідити характеристики, переваги та недоліки існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків.

4. Провести огляд ІТ-засобів, які можуть бути використані при розробці програмного забезпечення для автоматичної генерації стрес-тестів для веб-додатків.

5. Розробити програмне забезпечення для автоматичної генерації стрес-тестів для веб-додатків.

6. Здійснити тестування розробленого додатку для перевірки його функціональності та продуктивності.

Виходячи з цього, **об'єктом** дослідження є процес створення стрес-тестів, **предметом** дослідження – програмне забезпечення для автоматичної генерації стрес-тестів.

Методи дослідження. Для вирішення вищезгаданих завдань у роботі використано наступні методи: системно-структурні методи, порівняльний аналіз, методи тестування програмного забезпечення. Для розробки програмного забезпечення використані такі технології та інструменти, як .NET, ASP.NET Core, xUnit, Swagger та бібліотеки для HTTP запитів та застосунок для стрес-тестування Grafana K6.

Наукова новизна одержаних результатів. Наукова новизна полягає у тому, що у розробленому додатку більшість процесів стрес-тестування здійснюється автоматично, що дозволяє зменшити ресурси для проведення тестування програмного забезпечення.

Практична значущість результатів полягає у створенні програмного продукту у вигляді консольної утиліти, яка забезпечить автоматичну генерацію стрес-тестів для API веб-застосунків на основі специфікацій API та у форматі OpenAPI, а також генерацію звітів з результатами тестування та оцінкою продуктивності розробленого програмного забезпечення. Фінальний продукт може бути корисним розробникам, тестувальникам, замовникам веб-додатків для перевірки системи на швидкість роботи без витрати великих зусиль для ручного створення усіх необхідних тестових сценаріїв, а також у процесі вивчення дисципліни «Якість програмного забезпечення та тестування» студентами спеціальності 121 Інженерія програмного забезпечення.

1. АНАЛІЗ СУЧАСНОГО СТАНУ У ТЕОРІЇ ТА ПРАКТИЦІ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

1.1 Огляд технологій тестування

Метод тестування - це загальний підхід до тестування програмного забезпечення, який може включати в себе різні методики та техніки тестування, а також інструменти та процеси. З кожним роком розробляються нові методи та методики тестування, які задовольняють окремі потреби окремих продуктів під час розробки. За останні 15 років було описано більше ніж 30 технік тільки регресійного тестування [1].

1.1.1 Ручне тестування

Ручне тестування — це метод тестування певного функціоналу (функція, сервіс, сторінка в UI, тощо) вручну, тобто повністю проходження шляху користувача для даного функціоналу з натисканням усіх можливих кнопок та з прагненням зламати систему. Це робиться для того, щоб перевірити: кінцевий користувач точно не отримає не протестований функціонал. Як і всі методи, він має свої недоліки та свої позитивні сторони. Негативні включають в себе:

1. Ненадійність: люди можуть допускати помилки в процесі тестування, що може призвести до пропуску деяких проблем.

2. Витрати: ручне тестування може бути витратним з точки зору ресурсів, оскільки потребує багато людських ресурсів та часу.

3. Обмеження: люди можуть бути обмежені у своїй здатності до тестування з точки зору складності тестування та навичок, що необхідні для проведення тестування.

4. Повторюваність: ручне тестування може бути менш повторюваним і надійним у порівнянні з автоматизованим тестуванням.

В свою чергу проводження ручного тестування тільки збільшить кількість затраченого часу на розробку конкретного функціоналу. З позитивних сторін можна виділити наступні:

1. Легко доступне: ручне тестування не потребує спеціальних знань і навичок, що дозволяє більшій кількості людей займатися цим процесом.

2. Гнучкість: здатність взаємодіяти з іншими людьми дозволяє використовувати ручне тестування для багатьох різних типів додатків та в різних сценаріях.

3. Здатність до інтуїтивного сприйняття: люди здатні швидко виявляти проблеми, що можуть бути несприятливими для користувача.

4. Всебічність: ручне тестування дозволяє тестерам проаналізувати додаток з різних точок зору та знайти проблеми, які автоматизоване тестування може пропустити.

1.1.2 Модульне тестування

Модульне тестування (англ. unit testing) — це процес тестування окремих модулів або фрагментів коду програмного продукту. Модулем може бути функція, метод класу, клас, бібліотека або набір класів, що працюють разом.

Модульне тестування дозволяє виявити помилки в роботі окремих фрагментів програмного коду, зменшити кількість помилок на більш високому рівні і підвищити якість програмного продукту. Цей вид тестування вимагає написання спеціального коду, який перевіряє правильність роботи окремих модулів. Зазвичай, для модульного тестування використовуються спеціальні фреймворки, такі як JUnit для Java, xUnit для .NET, PHPUnit для PHP та інші.

Основною перевагою модульного тестування є те, що помилки виявляються на самому початковому етапі розробки, коли виправити їх легше і дешевше. Також модульні тести можуть бути виконані швидко і повторювано, що дозволяє швидко виявляти, зокрема, проблеми в коді.

Результатом впровадження модульного тестування є бібліотека автоматичних модульних тестів, яка запускається спеціальними фреймворками і на виході дає звіт, кількість пройдених та провалених тестів, можливі причини, тощо.

1.1.3 Інтеграційне тестування

Інтеграційне тестування - це процес перевірки взаємодії між різними модулями (або компонентами) програмного забезпечення для забезпечення їх правильного інтегрування та співпраці в рамках одного функціонального блоку або системи. Інтеграційне тестування є проміжним етапом між модульним тестуванням та системним тестуванням.

Існує багато способів впровадження інтеграційного тестування: у вигляді бібліотеки тестів, як з модульним тестуванням, або більш складні системи, які для своєї роботи потребують засобів автоматизованого розгортання всього продукту, наряду з самими інтеграційними тестами, які будуть тестувати «живий» продукт.

1.1.4 End-to-end тестування

End-to-end тестування (E2E) - це вид тестування, який охоплює тестування всієї системи з точки зору користувача, включаючи всі взаємодії між різними компонентами системи та зовнішніми системами. Це означає, що при проведенні E2E тестування тестувальник перевіряє, як система працює в цілому, від початку до кінця, з точки зору користувача.

При проведенні E2E тестування зазвичай використовуються сценарії, що відображають реальні умови використання системи користувачами. Наприклад, при E2E тестуванні інтернет-магазину тестувальник може перевірити, як працює процес замовлення товару, включаючи додавання товару в кошик, оформлення замовлення, оплату та доставку. Також можуть бути перевірені різні інтеграції та взаємодії між різними компонентами системи, такі як бази даних, сервери та інші зовнішні системи.

Одним із важливих аспектів E2E тестування є те, що цей вид тестування дозволяє виявляти проблеми, що можуть виникнути при взаємодії різних

компонентів системи. Також E2E тестування допомагає забезпечити відповідність системи бізнес-вимогам та вимогам користувачів, оскільки перевіряє роботу системи в цілому.

У процесі проведення end-to-end тестування зазвичай використовуються спеціальні інструменти, які дозволяють автоматизувати тестові сценарії та отримувати більш детальну звітність про результати тестування. Такі інструменти можуть симулювати реальне використання програми користувачами та перевіряти роботу усіх компонентів системи.

Для проведення ефективного end-to-end тестування важливо мати чіткі тестові сценарії, які охоплюють всі можливі варіанти взаємодії користувача з програмою. Також необхідно мати добре розуміння архітектури програмного продукту та його взаємодії з іншими системами, щоб забезпечити повноту та точність тестування.

Узагальнюючи, end-to-end тестування є важливим етапом в процесі тестування програмного продукту, оскільки воно дозволяє перевірити, чи працює весь функціонал системи правильно та забезпечити якість продукту перед його випуском в експлуатацію.

E2E тестування можна виконувати як вручну, так і за допомогою автоматизованих засобів. Автоматизовані E2E тести дозволяють швидше та ефективніше проводити тестування, зменшуючи витрати на тестування та забезпечуючи більшу точність результатів.

Одним з найбільш поширених інструментів для автоматизації E2E тестування є фреймворк Cypress. Він забезпечує зручний і простий у використанні інтерфейс для написання тестів та пропонує широкі можливості для перевірки функціональності веб-додатків.

1.1.5 Тестування на проникнення

Тестування на проникнення (англ. Penetration Testing, або скорочено Pentest) – це процес виявлення уразливостей в інформаційній системі шляхом спроб здійснення атак з боку зловмисника.

Цей вид тестування проводиться з метою виявлення слабких місць у системі авторизації та аутентифікації, інфраструктури та мережі додатків, що можуть бути використані зловмисниками для здійснення несанкціонованих дій. Тестування на проникнення можна проводити як зовнішньо, так і внутрішньо, досліджуючи можливості зловмисників, що працюють з різних локацій та використовують різні методи нападу.

Тестування на проникнення включає такі етапи: планування, ідентифікація цілей, збір інформації, виконання атак та оцінка результатів. Основна мета тестування на проникнення полягає в тому, щоб імітувати поведінку потенційного зловмисника, який намагається проникнути в систему зловживанням уразливостей. Для досягнення цієї мети спеціаліст-тестувальник може використовувати різноманітні методики та інструменти, такі як сканування портів, тестування на злам, тестування на переповнення буфера, тестування на ін’єкції коду (injection testing) та інші.

Тестування на проникнення є одним з найбільш складних та ресурсномістких видів тестування, оскільки вимагає великої кількості знань та досвіду в області безпеки, а також усунення можливих наслідків під час проведення тестування. Проте воно є необхідним для запобігання можливих атак та забезпечення надійного захисту інформаційної системи.

Результатом тестування на проникнення є звіт про знайдені уразливості в системі, їх опис, ризик, верхнеповерхневий опис того, як можна їх полагодити [2].

1.1.6 Тестування продуктивності

Тестування продуктивності є важливим етапом у розробці програмного забезпечення, щоб переконатись, що система може обробляти завантаження в реальному світі. Існують багато методів тестування продуктивності, включаючи spike тестування, stress тестування, load тестування, soak тестування, тощо.

- Тестування продуктивності при сплесках навантаження (англ. Spike testing) – це метод, коли навантаження різко збільшується до максимально

можливого рівня протягом короткого періоду часу, щоб перевірити, як система веде себе в критичних ситуаціях. Цей тип тестування може допомогти виявити слабкі місця системи та планувати можливі ризики.

- Стрес-тестування – це метод, коли навантаження на систему збільшується до максимального рівня, що система може витримати, а потім підтримується на цьому рівні протягом тривалого періоду часу. Мета цього тестування - визначити точку витримки системи та виявити можливі проблеми з продуктивністю.
- Навантажувальне тестування (англ. load testing) – це метод, коли навантаження на систему поступово збільшується до максимального рівня, який система може витримати, а потім підтримується на цьому рівні протягом тривалого періоду часу. Цей метод допомагає виявити, як система поводиться під навантаженням, що наближається до максимального.
- Тестування стабільності (soak testing) – це метод, коли система піддана постійному навантаженню протягом тривалого періоду часу (зазвичай декілька годин або навіть днів), щоб виявити, як вона поводиться при продовженні роботи. Цей тип тестування може допомогти виявити проблеми з ресурсами, такі як пам'ять, навантаження на процесор, використання мережі.

1.1.7 Мутаційне тестування

Мутаційне тестування - це метод тестування програмного забезпечення, який полягає у внесенні змін до програмного коду, щоб визначити, наскільки ефективно тести покривають код та виявляють помилки.

У цьому методі тестування, програмний код змінюється штучно, щоб створити "мутантів" - версії коду, у яких внесені деякі зміни, такі як додавання, видалення або зміна рядків коду. Для кожного мутанта, виконуються тестові сценарії, які використовуються для тестування основного коду, і результати порівнюються, щоб визначити, чи була виявлена помилка.

Мутаційне тестування допомагає виявляти дефекти, які можуть бути пропущені під час інших методів тестування, таких як функціональне тестування. Воно дозволяє перевірити якість тестування тестового набору, який використовується для тестування програмного забезпечення, та забезпечити, що тести виявляють всі можливі помилки.

Одним з недоліків мутаційного тестування є його складність та велика кількість генерованих мутантів, що може призвести до значної витрати часу та ресурсів на тестування. Однак, цей метод тестування може бути корисним для критичних систем або програм, де необхідна максимальна надійність та безпека. Приклади бібліотек для мутаційного тестування: Stryker.NET, Mutmut, PIT, тощо [3].

1.1.8 Системне тестування

Системне тестування є однією з фаз тестування програмного забезпечення, яка полягає у тестуванні повної системи для перевірки відповідності її вимогам та очікуванням користувачів.

Цей вид тестування проводиться після інтеграційного тестування та перед приймальним тестуванням, коли програмне забезпечення вже належним чином інтегроване та функціонує у відповідності з вимогами, визначеними в специфікації вимог.

Під час системного тестування виконується тестування всіх функцій, включаючи ті, що є пов'язані з базами даних, інтерфейсами, забезпеченням безпеки, продуктивністю, навантаженням, стабільністю та іншими. У процесі системного тестування також перевіряється взаємодія програмного забезпечення з іншими системами або пристроями, з якими воно має взаємодіяти, наприклад, з мережею або базою даних.

Метою системного тестування є виявлення дефектів, помилок або проблем, які можуть виникнути при використанні програмного забезпечення. Цей процес також допомагає забезпечити, що програмне забезпечення відповідає стандартам якості та безпеки та забезпечити високу якість продукту для користувачів.

У процесі системного тестування можуть використовуватися автоматизовані тестові сценарії та ручні тести. Після проходження системного тестування, якщо дефекти виявлені, вони виправляються, та тестування повторюється до того часу, поки продукт не буде відповідати вимогам та очікуванням користувачів.

Загалом існує дуже багато видів, методів та методик тестування, тут було розглянуто основні. Деякі методи тестування властиві для використання тільки для деяких типів продуктів.

1.2 Тестування веб-додатків, основні поняття та значущість

Основні поняття в тестуванні програмного забезпечення включають в себе:

Тестування - процес виявлення дефектів та помилок в програмному забезпеченні за допомогою запланованих тестів.

Тест - сукупність кроків та даних, які використовуються для виконання програми з метою перевірки її правильності.

Тест-кейс - документ, який описує конкретний тест та його очікуваний результат.

Дефект (баг) - виявлена некоректна поведінка програми, яка може призводити до неправильних результатів або збоїв в роботі програми.

Набір тестів (test suite) - сукупність тестів, які виконуються для перевірки певної функціональності або властивості програмного забезпечення.

Тестовий план - документ, який описує загальний підхід до тестування програмного забезпечення та планує процес його виконання.

Покриття коду - відсоток коду, який було протестовано під час виконання тестів.

Тестування веб-додатків є важливою складовою розробки програмного забезпечення, оскільки веб-додатки є важливою складовою бізнес інфраструктури компаній-постачальників будь-якого роду інтернет-ресурсів. Нестабільність або помилки в роботі веб-додатку можуть призвести до великих фінансових втрат та втрати довіри користувачів. Також важливо створювати веб-додатки безпечними,

оскільки вони зазвичай збирають та зберігають конфіденційну інформацію користувачів, таку як паролі, адреси електронної пошти та банківські дані.

Однією з основних переваг тестування є забезпечення якості програмного забезпечення. Це дозволяє виявити та виправити помилки та проблеми на ранній стадії розробки, що зменшує витрати на пізнішу корекцію та підвищує задоволеність користувачів. Крім того, автоматизоване тестування веб-додатків може допомогти зменшити час, необхідний для виконання тестів, та збільшити точність тестів, особливо для великих проєктів.

З урахуванням того, що розробка веб-додатків стає все більш популярною, тестування веб-додатків стає ще важливішою складовою розробки програмного забезпечення.

Існує багато різних процесів тестування і написання тестових сценаріїв. Деякі з найпоширеніших процесів включають:

- Waterfall (Каскадний процес): це лінійний процес розробки програмного забезпечення, який передбачає, що кожен етап розробки повинен завершуватися, перш ніж розпочати наступний етап. В процесі тестування це означає, що тести написані після того, як код було повністю розроблено.
- Agile (Гнучкий процес): цей процес передбачає, що розробка програмного забезпечення є неперервним процесом, де тестування виконується паралельно з розробкою. Це означає, що тестові сценарії написані на ранніх етапах розробки, і таким чином, дозволяє виявляти помилки раніше та вносити зміни в розробку.
- DevOps (Розробка та експлуатація програмного забезпечення): це підхід до розробки та експлуатації програмного забезпечення, який забезпечує плавний перехід від розробки до випуску та експлуатації. У цьому процесі тестування виконується автоматично під час збірки та релізу програмного забезпечення.

Процес тестування може бути досить різноманітним і залежить від багатьох факторів, таких як тип проєкту, вид програмного забезпечення, команда

розробників та тестувальників, строки, бюджет, тощо. Існує велика кількість процесів тестування та написання тестових сценаріїв, але наступні є основними:

1. Ручне тестування: це процес, у якому тестувальник вручну перевіряє функціональність програмного забезпечення. Ручне тестування може бути проведене як перед випуском програмного забезпечення, так і після випуску.

2. Автоматизоване тестування: це процес, у якому тестові сценарії виконуються за допомогою автоматичних тестових скриптів, що дозволяє швидше та ефективніше перевіряти функціональність програмного забезпечення.

3. Тестування на вимоги: це процес, у якому тестувальники перевіряють, чи відповідає програмне забезпечення вимогам, визначеним у специфікації вимог.

4. Тестування на прийняття: це процес, у якому замовник перевіряє програмне забезпечення на відповідність вимогам, перед тим, як воно буде випущене в експлуатацію.

5. Тестування на проникнення: це процес, у якому тестувальники намагаються знайти уразливості в програмному забезпеченні та перевірити його стійкість до атак.

6. Тестування на навантаження: це процес, у якому перевіряється, як програмне забезпечення працює під час навантаження або високого обсягу трафіку.

7. Тестування сумісності: це процес, у якому перевіряється, як програмне забезпечення працює на різних платформах.

В наш час методологія розробки Agile набирає все більшої популярності, особливо за рахунок того, що нею здебільшого користуються стартапи, оскільки у них бюджет здебільшого обмежений, потрібно якомога швидше випустити MVP та потреби бізнесу постійно змінюються, а з ними і вимоги до продукту [4]. Разом з більшим використанням Agile методологій, в таких компаніях більше використовуються Agile підходи до написання тестів. Одним з таких є TDD (англ. Test Driven Development - розробка, орієнтована на тестування). Основний принцип цього підходу є написання тестів перед написанням основного коду програми. Тут мається на увазі, що тести та код продукту пишуться разом, але код тестів пишеться трохи раніше[5]. Цей підхід суттєво відрізняється від традиційних

підходів написання тестів, які передбачають написання тестів, коли весь код уже був написаний. TDD як ідея зародилася у 2005 році Кентом Беком [6].

Написання тестів в рамках TDD, дотримуючись стандартів практик написання тестів за цим підходом дозволяє встановити рамки робочого циклу до 30 секунд. Якщо організувати все таким чином, це дозволяє писати десятки тестів кожен день. Через те, що тести пишуться разом з основним кодом продукту, це примушує людину писати код продукту, який легко тестувати, що, як показує практика позитивно відзначається на якості коду та його читабельності [5].

Для забезпечення коректної роботи продукту, особливо такого складного, як веб-застосунок, не можливо просто написати модульні тести і сказати, що все працює. Для підтвердження коректності роботи використовують різні набори тестів, і один метод тестування в цьому процесі може виконуватися автоматизовано, а інший вручну після підтвердження. що попередній етап вже пройдений. Модульне тестування само по собі дуже гарно підходить для тестування бізнес логіки, перевірку. що конкретна функція буде мати правильну поведінку незалежно від того. що туди буде передано. Проте модульні тести не здатні дати гарантію, що між собою модулі будуть взаємодіяти коректно. Це і є причиною об'єднання тестування у декілька етапів. Використання методів постійної інтеграції (англ. Continuous integration, скорочено CI) дозволило автоматизувати тестування застосунку: виставляти час, події для початку тестування, блокування розгортання продукту, якщо хоч один з тестів провалився.

Веб-додатки – це комплексні програми, які можуть ґрунтуватися на різних архітектурних моделях, мати різні компоненти, написані на різних мовах програмування та використовувати одразу різні типи баз даних. «Погані» архітектурні рішення роблять веб-додатки ще уразливішими до змін в коді. Модифікація старого функціоналу може призвести до неочікуваних змін у функціоналі та невизначеній поведінці інших модулів, функціонал яких були зачеплено змінами. Для зниження можливих наслідків використовують регресійне тестування. Регресійне тестування – це тип тестування застосунку або його частини після того, як були внесені зміни у його вихідних код. Це робиться для

підтвердження, що зміни не вплинули на поведінку інших модулів програми. Регресійне тестування часто входить у життєвий цикл розробки продукту та проводиться перед випуском програми. Для автоматизації регресивного тестування використовують багато застосунків, наприклад Selenium.

1.3 Автоматизація тестування

Ручне тестування продукту затрачає багато часу та зусиль тестувальників. Перед релізом завжди потрібно робити регресивне тестування, оскільки за час готування релізу, старий функціонал безумовно також змінювався, модифікувався, тощо. Цей процес затягує сам реліз, та в процесі можуть бути знайдені додаткові проблеми у продукті, що ще більше відкладе сам реліз.

Автоматизація - це процес використання інструментів та технологій для зменшення залучення людини, забезпечення автоматичного виконання задач, які раніше виконувалися вручну. У контексті тестування програмного забезпечення автоматизація тестування означає: використання спеціалізованих інструментів та програмних засобів для автоматичного виконання тестів замість виконання їх вручну.

Автоматизація тестування може бути застосована до різних видів тестів, таких як модульні тести, інтеграційні тести, системні тести, тести на проникнення тощо. Це дозволяє прискорити процес тестування, зменшити час виконання тестів та знизити вартість тестування в цілому. Крім того, автоматизовані тести можуть бути запущені в будь-який час, що дозволяє забезпечити постійну контроль за якістю програмного забезпечення в процесі розробки та після випуску.

Однак, автоматизація тестування не заміняє ручне тестування повністю, а слід розглядати її як допоміжний інструмент. Автоматизація не здатна повністю замінити інтуїцію тестувальника та його здатність до аналізу та оцінки програмного забезпечення. Тому автоматизацію слід використовувати розумно та в комбінації з іншими методами тестування для забезпечення високої якості програмного забезпечення.

Автоматизація тестування - це процес розробки тестів та їх виконання з використанням спеціальних інструментів і програмного забезпечення. Замість того, щоб виконувати тести вручну, автоматизація дозволяє розробити скрипти тестування, які можна запустити автоматично, щоб перевірити функціональність програмного продукту. Автоматизовані тести зазвичай швидші та більш точні, ніж ручні тести, оскільки вони виконуються однаково та за однакових умов кожного разу. Крім того, автоматизація тестування дозволяє зекономити час та ресурси, бо автоматичні тести можна запустити без участі людей в заданий час або після заданої події, такої як надсилання нового коду, успішна компіляція продукту, тощо. Автоматичні тести дають змогу побачити проблеми під час розробки продукту та гарантують, що частина продукту, покрита тестами буде працювати як передбачалося.

Автоматизована генерація тестів – це процес створення тестів з використанням спеціальних інструментів, створених для мінімізації залучення людини під час створення тестів. Тести, згенеровані такими інструментами, є від самого початку готовими, але для ідеального виконання можуть потребувати коректування людиною вручну.

1.3.1 Основні підходи до автоматизованої генерації тестів

Існує декілька підходів для автоматизованої генерації стрес-тестів для веб-API:

- Заснований на даних (data-driven).

Цей підхід полягає в генерації тестових даних для використання в тестуванні API. Це може включати генерацію випадкових даних або використання реальних даних, щоб перевірити, як добре API працює з різними типами даних.

- Заснований на структурі (structure-based).

Цей підхід полягає в генерації тестів на основі структури API, такої як структура запитів і відповідей. Це може включати генерацію тестів на основі стандартів, таких як OpenAPI, щоб забезпечити, що API відповідає стандартам.

- Заснований на динаміці (dynamically generated).

Цей підхід полягає в генерації тестів на основі динамічного аналізу API. Це може включати відслідковування даних, які передаються в API та аналіз їхньої структури, щоб забезпечити, що API працює правильно.

- Тестування на основі спостережень (англ. Observation-based Testing).

Цей підхід бере за основу спостереження за діями користувача та генерацією тестів на основі цих дій. Виходячи з описаних в минулому розділі прикладів застосунків для автоматизованої генерації тестів, даних метод є доволі популярним [7].

1.4 Сучасні засоби для автоматизованої генерації тестів веб-застосунків

1.4.1 Quotium Qtest

QTest від Quotium – це пропрієтанрий корпоративний інструмент для функціонального тестування, який дозволяє створювати та генерувати тести для веб-додатків та мобільних додатків. QTest використовує технологію штучного інтелекту для автоматичної генерації тестових сценаріїв на основі аналізу взаємодії користувача з додатком.

Основним принципом роботи QTest є моніторинг активності користувача в додатку, щоб визначити типові шляхи користувачів і сценарії використання програми. На основі цієї інформації QTest генерує тестові сценарії, які можуть відтворити типову поведінку користувача програми.

При створенні тестових сценаріїв QTest враховує не лише типові сценарії використання, але й несподівані сценарії, які можуть призвести до помилок у програмі. Це дозволяє створювати більш повні та надійні тести для додатків, покращуючи якість продукту та зменшуючи ймовірність помилок у продакшені.
Даний програмний застосунок використовує підхід тестування на основі спостережень. Підходи розглянуті у наступному розділі.

1.4.2 OpenText Silk Performer

Це пропрієтарний застосунок для автоматизації створення стрес-тестів для веб-застосунків.

Суть даної програми полягає у можливості створення навантажувальних тестів як вручну, так і автоматично за допомогою запису запитів, які виконуються на певній сторінці. Для цього необхідно вказати посилання на сторінку, виконати запис та визначити необхідні транзакції.

Даний програмний застосунок використовує підхід тестування на основі спостережень.

1.4.3 IBM Rational Performance Tester

IBM Rational Performance Tester це ще один застосунок для автоматизованого створення навантажувальних тестів. Він входить у пакет IBM Rational Test Workbench, наряду з IBM Rational Integration Tester та IBM Rational Functional Tester. Цей застосунок дозволяє створювати тести як і вручну так і автоматизовано, завдяки запису дій користувача з веб-застосунком. Записи зберігаються та генерується порядок виконання самих тестів та їх вміст. Цей застосунок також базується на підході генерування тестів на основі спостереження.

Дослідження можливих аналогів додатків було не легким та ускладнювалось фактом пропрієтарності вищеописаних засобів. Зі даного дослідження можна зробити висновок що засоби для автоматизованого тестування веб-додатків наразі недостатньо розвинуті та популярні. Це підтверджує твердження, що засіб для автоматизованого тестування веб додатків є актуальним та корисним, оскільки подібні застосунки займають певну нішу на ринку та користуються попитом.

1.4.4 Curiosity Test Modeller

Curiosity Test Modeller це веб-застосунок для автоматизованого створення тестових сценаріїв для тестування веб-застосунків. Цей застосунок використовує

оснований на даних (data-driven) підхід для автоматизованого створення тестових сценаріїв.

Для створення тестових сценаріїв потрібно спочатку створити модель даних для тестового сценарію. Для цього можна використати готовий csv файл з наявними даними для конкретного сценарію та імпортувати його у програму. Програма з цього набору даних створить комбінацію з різних тестових сценаріїв (рис. 1.4.4.1), які потім можна буде використати для тестування веб-застосунку, на базі основного сценарію (рис. 1.4.4.2) роботи для якого розробляється тест (створення нового користувача, додавання продуктів у корзину, тощо).

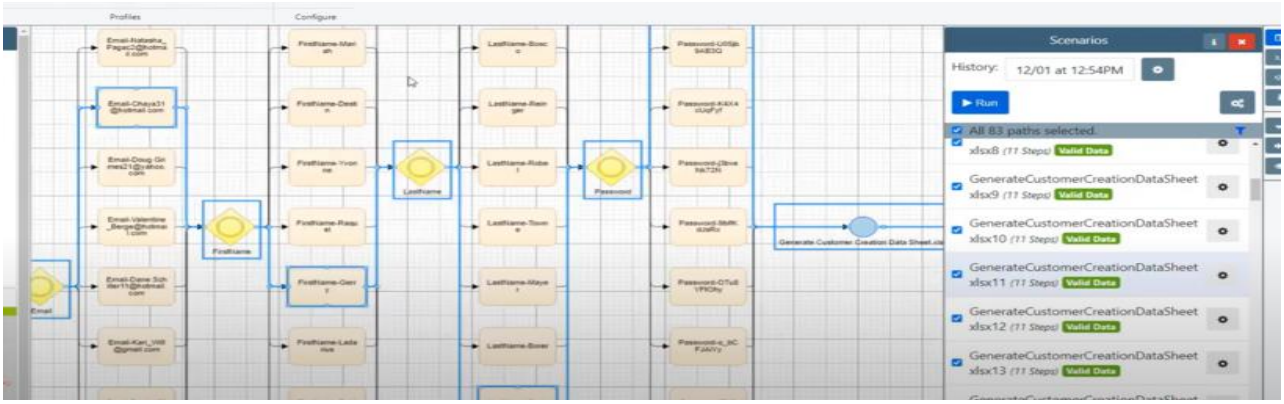


Рисунок 1.4.4.1 Комбінації тестових сценаріїв

Рисунок 1.4.4.2 Вигляд основного сценарію

1.4.5 Порівняльний аналіз додатків автоматизованого стрес-тестування

Аналіз літературних джерел дозволив виділити переваги та недоліки сучасних додатків автоматизованого стрес-тестування, результати якого представлено у таблиці 1.4.5.1.

Таблиця 1.4.5.1 Порівняння характеристик додатків автоматизованого стрес-тестування

Назва застосунку	Quotium QTest	OpenText Silk Performer	IBM Rational Performance Tester	Curiosity Test Modeller
Пропрієтарний	+	+	+	+
Має веб-інтерфейс	-	-	-	+
Можливість завантажувати приклади даних	-	-	+	+
Підхід для автоматизації	На основі спостережень	На основі спостережень	На основі спостережень	На основі даних
Автоматизація стрес тестування	-	-	+	+
Можливість інтеграції з іншими застосунками для проведення тестування	-	-	-	-

Слід відмітити, що проаналізовані додатки мать спільні проблеми – всі вони є пропрієтарними та жодна з них не має функціоналу інтегрування зі сторонніми засобами стрес-тестування.

2. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ СТРЕС-ТЕСТІВ

2.1 Огляд засобів реалізації додатку для автоматичної генерації стрес-тестів веб-додатків

Додаток для автоматичної генерації стрес-тестів веб-додатків в даній роботі буде розроблятися для персональних комп'ютерів. Через це вибір мов програмування необмежений. Не дивлячись на це, не всі мови програмування підходять для розробки деяких типів додатків. Зокрема, для персональних комп'ютерів виділяють наступні типи додатків:

2.1.1 Прогресивний веб-додаток

Це тип додатків, який безпосередньо відкривається у браузері користувача або у програмі яка побудована на основі одного з браузерних движків (Chrome, Firefox, тощо). Прикладами таких програм у повсякденності можуть бути Postman, Discord, тощо.

З плюсів такого підходу можна вирізнити те, що застосунок буде кросплатформеним та буде виглядати однаково на кожній платформі.

З мінусів такого підходу є те, що такі застосунки використовують в 2 рази більше оперативної пам'яті а ніж звичайні веб-додатки [8].

2.1.2 Нативний додаток

Це додаток, який для рендерингу та роботи використовує засоби операційної системи напряму. Кількість ресурсів на розробку такого типу застосунків напряму залежить від необхідних операційних систем для підтримки та обраних фреймворків та мов програмування.

Плюсами таких застосунків є швидкодія, доступність до найнижчих інструкцій, які дає змогу використовувати операційна система, напряму і з повним контролем.

З мінусів виступає кількість ресурсів необхідних на розробку, особливо, якщо треба забезпечити кросплатформність.

2.1.3 Консольні додатки

Це додатки, які виконуються напряму з командного інтерфейсу користувача (cli). Такі застосунки є найлегшими для розробки, оскільки не потрібно розробляти інтерфейс користувача у звичному сенсі цього слова. Потрібно розробити тільки довідник команд, які можна використовувати.

З недоліків можна виділити, що такий інтерфейс взаємодії є недружнім до початкових користувачів.

З позитивних сторін можна виділити той факт, що такі програми дуже легко автоматизувати для використання на підприємствах або для персональних цілей, якщо цього дозволяє інтерфейс програми.

Слід відмітити, що такі програми легше портувати на майже будь-яку операційну систему або архітектуру, оскільки для цього потрібно лише скомпілювати вихідних код для цільової платформи або архітектури.

У даній роботі розробляється консольний додаток під платформу Windows. Під час вибору мови програмування також приймалося до уваги можливість подальшого розвитку додатку та портування його на інші операційні системи.

C# підтримує усі основні методики в програмуванні — мультипоточність, асинхронне програмування, лямбда вирази, рефлексію, універсальні шаблони та ООП. В останніх специфікаціях мови почали додаватися елементи функціонального програмування, такі як зіставлення зразка (англ. Pattern matching), пусті змінні (англ. Discards), деконструкція типів (англ. Type deconstruction).

Для роботи з OpenApi було використано пакет Microsoft.OpenApi від Microsoft. Ця бібліотека спрощує процес роботи зі стандартом OpenApi та дає необхідні структури для обробки вхідних даних.

Для створення командного інтерфейсу користувача було використано відкриту бібліотеку McMaster.Extensions.CommandLineUtils [9]. Ця бібліотека дозволяє зекономити час розробника на розробці функціоналу для роботи з консоллю та більше зосередитися на розробці самого функціоналу застосунку.

Для роботи с текстовим форматом JSON було використано бібліотеку Newtonsoft.Json. Ця бібліотека є одною з найшвидших та найбільш розвинених бібліотек для роботи з форматом JSON для платформи .NET [10].

2.2 Grafana K6

В якості цільового застосунку для стрес-тестування використовується застосунок Grafana K6. Цей застосунок відрізняється від своїх аналогів, таких як Apache Jmeter тим, що є дуже простим в інсталюванні та використанні. Також цей застосунок відповідає усім потребам для стрес-тестування, оскільки дає користувачу всі необхідні метрики для розуміння продуктивності системи для конкретного сценарію, а також необхідні можливості для конфігурації стрес-тестового сценарію, такі як розбиття сценарію на етапи, розділені за тривалістю та цільовою кількістю запитів в секунду, які будуть підтримуватися на протязі часу даного етапу. На додаток до цього Grafana K6 використовує дуже зручний формат конфігурації — JavaScript файли, які потрібно описувати згідно з документації. Цей факт дає максимальний простір під час написання тестового сценарію, що дозволяє дуже легко маніпулювати з тестовими сценаріями та додавати можливі алгоритми, імпортувати інші бібліотеки та модулі під час виконання тестового сценарію [11]. Цей функціонал можна використовувати навіть для отримання необхідних даних з бази даних в реальному часі перед запуском самого стрес-тесту для того, щоб відбувалися реальні запити в базу даних, які зможуть максимально навантажити систему. Також Grafana K6 дає змогу користувачу використовувати сторонні засоби для аналізу роботи застосунку під час проходження стрес-тесту з показанням необхідних метрик під-час або після виконання тестового сценарію використовуючи вибраний користувачем додаток для роботи з даними в реальному

часі, наприклад: Apache Kafka, Amazon CloudWatch, Prometeus remote wire, InfluxDB, NetData, New Relic та інші [12].

Для тестування застосунку під час розробки було використано бібліотеку для тестування xUnit.

Для розробки тестового API було використано веб-фреймворк на базі .NET ASP.NET 7.

2.3 Стандарт OpenApi

OpenAPI (раніше відомий як Swagger) – це специфікація для опису REST API. Вона дозволяє розробникам описувати API у форматі машинного читання, що дозволяє автоматизувати процеси, пов'язані з розробкою, тестуванням та документуванням API.

OpenApi використовує формат JSON або YAML для опису API. У специфікації вказується інформація про ресурси, методи, параметри запиту та відповіді. Також можна вказати вимоги до аутентифікації, типи даних, схеми валідації даних та багато іншого.

Використання OpenAPI дозволяє розробникам та тестувальникам легко взаємодіяти з API та автоматизувати багато процесів. Наприклад, специфікація може використовуватися для автоматичної генерації коду клієнтів для API, для автоматичної генерації документації та для автоматичного тестування API.

OpenAPI є стандартом, що підтримується різними інструментами та сервісами. Це дозволяє розробникам легко інтегрувати специфікацію в свої проекти та використовувати різні інструменти для роботи з API.

OpenAPI є відкритим стандартом та є частиною проекту OpenAPI Initiative, який належить Linux Foundation. Проект підтримується великою кількістю компаній та розробників, що забезпечує його активний розвиток та підтримку [13].

Оскільки стандарт доволі поширений його і було обрано за основу додатку для автоматичної генерації стрес-тестів.

В наш час новітні IT системи можуть бути інтегровані з безліч сторонніх веб сервісів. Розробники повинні інтегрувати сторонні сервіси у новітні системи. Для полегшення цього, провайдери сторонніх сервісів використовують стандарт OpenApi, оскільки він дозволяє одразу оцінити корисні сервіси при інтеграції та зменшує час на створення документації [14].

2.4 Технологія OpenApi Generator

Технологія OpenAPI Generator - це інструмент, який дозволяє автоматично генерувати код для клієнтських бібліотек, серверних заглушок та документації на основі специфікацій API, написаних в форматі OpenAPI (раніше відомому як Swagger).

За допомогою OpenAPI Generator можна автоматично згенерувати код для декількох десятків мов програмування, таких як Java, Python, JavaScript, C#, Go, Ruby, Kotlin, Swift та інші. Також цей інструмент дозволяє згенерувати документацію у форматах HTML, Markdown та Confluence Wiki, тощо.

OpenAPI Generator базується на шаблонному підході до генерації коду, тобто код генерується на основі шаблонів, що дозволяє легко модифікувати та розширювати згенерований код.

Загалом, OpenAPI Generator дозволяє зекономити час та зусилля, які були б витрачені на ручне написання коду для взаємодії з API. Він спрощує розробку та підтримку API, дозволяючи розробникам сконцентруватись на реалізації функціональності своєї програми, а не на написанні базового коду для взаємодії з API [15].

2.5 Мова програмування C#

C# (C Sharp) - це об'єктно-орієнтована, сильно типізована мова програмування, розроблена компанією Microsoft. Вона була представлена у 2000

році і стала однією з основних мов розробки програмного забезпечення для платформи .NET.

C# має синтаксис, подібний на C++ чи Java, хоча з плином часу відмінностей стає все більше. Але це не виключає факту, що розробникам на цих мовах буде легше опанувати C#.

C# підтримує усі основні методики в програмуванні — мультипоточність, асинхронне програмування, лямбда вирази, рефлексію, універсальні шаблони та ООП. В останніх специфікаціях мови почали додаватися елементи функціонального програмування, такі як зіставлення зразка (англ. Pattern matching), пусті змінні (англ. Discards), деконструкція типів (англ. Type deconstruction). [https://learn.microsoft.com/uk-ua/dotnet/csharp/fundamentals/functional/pattern-matching] C# також підтримує використання SIMD інструкцій процесорів для максимальної оптимізації коду.

C# позиціонується як безпечна або спрямована на безпеку мова. В першу чергу має обмежений функціонал для роботи з пам'яттю напряму, хоча стандартна бібліотека надає велику кількість методів. Для роботи з вказівниками потрібно явно сказати компілятору, що код небезпечний параметром компіляції "unsafe". Також C# має вбудований механізм "чистки сміття", що дозволяє розробнику не думати про самостійну чистку пам'яті після виділення. Тут потрібно зазначити, що чистка сміття неможлива для небезпечного (unsafe) коду.

Також варто зазначити, що C# підтримує кросплатформенність, тобто програми, написані на цій мові, можуть працювати на різних операційних системах, включаючи Windows, Linux та macOS. Це дозволяє розробникам створювати програмне забезпечення, яке може працювати на різних пристроях та платформах.

Основними фреймворками для написання коду C# є .NET(раніше .NET Core), .NET Framework, Mono, UWP, ASP.NET.

C# є JIT-компільованою мовою. Це означає, що в процесі виконання програма докомпілює код в машинний та оптимізує його. Через це в деяких задачах

швидкість виконання коду на C# більша ніж коду на C++ чи C, хоча це скоріше виключення [16].

Код C# компілюється у так званий IL (англ. Intermediate Language - проміжний) код, який по своїй суті схожий на високоверхневий Assembler. IL код виконується CLR (англ. Common Language Runtime - Загальномовне середовище виконання).

Щоб CLR могло надавати послуги для керованого коду, компілятори мов програмування .NET видають метадані, які описують різні аспекти коду, такі як типи, члени та посилання. Ці метадані зберігаються разом із самим кодом, тобто будь-який портативний виконуваний файл, який використовується CLR, міститиме метадані. CLR використовує ці метадані для виконання різних функцій, включаючи пошук і завантаження класів, розміщення екземплярів об'єктів у пам'яті, вирішення викликів методів, генерування рідного коду, забезпечення безпеки та встановлення меж контексту часу виконання.

Через це, платформа .NET є дуже гнучкою: за допомогою компілятору MSBuild у розробника є можливість написати свій аналізатор коду або програму, яка буде виконуватися під час компіляції та створювати вихідний код (англ. Source code generator) – такі програми є досить популярними в наш час та можеть бути використані для багатьох різних типів задач: оптимізація роботи додатку, оптимізація налагодження (англ. debugging) коду. Навіть Майкросовт з початку існування C# і до сьогодні використовує такі засоби і включила їх у стандартну бібліотеку [17]. Використовуючи ці методи можна навіть вставляти IL код. Це використовується для оптимізації, і в останніх версіях мови його було включено для оптимізації роботи і це дозволило підвищити продуктивність роботи коду до 4 разів.

CLR призначене для полегшення створення компонентів і програм, об'єкти яких взаємодіють один з одним на різних мовах програмування. Об'єкти, написані різними мовами, можуть безперебійно спілкуватися один з одним, а їх поведінка може бути тісно інтегрована. Наприклад, можна визначити клас на одній мові, а потім використовувати іншу мову для отримання класу з оригінального класу або

виклику методу вихідного класу. Крім того, екземпляр класу можна передати методу класу, який був написаний іншою мовою. Ця міжмовна інтеграція стала можливою завдяки загальній системі типів (CTS, англ. Common type system), яка визначається середовищем виконання та використовується мовними компіляторами та інструментами, націленими на середовище виконання. Усі залучені сторони повинні дотримуватися правил створення, використання, збереження, зв'язування та визначення нових типів, встановлених стандартом ECMA для CLI (англ. Common Language Infrastructure — загальномовна інфраструктура) [18].

Популярність, як термін для мови програмування, доволі важко визначити. Деякі мови можуть бути дуже популярними и обожнюваними серед програмістів, але саме продуктового коду на них написано мало, інші ж навпаки, вже не згадуються але доволі велика кількість програм певних секторів індустрії написана і працює саме на них. Слід також сказати, що мови програмування розробляються для певних цілей і полегшення виконання певних задач і що неможливо написати одну універсальну мову яка буде ідеальною в усьому.

Не дивлячись на це, популярність мови росте, в 2022 році, наприклад, виросла аж на 2 відсотки, тому можна цілком впевнено сказати що вона користується попитом не менше аніж JavaScript, PHP, Go, тощо [19].

Факторами, які впливають на популярність мов програмування серед розробників залежить від багатьох факторів, не виключаючи такі як відкритість, кількість готових бібліотек, община (англ. Community) конкретної мови, продукти для розробників. В контексті C#, на її популярність позитивно вплинув ігровий рушій Unity. Цей ігровий рушій використовується в 47% незалежних ігор (англ. Indie games) та 13% від загальної кількості ігор онлайн платформи Steam [20]. За даними Unity налічує близнько 750 000 зареєстрованих користувачів, та близько 230 000 активних користувачів в місяць в 2022 році [21]. Ці факти не можуть не впливати на популярність мови програмування.

Зважаючи на всі ці факти мову C# було вибрано для написання цієї роботи.

3. РЕАЛІЗАЦІЯ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ СТРЕС-ТЕСТІВ НА ОСНОВІ OPENAPI МОВОЮ C#

3.1 Проєктування та постановка технічного завдання

Розробити програмний засіб у вигляді застосунку командного рядку, який дозволить користувачу мати можливість згенерувати необхідні для цільового застосунку стрес-тестування, додаючи можливість запуску цільового застосунку стрес-тестування одразу після генерації необхідних файлів конфігурації. На додаток до цього, фінальний програмний засіб повинен вести логування свого внутрішнього стану під час виконання з можливістю користувачу обирати, куди саме вести запис — до файлу або у командний рядок користувача. На додаток, користувач повинен мати змогу обирати, наскільки детальні логи він хоче бачити та мати змогу відключити логування повністю. Також користувач повинен мати змогу обирати звідки саме програма буде отримувати документацію у форматі OpenApi – з інтернету, в якості веб-ресурсу чи з файлу користувача. І користувач повинен мати змогу додавати необхідні йому приклади запитів в конфігурацію програми для опрацювання всеможливих збоїв програми чи просто використовуючи строго потрібний запит, знаючи, що саме така конфігурація буде спричиняти максимальне навантаження.

3.1.1 Функціональні вимоги

- Перевірка існуючих генераторів.

Оскільки застосунок може підтримувати одразу декілька цільових застосунків стрес-тестування, користувачу необхідно мати можливість перевірки, які засоби можуть бути використані, та обирати конкретний.

- Підтримка типів HTTP запитів: GET, POST, PUT.
- Підтримка роботи з форматом запитів JSON.
- Рандомізація запитів.

Створення запитів для цільової програми стрес-тестування повинно бути рандомізоване, тобто усі параметри запиту повинні генеруватися автоматично, виходячи з типу параметру, його допустимих значень (якщо тип - перелік). Якщо тип параметру комплексний - тобто складається з декількох параметрів, то програма повинна рекурсивно згенерувати усі внутрішні параметри. Задля запобігання переповнення та нескінченного зациклювання, програма повинна мати обмеження у вигляді 5 вкладень. Для масивів програма повинна генерувати 5 значень, включаючи комплексні об'єкти.

- Помилки мають логуватися але ігноруватися.

Автоматична робота з API може легко викликати різного роду помилки. Через це все помилки потрібно виводити користувачу, але в той самий час програма повинна ігнорувати запити де вони виникають і продовжувати працювати у штатному режимі.

3.1.2 Нефункціональні вимоги

- Масштабованість

З технічної точки зору потрібно розробити архітектуру програмного забезпечення таким чином, щоб у користувача була можливість самостійно розробити інтеграцію з цільовим застосунком стрес-тестування та використовувати її під час роботи без необхідності ре-компіляції програмного застосунку, таким чином додаючи функціонал під час виконання застосунку (анг. Runtime) у вигляді плагіну. Додавання цільового застосунку стрес-тестування повинно відбуватися за допомогою підключення .NET бібліотеки за допомогою вказування файлу бібліотеки у конфігурації застосунку.

- **Продуктивність**

Програма повинна відпрацьовувати до 10 секунд для API розміром до 1000 запитів якщо файл документації знаходиться на жорсткому диску.

3.1.3 Постановка завдання

Виходячи з вищеописаного, можна сформувати можливості, які засіб повинен давати можливість користувачу конфігурувати:

- Додавати приклади запитів.

В такому випадку програма не буде генерувати рандомізовані об'єкти для запитів, а використовувати надані користувачем.

- Додавати обов'язкові для використання заголовки.

Додані користувачем заголовки повинні бути використані в усіх запитах.

Даний функціонал може бути необхідний у багатьох сценаріях, коли, наприклад, користувачу потрібно включити авторизаційні заголовки для кожного запиту API.

- Додавати конфігурацію для цільового додатку стрес-тестування.

Цей функціонал необхідний для легкого маніпулювання конфігурацією цільової системи без необхідності робити зміни в різних місцях виконання.

- Визначати рівень логування.
- Визначати обрану цільову програму для якої буде генеруватися сценарій стрес-тестування.
- Визначати адресу в мережі інтернет, для якої потрібно буде запустити стрес-тест.

Це налаштування необхідне для проведення стрес-тесту одразу після генерування файлу конфігурації. Це налаштування не повинно бути обов'язковим.

- Вихідна папка для конфігурації.

Необхідно для зберігання файлу чи файлів конфігурації, які будуть створені програмним засобом

- Шлях до бібліотеки-плагіну

Дане налаштування дозволить користувачу включати бібліотеку-плагін для додавання додаткових цільових програм стрес-тестування.

- Шлях до цільової програми стрес-тестування.

Це необхідно для легкості в запуску цільової програми стрес-тестування. Дане налаштування дозволяє оминути різних ускладнень, таких як задання або перевірка наявності змінних середовища, для запуску цільової програми.

- Шлях до файлу документації.

Рисунок 3.1.2.1 Вигляд файлу конфігурації

Необхідне для отримання саме файлу документації API у форматі OpenApi. Може бути шляхом до файлу або посиланням на ресурс в інтернеті.

3.2 Діаграма варіантів використання

На даній діаграмі (рис. 3.2.1) зображено варіанти використання користувачем розробленого програмного забезпечення. Користувач може згенерувати конфігурації тестових сценаріїв з можливістю одразу їх запуску після генерації та показати список усіх цільових застосунків для стрес-тестування.

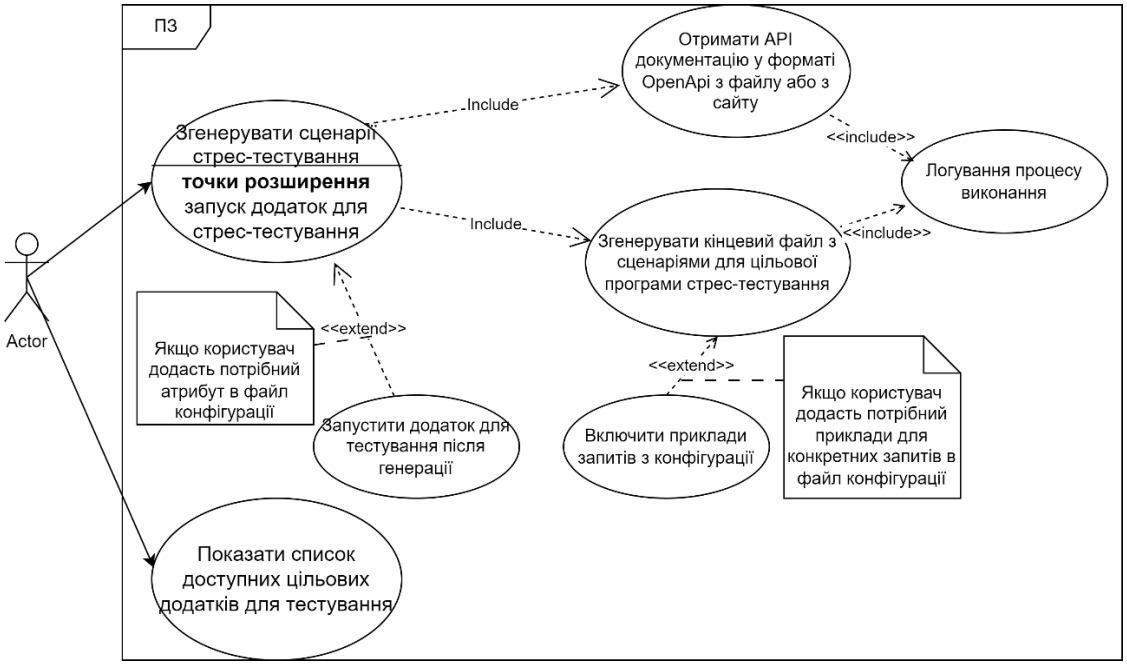


Рисунок 3.2.1 Діаграма варіантів використання

3.3 Архітектура системи

Отже, виходячи з технічного завдання, була продумана архітектура програмного застосунку.

Програмний засіб складається з двух основних складових: основної бібліотеки та консольного застосунку для роботи з бібліотекою. Така реалізація дозволяє розробляти будь-який кінцевий застосунок, такий як веб-застосунок, або графічниз застосунок, оскільки немає залежностей від інтерфейсу та функціоналу наданого консольним застосунком.

Консольний застосунок складається з основного класу Program, який виконує роботу вхідної точки та містить у собі усю логіку обробки аргументів командного рядку переданих користувачем під час виконання. Також, в бібліотеці консолького застосунку представлено статичний класс StressOptions, який представляє собою клас, в якому знаходяться методи розширення функціоналу бібліотеки CommandLineUtils для створення агрументів командного рядку які буде обробляти програма. Також представлено класи FileOutputWriter та ConsoleWriter, які є реалізаціями інтерфейсу з основної бібліотеки IOutputWriter та служать для ведення логування відповідно у файл та у консоль. Інтерфейс IOutputWriter описує контракт для логування, включаючи в себе 4 методи: WriteError - для логування помилок, WriteWarning - для логування попереджень, які являють собою некритичні помилки, WriteException - для логування виключень які виникають в системі під час виконання та являються корисними під час відладки програмного засобу та метод WriteOutput, який відповідає за логування інформаційних повідомлень під час роботи системи.

Основна бібліотека складається з компонентів внутрішньої реалізації та компонентів з можливістю зовнішнього доступу та використання. Наступних класи та інтерфейси визначено як публічні(інтерфейси починаються з суфіксу I-): DotnetStress, SubstitutionConfig, SubstitutionMetadata, StressConfig, SingletonWriter, EndpointMetadata, InputParameter, ArgumentLocation, ParameterMetadata, RunnerNotFoundException, ConfigurationException, IOutputWriter, IGenerator, IRunnerExecutor. Напротивагу, наступні класи являються компонентами внутрішньої реалізації: Extensions, K6Executor, K6Generator та OpenApiParser.
Кожен з даних класів було створено під визначені потреби:

- **DotnetStress**

Даний клас є основною точкою входу в бібліотеку. Він відповідальний за початок всіх процесів для роботи застосунку, передачу конфігурації у потрібні процеси.

- **StressConfig**

Є класом для опису конфігурації, яку можна передавати застосунку для роботи

- **SubstitutionMetadata**

Цей службовий клас є необхідним для парсеру бібліотеки Newtonsoft.JSON для роботи з конфігурацією для підставляння запитів, створених користувачем.

- **SubstitutionConfig**

Є допоміжним класом та використовується класом SubstitutionMetadata

- **SingletonWriter**

Цей клас побудований за паттерном проектування singleton. Він використовується іншими класами для логування їх внутрішнього стану.

- **EndpointMetadata**

Цей клас необхідний для опису окремого ендпоінту. Використовує у реалізації об'єкти класу InputParameter в якості опису параметрів, які використовуються в даному ендпоінті.

- **InputParameter**

Використовується для опису параметру запиту. Може описувати як параметр шляху запиту (www.test.com/api/buy/{parameter}) так і параметр запиту(англ. query parameter), наприклад www.test.com/api/show?{parameter}=1 або параметр заголовку або параметр тіла запиту.

- **ArgumentLocation**

Описує місце-знаходження параметру

- **ParameterMetadata**

Описує сам параметр, його поля, можливі значення, назву, тип.

- **RunnerNotFoundException**

Виключення, яке генерується під час виконання, якщо користувач передав програмі назву неіснуючого цільового додатку для стрес-тестування.

- **ConfigurationException**

Виключення, яке генерується, якщо користувач передав неправильну конфігурацію під час виконання. Наприклад якщо не було передано шлях до файлу або ресурсу який містить документацію у форматі OpenApi.

- **IOutputWriter**

Інтерфейс який описує контракт для реалізації логування в системі.

- **IGenerator**

Інтерфейс, який описує контракт для створення своєї реалізації інтеграції для цільового застосунку для стрес-тестування.

- **IRunnerExecutor**

Інтерфейс який описує контракт для повернення об'єкту для запуску цільового застосунку для стрес-тестування для запуску після коректної генерації конфігурації. Цей інтерфейс дає змогу власноруч конфігурувати поведінку під час запуску, це відкриває можливості виконувати дії будь-якої складності для запуску програми.

- **Extensions**

Це службовий статичний клас, який містить розширення різних методів, які дозволяють скоротити кількість фінального коду та зробити його зрозумілішим.

- **K6Executor**

Це клас, який реалізовує IRunnerExecutor, повертає структуру даних необхідну для запуску програми засобами операційної системи консольним застосунком.

- **K6Generator**

Це основний клас, який відповідальний за генерацію конфігураційних файлів для Grafana K6. Реалізує IGenerator.

- **OpenApiParser**

Цей службовий клас відповідає за отримання документації OpenApi зі стороннього ресурсу або файлу та є відповідальним за подальше перетворення структури документації OpenApi у файли внутрішньої реалізації, описані вище. Дана реалізація, у вигляді одного класу який виконує одну задачу, дозволяє у

подальшому дуже легко інтегрувати альтернативні засоби документації API за допомогою створення окремого інтерфейсу, і використання його по принципу використання IGenerator. Цього не було зроблено за відсутності необхідності.

3.4 Діаграма класів

На даній діаграмі (рис. 3.4.1) показано весь список класів, використаних в кінцевій версії програмного забезпечення. Також на даній діаграмі зображено зв’язки між класами системи.

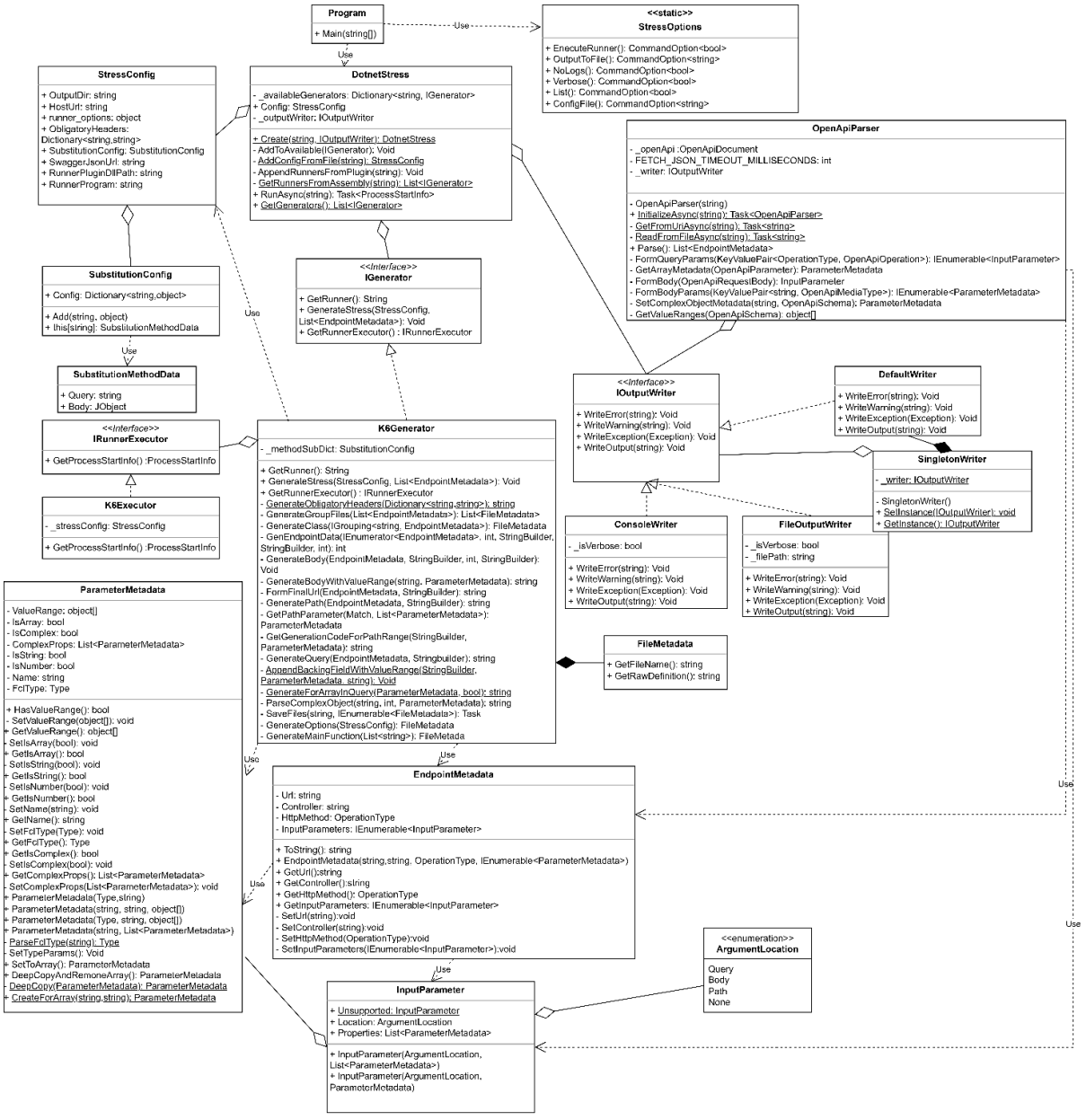


Рисунок 3.4.1 Діаграма класів

3.5 Алгоритм роботи застосунку

Виходячи з технічного завдання, було розроблено застосунок, який має наступний інтерфейс користувача командного рядку при його використанні (рис. 3.5.1):

```
PS E:\temp> dotnet-stress-cli.exe --help
Usage:  [options]

Options:
  -?|-h|--help                Show help information.
  -e|--exec                   Whether to run target stress application. Requires a valid path or for the program to be
                              accessible from PATH
                              Default value is: False.
  -r|--runner <string>        Desired runner name
  -c|--config <CONFIG_PATH>   Indicates where the config is. Default is ./config.json
                              Default value is: ./config.json.
  -l|--list                   List runners
  -v|--verbose                If set will show all output. If not set, only warnings and errors are shown
  -f|--outfile <OUTFILE_PATH> Sets file to write output to. Default is console
  --no-logs                   If set won't write any logs anywhere.
```

Рисунок 3.5.1 Екранна модель зі списком усіх команд

На даній екранній формі описані наступні команди:

- Аргумент запуску застосунку стрес-тестування -e|exec

Якщо даний аргумент присутній під час запуску програми, запускається алгоритм запуску застосунку стрес-тестування після завершення генерації для нього файлів конфігурації. За замовченням даний механізм не виконується.

- Аргумент вибору застосунку для стрес-тестування -r|--runner <string>

Даний аргумент використовується для вказування програмі який цільовий застосунок стрес-тестування використовувати. Як видно з опису, даний аргумент потребує передачі назви. Приклад використання: -r k6.

- Аргумент вказання файлу конфігурації програми -c|--config <CONFIG_PATH>

Значення даного аргументу вказує програмі, де потрібно шукати файл конфігурації. Якщо аргумент відсутній використовується файл config.json, локальний до шляху, звідки була запущена програма.

- Аргумент виводу на екран усіх доступних цільових застосунків для стрес-тестування -l|--list

Даний аргумент використовується в цілях розуміння користувачем, які засоби стрес-тестування можуть бути використані. Передача цього параметру запускає процес виводу в консоль користувача списку з назв застосунків, які користувач може використати для передачі в аргумент -r|--runner. Після виводу списку програма одразу завершується з кодом завершення 0 (рис. 3.5.2).

```
PS E:\temp> dotnet-stress-cli.exe -l
Found additional 1 runners: TestRunner
k6
TestRunner
```

Рисунок 3.5.2 Приклад з передачею аргументу -l коли в конфігурацію додано плагін з назвою TestRunner

- Аргумент детального логування -v|--verbose

Даний аргумент вказує програмі, що під час виконання потрібно виводити усі повідомлення, навіть інформаційні. Без передачі даного аргументу програма буде логувати тільки помилки (рис. 3.5.3).

```
PS E:\temp> dotnet-stress-cli.exe -v
[17:38:04] - Found additional 1 runners: TestRunner
[17:38:04] - Started init Openapi Parser with path http://localhost:5029/swagger/v1/swagger.json
[17:38:05] - Successfully fetched openapi json
[17:38:05] - ERROR - Declared path parameter "id" needs to be defined as a path parameter at either the path or operation level --- #/paths/~1Weat
herForecast~1{Id}~1{Name}~1{Type}/get/parameters/0/in
[17:38:05] - ERROR - Declared path parameter "name" needs to be defined as a path parameter at either the path or operation level --- #/paths/~1We
atherForecast~1{Id}~1{Name}~1{Type}/get/parameters/1/in
[17:38:05] - ERROR - Declared path parameter "type" needs to be defined as a path parameter at either the path or operation level --- #/paths/~1We
atherForecast~1{Id}~1{Name}~1{Type}/get/parameters/2/in
[17:38:05] - Successfully deserialized openapi json
[17:38:05] - Started parsing openapi
[17:38:05] - Parsing endpoint: Options /someOther/endpoint
[17:38:05] - Forming query params
[17:38:05] - Parsing endpoint: Post /someOther/endpoint
[17:38:05] - Forming query params
[17:38:05] - Forming body
[17:38:05] - Parsing endpoint: Get /WeatherForecast/GetWeatherForecast
[17:38:05] - Forming query params
[17:38:05] - Parsing endpoint: Get /WeatherForecast/GetWeatherForecastById
[17:38:05] - Forming query params
[17:38:05] - Parsing endpoint: Post /WeatherForecast/AddWeatherForecast
[17:38:05] - Forming query params
[17:38:05] - Forming body
[17:38:05] - Parsing endpoint: Post /WeatherForecast/Create
[17:38:05] - Forming query params
[17:38:05] - Forming body
[17:38:05] - Parsing endpoint: Get /WeatherForecast/{Id}/{Name}/{Type}
[17:38:05] - Forming query params
[17:38:05] - Parsing endpoint: Get /WeatherForecast/withObject
[17:38:05] - Forming query params
[17:38:05] - Ended parsing openapi
[17:38:05] - Generating class for SomeOther
[17:38:05] - Generating class for SomeOther
[17:38:05] - Generating class for WeatherForecast
[17:38:05] - Generating class for WeatherForecast
[17:38:05] - Generating class for WeatherForecast
[17:38:05] - Generating class for WeatherForecast
[17:38:05] - Generating class for WeatherForecast
[17:38:05] - Generating class for WeatherForecast
```

Рисунок 3.5.3 Приклад виконнаня програми з аргументом -v, який відповідає за детальне логування в системі

Як видно з рисунку 3.5.3, програма під час виконання логувала у консоль як інформаційні так і повідомлення про помилки.

- Аргумент файлу для логування -f|--outfile <OUTPUTFILE_PATH>

Якщо даний аргумент було передано користувачем, програма буде використовувати передане значення в якості шляху до файлу, в який потрібно буде вести логування стану системи. В разі передачі цього аргументу логування в консоль користувача вестись не буде (рис 3.5.4).

```
PS E:\temp> ls


    Directory: E:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         21-May-23  10:11 AM                output
d-----         20-May-23  10:45 AM                _old
-a----         27-May-23   4:36 PM            116 conf.ps1
-a----         27-May-23   4:58 PM           1172 config.json


PS E:\temp> dotnet-stress-cli.exe --outfile output.log
PS E:\temp> ls


    Directory: E:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         21-May-23  10:11 AM                output
d-----         20-May-23  10:45 AM                _old
-a----         27-May-23   4:36 PM            116 conf.ps1
-a----         27-May-23   4:58 PM           1172 config.json
-a----         27-May-23   5:11 PM           1583 output.log
```

Рисунок 3.5.4 Приклад виконання програми з логуванням у файл

Як видно з рисунку 3.5.4, було створено файл output.log, розмір якого 1583 байти.

- Аргумент відключення логування --no-logs

Даний аргумент вказує програмі, що логування не повинно вестись ні в якому вигляді.

Далі приведена діаграма діяльності основного алгоритму роботи (рис. 3.5.5).

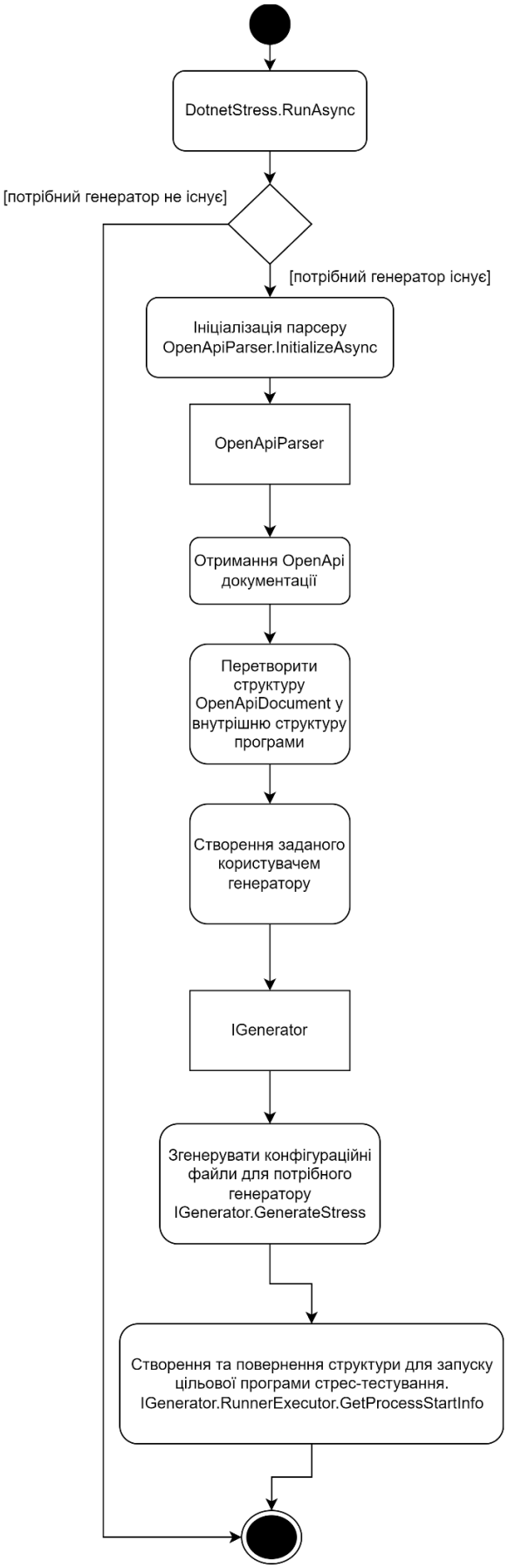


Рисунок 3.5.5 Діаграма активності основного алгоритму

Починається все з виклику методу RunAsync клас DotnetStress. Цей метод перевіряє, чи існує заданий користувачем цільовий додаток стрес-тестування (надалі в цьому прикладі - генератор). Якщо не існує - програма завершує роботу. Якщо існує – виповнюється ініціалізація парсеру. Під час ініціалізації завантажується документація у форматі OpenApi та перетворюється у внутрішні класи застосунку. Після цього створюється клас-генератор конфігурації для генератору та відбувається виклик методу IRunner.GenerateStress для генерування конфігураційних файлів. Далі відбувається виклик методу IRunner.RunnerExecutor.GetProcessStartInfo для отримання інформації для запуску генератору, яка повертається і обробляється консольним застосунком.

Далі наведена діаграма діяльності при передачі користувачем аргументу -l|--list (рис. 3.5.7)

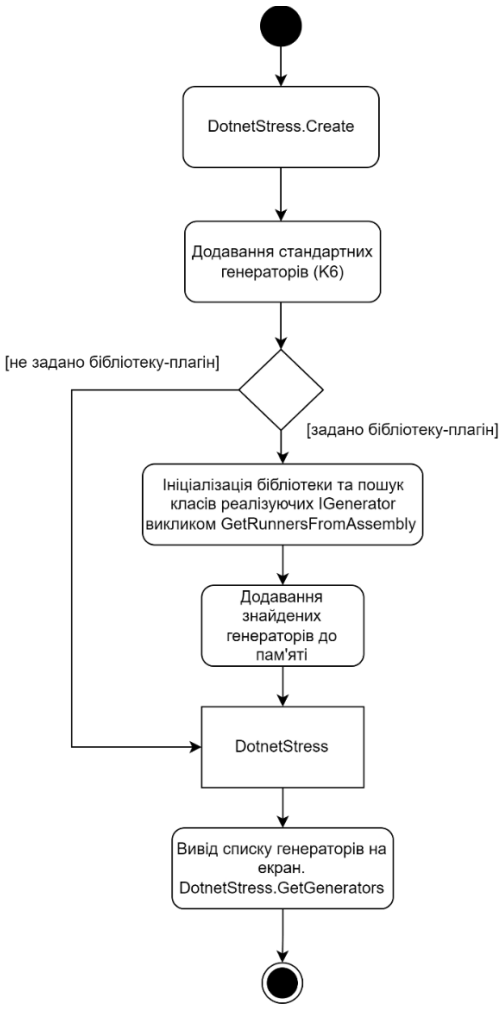


Рисунок 3.5.7 Діаграма активностей для виводу списку генераторів

Спочатку іде ініціалізація класу DotnetStress шляхом виклику статичного методу Create. Під час роботи цього методу спочатку додаються імплементовані генератори, тобто генератор для Grafana K6. Після цього йде перевірка чи було задано користувачем шляху до бібліотеки-плагіну. Якщо шлях було задано, то бібліотека завантажується у пам'ять та в ній відбувається пошук класів, реалізуючих інтерфейс IGenerator, викликом приватного методу DotnetStress.GetRunnersFromAssembly. Знайдені класи генераторів додаються у пам'ять та метод Create повертає створений об'єкт DotnetStress. Після цього консольний застосунок викликає метод класу DotnetStress GetGenerators для отримання доступних генераторів та повертає їх назви викликаючи IGenerator.Runner кожного та припинає роботу застосунку.

3.6 Тестування застосунку

Тестування застосунку відбувалося за допомогою бібліотеки xUnit.

Для тестування алгоритмів були створені юніт-тести на парсинг OpenApi документації та на генерацію конфігурації.

Також було створено проєкт веб-застосунку на основі фреймворку ASP.NET 7, в якому було створено API, використане для тестування. Використання власно-створеного API дало змогу дуже легко на гнучко тестувати та одразу виправляти помилки під час розробки. Що дозволяє пришвидшити розробку програми за рахунок зменшення покриття коду тестами.

В додатку А представлено результати роботи розробленого програмного забезпечення. Для тестування було створено веб API, яке складається з 8 кінцевих точок для запитів. В результаті було отримано 4 файли для цільової програми стрес-тестування K6. Сумарна кількість рядків коду становить 161, що для людини може зайняти близько двох годин роботи.

ВИСНОВКИ

Відповідно до поставленої мети та задач бакалаврського дослідження «Розробка програмного забезпечення для OpenAPI автоматичної генерації стрес-тестів на основі мовою C#» отримано наступні результати: проведено огляд та аналіз літературних джерел, методик існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків; проаналізовані програмні рішення, що використовуються на поточний момент для автоматичного тестування та/або автоматичного генерування тестових сценаріїв для автоматизації тестування веб-додатків; на основі переваг та недоліків існуючих технологій тестування сформовано вимоги та розроблено проєкт нового застосунку для автоматичної генерації стрес-тестів на основі OpenApi мовою C#; обґрунтовано вибір технологій для розробки програмного забезпечення веб-додатку; розроблено додаток і здійснено його тестування. Для реалізації програмного забезпечення було використано мову програмування C#, бібліотеку для тестування xUnit, фреймворк для розробки тестового API ASP.NET 7. Для роботи з форматом даних JSON було використано бібліотеку JSON.NET (яку ще називають Newtonsoft.Json). Для роботи зі стандартом OpenApi було використано бібліотеку Microsoft.OpenApi та Swagger. Для створення інтерфейсу командного рядку було використано бібліотеку CommandLineUtils. В якості цільового застосунку для стрес-тестування було використано Grafana K6.

Результати проведеного дослідження дають підстави зробити такі висновки:

1. Тестування – необхідний процес у розробці програмного забезпечення, відіграє важливу роль у забезпеченні якості коду та зменшення ризику помилок у програмі продукту.

2. Сучасні додатки автоматизованого стрес-тестування Quotium QTest, OpenText Silk Performer, BM Rational Performance Tester, Curiosity Test Modeller є пропрієтарними і жоден з них не має функціоналу інтегрування зі сторонніми засобами стрес-тестування, що підтверджує актуальність даного дослідження.

3. Створений програмний продукт у вигляді консольної утиліти, яка забезпечить автоматичну генерацію стрес-тестів для API веб-застосунків на основі специфікацій API та у форматі OpenAPI, а також генерацію звітів з результатами тестування та оцінкою продуктивності розробленого програмного забезпечення.

4. Тестування додатку показало задовільні результати.

5. Розроблене програмне забезпечення можна використовувати як для тестування реальних застосунків, так і для навчання. Застосунок є корисним розробникам, тестувальникам, замовникам веб-додатків для перевірки системи на швидкість роботи без витрати великих зусиль для ручного створення усіх необхідних тестових сценаріїв, а також у процесі вивчення дисципліни «Якість програмного забезпечення та тестування» студентами спеціальності 121 Інженерія програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Raúl R. 15 years of software regression testing techniques—a survey [Електронний ресурс] / R. Raúl, O. Gómez, G. Rodríguez // International Journal of Software Engineering and Knowledge Engineering. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/279530047_15_Years_of_Software_Regression_Testing_Techniques_--_A_Survey.
2. Penetration Testing and Vulnerability Assessment [Електронний ресурс] / [I. Yaqoob, S. Hussain, S. Mamoon та ін.] // Journal of Network Communications and Emerging Technologies (JNCET). – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/profile/Irfan-Yaqoob-2/publication/349077887_Penetration_Testing_and_Vulnerability_Assessment/links/601ea917299bf1cc26a981c5/Penetration-Testing-and-Vulnerability-Assessment.pdf.
3. Delamaro M. Interface mutation: An approach for integration testing [Електронний ресурс] / M. Delamaro, J. Maidonado, A. Mathur // IEEE transactions on software engineering. – 2001. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/910859.
4. Melegati J. What influences software startups to use lean startup? [Електронний ресурс] / Jorge Melegati // Proceedings of the 19th International Conference on Agile Software Development: Companion. – 2018. – Режим доступу до ресурсу: https://dl.acm.org/doi/abs/10.1145/3234152.3314990.
5. Martin R. C. Clean code: a handbook of agile software craftsmanship / Robert Cecil Martin., 2008. – 464 с. – (Pearson). – (1).
6. Sawalha S. Agile software development: Methodologies and trends [Електронний ресурс] / S. Sawalha, H. AbdelNabi, A. Samar // International Journal of Interactive Mobile Technologies. – 2020. – Режим доступу до ресурсу: https://pdfs.semanticscholar.org/2fef/154748093288894dbd0b98db1b9b54731c71.pdf.
7. Izquierdo C. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach [Електронний ресурс] / Cánovas Izquierdo // IEEE 22nd international

enterprise distributed object computing conference. – 2018. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/8536162/.

8. Comparison between Progressive Web App and Regular Web App. Diss. [Електронний ресурс] // Department of Software Engineering. – 2019. – Режим доступу до ресурсу: https://www.diva-portal.org/smash/get/diva2:1334458/FULLTEXT02.pdf.

9. Бібліотека CommandLineUtils [Електронний ресурс] – Режим доступу до ресурсу: https://natemcmaster.github.io/CommandLineUtils/.

10. Порівняння Newtonfosft.Json до аналогів [Електронний ресурс] – Режим доступу до ресурсу: https://www.newtonsoft.com/json.

11. Основні особливості застосунку K6 [Електронний ресурс] – Режим доступу до ресурсу: https://k6.io/docs/#key-features.

12. Можливості K6 працювати з результатами у реальному часі [Електронний ресурс] – Режим доступу до ресурсу: https://k6.io/docs/results-output/real-time/.

13. Стандарт OpenApi [Електронний ресурс] – Режим доступу до ресурсу: https://spec.openapis.org/oas/v3.1.0.

14. Schwichtenberg S. From open API to semantic specifications and code adapters [Електронний ресурс] / S. Schwichtenberg, C. Gerth, G. Engels // IEEE International Conference on Web Services (ICWS). – 2017. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/8029798/.

15. Технологія OpenApi Generator [Електронний ресурс] – Режим доступу до ресурсу: https://openapi-generator.tech/.

16. Порівняння швидкості роботи C# та C++ [Електронний ресурс] – Режим доступу до ресурсу: https://programming-language-benchmarks.vercel.app/cpp-vs-csharp.

17. Документація атрибуту який використовується для генерації вихідного коду зі стандартної бібліотеки [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.generatedregexattribute.

18. Опис CLR в .NET [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/standard/clr.

19. Індекс TIOБE [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.

20. Engström H. A taxonomy of game engines and the tools that drive the industry. [Електронний ресурс] / H. Engström, M. Toftedahl // The 12th Digital Games Research Association Conference. – 2019. – Режим доступу до ресурсу: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1352554..

21. Звіт UNITY за 2023 рік [Електронний ресурс] – Режим доступу до ресурсу: https://cdn.unity3d.com/media/2023%20Gaming%20Report.pdf.

ДОДАТОК А

Результати тестування застосунку

```
PS E:\temp> dotnet-stress-cli -v -e
[13:35:42] - Found additional 1 runners: TestRunner
[13:35:42] - Started init Openapi Parser with path http://localhost:5029/swagger/v1/swagger.json
[13:35:43] - Successfully fetched openapi json
[13:35:43] - ERROR - Declared path parameter "id" needs to be defined as a path parameter at either the path or operatio
n level --- #/paths/~1WeatherForecast~1{Id}~1{Name}~1{Type}/get/parameters/0/in
[13:35:43] - ERROR - Declared path parameter "name" needs to be defined as a path parameter at either the path or operat
ion level --- #/paths/~1WeatherForecast~1{Id}~1{Name}~1{Type}/get/parameters/1/in
[13:35:43] - ERROR - Declared path parameter "type" needs to be defined as a path parameter at either the path or operat
ion level --- #/paths/~1WeatherForecast~1{Id}~1{Name}~1{Type}/get/parameters/2/in
[13:35:43] - Successfully deserialized openapi json
[13:35:43] - Started parsing openapi
[13:35:43] - Parsing endpoint: Options /someOther/endpoint
[13:35:43] - Forming query params
[13:35:43] - Parsing endpoint: Post /someOther/endpoint
[13:35:43] - Forming query params
[13:35:43] - Forming body
[13:35:43] - Parsing endpoint: Get /WeatherForecast/GetWeatherForecast
[13:35:43] - Forming query params
[13:35:43] - Parsing endpoint: Get /WeatherForecast/GetWeatherForecastById
[13:35:43] - Forming query params
[13:35:43] - Parsing endpoint: Post /WeatherForecast/AddWeatherForecast
[13:35:43] - Forming query params
[13:35:43] - Forming body
[13:35:43] - Parsing endpoint: Post /WeatherForecast/Create
[13:35:43] - Forming query params
[13:35:43] - Forming body
[13:35:43] - Parsing endpoint: Get /WeatherForecast/{Id}/{Name}/{Type}
[13:35:43] - Forming query params
[13:35:43] - Parsing endpoint: Get /WeatherForecast/withObject
[13:35:43] - Forming query params
[13:35:43] - Ended parsing openapi
[13:35:43] - Generating class for SomeOther
[13:35:43] - Generating class for SomeOther
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Generating class for WeatherForecast
[13:35:43] - Executing runner

  execution: local
     script: main.js
     output: -

  scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
           * contacts: Up to 30 looping VUs for 30s over 1 stages (gracefulRampDown: 30s, gracefulStop: 30s)


running (0m31.6s), 00/30 VUs, 310 complete and 0 interrupted iterations
contacts   [======================================] 00/30 VUs  30s

     SomeOther

     WeatherForecast

     data_received..................: 684 kB  22 kB/s
     data_sent......................: 1000 kB 32 kB/s
     group_duration.................: avg=993.97ms min=231.34ms med=862.16ms max=2.15s    p(90)=1.84s    p(95)=1.92s
     http_req_blocked...............: avg=852.51µs min=0s       med=0s       max=228.51ms p(90)=0s       p(95)=0s
     http_req_connecting............: avg=13.24µs  min=0s       med=0s       max=3.61ms   p(90)=0s       p(95)=0s
     http_req_duration..............: avg=123.32ms min=6.16ms   med=115.63ms max=381.63ms p(90)=229.37ms p(95)=253.75ms
       { expected_response:true }...: avg=123.32ms min=6.16ms   med=115.63ms max=381.63ms p(90)=229.37ms p(95)=253.75ms
     http_req_failed................: 0.00%   0 0           4960
     http_req_receiving.............: avg=160.38µs min=0s       med=0s       max=4.06ms   p(90)=653.4µs  p(95)=872.4µs
     http_req_sending...............: avg=34.59µs  min=0s       med=0s       max=1.3ms    p(90)=0s       p(95)=505.91µs
     http_req_tls_handshaking.......: avg=821.95µs min=0s       med=0s       max=227.82ms p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=123.13ms min=5.25ms   med=115.2ms  max=381.63ms p(90)=229.37ms p(95)=253.75ms
     http_reqs......................: 4960    157.202843/s
     iteration_duration.............: avg=1.98s    min=1.1s     med=2s       max=2.92s    p(90)=2.59s    p(95)=2.63s
     iterations.....................: 310     9.825178/s
     vus............................: 18      min=10       max=29
     vus_max........................: 30      min=30       max=30

ERRO[0032] some thresholds have failed
[13:36:17] - Runner execution finished. Please see output directory for results.
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для автоматичної генерації стрес-тестів на основі OpenApi мовою C#

Виконав студент 4 курсу

групи ПД-41
Прокопчук Кирило Костянтинович
Керівникроботи

К.п.н, доц, доцент Шевченко Світлана Миколаївна

Київ – 2023

Об’єкт, предмет, мета бакалаврської роботи.

- Мета – підтримка процесу створення стрес-тестів у автоматичному режимі за допомогою додатку мовою C# та використанням стандарту OpenApi
- Об’єкт – процес створення стрес-тестів
- Предмет – програмне забезпечення для автоматичної генерації стрес-тестів

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз літературних джерел, методик існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків.
2. Проаналізувати програмні рішення, що використовуються на поточний момент для автоматичного тестування та/або автоматичного генерування тестових сценаріїв для автоматизації тестування веб -додатків.
3. Дослідити характеристики, переваги та недоліки існуючих засобів для автоматичної генерації стрес-тестів для веб-додатків.
4. Провести огляд IT -засобів, які можуть бути використані при розробці програмного забезпечення для автоматичної генерації стрес -тестів для веб-додатків.
5. Розробити програмне забезпечення для автоматичної генерації стрес -тестів для веб-додатків.

АНАЛІЗ АНАЛОГІВ

Назва застосунку	Відкритий(не пропрієтарний)	Має веб-інтерфейс	Завантаження прикладів даних	Підхід для автоматизації	Автоматизація стрес тестування	Інтеграція з іншими застосунками для проведення тестування
Quotium QTes	-	-	-	На основі спостережен	-	-
OpenText Silk Performer	-	+	-	На основі спостережен	+	-
IBM Rational Performance Tester	-	-	+	На основі спостережен	+	-
Curiosity Test Modeller	-	+	+	На основі даних	+	-
Dotnet-stress	+	-	+	На основі даних	+	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

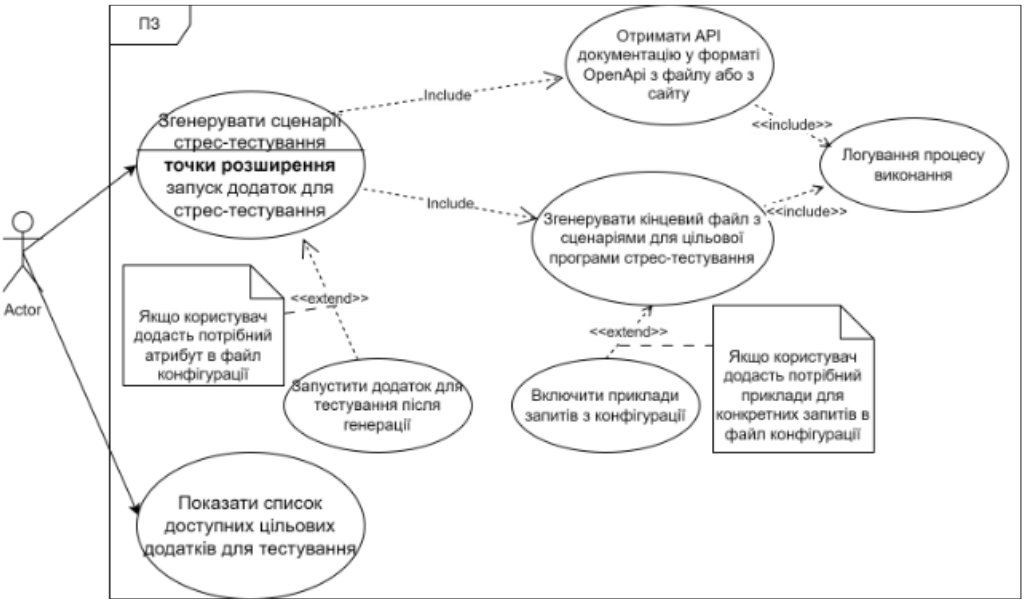
1. Автоматична генерація параметрів запитів.
2. Інтеграція з ПЗ для стрес-тестування – Grafana K6
3. Можливість використання заданих користувачем об’єктів для запитів.
4. Логування помилок та процесу роботи застосунку
5. Можливість додавати інтеграції з іншими додатками для тестування у виглядіdll файлів-плагінів.
6. Можливість задання нативної конфігурації додатку для тестування
7. Архітектура має дозволяти створювання похідних додатків на базі основної бібліотеки без внесення змін в основу бібліотеку ПЗ.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

Діаграма варінтів використання

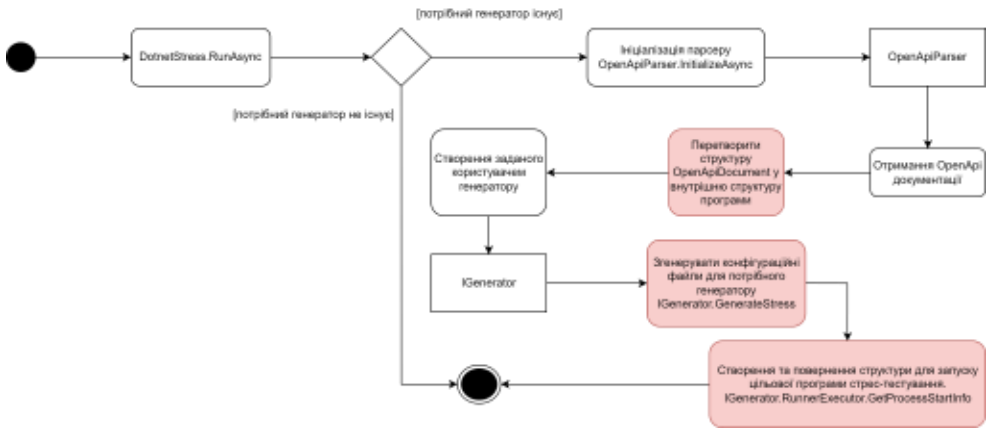


7

Діаграма класів

8

Діаграма діяльності



9

Формат файлу конфігурації

```
{
    "HostUrl" : "http://localhost:5029",
    "SwaggerJsonUrl" : "http://localhost:5029/swagger/v1/swagger.json",
    "RunnerProgram": "k6",
    "RunnerPluginDllPath": "E:/Desktop/Projects/dotnet-stress/src/dotnet-stress/TestpluginAssembly/bin/Debug/net7.0/TestpluginAssembly.dll",
    "SubstitutionConfig" : {
            "GET /WeatherForecast/GetComplex":{
                "query":"?id=21345&name=someCoolName&type=1"
            },
            "POST /WeatherForecast/AddWeatherForecast": {
                "body": {
                    "Test1": 12345,
                    "Field2": 123456,
                    "Obj": {
                        "SomeCollInside": "12345655yrgrtbet"
                    }
                }
            }
    },
    "ObligatoryHeaders":{
        "Authorization": "Bearer 123456"
    },
    "runner_options": {
        "discardResponseBodies": true,
        "scenarios": {
        "thresholds" : {
    }
}
```

10

Список команд для запуску додатку

```
PS E:\temp> dotnet-stress-cli.exe --help
Usage:  [options]

Options:
  -?|-h|--help                Show help information.
  -e|--exec                   Whether to run target stress application. Requires a valid path or for the program to be
                              accessible from PATH
                              Default value is: False.
  -r|--runner <string>        Desired runner name
  -c|--config <CONFIG_PATH>   Indicates where the config is. Default is ./config.json
                              Default value is: ./config.json.
  -l|--list                   List runners
  -v|--verbose                If set will show all output. If not set, only warnings and errors are shown
  -f|--outfile <OUTFILE_PATH> Sets file to write output to. Default is console
  --no-logs                   If set won't write any logs anywhere.
```

11

Робота додатку з логуванням та запуском цільової програми стрес-тестування

```
[20:26:34] - Executing runner
  execution: local
     script: main.js
     output: -

  scenarios: (100.00%) 1 scenario, 30 max VUs, 1m0s max duration (incl. graceful stop):
           * contacts: Up to 30 looping VUs for 30s over 1 stages (gracefulRampDown: 30s, gracefulStop: 30s)


running (0m31.9s), 00/30 VUs, 254 complete and 0 interrupted iterations
contacts ✓ [======================================] 00/30 VUs  30s

     ✓ SomeOther

     ✓ WeatherForecast

     data_received..................: 569 kB 18 kB/s
     data_sent......................: 805 kB 25 kB/s
     group_duration.................: avg=1.23s    min=293.74ms med=1.02s    max=2.83s    p(90)=2.35s    p(95)=2.52s
     http_req_blocked...............: avg=1.04ms   min=0s       med=0s       max=298.22ms p(90)=0s       p(95)=0s
     http_req_connecting............: avg=9.41µs   min=0s       med=0s       max=1.9ms    p(90)=0s       p(95)=0s
   ✓ http_req_duration..............: avg=153.08ms min=12.78ms  med=140.98ms max=631.24ms p(90)=293.62ms p(95)=332.72ms
     { expected_response:true }...: avg=153.08ms min=12.78ms  med=140.98ms max=631.24ms p(90)=293.62ms p(95)=332.72ms
     http_req_failed................: 0.00%  ✓ 0          ✗ 4064
     http_req_receiving.............: avg=143.48µs min=0s       med=0s       max=2.03ms   p(90)=643.1µs  p(95)=844.79µs
     http_req_sending...............: avg=25.48µs  min=0s       med=0s       max=1.02ms   p(90)=0s       p(95)=32.13µs
     http_req_tls_handshaking.......: avg=1.01ms   min=0s       med=0s       max=296.75ms p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=152.92ms min=11.86ms  med=140.63ms max=631.24ms p(90)=293.62ms p(95)=332.51ms
     http_reqs......................: 4064   127.469577/s
     iteration_duration.............: avg=2.46s    min=1.28s    med=2.51s    max=3.75s    p(90)=3.43s    p(95)=3.5s
     iterations.....................: 254    7.966848/s
     vus............................: 25     min=10       max=30
     vus_max........................: 30     min=30       max=30

ERRO[0032] some thresholds have failed
[20:27:07] - Runner execution finished. Please see output directory for results.
```

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Прокопчук К.К. Розробка програмного забезпечення для автоматизованої генерації стрес-тестів. Матеріали X Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології - 2023». 18.05.2023, м. Київ. – К.: Київський університет імені Бориса Грінченка, 2023. – С. 141 – 142.

   https://zcit.kubg.edu.ua/index.php/journal/issue/view/ 11/21

2. Прокопчук К.К., Негоденко О.В., Шевченко С.М. Генерація стрес-тестів за допомогою OpenApi мовою C#. – готується до друку

13

ВИСНОВКИ

1. Проаналізовані застосунки для автоматичного та/або автоматизованого стрес-тестування. Ключовими недоліками існуючих систем є пропрієтарність та той факт що вони є окремими програмними засобами для проведення тестування, без можливостей інтеграції з іншими.
2. Проведено аналіз засобів розробки програмного забезпечення, обгрунтовано їх вибір, а саме: мова програмування C#, цільовий засіб стрес-тестування Grafana K6, бібліотека тестування xUnit, фреймворк для розробки API ASP.NET, бібліотека роботи з форматом даних JSON JSON.NET, бібліотеку Microsoft.OpenApi та Swagger для роботи зі стандартом OpenApi та бібліотеку CommandLineUtils для написання інтерфейсу командного рядку.
3. Розроблено: програмне забезпечення, яке дозволяє зменшити необхідні ресурси та час на розробку стрес-тестів для API веб-застосунків.

14