НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Програмне забезпечення для автогенерації тестів за допомогою LLM

Виконав : студент __4__ курсу, групи _ІО-06_
(шифр групи)

_____Мотора Владислав Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ас. Нечай Д.О._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)_____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Мотори Владислава Сергійовича

1. Тема Програмне забезпечення для автогенерації тестів за допомогою LLM
керівник проєкту Нечай Д. О. Асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____2024 року №____

2. Термін здачі студентом закінченого проєкту *8 червня 2024 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд існуючих рішень для автогенерації автотестів

Розділ 2. Огляд інструментів для написання програми

Розділ 3. Деталі розробки системи

Розділ 4. Дослідження та аналіз розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) логічна схема системи, діаграма дата-акторів, діаграма повідомлень в консолі.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Ковальчук О. М.		

7. Дата видачі завдання «16» жовтня.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*14.02.2024-05.04.2024*	
2.	*Вивчення та аналіз завдання*	*06.04.2024-14.04.2024*	
3.	*Розробка архітектури та загальної структури системи*	*15.04.2024-24.04.2024*	
4.	*Розробка структур окремих підсистем*	*25.04.2024-4.05.2024*	
5.	*Програмна реалізація системи*	*5.05.2024-15.05.2024*	
6.	*Оформлення пояснювальної записки*	*16.05.2024-19.05.2024*	
7.	*Захист програмного продукту*	*25.05.2024*	
8.	*Передзахист*	*15.06.2024*	
9.	*Захист*	*22.06.2024*	

Студент-дипломник _____ Мотора В. С.
(підпис)

Керівник проєкту _____ Нечай Д. О.
(підпис)

АНОТАЦІЯ

У даній дипломній роботі пропонується розробка Програмного продукту для генерації автоматичних тестів за допомогою LLM.

Документ описує архітектуру та функціональність продукту, включаючи його здатність створювати, перевіряти та валідувати автоматичних тестів. Програма забезпечує інтеграцію з LLM-моделлю ChatGPT, використовує Mocha для перевірки роботоздатності та Stryker для валідації. Застосування фреймворку WebdriverIO забезпечує надійність і масштабованість при інтеграції згенерованих автоматичних тестів.

Важливою частиною роботи є реалізація механізмів генерації тестів, запуску тестів та їх валідація. Крім того програмне забезпечення прямо інтегрується в середовище розробки для зручності налаштування та роботи з ним.

Для оцінки ефективності та зручності використання програми проводяться тести та аналіз результатів її роботи у різних умовах. Нарешті, дипломна робота демонструє важливість і можливості використання подібних сервісів для оптимізації процесів написання автоматичних тестів та підвищення загальної ефективності покриття тестами кінцевого продукту.

ANNOTATION

This thesis proposes the development of a software product for the generation of automatic tests using LLM.

The document describes the architecture and functionality of the product, including its ability to create, verify, and validate automated tests. The service provides integration with the ChatGPT LLM model, uses Mocha for functionality testing and Stryker for validation. The use of the WebdriverIO framework ensures reliability and scalability when integrating generated automatic tests.

An important part of the work is the implementation of test generation mechanisms, test launch and their validation. In addition, the software is directly integrated into the development environment for ease of configuration and operation.

To evaluate the effectiveness and ease of use of the program, tests and analysis of the results of its operation in various conditions are carried out. Finally, the thesis demonstrates the importance and possibilities of using such services to optimize the processes of writing automatic tests and increase the overall efficiency of test coverage of the final product.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Програмне забезпечення	4		
			для автогенерації тестів			
			за допомогою LLM			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Програмне забезпечення	63		
			для автогенерації тестів			
			за допомогою LLM			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Програмне забезпечення	1		
			для автогенерації тестів			
			за допомогою LLM			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.467200.005 Д2***	Програмне забезпечення	1		
			для автогенерації тестів			
			за допомогою LLM			
			Діаграма дата-акторів			
	А4	***ІАЛЦ.467200.006 Д3***	Програмне забезпечення	1		
			для автогенерації тестів			
			за допомогою LLM			
			Діаграма повідомлень			
			в консолі			
	А4	***ІАЛЦ.467200.007 Д4***	Програмне забезпечення	15		
			для автогенерації тестів			
			за допомогою LLM			
			Текст програмного коду			

					# *ІАЛЦ.467200.001 ОА*		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Мотора В. С.			Програмне забезпечення для автогенерації тестів за допомогою LLM Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Нечай Д.О.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-06	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмне забезпечення для автогенерації тестів за допомогою LLM»

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Областю застосування розроблюваної системи є програмний продукт, призначений для інженерів-автоматизаторів з контролю якості. Засіб дозволяє інженерам створювати, запускати та валідувати автоматичні тести. Система надає можливість інтеграції з будь-яким популярним тестранером. Програма забезпечує інтеграцію з LLM-моделлю ChatGPT, використовує Mocha для перевірки роботоздатності та Stryker для валідації. Застосування фреймворку WebdriverIO забезпечує надійність і масштабованість при інтеграції згенерованих автоматичних тестів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Основною метою розробки засобу є створення надійного, ефективного і зручного інструменту для створення автоматичних тестів, вирішення проблеми пришвидшення розробки програмних продуктів через застосування інженерами-розробниками LLM-моделей. Це включає в себе можливість створення, перевірки та валідації автотестів. Використання LLM-моделі ChatGPT забезпечує надійність, масштабованість та гнучкість. Система також підтримує інтеграцію у вже існуючий проект. Це допоможе інженерам оптимізувати процес розробки автотестів та підвищити ефективність роботи.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література, технічна документація, публікації в періодичних виданнях та мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Система, що розроблюється, повинна відповідати поставленим вимогами, що наведені нижче:

- Верифікація та валідація згенерованих тестів.
- Зручна взаємодія із згенерованими тестами
- Зрозуміла відповідь на всі типи вхідних данних, включаючи згенеровану LLM відповідь на запит

5.2. Вимоги до програмного забезпечення

- Операційна система Windows/MacOS
- Встановлена платформа Node.js для розробки проектів для тестування продуктів.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Назва етапу виконання:	Термін виконання:
Затвердження теми проекту	14.02.2024-05.04.2024
Вивчення та аналіз завдання	06.04.2024-14.04.2024
Розробка архітектури та загальної структури системи	15.04.2024-04.05.2024
Розробка структур окремих підсистем	25.04.2024-15.05.2024
Програмна реалізація системи	5.05.2024-15.05.2024
Оформлення пояснювальної записки	16.05.2024-19.05.2024
Захист програмного продукту	25.05.2024
Предзахист	15.06.2024
Захист	22.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмне забезпечення для автогенерації тестів за допомогою LLM»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

LLM (Large Language Models) — це великі мовні моделі, які використовують машинне навчання для обробки та генерації природної мови. Вони здатні аналізувати текст, розуміти контекст та генерувати відповіді на основі великих обсягів даних.

Автотести (Automated test)

Автотести — це автоматизовані тести, які виконуються програмним забезпеченням для перевірки функціональності, продуктивності та безпеки іншого програмного забезпечення. Вони дозволяють швидко виявляти помилки та забезпечувати якість продукту. Автотести значно прискорюють процес тестування на різних етапах розробки.

API (Application Programming Interface)

API — це набір правил і протоколів, що дозволяють різним програмам взаємодіяти між собою. API визначає методи та формати обміну даними, забезпечуючи інтеграцію та комунікацію між різними програмними компонентами. Вони широко використовуються для зв'язку між серверами, додатками та сервісами.

Фреймворк — зовнішній програмний продукт.

JSON (JavaScript Object Notation) — це компактний формат для обміну даними, який зручний для читання та написання як людьми, так і машинами. Він є текстовим форматом, що використовується для представлення структурованих даних, базуючись на синтаксисі об'єктів JavaScript, але при цьому залишається незалежним від конкретної мови програмування.

Промпт (Prompt) — це вхідний текст або запит, який подається до моделі штучного інтелекту, щоб отримати відповідь або завершення. Він служить як інструкція або контекст для генерації відповідного тексту моделлю.

ВСТУП

У сучасному світі програмування, особливо в контексті швидкого розвитку технологій та збільшення обсягів програмного забезпечення, важливість ефективного контролю якості стає все більш актуальною. Інженери з контролю якості постійно шукають нові методи та інструменти для оптимізації процесів тестування. Однією з перспективних областей у цьому контексті є використання штучного інтелекту, зокрема моделей великих мовних моделей (Large Language Models, LLM), для автоматизації створення тестових сценаріїв.

Розробка такого програмного забезпечення перш за все має на меті зменшити час покриття тестами нового функціоналу, адже інженери-розробники вже застосовують ШІ для генерації коду для нового функціоналу. Через зменшення часу на написання нового коду для продукту, баланс між кодом продукту, та кодом який перевіряє роботу коду продукту змінюється в сторону коду продукту, що збільшує тиск на інженерів з контролю якості, що в свою чергу впливає на якість кінцевого продукту.

Використання ШІ завжди несе в собі ризик що мовна модель галюцинує. Галюцинація мовної моделі це явище, коли штучний інтелект, зокрема мовна модель, генерує інформацію, яка не є точною або не базується на достовірних даних. Це може включати в себе створення неправдивих фактів, нелогічних висновків або неконсистентних відповідей, які не відповідають дійсності або очікуванням. Якщо для розроки нового функціоналу перевірка проста – тобто просто запуск коду, то для автотестів ця перевірка складніша, бо навіть для автотестів написаних інженерами вручну проводяться перевірки валідності тестів.

Завданням дипломного проекту є розробка програмного забезпечення яке буде генерувати і відразу валідувати створені тести для подальшого використання інженером.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ АВТОГЕНЕРАЦІЇ АВТОТЕСТІВ

Автоматизація тестування відіграє вирішальну роль у підвищенні продуктивності розробки програмного забезпечення та забезпеченні його якості. З появою технологій штучного інтелекту, зокрема методів машинного навчання та обробки природної мови, відкриваються нові можливості для автоматизації створення тестових сценаріїв. Аналіз існуючих рішень у сфері автогенерації автотестів дозволяє оцінити різноманітні підходи та технології, що використовуються для цієї мети.

Розгляд ключових систем та платформ, що застосовуються в індустрії для генерації тестових випадків, включає аналіз їхніх методологій, переваг та обмежень. Важливо відзначити, як ці системи інтегруються з існуючими процесами розробки та які технічні та технологічні виклики вони долають. Особлива увага приділяється використанню штучного інтелекту, зокрема мовних моделей, для оптимізації процесів генерації тестів.

1.1 Accelq

AccelQ є інноваційною платформою для автоматизації тестування, яка дозволяє користувачам ефективно управляти циклом тестування програмного забезпечення. Ця платформа використовує безкодовий підхід, що робить її доступною для тестувальників без глибоких знань у програмуванні. AccelQ підтримує автоматизацію тестування веб-додатків, мобільних додатків та API, інтегруючись з різними середовищами розробки та системами CI/CD. [1]

Плюси AccelQ:

- Безкодова автоматизація: AccelQ дозволяє тестувальникам створювати та управляти тестовими сценаріями без необхідності

писати код, що спрощує процес тестування та знижує поріг входження для нових користувачів.

- Інтеграція з CI/CD: Платформа легко інтегрується з популярними системами безперервної інтеграції та доставки, що дозволяє автоматизувати тестування в рамках DevOps процесів.
- Підтримка різних платформ: AccelQ підтримує тестування веб-додатків, хмарних сервісів та мобільних додатків, забезпечуючи широкий спектр тестувальних можливостей.
- Візуальне моделювання: Платформа використовує візуальне моделювання для створення тестових сценаріїв, що дозволяє користувачам легко візуалізувати та розуміти тестові процеси.
- Аналітика та звітність: AccelQ надає розширені можливості аналітики та звітності, що допомагає командам відстежувати прогрес тестування та ідентифікувати ключові проблеми.

Мінуси AccelQ:

- Вартість: Як комерційний продукт, AccelQ може бути відносно дорогим рішенням, особливо для малих та середніх підприємств або стартапів.
- Крива навчання: Незважаючи на безкодовий підхід, новим користувачам може знадобитися час, щоб повністю освоїти всі можливості платформи та ефективно їх використовувати.
- Залежність від платформи: Компанії, які інтегрують AccelQ у свої процеси, можуть стати залежними від цього інструменту, що може ускладнити перехід на інші рішення в майбутньому.

AccelQ є потужним інструментом для автоматизації тестування, який пропонує значні переваги для підвищення ефективності та якості процесів тестування в компаніях. Його безкодовий підхід, широкі можливості інтеграції

та підтримка різних платформ роблять його відмінним вибором для багатьох організацій. Однак, потенційні користувачі повинні враховувати вартість інструменту та потребу в навчанні при прийнятті рішення про його впровадження.

1.2 ReadyAPI Platform

ReadyAPI Platform від SmartBear є комплексним рішенням для тестування API, яке підтримує REST, SOAP, GraphQL та інші веб-сервіси. Ця платформа дозволяє розробникам і тестувальникам ефективно створювати, управляти та виконувати автоматизовані тести, забезпечуючи високу якість інтерфейсів програмування додатків. ReadyAPI об'єднує в собі кілька ключових інструментів, включаючи SoapUI для функціонального тестування, LoadUI для тестування продуктивності, і Secure для тестування безпеки API. [2]

Плюси ReadyAPI Platform:

- Інтегровані можливості: ReadyAPI надає все необхідне для комплексного тестування API, включаючи функціональне тестування, тестування продуктивності та безпеки, що дозволяє користувачам зосередитися на одному інструменті замість використання декількох рішень.
- Легка інтеграція: ReadyAPI може легко інтегруватися з іншими інструментами та сервісами, такими як Swagger для документації API, а також з популярними системами CI/CD, що сприяє автоматизації процесів розробки.
- Підтримка різних протоколів: Платформа підтримує широкий спектр протоколів і стандартів API, що робить її універсальним інструментом для різних типів додатків.

Мінуси ReadyAPI Platform:

- Висока вартість: Як комерційний продукт, ReadyAPI може бути дорогим для деяких організацій, особливо для малих та середніх підприємств.
- Складність: Через свої розширені можливості та численні функції, ReadyAPI може здатися складним для нових користувачів або тим, хто не має досвіду в тестуванні API.
- Ресурсоємність: Для ефективної роботи ReadyAPI може вимагати значних ресурсів системи, особливо при виконанні складних тестів на продуктивність або великих наборів даних.

ReadyAPI Platform є потужним інструментом для комплексного тестування API, який забезпечує розробникам і тестувальникам великі можливості для забезпечення якості та безпеки веб-сервісів. Його інтегрованість, підтримка різних протоколів і легкість інтеграції роблять його відмінним вибором для великих організацій та проектів, які вимагають високого рівня автоматизації тестування. Однак, висока вартість і складність можуть бути перешкодою для деяких компаній, особливо з обмеженим бюджетом або без спеціалізованих знань у тестуванні API.

1.3 Swagger_meqa

Swagger Meqa є інструментом, розробленим для автоматизації тестування API, який використовує специфікації Swagger (OpenAPI) для генерації тестових випадків. Цей інструмент дозволяє розробникам та тестувальникам ефективно створювати тестові сценарії, що забезпечують глибоке покриття функціональності API, виявляючи потенційні помилки та вразливості. Swagger Meqa аналізує структуру API і на основі цього аналізу автоматично генерує тестові кейси, що значно зменшує час, необхідний для ручного написання тестів. [3]

Плюси Swagger Meqa:

- Автоматизація генерації тестів: Swagger Meqa автоматично генерує тестові випадки на основі специфікацій API, що дозволяє швидко розпочати тестування і зосередитися на більш складних аспектах тестування.
- Забезпечення глибокого покриття: Інструмент забезпечує глибоке покриття функціональності API, виявляючи не тільки очевидні помилки, але й потенційні вразливості та проблеми, які можуть не бути очевидними при ручному тестуванні.
- Інтеграція з існуючими інструментами: Swagger Meqa може інтегруватися з іншими інструментами тестування та розробки, що дозволяє використовувати його в рамках більших систем автоматизації.

Мінуси Swagger Meqa:

- Залежність від специфікацій: Ефективність Swagger Meqa сильно залежить від якості та повноти специфікацій Swagger. Неповні або неточні специфікації можуть призвести до неповного або некоректного тестового покриття.
- Потреба в спеціалізованих знаннях: Для ефективного використання Swagger Meqa необхідно мати глибокі знання специфікацій Swagger та принципів тестування API, що може бути бар'єром для нових користувачів.

Swagger Meqa є потужним інструментом для автоматизації тестування API, який може значно підвищити ефективність тестування за рахунок автоматичної генерації тестових випадків. Цей інструмент забезпечує глибоке покриття функціональності API, допомагаючи виявити потенційні проблеми на ранніх етапах розробки. Однак, ефективність використання Swagger Meqa сильно залежить від якості специфікацій API та вимагає від користувачів

спеціалізованих знань. Таким чином, Swagger Meqa є відмінним вибором для організацій, які мають добре структуровані специфікації API та команди, кваліфіковані в тестуванні API.

1.4 Tcases

Tcases є відкритим програмним забезпеченням, призначеним для генерації тестових випадків. Цей інструмент використовує модель-орієнтований підхід для автоматизації процесу створення тестів, що дозволяє розробникам і тестувальникам ефективно покривати різні сценарії використання програмного забезпечення. Tcases аналізує вхідні моделі, які описують можливі стани та переходи системи, і на основі цих моделей генерує тестові випадки, які можуть бути використані для ручного або автоматизованого тестування. [4]

Плюси Tcases:

- Автоматизація генерації тестів: Tcases знижує час і зусилля, необхідні для створення тестових випадків, завдяки автоматичній генерації на основі визначених моделей.
- Покращене покриття тестами: Завдяки модель-орієнтованому підходу, Tcases допомагає забезпечити глибоке тестове покриття, виявляючи потенційні проблеми, які можуть бути пропущені при ручному підході до написання тестів.
- Гнучкість виводу: Tcases підтримує генерацію тестових випадків у різних форматах, що дозволяє легко інтегрувати їх у різні інструменти та середовища тестування.

Мінуси Tcases:

- Залежність від якості моделей: Ефективність Tcases сильно залежить від точності та повноти моделей, які використовуються для генерації тестів. Неповні або неточні моделі можуть призвести до неповного або некоректного тестового покриття.

- Потреба в спеціалізованих знаннях: Для ефективного використання Tcases необхідно мати глибокі знання в області моделювання та тестування, що може бути бар'єром для нових користувачів або малих команд.

Tcases є потужним інструментом для автоматизації генерації тестових випадків, який може значно підвищити ефективність тестування програмного забезпечення. Його здатність до глибокого тестового покриття та автоматизації процесу створення тестів робить його відмінним вибором для проектів, які вимагають високої якості та надійності. Однак, для досягнення максимальної ефективності використання Tcases, командам потрібно забезпечити високу якість моделей та мати достатні знання у сфері моделювання та тестування. Tcases буде особливо корисним у середовищах, де вже існує досвід роботи з модель-орієнтованими методами розробки.

ВИСНОВОК ДО РОЗДІЛУ 1

На основі оглянутих вище рішень що дозволяють автоматично генерувати тестові випадки можна зробити висновок що кожне з них намагається надати багато інших послуг окрім самої генерації тестів. Розглянуті рішення пропонують інтеграції з CI/CD, веб-сервіси для управління тестами, свої платформи або генерують тестові сценарії.

Платформа AccelQ, орієнтована на підхід "no-code", може здатися непідходящим вибором для досвідчених інженерів, які звикли писати код для тестових випадків. Вони можуть відчути обмеженість у функціональності та контролі, що надається такою платформою, оскільки багато складних тестових сценаріїв можуть вимагати глибшого втручання та налаштувань на рівні коду. Окрім цього, інтеграція AccelQ у вже існуючий проєкт може бути доволі незручною. Це пов'язано з тим, що AccelQ не завжди легко поєднується з іншими інструментами та фреймворками, які вже використовуються в інженерних процесах, що призводить до непотрібних затримок та конфліктів у робочих процесах.

ReadyAPI Platform надає користувачам не просто інструмент для тестування, а повноцінну платформу, що забезпечує комплексний підхід до управління тестуванням. На відміну від інших інструментів, які пропонують інтеграцію у вже існуючі проєкти, ReadyAPI створена для того, щоб замінити собою багато з них. Це створює певну складність для команд, які вже мають налаштовані процеси та використовують різні технології та інструменти. Замість того, щоб адаптуватися до поточних умов, ReadyAPI вимагає від користувачів переходу на нову платформу, що може супроводжуватися значними затратами часу та ресурсів.

Tcases, побудований на моделях, може виявитися проблематичним для інтеграції у вже існуючі проєкти через низку факторів. Перш за все, модельно-орієнтований підхід вимагає суттєвих змін у поточних процесах та

архітектурних підходах, що може бути не завжди можливим або бажаним для розробницьких команд. Крім того, Tcases часто стикається з проблемами генерації працюючих тестових випадків, що призводить до виникнення неуспішних тестів, які потребують додаткового аналізу та коригування. Це не лише затримує цикл розробки, але й зменшує надійність автоматизованого тестування, що є критично важливим для забезпечення якості програмного забезпечення.

Swagger_meqa, інструмент на основі підходу "no-code", може не відповідати очікуванням команд, які звикли працювати з кодом, особливо у випадку з генерацією тестових файлів. Однією з основних проблем є те, що Swagger_meqa не генерує тестові файли у форматі, який часто використовується розробниками, наприклад, .js файли. Натомість цей інструмент генерує тестові випадки у форматі YAML або JSON, що потребує додаткових кроків для перетворення у зрозумілий код для тестових фреймворків, таких як Jest чи Mocha.

Формат YAML або JSON, хоч і є зрозумілим структурованим форматом даних, проте не забезпечує тієї гнучкості та контролю, які надаються при безпосередньому написанні коду. Розробникам доводиться витрачати додатковий час на конвертацію та налаштування цих файлів, що приводить до збільшення часу на розробку. [5]

Підсумовуючи описані рішення, можна сказати що вони потребують від інженера з контролю якості або команди інженерів інтегрувати їх продукт у вже існуючий проект. В разі використання такого програмного продукту інженерам потрібно змінювати підхід до тестування, або інтегрувати важку програму в якій буде використовуватись лише частинка функціоналу.

З огляну на цей аналіз буде доцільним розробити рішення яке буде поєднувати ефективність генерації, гнучкість та простоту використання.

РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ ДЛЯ НАПИСАННЯ ПРОГРАМИ

Основна мета цього дипломного проекту полягає у розробці програми, яка автоматизує створення тестів для API, використовуючи можливості великих мовних моделей (LLM – Large Language Models). Така система дозволяє зменшити час на тестування, підвищити точність тестів і полегшити роботу інженерів з якості (QA).

Перший етап роботи програми полягає в отриманні опису кінцевої точки API від користувача. Цей опис може містити URL кінцевої точки, метод HTTP (GET, POST, PUT, DELETE тощо), параметри запиту, формат даних, які очікуються у відповіді. Інтерфейс програми для вставки опису може бути реалізований у вигляді форми на веб-сторінці або десктопному додатку.

На основі отриманого опису програма формує запит до великої мовної моделі (LLM). Модель може приймати до уваги деталізований опис кінцевої точки API, включаючи документи та специфікації, такі як OpenAPI/Swagger.

Після отримання відповіді від LLM, яка містить пропозиції щодо тестів, програма аналізує результат і розбиває його на окремі тест-кейси. Відповідь може включати різні сценарії використання кінцевої точки API: позитивні, негативні, граничні випадки і т.д. [6]

Кожен згенерований тест запускається у середовищі тестування. На цьому етапі програма відправляє HTTP-запити до кінцевої точки API відповідно до згенерованих тест-кейсів і аналізує відповіді на відповідність очікуваним результатам.

Після першого запуску тестів програма відкидає ті, що не пройшли. Це допомагає зосередитися на тестах, які виявилися коректними з першого запуску і не потребують додаткового налагодження. [7]

Після відсіву некоректних тестів ті, що залишилися, проходять процес валідації за допомогою мутацій. Цей етап включає невеликі зміни (мутації) в

параметрах запитів і перевірку стійкості тестів до таких змін. Валідація дозволяє гарантувати, що тести є достатньо міцними та правильно реагують на очікувані і неочікувані сценарії.

Тільки ті тест-кейси, які успішно пройшли всі етапи валідації, залишаються для подальшого використання. Це забезпечує високу якість і релевантність тестів.

На завершальному етапі програма записує всі успішно пройдені тести у окремий файл, який можна легко інтегрувати у існуючу структуру проекту. Файл може містити сценарії для автоматизованих тестових фреймворків, таких як Postman, JUnit, або будь-який інший інструмент, що використовується командами розробки та якості.

2.1 Огляд існуючих LLM для генерації автотестів

Великі мовні моделі, такі як ChatGPT-4 turbo від компанії OpenAI та Gemini від Google, представляють нове покоління інструментів для обробки природної мови, здатних розуміти та генерувати текст на основі дуже об'ємних та різноманітних даних. Ці моделі вже знайшли своє застосування у різних галузях, включаючи написання текстів, створення чат-ботів, переклад мов та інше. В останні роки з'явилися приклади використання LLM для автоматизації завдань, пов'язаних з розробкою програмного забезпечення, таких як написання коду та створення тестів.

Основна перевага використання LLM для генерації автотестів полягає в здатності моделей до розпізнавання і розуміння контексту, а також у вмінні генерувати текст, який відповідає складним вимогам. Наприклад, ChatGPT-4 turbo має 175 мільярдів параметрів і може генерувати високоякісні тексти, що робить її потужним інструментом для створення тестів, які можуть враховувати різні сценарії взаємодії з API. Інші моделі, такі як BERT, використовуються в основному для завдань, що потребують розуміння тексту, але також можуть

бути адаптовані для створення тест-кейсів, аналізуючи специфікації та документацію API. [8]

Новітні дослідження та розробки у цій галузі також включають поєднання LLM з іншими технологіями, такими як інструменти автоматизованого тестування (наприклад, Selenium, Postman) та платформи для безперервної інтеграції і доставки (CI/CD), що дозволяє створювати високоефективні системи автоматизації тестування. У цьому розділі ми детально розглянемо основні властивості та можливості різних LLM, їх сильні та слабкі сторони, а також практичні аспекти використання їх для генерації автотестів.

2.1.1 ChatGPT-4 turbo

ChatGPT-4 Turbo представляє собою вдосконалену версію попередньої моделі ChatGPT, яка була розроблена компанією OpenAI. Дана модель має на меті забезпечення підвищеної продуктивності та ефективності у порівнянні зі своїми попередниками. У цьому документі будуть розглянуті переваги та недоліки ChatGPT-4 Turbo, а також можливості його налаштування. [9]

Переваги ChatGPT-4 Turbo

1. Підвищена продуктивність

Однією з ключових переваг ChatGPT-4 Turbo є його підвищена продуктивність. Завдяки оптимізованому архітектурному дизайну та вдосконаленим алгоритмам навчання, ця модель здатна обробляти запити значно швидше, ніж попередні версії. Це дозволяє використовувати її у реальному часі, що є критичним фактором для додатків, які вимагають миттєвих відповідей, таких як чат-боти та системи підтримки клієнтів.

2. Покращена якість відповідей

ChatGPT-4 Turbo має покращену здатність генерувати змістовні та релевантні відповіді. Це досягається завдяки кращому розумінню контексту та більш точної обробки природної мови. Модель здатна враховувати деталі

запиту та надавати більш точні відповіді, що особливо важливо для професійних застосувань, де точність і релевантність інформації мають велике значення.

3. Масштабованість

Модель ChatGPT-4 Turbo добре підходить для великих масштабів використання. Завдяки оптимізаціям у обробці запитів та архітектурі, ця модель може обробляти одночасно велику кількість запитів, що робить її ідеальною для використання у великих організаціях або платформах з високим рівнем користувацької активності.

4. Гнучкість використання

ChatGPT-4 Turbo надає значну гнучкість у налаштуванні та інтеграції до різних систем. Модель може бути легко інтегрована в існуючі інфраструктури через API, що дозволяє розробникам швидко впроваджувати її у свої додатки без суттєвих змін у вже наявних системах.

Недоліки ChatGPT-4 Turbo

1. Високі вимоги до обчислювальних ресурсів

Одним з основних недоліків ChatGPT-4 Turbo є високі вимоги до обчислювальних ресурсів. Модель потребує значних потужностей для обробки запитів у реальному часі, що може бути проблематичним для невеликих компаній або додатків, що працюють на обмежених ресурсах. Це може спричинити додаткові витрати на апаратне забезпечення або оренду серверів.

2. Вартість

Ще одним важливим аспектом є висока вартість використання цієї моделі. Окрім витрат на обчислювальні ресурси, використання ChatGPT-4 Turbo може бути пов'язане з значними ліцензійними зборами та платою за доступ до API. Це може обмежити доступ до моделі для невеликих проектів або стартапів з обмеженим бюджетом.

3. Складність налаштування

Попри те, що модель пропонує широку гнучкість у налаштуванні, процес адаптації ChatGPT-4 Turbo до специфічних потреб певного додатку може бути досить складним і вимагати високої кваліфікації від розробників. Це включає налаштування параметрів моделі, тонке налаштування гіперпараметрів та інтеграцію з існуючими системами.

4. Обмеження в розумінні контексту

Хоча ChatGPT-4 Turbo має покращену здатність розуміти контекст, все ще існують обмеження. Модель може погано обробляти довготривалі контексти або специфічні деталі, що можуть вимагати особливої уваги. Також, як і інші моделі глибокого навчання, ChatGPT-4 Turbo може генерувати помилкові або нерелевантні відповіді у нестандартних ситуаціях.

Можливості налаштування ChatGPT-4 Turbo

1. Налаштування параметрів моделі

ChatGPT-4 Turbo пропонує широкі можливості для налаштування параметрів моделі. Користувачі можуть регулювати різні гіперпараметри, такі як температура, довжина генерованого тексту та інші, щоб досягти оптимальних результатів для їх конкретного сценарію. Наприклад, зниження температури може сприяти отриманню більш детермінованих відповідей, тоді як підвищення температури може зробити відповіді більш різноманітними.

2. Тонке налаштування з використанням навчання на додаткових даних

Для користувачів, які хочуть адаптувати модель до специфічних потреб, ChatGPT-4 Turbo допускає тонке налаштування за допомогою донавчання на додаткових даних. Це означає, що можна використовувати власні набори даних для подальшого навчання моделі, що дозволяє підвищити точність і релевантність відповідей у вузькоспецифічних областях. Наприклад, компанія може навчати модель на даних з власної бази знань, щоб покращити якість відповідей, що стосуються їхньої продукції або послуг.

3. Інтеграція з існуючими системами

Одна з сильних сторін ChatGPT-4 Turbo – це здатність до легкої інтеграції з існуючими системами. Модель надає API, через який можна виконувати запити та отримувати відповіді від моделі. Це дозволяє інтегрувати ChatGPT-4 Turbo у широке різноманіття додатків, від веб- і мобільних застосунків до корпоративних систем підтримки клієнтів. Інтерфейс API є гнучким і дозволяє легко розширювати функціональність додатку без необхідності значних змін у його архітектурі.

4. Використання шаблонів і попередньо налаштованих сценаріїв

ChatGPT-4 Turbo підтримує можливість використання шаблонів і попередньо налаштованих сценаріїв для різних задач. Це дозволяє розробникам швидко налаштувати модель для типових використань, таких як автоматизовані служби підтримки, створення контенту або управління соціальними мережами. Використання таких шаблонів значно скорочує час розгортання і полегшує процес інтеграції моделі у виробниче середовище.

ChatGPT-4 Turbo представляє собою потужний інструмент для автоматизації широкого спектру задач, завдяки своїй підвищеній продуктивності, покращеній якості відповідей і гнучкості у налаштуванні. Незважаючи на деякі недоліки, зокрема високі вимоги до обчислювальних ресурсів та вартість, ця модель має великий потенціал для використання у різних сферах, від підтримки клієнтів до генерації контенту та наукових досліджень. Завдяки широким можливостям налаштування, ChatGPT-4 Turbo здатна адаптуватися до специфічних потреб користувачів, що робить її цінним інструментом для сучасних рішень у галузі штучного інтелекту.

2.1.2 Gemini

Gemini представляє собою новітню платформу від компанії Google, яка об'єднує передові технології штучного інтелекту (ШІ) та машинного навчання

(МН). Розглянемо переваги та недоліки Google Gemini, а також можливості його налаштування. [10]

Переваги Google Gemini

1. Інтеграція різних технологій

Однією з ключових переваг Google Gemini є його здатність інтегрувати різні технології ШІ в одну платформу. Це дозволяє користувачам отримувати доступ до різних інструментів і сервісів з одного місця, що значно спрощує процес роботи з ШІ. Наприклад, користувачі можуть одночасно використовувати обробку природної мови для аналізу тексту та комп'ютерне бачення для аналізу зображень.

2. Підвищена ефективність

Завдяки інтеграції різних технологій, Google Gemini може забезпечити більш ефективні рішення для різних завдань. Система може використовувати дані з різних джерел для створення більш точних моделей прогнозування, що дозволяє зменшити час і ресурси, необхідні для виконання складних завдань.

3. Інтуїтивний інтерфейс

Google Gemini пропонує інтуїтивний інтерфейс, який спрощує роботу з ШІ навіть для користувачів без спеціальних знань у цій галузі. Це дозволяє залучити до використання технологій ШІ широку аудиторію, включаючи бізнес-користувачів, дослідників та розробників.

4. Підтримка різних мов

Одна з важливих переваг Google Gemini — це підтримка різних мов. Це дозволяє користувачам з різних країн і регіонів використовувати платформу на своїй рідній мові, що значно підвищує зручність і доступність.

Недоліки Google Gemini

1. Високі вимоги до ресурсів

Одним з основних недоліків Google Gemini є високі вимоги до обчислювальних ресурсів. Для ефективної роботи платформи необхідні

потужні сервери та велика кількість оперативної пам'яті. Це може бути проблемою для малих і середніх підприємств, які не мають достатніх ресурсів для забезпечення необхідної інфраструктури.

2. Проблеми з конфіденційністю

Інтеграція різних технологій ШІ може викликати проблеми з конфіденційністю даних. Обробка великих обсягів даних може призвести до витоку конфіденційної інформації. Це особливо важливо для компаній, які працюють з чутливими даними, такими як медичні або фінансові установи.

3. Складність налаштування

Незважаючи на інтуїтивний інтерфейс, налаштування Google Gemini може бути складним для користувачів без спеціальних знань у галузі ШІ. Це може вимагати додаткових витрат на навчання персоналу або залучення зовнішніх консультантів.

4. Залежність від інтернет-з'єднання

Google Gemini є хмарною платформою, що означає, що для її використання необхідне стабільне інтернет-з'єднання. Це може бути проблемою в регіонах з поганою інтернет-інфраструктурою або в умовах, де доступ до інтернету обмежений.

Можливості налаштування Google Gemini

1. Налаштування параметрів моделі

Google Gemini пропонує широкі можливості для налаштування параметрів моделі. Користувачі можуть регулювати різні гіперпараметри, такі як температура, довжина генерованого тексту та інші, щоб досягти оптимальних результатів для їх конкретного сценарію. Наприклад, зниження температури може сприяти отриманню більш детермінованих відповідей, тоді як підвищення температури може зробити відповіді більш різноманітними.

2. Тонке налаштування з використанням навчання на додаткових даних

Для користувачів, які хочуть адаптувати платформу до специфічних потреб, Google Gemini допускає тонке налаштування за допомогою донавчання на додаткових даних. Це означає, що можна використовувати власні набори даних для подальшого навчання моделі, що дозволяє підвищити точність і релевантність відповідей у вузькоспецифічних областях.

3. Інтеграція з існуючими системами

Одна з сильних сторін Google Gemini – це здатність до легкої інтеграції з існуючими системами. Платформа надає API, через який можна виконувати запити та отримувати відповіді від моделі. Це дозволяє інтегрувати Google Gemini у широке різноманіття додатків, від веб- і мобільних застосунків до корпоративних систем підтримки клієнтів.

4. Використання шаблонів і попередньо налаштованих сценаріїв

Google Gemini підтримує можливість використання шаблонів і попередньо налаштованих сценаріїв для різних задач. Це дозволяє розробникам швидко налаштувати платформу для типових використань, таких як автоматизовані служби підтримки, створення контенту або управління соціальними мережами. Використання таких шаблонів значно скорочує час розгортання і полегшує процес інтеграції платформи у виробниче середовище.

Google Gemini представляє собою потужний інструмент для автоматизації широкого спектру задач, завдяки своїй інтеграції різних технологій, підвищеній ефективності, інтуїтивному інтерфейсу та гнучкості у налаштуванні. Незважаючи на деякі недоліки, зокрема високі вимоги до обчислювальних ресурсів та проблеми з конфіденційністю, ця платформа має великий потенціал для використання у різних сферах, від підтримки клієнтів до генерації контенту та наукових досліджень. Завдяки широким можливостям налаштування, Google Gemini здатна адаптуватися до специфічних потреб користувачів, що робить її цінним інструментом для сучасних рішень у галузі штучного інтелекту.

2.1.3 Висновок

Порівнюючи ChatGPT-4 Turbo та Google Gemini, можна відзначити, що обидві платформи демонструють високий рівень здатності генерувати точні та релевантні відповіді на запити користувачів. Обидві системи також пропонують значні можливості для налаштування, що дозволяє адаптувати їх до специфічних потреб різних користувачів та додатків.

З точки зору здатності генерувати точні відповіді, як ChatGPT-4 Turbo, так і Google Gemini використовують передові технології обробки природної мови та машинного навчання. Це дозволяє їм ефективно розуміти контекст запитів і надавати змістовні відповіді, що є критично важливим для професійних застосувань.

Що стосується можливостей налаштування, обидві платформи пропонують широкий спектр інструментів для тонкого налаштування моделей. Користувачі можуть регулювати різні параметри, використовувати додаткові дані для навчання та інтегрувати платформи з існуючими системами через API. Це забезпечує високу гнучкість у використанні та дозволяє швидко адаптувати платформи до конкретних завдань.

Головною відмінністю між ChatGPT-4 Turbo та Google Gemini є великий вибір різних моделей, які пропонує OpenAI, а також частота оновлень та зміни, що відбуваються з кожною новою версією. OpenAI регулярно випускає нові версії своїх моделей, що включають покращення продуктивності, точності та нові функціональні можливості. Це дозволяє користувачам завжди мати доступ до найсучасніших технологій і швидко впроваджувати нові рішення у свої проекти.

Враховуючи всі ці фактори, я вирішив використовувати ChatGPT через те, що він зайняв велику долю ринку, що впливає на спільноту, яка сформувалася навколо нього. Велика спільнота користувачів та розробників створює додаткові можливості для обміну досвідом, підтримки та розвитку

нових рішень. Це робить ChatGPT-4 Turbo привабливим вибором для широкого спектру завдань у галузі штучного інтелекту.

2.2 Вибір системи опису документації

Вибір системи опису документації є критично важливим етапом у розробці API, оскільки від цього залежить ефективність інтеграції та взаємодії з розробниками. У цьому розділі будуть розглянуті три найкращі технології для документування API: Swagger (OpenAPI), Postman та Redoc. Swagger, тепер відомий як OpenAPI, є стандартом де-факто для створення і візуалізації специфікацій API, пропонуючи широкий набір інструментів та інтеграцій. Postman, відомий своєю потужною платформою для тестування API, також надає розширені можливості для генерації та підтримки документації. Redoc, у свою чергу, відзначається своєю здатністю генерувати елегантну та читабельну документацію з існуючих специфікацій OpenAPI. Ці три інструменти забезпечують комплексний підхід до створення і підтримки документації API, що робить їх найкращими варіантами для розгляду.

2.2.1 Swagger

Swagger — це популярний інструмент для документування та тестування RESTful API, який був створений для полегшення роботи розробників з API. Його історія починається у 2010 році, коли Тоні Там, один із засновників компанії Reverb, розробив Swagger як внутрішній інструмент для документування API. Згодом інструмент став відкритим та доступним для широкого загалу, що дозволило йому швидко здобути популярність серед розробників. [11][12]

У 2015 році компанія SmartBear Software придбала Swagger, що сприяло подальшому розвитку та вдосконаленню інструменту. У 2016 році специфікація Swagger була перейменована в OpenAPI Specification (OAS) і передана до Linux

Foundation, де була створена OpenAPI Initiative. Це забезпечило ще більшу підтримку та розвиток стандарту.

Переваги Swagger:

- Автоматизація документації: Swagger дозволяє автоматично генерувати документацію для API, що значно зменшує час і зусилля, необхідні для її створення та підтримки.
- Легкість у використанні: Інтуїтивно зрозумілий інтерфейс Swagger UI дозволяє швидко переглядати та тестувати API, що спрощує роботу як для розробників, так і для тестувальників.
- Підтримка різних мов програмування: Swagger Codegen підтримує генерацію клієнтських бібліотек та серверних скелетів для багатьох мов програмування, що робить його універсальним інструментом для різних проектів.
- Спільна робота: Swagger Hub дозволяє командам розробників спільно працювати над API, забезпечуючи централізоване місце для зберігання та управління специфікаціями.
- Відкритий стандарт: Swagger використовує формат OpenAPI, який є відкритим стандартом для опису RESTful API. Це забезпечує сумісність з іншими інструментами та платформами, що підтримують OpenAPI.

Недоліки Swagger:

- Обмеження у складних сценаріях: Хоча Swagger добре підходить для більшості RESTful API, він може бути обмеженим у випадках складних або специфічних сценаріїв, які не можуть бути повністю описані у форматі OpenAPI.
- Вимоги до навчання: Незважаючи на інтуїтивність інтерфейсу, новачкам може знадобитися час для ознайомлення з усіма можливостями та функціями Swagger.

- Залежність від специфікацій: Автоматична генерація документації та коду залежить від точності та повноти специфікацій OpenAPI. Помилки або неточності у специфікаціях можуть призвести до некоректної документації або коду.
- Обмеження у налаштуванні: Хоча Swagger надає багато можливостей для налаштування, деякі користувачі можуть виявити, що їм не вистачає гнучкості у певних аспектах, таких як налаштування вигляду документації або генерації коду.
- Вимоги до підтримки: Як і будь-який інший інструмент, Swagger потребує регулярного оновлення та підтримки, щоб залишатися сумісним з новими версіями мов програмування та стандартів.

2.2.2 Postman

Postman це інструмент для тестування та документування API, який використовується розробниками для спрощення процесу розробки, тестування та інтеграції API. Він надає зручний інтерфейс для створення, відправки та аналізу HTTP-запитів, а також для автоматизації тестування API. Postman підтримує різні типи запитів, включаючи GET, POST, PUT, DELETE, PATCH та інші, що робить його універсальним інструментом для роботи з API. [13]

Історія Postman починається у 2012 році, коли Абхінад Кашьяп, один із засновників компанії Postdot Technologies, створив Postman як розширення для Google Chrome. Інструмент швидко здобув популярність серед розробників завдяки своїй простоті та функціональності. Згодом Postman перетворився на повноцінний додаток, доступний для різних операційних систем, включаючи Windows, macOS та Linux.

Переваги Postman:

- Інтуїтивно зрозумілий інтерфейс: Postman надає зручний графічний інтерфейс для створення та відправки HTTP-запитів, що робить його доступним для розробників з різним рівнем досвіду.
- Підтримка різних типів запитів: Postman підтримує всі основні типи HTTP-запитів, включаючи GET, POST, PUT, DELETE та PATCH, що дозволяє тестувати різні аспекти API.
- Автоматизація тестування: Postman дозволяє створювати колекції запитів та автоматизувати їх виконання за допомогою вбудованого Runner. Це дозволяє автоматизувати тестування API та інтегрувати його у процеси CI/CD.
- Спільна робота: Postman надає можливість спільної роботи над колекціями запитів, що дозволяє командам розробників ефективно співпрацювати та ділитися результатами тестування.
- Інтеграція з іншими інструментами: Postman підтримує інтеграцію з різними інструментами та сервісами, такими як Jenkins, GitHub, Slack та інші, що дозволяє інтегрувати його у існуючі робочі процеси.

Недоліки Postman:

- Обмеження у складних сценаріях: Хоча Postman добре підходить для більшості сценаріїв тестування API, він може бути обмеженим у випадках складних або специфічних сценаріїв, які потребують більш гнучких налаштувань.
- Вимоги до навчання: Незважаючи на інтуїтивність інтерфейсу, новачкам може знадобитися час для ознайомлення з усіма можливостями та функціями Postman.
- Обмеження у налаштуванні: Хоча Postman надає багато можливостей для налаштування, деякі користувачі можуть виявити, що їм не

вистачає гнучкості у певних аспектах, таких як налаштування вигляду інтерфейсу або генерації коду.

- Вимоги до ресурсів: Postman може споживати значну кількість системних ресурсів, особливо при роботі з великими колекціями запитів або складними сценаріями тестування.
- Залежність від інтернет-з'єднання: Деякі функції Postman, такі як синхронізація колекцій та спільна робота, потребують стабільного інтернет-з'єднання, що може бути обмеженням у деяких умовах.

2.2.3 Redoc

Redoc — це інструмент для генерації документації для API на основі специфікацій OpenAPI (раніше відомих як Swagger). Він дозволяє створювати зручну для користувача документацію, яка може бути легко інтегрована у веб-сайти та інші платформи. Redoc є відкритим програмним забезпеченням, що дозволяє розробникам вільно використовувати та налаштовувати його відповідно до своїх потреб. [14]

Історія Redoc починається у 2015 році, коли компанія Rebilly, яка спеціалізується на платіжних рішеннях, створила цей інструмент для внутрішнього використання. Згодом Redoc був випущений як відкритий проект, що дозволило йому здобути популярність серед розробників завдяки своїй простоті та функціональності.

Переваги Redoc:

- Зручний дизайн: Redoc надає сучасний та зручний для користувача інтерфейс для документації API, що робить її легкою для читання та навігації.
- Підтримка OpenAPI: Redoc повністю підтримує специфікації OpenAPI, що дозволяє автоматично генерувати документацію на основі існуючих специфікацій API.

- Легкість інтеграції: Redoc можна легко інтегрувати у веб-сайти та інші платформи за допомогою простого HTML-файлу або JavaScript-бібліотеки.
- Налаштування: Redoc надає багато можливостей для налаштування вигляду та поведінки документації, що дозволяє адаптувати її під конкретні потреби проекту.
- Відкритий код: Redoc є відкритим програмним забезпеченням, що дозволяє розробникам вносити зміни та покращення у код відповідно до своїх вимог.

Недоліки Redoc:

- Обмеження у складних сценаріях: Хоча Redoc добре підходить для більшості сценаріїв документування API, він може бути обмеженим у випадках складних або специфічних сценаріїв, які потребують більш гнучких налаштувань.
- Вимоги до навчання: Незважаючи на інтуїтивність інтерфейсу, новачкам може знадобитися час для ознайомлення з усіма можливостями та функціями Redoc.
- Залежність від специфікацій: Автоматична генерація документації залежить від точності та повноти специфікацій OpenAPI. Помилки або неточності у специфікаціях можуть призвести до некоректної документації.
- Вимоги до підтримки: Як і будь-який інший інструмент, Redoc потребує регулярного оновлення та підтримки, щоб залишатися сумісним з новими версіями специфікацій OpenAPI та стандартів.
- Обмеження у налаштуванні: Хоча Redoc надає багато можливостей для налаштування, деякі користувачі можуть виявити, що їм не

вистачає гнучкості у певних аспектах, таких як налаштування вигляду інтерфейсу або додавання специфічних функцій.

2.3 Використання Styker для мутаційних тестів

Мутаційне тестування є однією з технік тестування програмного забезпечення, яка використовується для оцінки якості тестових випадків. Основна ідея мутаційного тестування полягає в тому, щоб внести невеликі зміни (мутації) в програмний код і перевірити, чи здатні існуючі тестові випадки виявити ці зміни. Мутації можуть включати зміну операторів, значень змінних, умовних виразів та інших елементів коду. Якщо тестові випадки не виявляють змін, це може свідчити про їх недостатню ефективність. [15][16][17]

Мутаційне тестування має кілька важливих переваг:

- Підвищення якості тестових випадків. Оскільки мутаційне тестування вимагає виявлення навіть незначних змін у коді, це сприяє створенню більш ретельних і детальних тестових випадків.
- Виявлення слабких місць у тестах. Мутаційне тестування допомагає виявити тестові випадки, які не покривають певні частини коду або не перевіряють важливі аспекти функціональності.
- Підвищення надійності програмного забезпечення. Завдяки більш ретельному тестуванню, програмне забезпечення стає більш стійким до помилок і несподіваних ситуацій.

Swagger є інструментом для автоматизації створення, документування та тестування API. Він забезпечує зручний інтерфейс для опису API, що дозволяє розробникам легко створювати та підтримувати документацію. У контексті мутаційного тестування Swagger має кілька переваг:

- Автоматизація тестування API. Swagger дозволяє автоматично генерувати тестові випадки для API на основі його опису. Це значно спрощує процес створення тестів і забезпечує їх відповідність актуальній версії API.
- Підтримка різних форматів даних. Swagger підтримує різні формати даних, такі як JSON і YAML, що робить його гнучким інструментом для роботи з різними типами API.
- Інтеграція з іншими інструментами. Swagger легко інтегрується з іншими інструментами для тестування та автоматизації, такими як Postman, JUnit, та інші. Це дозволяє використовувати його в комплексних процесах тестування.
- Підвищення якості документації. Завдяки автоматичному генеруванню документації, Swagger забезпечує її актуальність і точність, що сприяє кращому розумінню API розробниками та тестувальниками.
- Полегшення валідації тестових випадків. Swagger дозволяє автоматично перевіряти відповідність тестових випадків специфікації API, що значно спрощує процес валідації і забезпечує високу якість тестів.

Таким чином, мутаційне тестування є потужним інструментом для підвищення якості тестових випадків і програмного забезпечення в цілому. Використання Swagger у цьому контексті дозволяє автоматизувати процеси тестування API, забезпечити їх відповідність специфікаціям і підвищити загальну ефективність тестування. [18]

2.4 Використання платформи NodeJS

Node.js є популярною платформою для розробки серверних додатків, яка базується на JavaScript і працює на рушії V8 від Google. Вона дозволяє розробникам створювати високопродуктивні, масштабовані мережеві додатки з

мінімальними зусиллями. Вибір платформи Node.js для розробки проекту має кілька ключових переваг, які роблять її привабливою для багатьох розробників. [19]

- Широка доступність фреймворків і бібліотек. Node.js має величезну екосистему модулів і фреймворків, які значно спрощують процес розробки. Одним з найпопулярніших фреймворків є Express.js, який забезпечує простий і гнучкий інтерфейс для створення веб-додатків і API. Крім того, існують інші фреймворки, такі як Koa.js, Nest.js, Sails.js, які пропонують різні підходи до розробки і можуть задовольнити потреби різних проектів. Завдяки цій різноманітності, розробники можуть вибрати інструменти, які найкраще відповідають їхнім вимогам і стилю роботи.
- Швидкість написання коду. Node.js дозволяє розробникам писати код швидко і ефективно завдяки асинхронній моделі програмування і неблокуючому вводу-виводу. Це означає, що додатки можуть обробляти багато запитів одночасно без затримок, що значно підвищує їх продуктивність. Крім того, використання JavaScript як мови програмування дозволяє розробникам використовувати свої знання і досвід з фронтенд-розробки для створення серверних додатків, що зменшує криву навчання і прискорює процес розробки.
- Знайомство з платформою. Вибір Node.js також обумовлений тим, що я добре знайомий з цією платформою. Мій досвід роботи з Node.js дозволяє мені швидко і ефективно вирішувати завдання, що виникають під час розробки, і використовувати найкращі практики для забезпечення високої якості коду. Знання особливостей платформи, її сильних і слабких сторін дозволяє мені приймати обґрунтовані рішення і оптимізувати процес розробки.

Node.js також має кілька інших переваг, які роблять її привабливою для розробників:

- Масштабованість. Завдяки своїй архітектурі, Node.js дозволяє легко масштабувати додатки як горизонтально, так і вертикально. Це означає, що можна додавати нові вузли або збільшувати потужність існуючих, щоб обробляти більше запитів і забезпечувати стабільну роботу додатка під високим навантаженням.
- Активна спільнота. Node.js має велику і активну спільноту розробників, яка постійно працює над покращенням платформи і створенням нових інструментів і бібліотек. Це забезпечує швидке вирішення проблем, доступ до великої кількості ресурсів і підтримку з боку інших розробників.
- Кросплатформеність. Node.js працює на різних операційних системах, таких як Windows, macOS і Linux, що робить її універсальною платформою для розробки додатків. Це дозволяє розробникам працювати на будь-якій зручній для них операційній системі і забезпечує широку сумісність додатків.
- Легкість у розгортанні. Завдяки своїй простоті і гнучкості, Node.js дозволяє легко розгортати додатки на різних платформах і хмарних сервісах. Це забезпечує швидке і безпроблемне розгортання додатків у виробниче середовище.

Отже, вибір платформи Node.js для розробки проекту є обґрунтованим і логічним рішенням, враховуючи її численні переваги. Широка доступність фреймворків і бібліотек, швидкість написання коду, а також мій досвід роботи з цією платформою роблять її ідеальним вибором для створення високоякісних, продуктивних і масштабованих додатків. Node.js дозволяє ефективно

використовувати ресурси і забезпечувати стабільну роботу додатків під високим навантаженням, що є ключовим фактором для успішної реалізації проекту.

2.5 Використання фреймворку WebdriverIO з тестранером Mocha

WebdriverIO є потужним інструментом для автоматизації тестування веб-додатків, який базується на стандарті WebDriver. Він надає розробникам і тестувальникам можливість писати тести для веб-додатків на різних браузерах і платформах, забезпечуючи високу надійність і ефективність тестування. WebdriverIO підтримує різні фреймворки для написання тестів, серед яких одним з найпопулярніших є Mocha. [20][21]

Mocha є гнучким і простим у використанні тестовим фреймворком для JavaScript, який дозволяє писати як синхронні, так і асинхронні тести. Він забезпечує зручний інтерфейс для організації тестів, підтримує різні стилі написання тестів (BDD, TDD) і має багатий набір функцій для асерцій, моків і шпигунів. Використання Mocha разом з WebdriverIO дозволяє створювати ефективні і зрозумілі тести для веб-додатків, що значно спрощує процес автоматизації тестування.

Однією з ключових переваг використання WebdriverIO і Mocha є підтримка Mocha від Stryker. Підтримка Mocha від Stryker дозволяє інтегрувати мутаційне тестування в процес автоматизації тестування, що значно підвищує якість тестів і програмного забезпечення в цілому.

Таким чином, використання WebdriverIO і Mocha з підтримкою Stryker є потужним рішенням для автоматизації тестування кінцевих точок API. Вони забезпечують високу якість тестів, гнучкість і розширюваність, а також інтеграцію з іншими інструментами і процесами. Це дозволяє створювати

надійні і ефективні тести, які забезпечують стабільну роботу веб-додатків і підвищують їх якість.

2.6 Використання VS Code

Visual Studio Code (VS Code) є одним з найпопулярніших редакторів коду серед розробників, і це не випадково. Він пропонує безліч функцій, які роблять його ідеальним вибором для написання програмного забезпечення на платформі Node.js. Ось кілька ключових причин, чому VS Code є відмінним рішенням для розробки на Node.js. [22]

Підтримка всіх популярних Node.js фреймворків. VS Code підтримує всі основні фреймворки для Node.js. Завдяки великій кількості розширень, розробники можуть легко налаштувати середовище під свої потреби, отримуючи доступ до спеціалізованих інструментів і функцій для кожного фреймворку.

VS Code має вбудовану підтримку для npm, що дозволяє легко керувати залежностями проекту, встановлювати нові пакети та виконувати скрипти безпосередньо з редактора. Це значно спрощує процес розробки і забезпечує зручний інтерфейс для роботи з пакетними менеджерами.

Автодоповнення і інтелектуальні підказки. Завдяки інтеграції з TypeScript і JavaScript, VS Code надає розробникам потужні інструменти для автодоповнення коду і інтелектуальних підказок. Це допомагає писати код швидше і з меншою кількістю помилок, підвищуючи продуктивність і якість розробки.

Налагодження і тестування. VS Code має вбудовані можливості для налагодження Node.js додатків, що дозволяє легко знаходити і виправляти помилки. Крім того, існують розширення для інтеграції з популярними тестовими фреймворками, такими як Mocha, Jest і Jasmine, що забезпечує зручне середовище для написання і виконання тестів.

Розширюваність і налаштування. Однією з головних переваг VS Code є його розширюваність. Існує безліч розширень, які можна встановити для додавання нових функцій і покращення робочого процесу. Розробники можуть налаштувати редактор під свої потреби, додаючи підтримку для різних мов програмування, інструментів і фреймворків.

Інтеграція з системами контролю версій. VS Code має вбудовану підтримку для Git та інших систем контролю версій, що дозволяє легко керувати версіями коду, виконувати коміти, створювати гілки і вирішувати конфлікти безпосередньо з редактора.

Таким чином, VS Code є потужним і гнучким інструментом для розробки на платформі Node.js. Він підтримує всі популярні фреймворки, забезпечує зручне середовище для написання, налагодження і тестування коду, а також має безліч розширень для налаштування під конкретні потреби розробників. Це робить його ідеальним вибором для створення високоякісних програмних рішень на Node.js.

ВИСНОВОК ДО РОЗДІЛУ 2

Після ретельного аналізу різних інструментів та методів, ми дійшли висновку, що для нашого проекту найбільш підходящим вибором є використання ChatGPT-4 Turbo для створення гнучкої та адаптивної системи, а також Swagger для генерації тестів на основі опису API. Цей підхід забезпечить високий рівень ефективності та зручності при розробці та підтримці нашого програмного забезпечення. [23]

ChatGPT-4 Turbo від OpenAI є інструментом, який пропонує значну гнучкість налаштувань і зручний API для інтеграції в різні системи. Однією з ключових переваг цього інструменту є можливість тонкої настройки для кращого розуміння контексту, що особливо важливо для складних і багатокомпонентних систем. Інтерфейс програмування (API) ChatGPT-4 Turbo спроектований таким чином, щоб бути легким у використанні та інтеграції, що значно спрощує процес підключення моделі до існуючих програмних рішень та забезпечує безшовну взаємодію з іншими компонентами системи. Крім того, ChatGPT-4 Turbo може адаптуватися до різних сценаріїв використання, що робить його ідеальним для проектів, де потрібна висока гнучкість, а також забезпечує можливість масштабування, що дозволяє обробляти великі обсяги запитів без втрати продуктивності.

Swagger (OpenAPI) є загальновизнаним стандартом для опису RESTful API, і його використання для генерації тестів має численні переваги. Swagger є надзвичайно поширеним інструментом, який підтримується багатьма платформами та мовами програмування, що забезпечує широку сумісність і підтримку. Специфікації Swagger дозволяють створювати детальні та структуровані описи API, які легко зрозуміти і використовувати, що значно спрощує процес розробки та тестування. Це особливо важливо, оскільки розробники можуть легко знайти необхідну інформацію про кінцеві точки API, параметри запитів та відповіді.

Використання Swagger для генерації тестів дозволяє автоматизувати процес тестування API, що забезпечує високу точність та ефективність, зменшуючи ризик людських помилок та підвищуючи надійність програмного забезпечення. Крім того, Swagger UI надає інтерактивну документацію, яка дозволяє розробникам виконувати запити безпосередньо з браузера. Це особливо корисно для тестування та відлагодження API, оскільки дозволяє швидко перевіряти поведінку різних кінцевих точок.

Зважаючи на вищенаведені переваги, ми прийняли рішення використовувати ChatGPT-4 Turbo як основний інструмент для розробки та підтримки нашого програмного забезпечення завдяки його гнучкості, зручному API та здатності адаптуватися до різних сценаріїв. Для генерації тестів ми будемо використовувати Swagger, оскільки він є загальновизнаним стандартом з широкою підтримкою в індустрії та зручними можливостями для створення детальних і структурованих описів API. Це дозволить нам забезпечити високу якість та надійність нашого програмного забезпечення, спрощуючи процес розробки та тестування.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Відповідно до досліджених матеріалів та обраних засобів для реалізації програмного продукту ми можемо перейти до стадії реалізації. Для того щоб створити гнучку та настроювану програму для генерації автотестів необхдіно описати основні частини програми, функціональні блоки програми та схему їх взаємодії.

3.1 Розробка компонентів для налаштування роботи з ChatGPT

В цьому розділі будуть описані основні компоненти для роботи з ChatGPT

3.1.1 Запит до ChatGPT

```
const chatCompletion = await openai.chat.completions.create({
    messages: [{ role: 'user', content: promptText }],
    model: 'gpt-4-turbo',
    temperature: 0,
    max_tokens: 3000,
    top_p: 0.5,
    frequency_penalty: 0,
    presence_penalty: 0
});
```

Рисунок 3.1 – функція запиту до ChatGPT4-turbo

Функція запиту до ChatGPT використовує API OpenAI для створення завершення чату на основі заданого тексту. [24]

Функція create приймає об'єкт з кількома параметрами, які визначають поведінку та характеристики запиту. Розглянемо кожен з них окремо.

messages: [{ role: 'user', content: promptText }]

Цей параметр є масивом об'єктів, кожен з яких представляє повідомлення в чаті. Кожне повідомлення має два ключі: role та content. У цьому випадку, role

встановлено як 'user', що означає, що повідомлення надходить від користувача. content містить текст запиту, який зберігається у змінній promptText.

model: 'gpt-4-turbo'

Цей параметр визначає модель, яка буде використовуватися для генерації відповіді. У даному випадку використовується модель gpt-4-turbo, яка є однією з варіацій GPT-4, оптимізованою для швидкості та ефективності.

temperature: 0

Параметр temperature контролює випадковість відповіді. Значення 0 означає, що модель буде намагатися генерувати найбільш передбачувану відповідь, що може бути корисним для отримання точних та детермінованих результатів. Вищі значення temperature (до 1) роблять відповіді більш різноманітними та творчими.

max_tokens: 3000

Цей параметр визначає максимальну кількість токенів (слова або частини слів), які можуть бути використані для генерації відповіді. У даному випадку встановлено значення 3000, що дозволяє отримати досить довгу та детальну відповідь.

top_p: 0.5

Параметр top_p використовується для налаштування семплінгу за допомогою методу "ядра" (nucleus sampling). Значення 0.5 означає, що модель буде враховувати тільки ті токени, які разом складають 50% ймовірності, що допомагає уникнути менш ймовірних варіантів і зробити відповіді більш узгодженими.

frequency_penalty: 0

Цей параметр контролює, наскільки модель буде штрафувати часте повторення тих самих токенів у відповіді. Значення 0 означає, що штраф не застосовується, тобто модель не буде спеціально уникати повторень.

presence_penalty: 0

Параметр presence_penalty визначає, наскільки модель буде штрафувати використання нових токенів, які ще не з'являлися у відповіді. Значення 0 означає, що штраф не застосовується, тобто модель не буде уникати введення нових тем або слів.

3.1.2 Опис за об’єднання промпту для запиту

Промпт у створеній програмі складається з двох відокремлених частин які пізніше з’єднуються в один текст запиту

Перша частина описує вимоги до відвовіді від моделі на запит

```
Write an API test for this endpoint, described by Swagger with these requirements:
1. Use webdriverio framework
2. Write only it-blocks
3. For API requests use only "fetch" library
4. Don't write anything but it-blocks
5. Don't write any import code
6. Don't write ANY comments
7. Generate random data for request data
8. Use the api key "special-key" to test the authorization filters
9. Write the output in the format of .json.
10. use the example below: underscore should replac spaces in the names
{
    {
        "name" : name_of_the_script,
        "script_code" : script_code
    },
    {
        "name" : name_of_the_script,
        "script_code" : script_code
    }
}

use CORRECT ammount of brackets, the JSON you provide to me needs to be parsable via code
don't put commas at the end of scopes
in JSON format, message starts and ends with brackets "[]"

11. Don't write anything but this json
12. Make script_code parsable via JSON.parse() function
13. Name of the it-block should always match the name_of_the_script
14. Generate one it-block for each response code, they all should be in JSON you will send me back
15. DO NOT HIGHLIGHT JSON WITH ```, I NEED ONLY JSON ITSELF
16. Webcite URL is - https://petstore.swagger.io/

here's the swagger description:
```

Рисунок 3.2 - promptRequirements.txt

Цей промпт описує вимоги до написання тесту для API за допомогою фреймворку WebdriverIO. Він містить конкретні інструкції щодо того, як слід реалізувати тестування кінцевого пункту (endpoint), описаного у Swagger. Нижче наведено детальний опис кожної з вимог:

Використання WebdriverIO: Тест повинен бути написаний за допомогою фреймворку WebdriverIO. Це означає, що всі дії, пов'язані з тестуванням, повинні бути реалізовані за допомогою цього інструменту.

Використання однієї бібліотеки для запитів у мережу: Код повинен бути створений за допомогою однієї бібліотеки для запитів у мережу. Це забезпечує уніфікацію та стандартизацію модулю для запитів.

Форматування JSON: У JSON-форматі слід використовувати підкреслення для розділення пробілів у назвах. Наприклад, замість "script code" слід використовувати "script_code".

Парсинг JSON: Результат повинен бути здатним для парсингу за допомогою функції JSON.parse(). Це означає, що JSON повинен бути коректно сформатований для аналізу.

Фільтри авторизації: Фільтри авторизації повинні бути перевірені для кожного кінцевого пункту. Це означає, що тест повинен включати перевірку механізмів аутентифікації та авторизації.

Ці вимоги забезпечують, що тест буде написаний відповідно до стандартів, ефективно взаємодіятиме з API та коректно оброблятиме JSON-дані.

Такий підхід з розділення запитів на окремі не з’єдані частини забезпечує гнучкість програми та можливість її підлаштування під необхідні в данний момент задачі. До прикладу інший розробник може захотіти використати інший фреймворк

```yaml
/pet:
    put:
      tags:
        - pet
      summary: Update an existing pet
      description: Update an existing pet by Id
      operationId: updatePet
      requestBody:
        description: Update an existent pet in the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
          application/xml:
            schema:
              $ref: '#/components/schemas/Pet'
          application/x-www-form-urlencoded:
            schema:
              $ref: '#/components/schemas/Pet'
        required: true
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
            application/xml:
              schema:
                $ref: '#/components/schemas/Pet'
        '400':
          description: Invalid ID supplied
        '404':
          description: Pet not found
        '422':
          description: Validation exception
      security:
        - petstore_auth:
            - write:pets
            - read:pets
```

Рисунок 3.3 – приклад Swagger опису кінцевої точки API

Потім обидві частини запиту об’єднуються в один за допомогою функції getFinalPrompt() класу PromptManager

3.2 Перша перевірка згенерованих тестів

В цьому розділі будуть описані кроки під час першої перевірки згенерованих тестів

3.2.1 Запуск тестів у WebdriverIO

Після отримання даний тестів вони зберігаються в папці generatedScripts у файл з назвою Scripts-дата-запуску-програми-з-точністю-до-секунди.spec.js

Рисунок 3.4 – приклад згенерованих автотетсів

Далі вони запускаються npm командою за допомогою фреймворку WebdriverIO

```
async runTheGeneratedScript(fileName) {
    try {
        const { stdout, stderr } = await exec(`./node_modules/.bin/wdio --spec ./test/specs/api/generatedScripts/${fileName}`);
        console.log('stdout:', stdout);
        console.log('stderr:', stderr);
    } catch (e) {
        console.error(e)
    }
}
```

Рисунок 3.5 – функція для запуску файлу із згенерованими тестами

Після відпрацювання, фреймворк записує результати у .json файл для подальшого використання програмою

```
reporters: [
    'spec',
    ['ctrf-json', {}]
],
```

Рисунок 3.6 – налаштування wdio.conf.js для запису результатів у .json файл

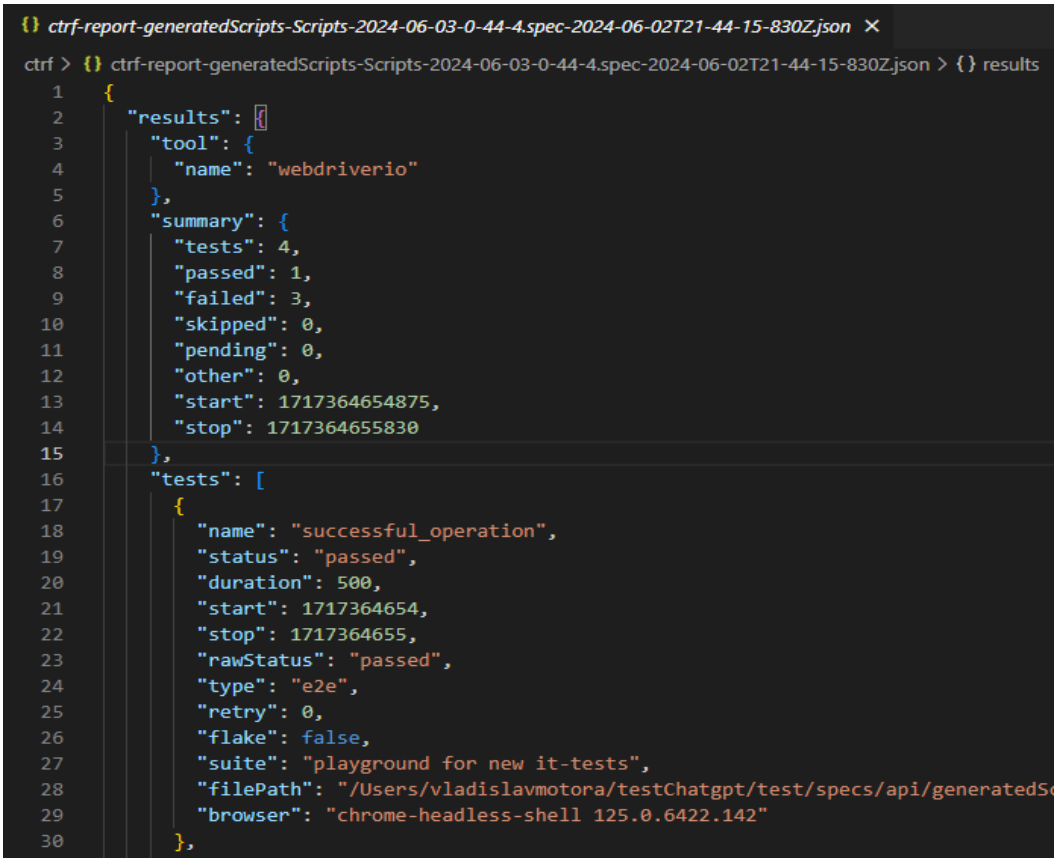


Рисунок 3.7 – приклад .json файлу-звіту

3.2.2 Відправка тестів які пройшли до наступного файлу для перевірки на мутації від Stryker

Після відпрацювання першої перевірки функція filterPassedScriptsByTheirNames() за іменем фільтрує тести які задовільняють вимоги.

```
filterPassedScriptsByTheirNames(initialJson, arrWithPassedScriptsNames){
    return initialJson.filter(({ name }) => arrWithPassedScriptsNames.some(m => name === m))
}
```

Рисунок 3.8 – код функції що фільтрує згенеровані тести

Далі відфільтровані тести записуються у файл з назвою Scripts-дата-запуску-програми-з-точністю-до-секунди.spec.js у папці generatedScriptsForStrykerCheck [25]

3.3 Друга перевірка згенерованих тестів від Stryker

Функція runTheGeneratedScriptByStryker() запускає Stryker

```
async runTheGeneratedScriptByStryker() {
    try {
        const { stdout, stderr } = await exec(`npm run launch:Stryker`);
          console.log('stdout:', stdout);
          console.log('stderr:', stderr);
    } catch (e) {
        console.error(e)
    }
}
```

Рисунок 3.9 – функція запуску Sryker

Запуск Stryker вимагає правильної конфігурації щоб фреймворк зміг бачити за пускати тести формату що його використовує WebdriverIO. Їх точкою пересічення є тестранер mocha.

```
const config = {
  packageManager: "npm",
  mutator: {
    plugins: []
  },
  mutate: [ "./test/specs/api/generatedScriptsForStrykerCheck/*.js" ],
  reporters: ['json', 'clear-text'],
  testRunner: "mocha",
  disableTypeChecks : true,
  cleanTempDir : true,
  jsonReporter : {
    fileName: "reports/mutation/mutation.json"
  },
  tempDirName: 'strykerTmp',
  logLevel : "info",
  coverageAnalysis: "perTest",
  "mochaOptions": {
    "spec": [ "./test/specs/api/generatedScriptsForStrykerCheck/*.js" ]
  }
};
export default config;
```

Рисунок 3.10 – налаштування кофігурації у файлі stryker.config.mjs

В полі reporters – записані json та crear-text. Json потрбіен для подальшого аналізу в програмі, а clear-text для показу результатів у консолі під час виконання головної програми.

Після виконання перевірки від Stryker, функція countScoreFromJSON рахує коефіцієнт ефективності згенерованих тестів. [26][27]

```
countScoreFromJSON() {
    let JSONString = fs.readFileSync(PATH_TO_JSON, 'utf8', (err) => {
        if (err) {
            console.error(err);
            return;
        }
    });
    const survivedNum = this.#countInstances(JSONString, '"status":"Killed"')
    const allNum = this.#countInstances(JSONString, '"status":')
    return ((survivedNum / allNum) * 100).toFixed(2);
}
```

Рисунок 3.11 – код функції countScoreFromJSON

Якщо коефіцієнт ефективності після запуску задовільняє вибраний інженером поріг, - такі тести будуть вважатись перевіреними та валідованими

та будуть записані з назвою Scripts-дата-запуску-програми-з-точністю-до-секунди.spec.js у папці generatedScriptsCheckedAndVerified

Такі тести вже можуть йти у вже існуючий проект з тестування API відповідно до структури яку для себе обрав\обрали інженери, так як згенеровані тести представляють з себе it-блоки, та використовують лише базові модулі.

Логічна схема роботи програми описана у додатку (див. додаток 1)

Діаграма дата-акторів дедально описана у додатку (див. додаток 2)

ВИСНОВОК ДО РОЗДІЛУ 3

В результаті проведеної роботи щодо розробки програмного забезпечення для генерації автотестів за допомогою LLM можна сказати:

- Були використані з опису другого розділу LLM-моделі, фреймворки для тестування, системи опису API-документації, середовище розробки та фреймворк для мутаційних тестів.
- Були продемонстровані фрагменти коду у вигляді скріншотів розроблюваної системи.
- Була пояснена логіка роботи програми
- Були описані залежності розробленої програми
- Були створені логічна схема системи (додаток 1) та діаграма дата-акторів (додаток 2)

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

Базуючись на розробленій системі з генерації автотестів за допомогою LLM метою данного розділу є показати результати її роботи. Провести дослідження з ефективності роботи розробленої системи, та вказати на її можливі покращення. [28]

4.1 Дослідження роботи створеної програми

Вхідні данні для генерації автотестів представляють з себе API-опис кінцевої точки в форматі Swagger:

```
/pet:
    put:
      tags:
        - pet
      summary: Update an existing pet
      description: Update an existing pet by Id
      operationId: updatePet
      requestBody:
        description: Update an existent pet in the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
          application/xml:
            schema:
              $ref: '#/components/schemas/Pet'
          application/x-www-form-urlencoded:
            schema:
              $ref: '#/components/schemas/Pet'
        required: true
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
            application/xml:
              schema:
                $ref: '#/components/schemas/Pet'
        '400':
          description: Invalid ID supplied
        '404':
          description: Pet not found
        '422':
          description: Validation exception
      security:
        - petstore_auth:
            - write:pets
            - read:pets
```

Рисунок 4.1 – API-опис у форматі Swager для pet/put

Запуск програми генерує наступний тестовий випадок:

```
it('successful_operation', async () => {
    const response = await fetch('https://petstore.swagger.io/v2/pet', {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json',
            'api_key': 'special-key'
        },
        body: JSON.stringify({
            id: Math.floor(Math.random() * 1000),
            name: 'Doggie',
            status: 'available'
        })
    });
    const data = await response.json();
    expect(response.status).toBe(200);
    expect(data).toHaveProperty('id');
    expect(data).toHaveProperty('name', 'Doggie');
    expect(data).toHaveProperty('status', 'available');
});
```

Рисунок 4.2 – згенерований автотест

Результат запуску згенерованого автотесту:

```
Execution of 1 workers started at 2024-06-03T15:48:34.763Z

[0-0] RUNNING in chrome - file:///test/specs/api/Playground.spec.js
[0-0] (node:26918) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
[0-0] (Use `node --trace-deprecation ...` to show where the warning was created)
[0-0] wdio-ctrf-json-reporter: successfully written ctrf json
[0-0] PASSED in chrome - file:///test/specs/api/Playground.spec.js

 "spec" Reporter:
------------------------------------------------------------------
[chrome-headless-shell 125.0.6422.142 mac #0-0] Running: chrome-headless-shell (v125.0.6422.142) on mac
[chrome-headless-shell 125.0.6422.142 mac #0-0] Session ID: 30d24745db290695cd2f8ba341409448
[chrome-headless-shell 125.0.6422.142 mac #0-0]
[chrome-headless-shell 125.0.6422.142 mac #0-0] » /test/specs/api/Playground.spec.js
[chrome-headless-shell 125.0.6422.142 mac #0-0] playground for new it-tests
[chrome-headless-shell 125.0.6422.142 mac #0-0]    ✓ successful_operation
[chrome-headless-shell 125.0.6422.142 mac #0-0]
[chrome-headless-shell 125.0.6422.142 mac #0-0] 1 passing (889ms)


Spec Files:      1 passed, 1 total (100% completed) in 00:00:02
```

Рисунок 4.3 – результат запуску згенерованого автотесту

```yaml
/pet/findByTags:
    get:
      tags:
      - "pet"
      summary: "Finds Pets by tags"
      description: "Multiple tags can be provided with comma separated strings. Use\
        \ tag1, tag2, tag3 for testing."
      operationId: "findPetsByTags"
      produces:
      - "application/json"
      - "application/xml"
      parameters:
      - name: "tags"
        in: "query"
        description: "Tags to filter by"
        required: true
        type: "array"
        items:
          type: "string"
        collectionFormat: "multi"
      responses:
        200:
          description: "successful operation"
          schema:
            type: "array"
            items:
              $ref: "#/definitions/Pet"
        400:
          description: "Invalid tag value"
      security:
      - petstore_auth:
        - "write:pets"
        - "read:pets"
      deprecated: true
```

Рисунок 4.4 – API-опис у форматі Swager для pet/findByTags

Запустимо генерацію автотестів:

```javascript
it('successful_operation', async () => {
    const response = await fetch('https://petstore.swagger.io/v2/pet/findByTags?tags=tag1,tag2,tag3', {
        method: 'GET',
        headers: {
            'api_key': 'special-key'
        }
    });
    const data = await response.json();
    expect(response.status).toBe(200);
    expect(Array.isArray(data)).toBe(true);
});
```

Рисунок 4.5 – згенерований автотест

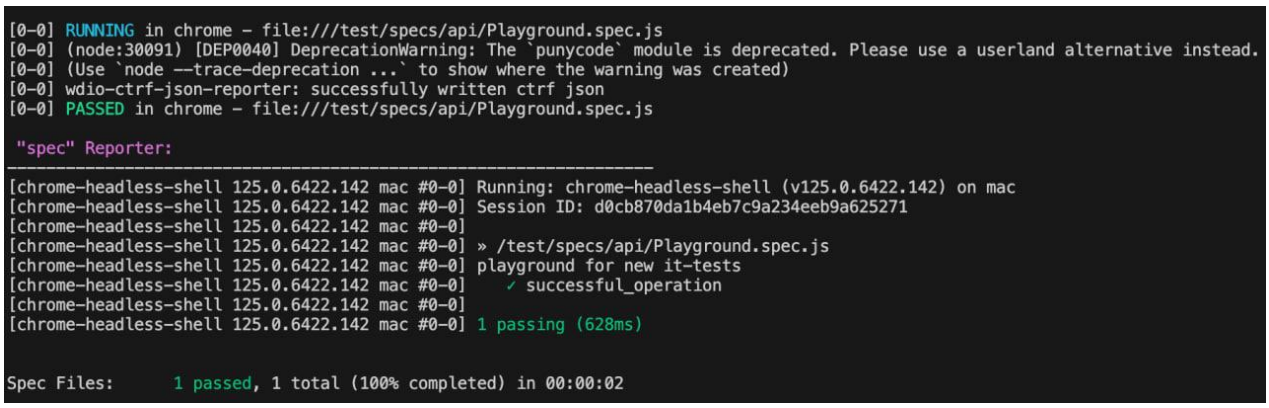


Рисунок 4.6 – результат запуску згенерованого автотесту

Дослідження роботи показало що програма успішно генерує тести, відкидає невалідні та перевіряє API на роботу

4.2 Огляд інтерфейсу програми

Програма використовує консоль як основний інтерфейс комунікації з користувачем. Консольний інтерфейс не передбачає взаємодію з користувачем. Комунікація користувача з програмаю відбувається у файлах промпту та Swagger-опису.

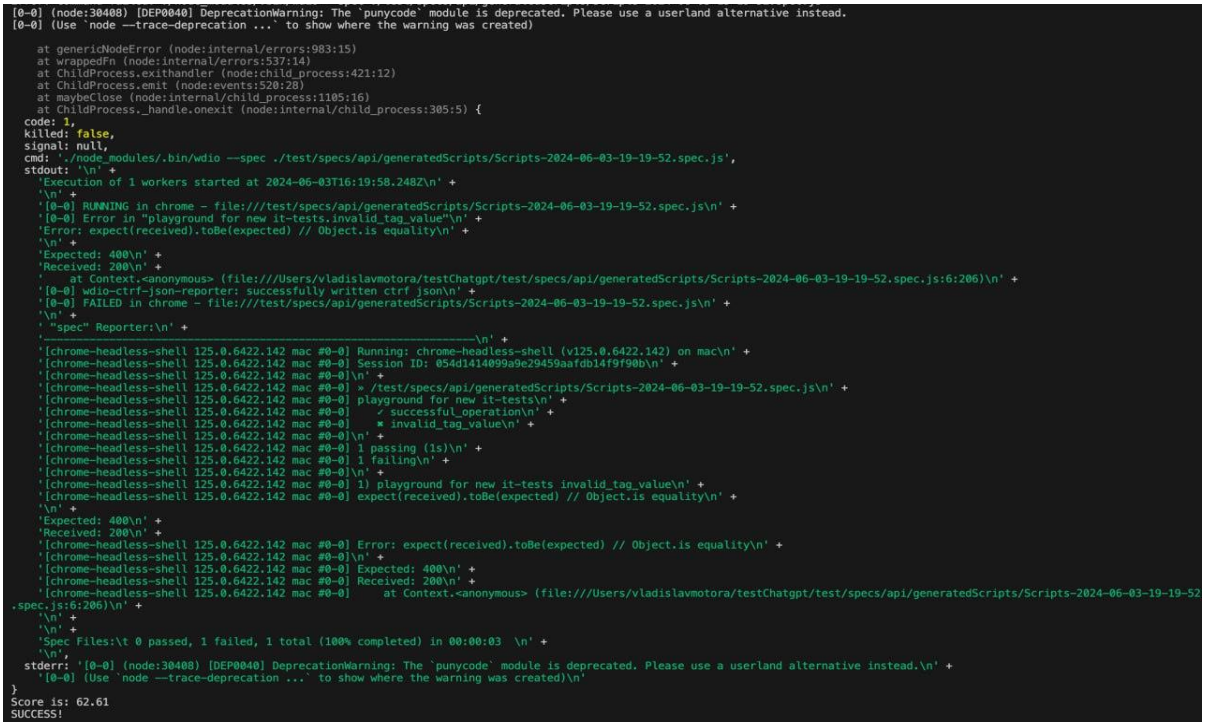


Рисунок 4.7 – консоль під час використання програми

Консоль надає користувачеві усю необхідну інформацію про хід виконання програми, її кроки, а також показує коефіцієнт наданий Swagger про ефективність проходження мутаційного тестування.

Детально діаграма повідомлень в консолі описана в додатку (див. додаток 3)

4.3 Рекомендації щодо розвитку та вдосконалення програми

Для забезпечення подальшого розвитку та вдосконалення програми, яка використовується для генерації та валідації автотестів, пропонується реалізація наступних покращень. Ці рекомендації спрямовані на підвищення ефективності, гнучкості та зручності використання програми для інженерів та розробників.

4.3.1 Ітераційний підхід

Одним з ключових напрямків вдосконалення програми є впровадження ітераційного підходу до генерації та валідації автотестів. Це дозволить програмі тричі відпрацьовувати цикл генерації та валідації створених автотестів. Ітераційний підхід включає наступні етапи:

Перша ітерація: Програма генерує початковий набір автотестів на основі наданих специфікацій та вимог. Ці автотести проходять первинну валідацію для виявлення основних помилок та невідповідностей.

Друга ітерація: На основі результатів первинної валідації програма вносить корективи та покращення до автотестів. Цей оновлений набір автотестів проходить повторну валідацію для перевірки виправлених помилок та виявлення нових можливих проблем.

Третя ітерація: Програма виконує остаточне доопрацювання автотестів, враховуючи результати попередніх ітерацій. Після цього автотести проходять фінальну валідацію для забезпечення їхньої відповідності вимогам та специфікаціям.

Такий підхід дозволяє значно підвищити якість та точність автотестів, зменшуючи кількість помилок та невідповідностей. Крім того, ітераційний підхід забезпечує більш глибоке тестування та валідацію, що сприяє підвищенню надійності та стабільності програмного забезпечення.

4.3.2 Покращення гнучкості

Для забезпечення більшої гнучкості програми пропонується додати можливість налаштування, де інженер зможе самостійно вибирати фреймворк, з яким він хоче працювати, або використовувати іншу систему опису документації замість Swagger. Це включає наступні аспекти:

Вибір фреймворку: Програма повинна надавати інженерам можливість вибору з декількох популярних фреймворків для генерації автотестів, таких як JUnit, TestNG, PyTest та інші. Це дозволить інженерам використовувати ті інструменти, з якими вони найбільш знайомі та які найкраще відповідають їхнім потребам.

Альтернативні системи опису документації: Крім Swagger, програма повинна підтримувати інші системи опису документації, такі як OpenAPI, RAML або API Blueprint. Це забезпечить більшу гнучкість у виборі інструментів для опису API та дозволить інженерам використовувати ті системи, які найбільше підходять для їхніх проектів.

Реалізація цих покращень дозволить програмі стати більш універсальною та адаптивною до різних потреб користувачів. Інженери зможуть налаштовувати програму відповідно до своїх вимог та уподобань, що сприятиме підвищенню ефективності їхньої роботи та якості створених автотестів.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі представлене готове програмне забезпечення розроблюване протягом виконання дипломної роботи. Надані вказівки щодо того як користуватись продуктом, описані залежності та можливості конфігурації.

Було проведено запуск та дослідження виконання роботи на прикладах декількох кінцевих точок та було описано налаштування запитів до LLM-моделі.

В кінці розділу пропонуються можливі способи покращення програмного продукту що можуть бути реалізовані в майбутньому

ВИСНОВКИ

Під час виконання дипломної роботи було розроблено комплексну систему для генерації та валідації автотестів, яка базується на сучасних технологіях штучного інтелекту та машинного навчання. Ця система надає можливість інженерам та розробникам автоматизувати процес створення тестів, що значно підвищує ефективність та якість програмного забезпечення.

У першому розділі роботи було проведено детальний аналіз існуючих методів генерації автотестів та їхніх недоліків. Було розглянуто різні підходи до автоматизації тестування, включаючи використання штучного інтелекту та машинного навчання. Окрему увагу було приділено аналізу сучасних інструментів та фреймворків, які використовуються для створення автотестів, таких як JUnit, TestNG, PyTest та інші.

У другому розділі було описано архітектуру розробленої системи, а також технології, які були використані для її реалізації. Було обґрунтовано вибір цих технологій та їхні переваги у контексті автоматизації тестування. Також у цьому розділі було розглянуто можливості інтеграції системи з існуючими інфраструктурами через API, що забезпечує високу гнучкість та адаптивність рішення.

У третьому розділі було детально описано процес розробки системи, включаючи реалізацію основних алгоритмів генерації та валідації автотестів. Було продемонстровано фрагменти коду та інтерфейс користувача, що дозволяє інженерам легко налаштовувати та використовувати систему.

У четвертому розділі було проведено детальний аналіз ефективності розробленої системи. Було представлено результати тестування системи на різних наборах даних та сценаріях, що дозволило оцінити її продуктивність та

точність. Також було розглянуто можливі проблеми та обмеження системи, а також запропоновано шляхи їх вирішення.

На основі проведеного дослідження було розроблено рекомендації щодо подальшого розвитку та вдосконалення програми. Зокрема, запропоновано впровадження ітераційного підходу до генерації та валідації автотестів, що дозволить значно підвищити їхню якість та точність. Також було запропоновано додати можливість налаштування, де інженери зможуть самостійно вибирати фреймворк для роботи та використовувати альтернативні системи опису документації замість Swagger.

Таким чином, розроблена система відповідає всім зазначеним на початку вимогам та має великий потенціал для подальшого розвитку та вдосконалення. Вона забезпечує високу ефективність та точність генерації автотестів, що сприяє підвищенню якості програмного забезпечення та зменшенню витрат на його тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ACCELQ: #1 AI-Powered Codeless Test Automation QA Tool. *ACCELQ Inc*. URL: https://www.accelq.com/
2. Automated API Functional Testing | ReadyAPI Platform. *Software Testing, Monitoring, Developer Tools | SmartBear*. URL: https://smartbear.com/product/ready-api/api-functional-testing/
3. GitHub - meqaio/swagger_meqa: Auto generate and run tests using swagger/OpenAPI spec, no coding needed. *GitHub*. URL: https://github.com/meqaio/swagger_meqa
4. GitHub - Cornutum/tcases: A model-based test case generator. *GitHub*. URL: https://github.com/Cornutum/tcases
5. Kariyawsam R. Codeless Test generation for API test Automation. *Medium*. URL: https://ksrajith.medium.com/codeless-test-generation-for-api-test-automation-cf0bd5c01d04
6. How to Build an API Using ChatGPT. *Stoplight*. URL: https://blog.stoplight.io/how-to-build-an-api-using-chatgpt
7. Generate Test Case | Provar Documentation. *Provar Documentation*. URL: https://documentation.provar.com/documentation/using-the-test-palette/generate-test-case/
8. Enhancing Large Language Models for Text-to-Testcase Generation. *arXiv.org e-Print archive*. URL: https://arxiv.org/html/2402.11910v1
9. OpenAI [Інтернет-ресурс] - https://openai.com/
10. Google AI for Developers | Build with the Google Gemini API and Gemma open models | Google for Developers. *Google for Developers*. URL: https://ai.google.dev

11. API Documentation & Design Tools for Teams | Swagger. *API Documentation & Design Tools for Teams | Swagger*. URL: https://swagger.io/
12. How to Get Started with Swagger API Testing | BlazeMeter by Perforce. *BlazeMeter*. URL: https://www.blazemeter.com/blog/swagger-api-testing
13. Postman API Platform [Інтернет-ресурс] - https://www.postman.com/
14. GitHub - Redocly/redoc: OpenAPI/Swagger-generated API Reference Documentation. *GitHub*. URL: https://github.com/Redocly/redoc
15. An Automated Approach to Generate Test Cases From Use Case Description Model. *Tech Science Press*. URL: https://www.techscience.com/CMES/v119n3/29796
16. Mutation Testing mit Stryker - Testautomatisierung.org. *Testautomatisierung.org*. URL: https://www.testautomatisierung.org/mutation-testing-mit-stryker/
17. Gillis A. S. What is Mutation Testing? | Definition from TechTarget. *IT Operations*. URL: https://www.techtarget.com/searchitoperations/definition/mutation-testing
18. Enhancing Large Language Models for Text-to-Testcase Generation. *arXiv.org e-Print archive*. URL: https://arxiv.org/html/2402.11910v1
19. Index | Node.js v22.2.0 Documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/docs/latest/api/
20. Getting Started | WebdriverIO. *WebdriverIO Â· Next-gen browser and mobile automation test framework for Node.js | WebdriverIO*. URL: https://webdriver.io/docs/gettingstarted
21. Mocha - the fun, simple, flexible JavaScript test framework. *Mocha - the fun, simple, flexible JavaScript test framework*. URL: https://mochajs.org/

22. Microsoft. Visual Studio Code - Code Editing. Redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (date of access: 04.06.2024).
23. Large Language Models as Test Case Generators: Performance Evaluation and Enhancement. *arXiv.org e-Print archive*. URL: https://arxiv.org/html/2404.13340v1
24. Large Language Models as Test Case Generators: Performance Evaluation and Enhancement. *arXiv.org e-Print archive*. URL: https://arxiv.org/html/2404.13340v1
25. Stack Overflow - Where Developers Learn, Share, & Build Careers. *Stack Overflow*. URL: https://stackoverflow.com/
26. shveta parnami. Empirical Validation of Test Case Generation based on All-edge Coverage Criteria. *Academia.edu - Share research*. URL: https://www.academia.edu/59516905/Empirical_Validation_of_Test_Case_Generation_based_on_All_edge_Coverage_Criteria?uc-sb-sw=9063775
27. Validating Test Case Migration via Mutation Analysis | Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test. *ACM Conferences*. URL: https://dl.acm.org/doi/abs/10.1145/3387903.3389319
28. Language Models are Few-Shot Learners. *arXiv.org*. URL: https://arxiv.org/abs/2005.14165

ДОДАТОК 1

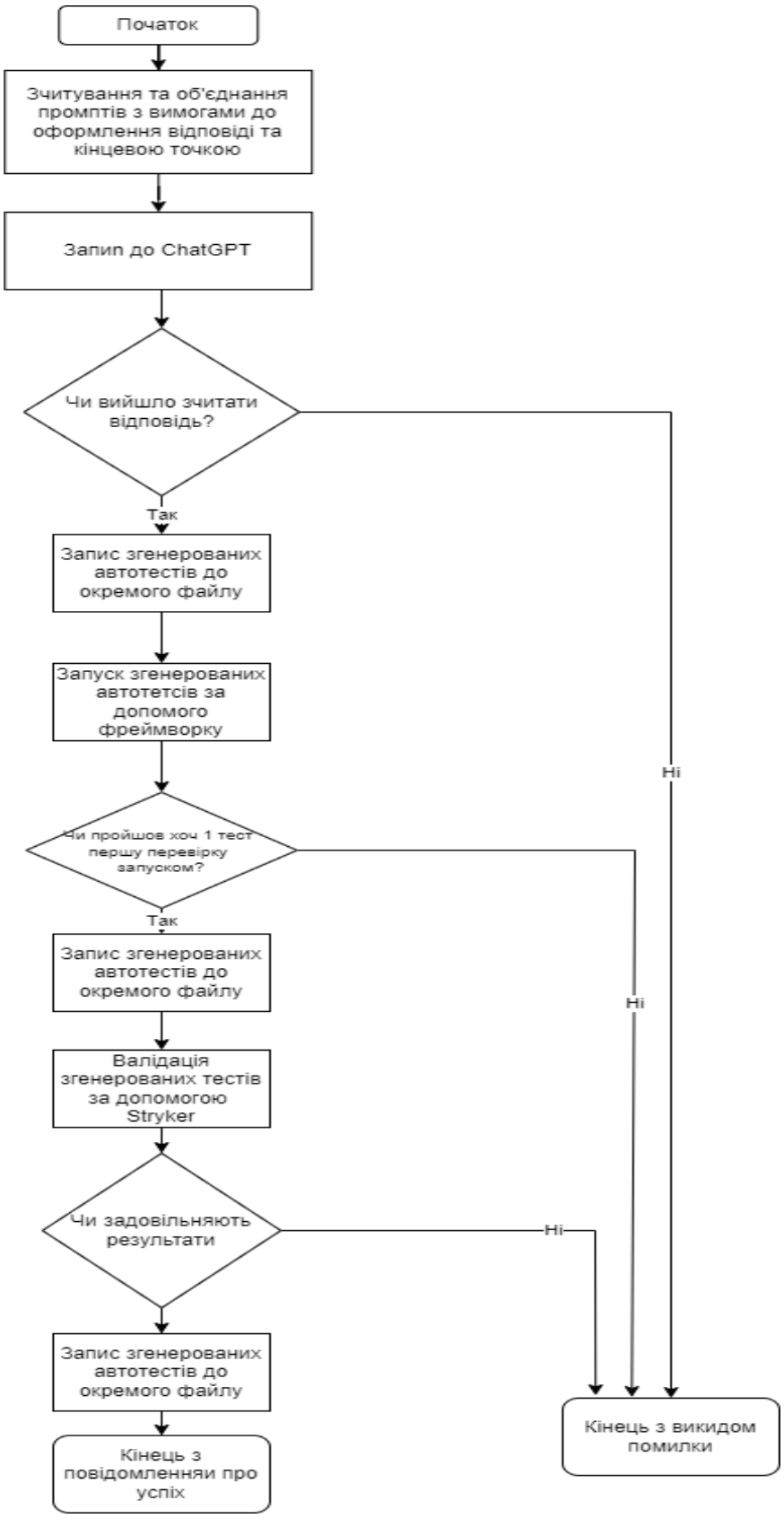
Програмне забезпечення для автогенерації тестів за допомогою LLM

Логічна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Мотора В. С.			**Програмне забезпечення для автогенерації тестів за допомогою LLM** Логічна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Нечай Д. А.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-06	
Н. Контр.		Ковальчук О. М.					
Затвердив							

ДОДАТОК 2

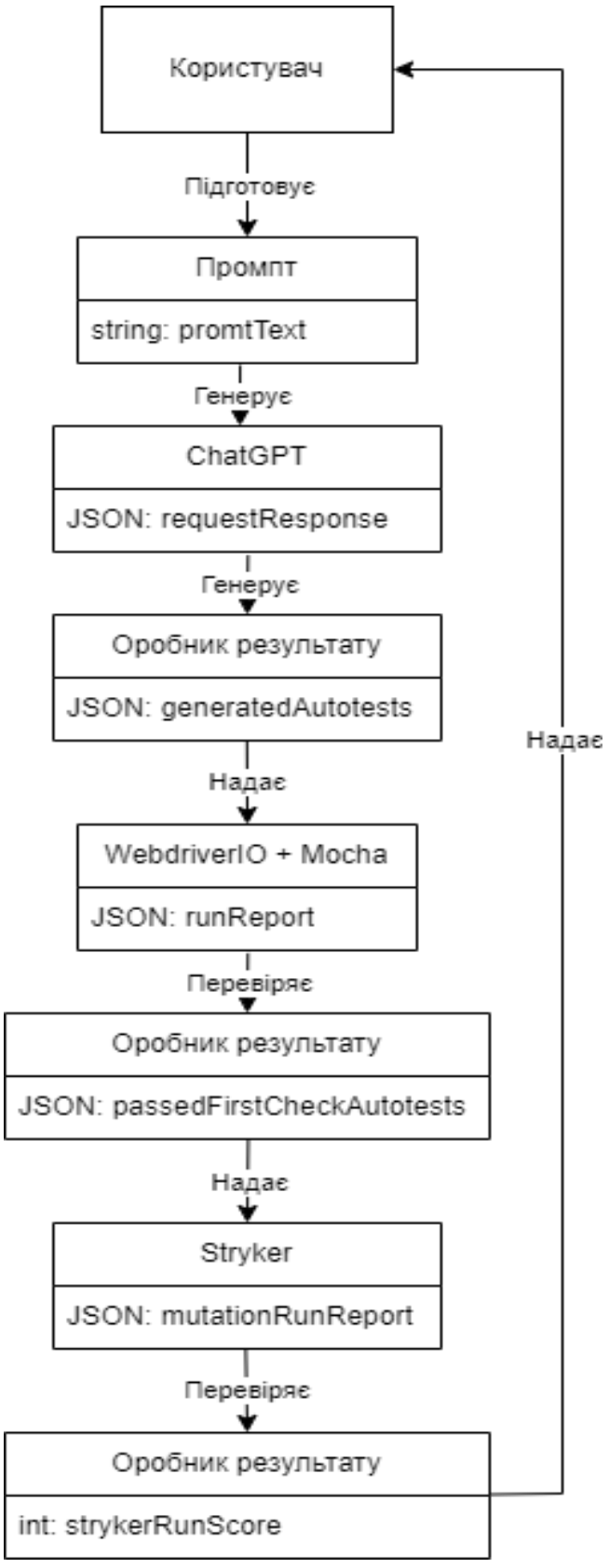
Програмне забезпечення для автогенерації тестів за допомогою LLM

Діаграма дата-акторів

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Мотора В. С.			**Програмне забезпечення для автогенерації тестів за допомогою LLM** Діаграма дата-акторів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Нечай Д. .				
Н. Контр.		Ковальчук О. М.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-06
Затвердив						

ДОДАТОК 3

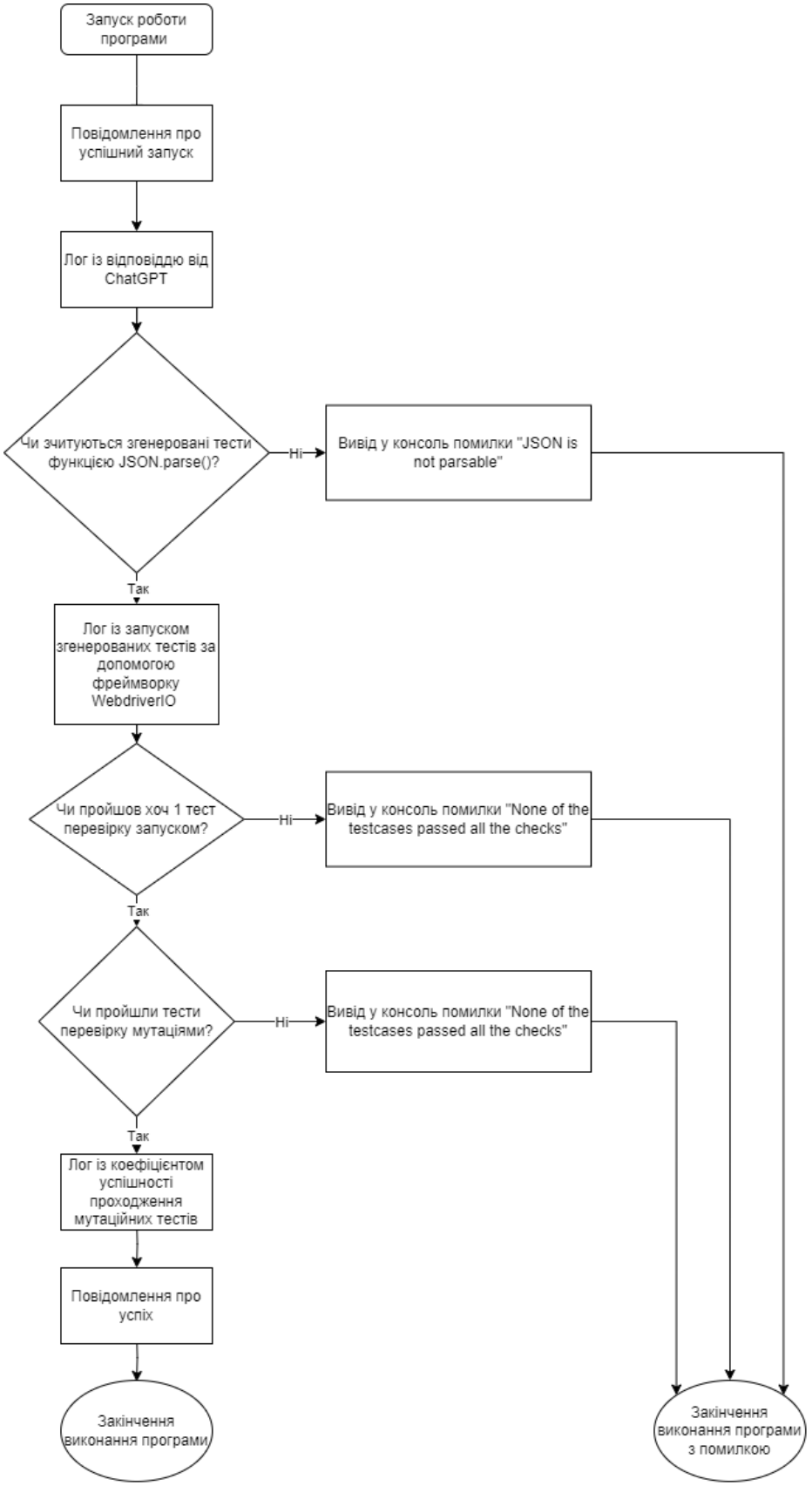
Програмне забезпечення для автогенерації тестів за допомогою LLM

Діаграма повідомлень в консолі

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Мотора В. С.			**Програмне забезпечення для автогенерації тестів за допомогою LLM**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Нечай Д. А.				
Н. Контр.		Ковальчук О. М.			Діаграма повідомлень в консолі	КПІ ім. Ігоря Сікорського, ФІОТ, ІО-06
Затвердив						

ДОДАТОК 4

Програмне забезпечення для автогенерації тестів за допомогою LLM

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 15

Київ 2024 р

```
const config = {
  packageManager: "npm",
  mutator: {
    plugins: []
  },
  mutate: [ "./test/specs/api/generatedScriptsForStrykerCheck/*.js" ],
  reporters: ['json', 'clear-text'],
  testRunner: "mocha",
  disableTypeChecks : true,
  cleanTempDir : true,
  jsonReporter : {
    fileName: "reports/mutation/mutation.json"
  },
  tempDirName: 'strykerTmp',
  logLevel : "info",
  coverageAnalysis: "perTest",
  "mochaOptions": {
    "spec": [ "./test/specs/api/generatedScriptsForStrykerCheck/*.js" ]
  }
};
export default config;

export const config = {
    runner: 'local',
```

```
    ],

suites: {
        playground: [
            './test/specs/api/Playground.spec.js',
        ]
    },
    exclude: [
    ],
    maxInstances: 4,
    capabilities: [{
        browserName: 'chrome',
        'goog:chromeOptions': {
            args:
},
    }],
    logLevel: 'silent',
    bail: 0,
    waitforTimeout: 1111,
    connectionRetryTimeout: 22222,
    connectionRetryCount: 3,
    framework: 'mocha',
    reporters: [
        'spec',
        ['ctrf-json', {}]
    ],
    mochaOpts: {
```

```
    ui: 'bdd',
    timeout: 55555
  },
                                    import fetch from 'node-fetch';
import {
  expect
} from 'expect';


describe('playground for new it-tests', async () => {
  it('successful_operation', async () => {
    const response = await fetch('https://petstore.swagger.io/v2/pet', {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        'api_key': 'special-key'
      },
      body: JSON.stringify({
        id: Math.floor(Math.random() * 1000),
        name: 'Doggie',
        status: 'available'
      })
    });
    const data = await response.json();
    expect(response.status).toBe(200);
    expect(data).toHaveProperty('id');
    expect(data).toHaveProperty('name', 'Doggie');
```

```
        expect(data).toHaveProperty('status', 'available');
    });


})
import fetch from 'node-fetch';
import {
    expect
} from 'expect';



describe('playground for new it-tests', async () => {
    it('successful_operation', async () => {
        const response = await fetch('https://petstore.swagger.io/v2/pet', {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
                'api_key': 'special-key'
            },
            body: JSON.stringify({
                id: Math.floor(Math.random() * 1000),
                name: 'Doggie',
                status: 'available'
            })
        });
        const data = await response.json();
        expect(response.status).toBe(200);
        expect(data).toHaveProperty('id');
```

```
        expect(data).toHaveProperty('name', 'Doggie');
        expect(data).toHaveProperty('status', 'available');
    });


})
```

Write an API test for this endpoint, described by Swagger with these requirements:

1. Use webdriverio framework
2. Write only it-blocks
3. For API requests use only "fetch" library
4. Don't write anything but it-blocks
5. Don't write any import code
6. Don't write ANY comments
7. Generate random data for request data
8. Use the api key "special-key" to test the authorization filters
9. Write the output in the format of .json.
10. use the example below: underscore should replac spaces in the names

```
{
    {
        "name" : name_of_the_script,
        "script_code" : script_code
    },
    {
        "name" : name_of_the_script,
        "script_code" : script_code
    }
}
```

use CORRECT ammount of brackets, the JSON you provide to me needs to be parsable via code

don't put commas at the end of scopes

in JSON format, message starts and ends with brackets "[]"

11. Don't write anything but this json

12. Make script_code parsable via JSON.parse() function

13. Name of the it-block should always match the name_of_the_script

14. Generate one it-block for each response code, they all should be in JSON you will send me back

15. DO NOT HIGHLIGHT JSON WITH ```, I NEED ONLY JSON ITSELF

16. Webcite URL is - https://petstore.swagger.io/

here's the swagger description:

```
import fs from 'node:fs';

class PromptManager {
    #readSwaggerDescription() {
        let promptText
        try {
            const data = fs.readFileSync('./promptManager/swaggerDescription.txt', 'utf8');
            promptText = data;
        } catch (err) {
            console.error(err);
        }
```

```
            return promptText
        }
        #readPromptRequirements() {
            let promptText
            try {
                const data = fs.readFileSync('./promptManager/promptRequirements.txt', 'utf8');
                promptText = data;
            } catch (err) {
                console.error(err);
            }

            return promptText
        }

        getFinalPrompt() {
            return `${this.#readPromptRequirements()} + \n + \`\`\` + \n + ${this.#readSwaggerDescription()} + \n + \`\`\``
        }
    }

    export default new PromptManager();
    import fs from 'node:fs';
    import path from 'node:path';
    import util from 'util';
    import child_process from 'child_process';
```

```
const exec = util.promisify(child_process.exec);

const PATH_TO_JSON = './reports/mutation/mutation.json'
const PATH_TO_CTRF = './ctrf/'


class Scripts {
    #countInstances(str, word) {
        return str.split(word).length - 1;
    }

    countScoreFromJSON() {
        let JSONString = fs.readFileSync(PATH_TO_JSON, 'utf8', (err) => {
            if (err) {
              console.error(err);
              return;
            }
        });
        const survivedNum = this.#countInstances(JSONString, '"status":"Killed"')
        const allNum = this.#countInstances(JSONString, '"status":')
        return ((survivedNum / allNum) * 100).toFixed(2);
    }


    clearAllCTRFFolder(){
        async function removeRecursively(directory) {
```

```
            await fs.rmdir(directory, { recursive: true }, (err) => err &&
console.error(err));
        }

        // Call the function
        removeRecursively('./ctrf')
            .then(() => console.log('Directory and all its contents have been removed
recursively.'))
    }

    dataToFillToWdioRun(itBlock){
        return`
        import fetch from 'node-fetch';

        describe('playground for new it-tests', async () => {
            ${itBlock}
        })
        `
    }

    dataToFillToStrykerRun(itBlock){
        return`
    import fetch from 'node-fetch';
    import { expect } from 'expect';


    describe('playground for new it-tests', async () => {
```

```
    ${itBlock}
})
`
    }

    getCurrentDateAsString(){
        let date_ob = new Date();

        // current date
        // adjust 0 before single digit date
        let date = ("0" + date_ob.getDate()).slice(-2);

        // current month
        let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);

        // current year
        let year = date_ob.getFullYear();

        // current hours
        let hours = date_ob.getHours();

        // current minutes
        let minutes = date_ob.getMinutes();

        // current seconds
        let seconds = date_ob.getSeconds();
```

```
        return year + "-" + month + "-" + date + "-" + hours + "-" + minutes + "-" +
seconds
    }

    async runTheGeneratedScriptByStryker() {
        try {
            const { stdout, stderr } = await exec(`npm run launch:Stryker`);
              console.log('stdout:', stdout);
              console.log('stderr:', stderr);
        } catch (e) {
            console.error(e)
        }
    }

    async sourceFromBashProfile() {
        try {
            const { stdout, stderr } = await exec(`source ~/.bash_profile`);
            //   console.log('stdout:', stdout);
            //   console.log('stderr:', stderr);
        } catch (e) {
            console.error(e)
        }
    }

    async runTheGeneratedScript(fileName) {
        try {
```

```
            const { stdout, stderr } = await exec(`./node_modules/.bin/wdio --spec ./test/specs/api/generatedScripts/${fileName}`);
              console.log('stdout:', stdout);
              console.log('stderr:', stderr);
          } catch (e) {
            console.error(e)
          }
        }

        #getNameOfTheCtrfFile(pathToCtrf){ //PATH_TO_CTRF
          return (fs.readdirSync(pathToCtrf).filter((allFilesPaths) =>
            allFilesPaths.match(/\.json$/) !== null))[0]
        }

        getNamesOfTheScriptsWithStatusPassedFromJSON(){
          //      console.log({'this.#getNameOfTheCtrfFile(PATH_TO_CTRF)' : this.#getNameOfTheCtrfFile(PATH_TO_CTRF)})
            const CTRFReportName = this.#getNameOfTheCtrfFile(PATH_TO_CTRF)

            const reportJSON = JSON.parse(fs.readFileSync(path.join(PATH_TO_CTRF, CTRFReportName), 'utf8', (err) => {
              if (err) {
                console.error(err);
                return;
              }
```

```
        }))

        // console.log({'reportJSON' : reportJSON})

        const JSONOfPassedTests = reportJSON.results.tests.filter(element =>
element.status == "passed")
        return JSONOfPassedTests.map(element => element.name)
      }
      // map(select(.product.type == "mobile")) | .[] | {id: .product.id, service:
.product.service}

      filterPassedScriptsByTheirNames(initialJson, arrWithPassedScriptsNames){
        return initialJson.filter(({ name }) => arrWithPassedScriptsNames.some(m
=> name === m))
      }
    }

    export default new Scripts();
    import fs from 'node:fs/promises';
    import path from "node:path";

    const directory = "./test/specs/api/generatedScripts";
    const strykerDirectory = "./test/specs/api/generatedScriptsForStrykerCheck";

    async function clearGeneratedScriptsFolder(){
      for (const file of await fs.readdir(directory)) {
        await fs.unlink(path.join(directory, file));
```

```
    }
}
async function clearGeneratedScriptsForStrykerFolder(){
    for (const file of await fs.readdir(strykerDirectory)) {
      await fs.unlink(path.join(strykerDirectory, file));
    }
}
clearGeneratedScriptsFolder()
clearGeneratedScriptsForStrykerFolder()
import OpenAI from 'openai';
import * as dotenv from "dotenv";
import fs from 'node:fs';

import scripts from './helper/scripts.js';
import promptManager from './promptManager/promptManager.js';

dotenv.config();
scripts.clearAllCTRFFolder()
const currentDate = scripts.getCurrentDateAsString()
const fileName = `Scripts-${currentDate}.spec.js`
const promptText = promptManager.getFinalPrompt()
const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});

async function main() {
    await scripts.sourceFromBashProfile()
```

```
    const chatCompletion = await openai.chat.completions.create({
      messages: [{ role: 'user', content: promptText }],
      model: 'gpt-4-turbo',
      temperature: 0,
      max_tokens: 3000,
      top_p: 0.5,
      frequency_penalty: 0,
      presence_penalty: 0
    });

    const content = chatCompletion.choices[0].message.content
    // console.log(content)

    const outputJson = JSON.parse(chatCompletion.choices[0].message.content)

    let textToFill = ''

    outputJson.forEach(element => {
      textToFill += element.script_code + '\n'
    });

    fs.writeFile(`./test/specs/api/generatedScripts/${fileName}`,
scripts.dataToFillToWdioRun(textToFill), (err) => err && console.error(err));

    await scripts.runTheGeneratedScript(fileName)
```

```
const arrayOfNamesOfPassedTests = scripts.getNamesOfTheScriptsWithStatusPassedFromJSON()
// console.log({'arrayOfNamesOfPassedTests' : arrayOfNamesOfPassedTests})

if(arrayOfNamesOfPassedTests.length === 0){
   throw new Error("None of the testcases passed all the checks")
}

const passedSciptsJson = scripts.filterPassedScriptsByTheirNames(outputJson, arrayOfNamesOfPassedTests)

// console.log({'passedSciptsJson' : passedSciptsJson})
let codeToFillToStrykerTest = ''

passedSciptsJson.forEach(element => {
   codeToFillToStrykerTest += element.script_code + '\n'
});

fs.writeFile(`./test/specs/api/generatedScriptsForStrykerCheck/${fileName}`, scripts.dataToFillToStrykerRun(codeToFillToStrykerTest), (err) => err && console.error(err));

await scripts.runTheGeneratedScriptByStryker(fileName)
```

```
        if(scripts.countScoreFromJSON() < 60){
            throw new Error("None of the testcases passed all the checks")
        }

        console.log('Score is: ' + scripts.countScoreFromJSON())

    fs.writeFile(`./test/specs/api/generatedScriptsCheckedAndVerified/${fileName}`, scripts.dataToFillToStrykerRun(codeToFillToStrykerTest), (err) => err && console.error(err));

        console.log('SUCCESS!')
        console.log('check ./test/specs/api/generatedScriptsCheckedAndVerified folder for Checked And Verified Testcases')
    }
main();
```