

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ центр післядипломної освіти _____
(повна назва)

Кафедра _____ програмної інженерії _____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ другий (магістерський) _____

Дослідження та реалізація методів
автоматизованого тестування веб-додатків
_____ (тема)

Виконав:
студент(ка) 2 курсу, групи ІПЗЗдМ-22-1

_____ Дем'яненко М.С. _____
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного
забезпечення
(код і повна назва спеціальності)

Тип програми _____ освітньо-наукова _____

Керівник доц. Чуприна А.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри

_____ (підпис)

_____ З.В.Дудар _____
(прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет _____ центр післядипломної освіти _____
 Кафедра _____ програмної інженерії _____
 Рівень вищої освіти _____ другий (магістерський) _____
 Спеціальність _____ 121 – Інженерія програмного забезпечення _____
 Тип програми _____ освітньо-наукова програма _____
 Освітня програма _____ Інженерія програмного забезпечення _____
 (шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« ____ » _____ 2024 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____ Дем'яненко Марині Сергіївні _____
 (прізвище, ім'я, по батькові)

1. Тема роботи _____ «Дослідження та реалізація методів автоматизованого тестування веб-додатків» _____

Затверджена наказом по університету від _____ 22.04.2024 р. № 60Стз _____

2. Термін подання студентом роботи до екзаменаційної комісії _____ 07.06.2024 _____

3. Вихідні дані до роботи науково-методична та науково-технічна література, дані з інтернет джерел, статті _____

4. Перелік питань, що потрібно опрацювати в роботі _____

Аналіз поточних трендів використання AI і мовних моделей у тестуванні, дослідження та впровадження новітніх методів автоматизованого тестування веб-додатків за допомогою генеративного штучного інтелекту, аналіз результатів дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	<i>виконано</i>
2	Вивчення і аналіз існуючих мовних моделей	15.02 – 24.02.24	<i>виконано</i>
3	Аналіз поточних трендів використання AI і мовних моделей у тестуванні.	17.02 – 28.02.24	<i>виконано</i>
4	Планування експериментів	25.02 – 28.02.24	<i>виконано</i>
5	Дослідження застосування великих мовних моделей у тестуванні веб-додатків	25.02 – 01.04.24	<i>виконано</i>
6	Експериментальні дослідження	02.04 – 20.04.24	<i>виконано</i>
7	Аналіз результатів дослідження	20.04 – 23.04.24	<i>виконано</i>
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	<i>виконано</i>
9	Підготовка пояснювальної записки	01.04 – 26.04.24	<i>виконано</i>
10	Підготовка презентації та доповіді	26.04 – 2.05.24	<i>виконано</i>
11	Нормоконтроль	3.05 – 08.05.24	<i>виконано</i>
12	Рецензування	08.05 – 11.05.24	<i>виконано</i>
13	Занесення диплома в електронний архів	11.05.2024	<i>виконано</i>

14	Попередній захист	11.05.2024	<i>виконано</i>
15	Допуск до захисту у зав. кафедри	13.05.2024	<i>виконано</i>

Дата видачі завдання 20 січня 2024р.

Студент (ка) _____
(підпис)

Дем'яненко М.С.

Керівник роботи _____
(підпис)

доц. Чуприна А.С.
(посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка містить: 86 с., 30 рис., 10 табл., 24 джерела.

МОВНІ МОДЕЛІ, ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ, ШТУЧНИЙ ІНТЕЛЕКТ, GPT, OPENAI, CYPRESS, REACT, JAVASCRIPT, NODE.JS, EXPRESS.

Об'єктом дослідження виступає застосування генеративного штучного інтелекту у процесі автоматизованого тестування веб-додатків, особлива увага приділяється аналізу переваг цього підходу в контексті сучасних реалій.

Метою цього дослідження є визначення та оцінка ефективності використання генеративних моделей штучного інтелекту у процесах автоматизації тестування веб-додатків. Це дослідження дозволяє глибше зрозуміти, як новітні розробки в області штучного інтелекту можуть трансформувати традиційні підходи до тестування веб-додатків, забезпечуючи значні переваги у швидкості, обсязі, і якості виконання робіт.

Дослідження базується на методах машинного навчання і обробки природної мови, використовуючи алгоритми генеративних мовних моделей, таких як GPT від OpenAI.

Результати дослідження показали, що використання генеративних мовних моделей значно збільшує ефективність тестування веб-додатків за рахунок автоматизації тестування вимог, генерації тестових сценаріїв та скриптів, а також створення комплексного звіту з тестування.

LANGUAGE MODELS, WEB APPLICATION TESTING, ARTIFICIAL INTELLIGENCE, GPT, OPENAI, CYPRESS, REACT, JAVASCRIPT, NODE.JS, EXPRESS.

The object of this research is the use of generative artificial intelligence in the process of automated web application testing, with a particular emphasis on analyzing the advantages of this approach within the context of contemporary realities.

This research aims to assess and quantify the effectiveness of using generative artificial intelligence models in the automation of web application testing processes. This research provides deeper insights into how recent advances in artificial intelligence can transform traditional approaches to web application testing, offering significant improvements in terms of speed, coverage, and quality of execution.

The study is based on machine learning and natural language processing methods, utilizing algorithms from generative language models such as GPT by OpenAI.

The research results demonstrated that the use of generative language models significantly enhances the efficiency of web application testing through the automation of requirement testing, generation of test scenarios and scripts, and the creation of comprehensive testing reports.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу EIArKhNURE.

Я, Дем'яненко Марина Сергіївна, студентка гр. ПЗЗдм-22-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження та реалізація методів автоматизованого тестування веб-додатків», що буде представлена на екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу EIArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	10
1.1 Сучасні методи тестування веб-додатків	10
1.2 Застосування штучного інтелекту та великих мовних моделей у тестуванні веб-додатків	13
1.3 Постановка задачі дослідження	17
2 ОПИС ПРОЕКТНИХ РІШЕНЬ	19
2.1 Аналіз існуючих мовних моделей	19
2.2 Математичний опис проблемної області дослідження	20
2.3 Проектування засобів для забезпечення експерименту	27
2.4 Планування експериментального дослідження	30
3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ У ТЕСТУВАННІ ВЕБ-ДОДАТКІВ	33
3.1 Дослідження тестування вимог (Requirements Testing)	33
3.2 Дослідження генерації тестових сценаріїв (Test Case Generation)	36
3.3 Дослідження генерації авто тестів	40
3.4 Дослідження створення звіту з тестування (Test Report Creation)	54
4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ	58
4.1 Аналіз ефективності за критеріями	58
4.2 Рекомендації щодо подальшого застосування	62
ВИСНОВКИ	64
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	66
ДОДАТОК А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії	69
ДОДАТОК Б Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ	70
ДОДАТОК В Слайди презентації	71

	9
ДОДАТОК Г Апробація результатів роботи	75
ДОДАТОК Д Код веб-додатку для експерименту	78
ДОДАТОК Е Код Python програми для експерименту	
ДОДАТОК Ж Експертний висновок нормоконтроль	79

ВСТУП

Сучасний технологічний прогрес із стрімкими темпами розвитку впливає на вимоги до якості та надійності програмного забезпечення, зокрема, веб-додатків. Розробка веб-додатків як невід'ємної частини сучасного інформаційного простору, вимагає постійного вдосконалення методів їх тестування. В цьому контексті, автоматизоване тестування веб-додатків стає не тільки актуальним, але й невід'ємним елементом забезпечення високої якості і ефективності продукції. на гарантування надійності та відмінності програмного продукту. [1]

Метою цієї роботи є дослідження та впровадження новітніх методів автоматизованого тестування веб-додатків за допомогою генеративного штучного інтелекту. Робота спрямована на поглиблене вивчення інструментів автоматизації в сучасному програмуванні, аналіз доступних засобів та їх ефективність у процесах розробки програмного забезпечення, та виявлення найбільш придатних інструментів для специфічних задач.

Актуальність дослідження зумовлена стрімким розвитком сучасних технологій та активною цифровою трансформацією в різних галузях. Умови постійного збільшення складності веб-додатків та розширення їх обсягу підкреслюють необхідність своєчасного та ефективного забезпечення високої якості програмного забезпечення. Автоматизоване тестування веб-додатків стає важливим інструментом для вирішення цього завдання, забезпечуючи виявлення та виправлення дефектів, мінімізацію витрат на виправлення дефектів, підтримку надійності та стабільності веб-додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасні методи тестування веб-додатків

Важливість проведення тестування важко переоцінити, оскільки сучасні веб-додатки стають все більш складними та удосконаленими. Вони відрізняються складним та різноманітним інтерфейсом, який активно взаємодіє з користувачем, витримує інтенсивне навантаження, використовує асинхронне завантаження контенту та оновлень, працює з кешуванням інформації та навіть може працювати в режимі офлайн. [1]

У процесі розробки програмного забезпечення існують різні етапи тестування, які більш відомі як рівні тестування (див. рис. 1.1). Основна мета кожного рівня полягає в тому, щоб зосередитися на конкретному аспекті функціонування програмного продукту, забезпечити високу якість, та мінімізувати кількість потенційних дефектів, що в свою чергу мінімізує витрати на розробку та підтримку, покращує авторитет компанії та користувацький досвід. [2]

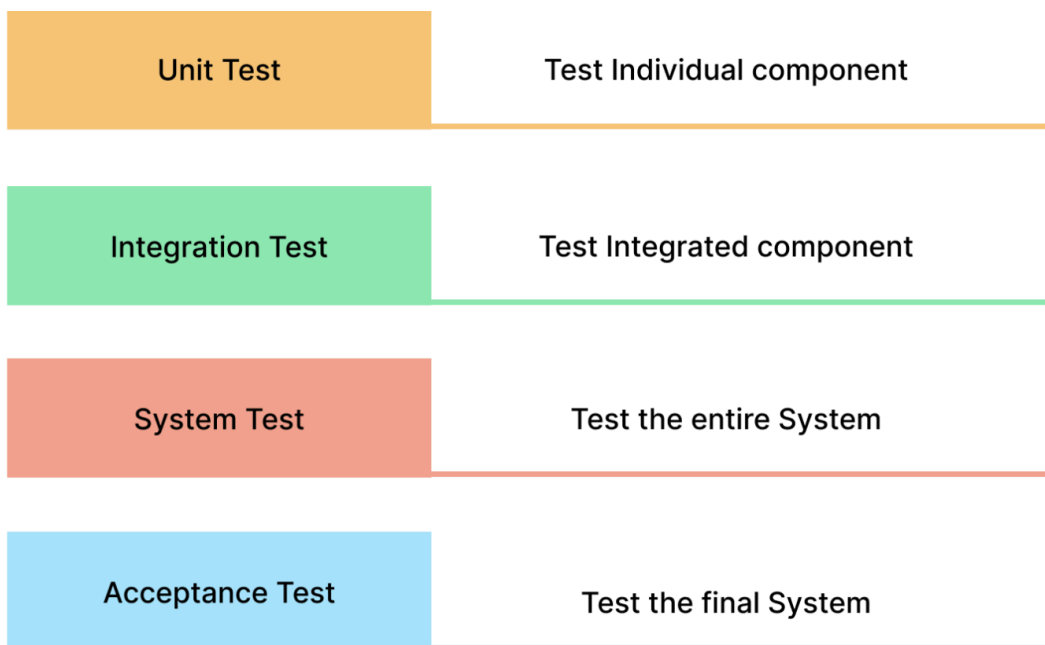


Рисунок 1.1 - Типи рівнів тестування (за даними [2])

На першому рівні тестування, що називається юніт-тестуванням або тестуванням модулів, відбувається оцінка найменших складових системи, які можна скомпілювати. Цей підхід дозволяє випробувати кожен модуль програмного

забезпечення окремо, перевіряючи, чи виконують компоненти системи свої функції належним чином.

На наступному етапі проводиться аналіз кількох модулів як єдиної групи, де перевіряється потік даних від одного модуля до іншого, також проводиться функціональне тестування. Цей процес допомагає виявити помилки або проблеми, що можуть виникнути через неправильне кодування або недоліки в інтеграції між модулями. [3]

Під час системного тестування оцінюється загальна функціональність та продуктивність повного та повністю інтегрованого програмного рішення.

До типів системного тестування відносяться:

- тестування продуктивності, тобто здійснюється перевірка швидкодії, масштабованості, стабільності та надійності програмного продукту або застосунку;
- тестування навантаження, що здійснюється для визначення поведінки системи або програмного продукту під екстремальним навантаженням;
- тестування на міцність, тобто перевірки надійності системи під різними навантаженнями;
- тестування масштабованості – проводиться для перевірки продуктивності програмного застосунку або системи з точки зору її здатності масштабуватися вгору або вниз в залежності від навантаження [4].

На останньому етапі здійснюється тестування прийняття (Acceptance testing), яке включає тестування функціональних та нефункціональних аспектів системи, таких як продуктивність, безпека, зручність використання, доступність, сумісність та надійність. Залежно від складності системи, це може бути зроблено вручну або за допомогою автоматизованих засобів. [2]

Основна мета тестування прийняття - перевірити, чи відповідає розроблений продукт вимогам і очікуванням користувачів, бізнесу та інших зацікавлених сторін.

Під час аналізу розглянутих етапів тестування стає очевидним, що функціональне та нефункціональне тестування здійснюються на різних етапах життєвого циклу розробки програмного забезпечення і мають велике значення.

Функціональне тестування - це вид тестування, що зосереджений на оцінці функціональності програмного забезпечення. Це тестування забезпечує виконання програмним забезпеченням вимог та специфікацій, які були визначені на етапі розробки. З іншого боку, нефункціональне тестування оцінює нефункціональні аспекти програмного забезпечення, такі як продуктивність, масштабованість, безпека та зручність використання. [5]

Під час проведення функціонального тестування тестувальник намагається перевірити, що програмне забезпечення працює ("функціонує") відповідно до технічних, бізнесових і вимог користувачів. Це форма тестування "чорної скриньки", під час якої перевіряється кожна функція веб-сайту / додатка, щоб вона відповідала відповідним вимогам.

Кілька типових прикладів того, що функціональний тест перевіряє:

- чи можуть користувачі легко та безперешкодно увійти (login) на сайт / додаток за допомогою правильних облікових даних?
- коли користувач натискає "Купити зараз", чи переходить інтерфейс користувача безпосередньо на наступну необхідну сторінку?
- якщо користувач вводить спеціальні символи в поле, де вони не дозволені, чи система виводить повідомлення про помилку і відмовляється продовжувати дію?

Узагальнюючи, функціональне тестування спрямоване на перевірку кожної функції з метою переконання, що вона виконує свої завдання, особливо враховуючи вимоги бізнесу та технічні вимоги. Воно також включає дослідження того, як програмне забезпечення працює з кожним набором специфікацій бізнес-логіки для досягнення очікуваних результатів. Цей процес включає як позитивні, так і негативні тести, що дозволяють визначити, як програма буде працювати в неідеальних умовах.

Нефункціональне тестування, у свою чергу, охоплює всі аспекти, які не були протестовані за допомогою функціонального тестування. Ці нефункціональні елементи програмного забезпечення включають продуктивність, зручність використання, стабільність, ефективність, підтримуваність, переносимість та інші. Подібно до функціональних тестів, у цій формі перевірки якості також використовується специфікація для створення тестових сценаріїв, наборів тестів та скриптів. Її основна мета полягає в оптимізації всіх нефункціональних елементів, щоб програмне забезпечення під час тестування могло надати найкращий можливий користувацький досвід. [6]

Обидва види тестування, як функціональне, так і нефункціональне, є важливими складовими процесу розробки програмного забезпечення. Спільно ці два види тестування сприяють покращенню якості програмного продукту та забезпеченню задоволеності користувачів найкращим користувацьким досвідом.

1.2 Застосування штучного інтелекту та великих мовних моделей у тестуванні веб-додатків

У постійно змінюваному світі розробки програмного забезпечення, інтеграція автоматизованого тестування стала незамінною для забезпечення якості, надійності та продуктивності застосунків. Однак, традиційні методи автоматичного тестування часто не в змозі встигати за зростаючою складністю та різноманіттям сучасних програмних систем. Саме тут автоматизація тестування за допомогою штучного інтелекту виходить на передову, стаючи справжнім проривом.

Штучний інтелект стає ключовим елементом у сфері автоматичного тестування, розширюючи межі можливостей тестового охоплення, підвищуючи точність та поліпшуючи продуктивність. [7]

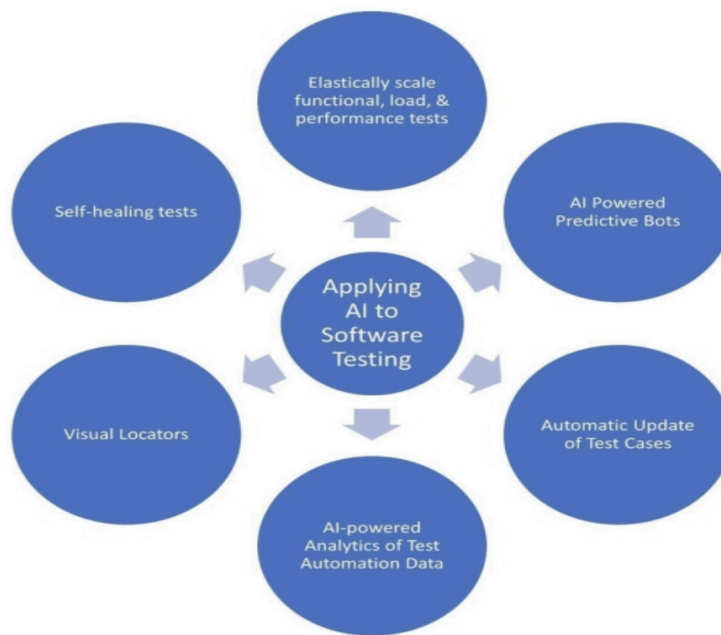


Рисунок 1.2 - Переваги використання ШІ в автоматизованому тестуванні (за даними [8])

Використовуючи потужність штучного інтелекту, процеси автоматизації тестування можуть виходити за межі обмежень ручного скриптингу та статичних тестових сценаріїв. У 2024 році автоматизація тестування на базі ШІ займає провідні позиції у сфері інновацій, обіцяючи кардинально змінити підходи до перевірки та валідації програмного забезпечення.

Впровадження штучного інтелекту у процеси автоматизованого тестування пропонує численні переваги, які значно покращують процес тестування. Однією з ключових переваг є можливість досягти неперевершених рівнів охоплення тестів та точності. Традиційні методи тестування часто не в змозі передбачати складні сценарії та швидко знаходити крайові випадки, що призводить до прогалин у охопленні тестів та потенційних вразливостей. З іншого боку, інструменти тестування на основі ШІ відзначаються вмінням аналізувати різноманітні набори даних, визначати критичні шляхи та генерувати тест-кейси, які всебічно валідують програмне забезпечення під тестуванням. Широке охоплення тестами не тільки зменшує ймовірність пропуску дефектів, але й покращує загальну якість та надійність програмного забезпечення. Інструменти ШІ також можуть

аналізувати ваше поточне охоплення тестами та вказувати на зони з недостатнім охопленням або на ділянки вашого застосунку, які можуть бути ризикованими.

Генерація тестових даних за допомогою штучного інтелекту є значним прогресом у сфері автоматизованого тестування. Традиційні методики генерації тестових даних часто покладаються на самотужки підібрані набори даних, які можуть не точно відображати сценарії реального світу. Однак алгоритми ШІ можуть аналізувати вимоги до застосунків та поведінку користувачів, щоб генерувати синтетичні тестові дані, які тісно імітують виробниче середовище. Це не лише покращує охоплення тестів, але й підвищує ефективність тест-кейсів у виявленні прихованих помилок та вразливостей. [9]

Одним з найбільш перспективних застосувань ШІ у автоматизованому тестуванні є прогнозна аналітика. Аналізуючи історичні дані про тестування та ідентифікуючі патерни, алгоритми ШІ можуть передбачати потенційні зони збоїв та відповідно пріоритезувати зусилля щодо тестування. Цей проактивний підхід до оптимізації тестування дозволяє командам розробників більш ефективно розподіляти ресурси, зосереджуючись на тих ділянках застосунку, які найімовірніше зіткнуться з проблемами. [9]

Наразі найпопулярнішою сферою автоматизації з використанням ШІ є використання машинного навчання для автоматичного створення тестів для застосунку шляхом "перегляду". Наприклад, достатньо спрямувати деякі з новітніх інструментів ШІ/МН на веб-додаток, щоб вони автоматично почали "переглядати" застосунок. Під час "перегляду" інструмент збирає дані, пов'язані з функціоналом, роблячи знімки екрану, завантажуючи HTML кожної сторінки, вимірюючи час завантаження тощо.

Таким чином, з часом він накопичує набір даних і тренує моделі машинного навчання (МН) на виявлення очікуваних шаблонів поведінки застосунку. Коли інструмент працює, він порівнює свій поточний стан із усіма відомими шаблонами, які вже були навчені. Якщо виникає відхилення (наприклад, сторінка, на якій зазвичай не буває помилок JavaScript, але зараз вони з'явилися), візуальна

різниця або проблема зі швидкістю завантаження нижчою за середню, інструмент позначить це як потенційну проблему.

Інноваційні інструменти тестування з використанням ШІ/МН здатні вивчати будь-які нюанси веб-додатка, розпізнавати взаємодію між частинами моделі об'єктів документів та слідкувати за еволюцією змін. Коли такий інструмент навчається та аналізує динаміку змін у додатку, він автономно вирішує, які ідентифікатори використовувати для розпізнавання елементів, що істотно спрощує підтримку актуальності та стабільності тестів без потреби втручання з боку тестувальника. [7]

Приклад використання ШІ, який розроблений в цій роботі може бути представлений в наступному вигляді:

Ваше завдання, як тестувальника, передбачає застосування структурованого підходу для забезпечення повного покриття тестами та контролю якості програмного забезпечення. Ось деталізація кожного кроку та його зміст:

- розуміння задачі (User story) – проведіть тестування специфікації, тобто вимог до додатку;
- визначення тестових сценаріїв – створіть тест-кейси для перевірки функціоналу додатку, включаючи як позитивні, так і негативні сценарії;
- підготовка тестових даних – наприклад: згенеруйте необхідні дані для майбутнього тестування, такі як електронна пошта/пароль (для тестування логін форми);
- підготовка тестового середовища – створіть новий проект для автоматизованого тестування або підготуйте файли для існуючого проекту;
- створення автоматизованих тестових скриптів – згенеруйте автоматизовані тести для перевірки функціоналу з використанням підготовлених даних та тест-кейсів;
- виконання тестування – запусіть тести;
- перегляд та створення звіту – проаналізуйте результати виконання тестів та напишіть звіт і висновки за результатами тестування.

Кожен із цих кроків є важливим для забезпечення якості програмного забезпечення і допомагає переконатись, що кінцевий продукт відповідає вимогам зацікавлених сторін.

Незважаючи на численні переваги ШІ в автоматизованому тестуванні, існують певні виклики та обмеження, які потребують уваги. Для ефективної роботи алгоритмів ШІ необхідні якісні тренувальні дані та можливість інтерпретації рішень ШІ, що вимагає технік як пояснюваного ШІ, так і безперервного моніторингу та адаптації моделей. [8]

Для ефективного використання ШІ організаціям необхідно знайти баланс між інвестуванням у кваліфіковані QA команди та розумні інструменти, надавати пріоритет тестуванню на ранніх етапах розробки та забезпечувати сучасні практики автоматизації для мінімізації навантаження на обслуговування. ШІ не тільки допомагає в написанні тестів та підтримці їх стабільності, але й адаптується до динамічних змін, зменшуючи час на оновлення та забезпечуючи максимальну ефективність виконання тестів [10].

1.3 Постановка задачі дослідження

Метою цього дослідження є кількісна оцінка ефективності застосування великих мовних моделей у процесах QA тестування сучасних веб-додатків.

Це дослідження спрямоване на вимірювання ефективності тестування за допомогою великих мовних моделей з точки зору релевантності генерації тестових сценаріїв, ефективності виявлення дефектів, повноти охоплення тестами і загального покращення часу циклу тестування.

Застосування великих мовних моделей може суттєво трансформувати традиційні підходи до тестування, пропонуючи автоматизацію складних і неявних аспектів генерації та аналізу тестів, що відкриває нові можливості для підвищення ефективності та якості процесів розробки програмного забезпечення.

Така інтеграція у повсякденні процеси тестування може не лише зменшити часові та трудові витрати на створення та аналіз тестів, але й забезпечити більш

високу якість продукту за рахунок глибшого і більш всебічного охоплення тестування.

Отримані результати дослідження допоможуть визначити ключові переваги та можливі обмеження застосування великих мовних моделей у сфері QA тестування, надаючи цінну інформацію для розробників програмного забезпечення, інженерів з тестування та керівників проектів щодо потенційних напрямів оптимізації та інновацій у своїх робочих процесах.

Також, розуміння впливу великих мовних моделей на процеси тестування дозволить розробити рекомендації щодо найбільш ефективних практик їх впровадження та використання, спрямованих на досягнення оптимального балансу між автоматизацією та людським втручанням. Таким чином, це дослідження сприятиме формуванню міцної основи для прийняття обґрунтованих рішень щодо інтеграції інноваційних технологій на основі штучного інтелекту у сферу забезпечення якості та тестування програмного забезпечення.

2 ОПИС ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз існуючих мовних моделей

Сучасні мовні моделі кардинально змінили сферу обробки мови, забезпечивши значний прогрес у розробці систем розмовних агентів, пошуку та аналізу великих масивів інформації.

Видатними прикладами є GPT від Open AI та Google Bard AI - розуміння їх можливостей, сильних сторін та обмежень допомагає фахівцям правильно інтегрувати такі моделі в свої системи та програми для ефективного їх використання.

У порівняльному аналізі мовних моделей GPT та Google Bard важливо звернути увагу на різні аспекти їхньої функціональності, архітектури та застосування. Кожна з цих моделей має свої унікальні характеристики, які можуть бути використані для оптимізації процесів тестування.

Заснований на архітектурі трансформера, GPT є прикладом ітерації моделі GPT (Generative Pre-trained Transformer), розробленої OpenAI. Модель тренується на диверсифікованих текстових даних, що охоплюють широкий спектр тем та стилів. В той час як Google Bard AI використовує LaMDA (Language Model for Dialogue Applications) для покращення взаємодії з користувачами через розмови. Bard намагається забезпечити більш зв'язну і природню комунікацію, навчаючись на величезній кількості даних, включно з веб-сторінками та іншими джерелами. [11]

GPT чудово підходить для створення тексту, відповідей на запитання та роботи з автоматизацією обслуговування клієнтів. Його використовують для навчання, розваг та програмування, завдяки його здатності генерувати код. Google Bard особливо корисний у сферах, де потрібна висока точність інформації та актуальність даних, таких як академічні дослідження або пошук конкретної інформації, оскільки він має інтеграцію з пошуковою системою Google, що забезпечує доступ до актуальних і точних даних [11].

GPT може бути використаний для автоматизованого створення тестових сценаріїв на основі вимог до функціоналу веб-додатків. Його здатність генерувати зрозумілий та контекстно-адекватний текст дозволяє швидко формулювати запитання та випадки використання, що можуть автоматизувати створення документації для тестування.

Завдяки тренуванню на різноманітних даних, GPT може імітувати реальні користувацькі запити та діалоги, що дозволяє тестувальникам виявляти помилки, які можуть виникнути під час реального використання додатків.

Google Bard, в свою чергу, користуючись можливостями пошуку Google, може допомагати у виявленні більш складних проблем у програмному забезпеченні, забезпечуючи доступ до актуальних даних і трендів в інтернеті, що може вплинути на тестування. Завдяки потужностям обробки природної мови, Google Bard може аналізувати великі обсяги відгуків користувачів для виявлення потенційних недоліків продукту, що може вплинути не тільки на план тестування, а й змінити вектор розробки функціоналу продукту.

Обидві моделі пропонують значні переваги, але їх застосування залежить від конкретних потреб проекту. Для розробки нашої моделі GPT може бути більш ефективним особливо в ситуаціях, де потрібно швидко генерувати багато тестових сценаріїв з мінімальними вхідними даними.

Вибір між GPT і Google Bard для тестування веб-додатків залежить від специфіки завдань, які потребують рішення. GPT ідеально підходить для автоматизації і створення різноманітних тестових сценаріїв, тоді як Google Bard має переваги в глибокому аналізі та використанні оновленої інформації з інтернету для виявлення та аналізу потенційних проблем. Обидві моделі вносять значний вклад у підвищення ефективності тестування веб-додатків, забезпечуючи важливі інструменти для сучасних розробників і QA фахівців [12].

2.2 Математичний опис проблемної області дослідження

Для визначення ефективності застосування великих мовних моделей у процесах QA тестування сучасних веб-додатків у нашому дослідженні,

звернемося до принципів багатокритеріального аналізу рішень. Цей підхід дозволяє інтегрувати різноманітні критерії в єдину зведену оцінку шляхом призначення вагових коефіцієнтів кожному критерію на основі його відносної важливості стосовно загальної мети [13].

Розрахунок показника ефективності базується на моделі зваженої суми (Weighted Sum Model - WSM), широко визнаному методі в галузі багатокритеріального аналізу рішень (Multi-Criteria Decision Analysis - MCDA).

Такий метод вибрано через його простоту, обчислювальну ефективність та інтуїтивну інтерпретацію його результатів, що робить його придатним для оцінки багатогранних аспектів ефективності тестування за допомогою великих мовних моделей.

Застосування методу в нашому дослідженні ґрунтується на принципах, викладених у посібнику «Багатокритеріальний аналіз рішень для прийняття стратегічних рішень» який описує всебічне дослідження методів MCDA та їх практичної реалізації [13].

Отже, для втілення принципів багатокритеріального аналізу необхідно сформулювати критерії для оцінювання ефективності застосування великих мовних моделей у процесах QA тестування, а також задати вагу цих критеріїв відповідно до формули розрахунку рівня ефективності (E) 2.1:

$$E = \sum_{i=1}^n (w_i \times C_i) \quad (2.1)$$

де w_i - вагові коефіцієнти, що відображають відносний внесок окремих критеріїв, C_i - нормалізоване значення і-го критерію.

Вагові коефіцієнти попередньо визначаються на основі оцінки експертів або інших відповідних критеріїв, а сума всіх вагових коефіцієнтів має дорівнювати 1. Критерії слід нормалізувати, щоб забезпечити узгодженість шкали за всіма критеріями [14].

Для нашого дослідження, вибір критеріїв для оцінки ефективності великих мовних моделей у QA тестуванні є критичним кроком, який безпосередньо впливає на достовірність і релевантність результатів дослідження. Вони повинні охоплювати ключові аспекти робочого процесу тестування, від швидкості та корисності згенерованих відповідей до взаємодії між QA інженером та системою, забезпечуючи комплексну оцінку практичності застосування моделі в сценаріях тестування в реальному середовищі.

У сфері тестування за допомогою штучного інтелекту кілька показників критично впливають на ефективність і результативність процесів тестування. До них належать час вирішення завдання (Time to Resolve the Task - TRT), час обробки результатів (Time to Process Results - TPR), релевантність отриманих результатів (Relevance of the Generated Result- RGR) і кількість уточнень (Number of Clarifications - NC).

Кожен з них відіграє важливу роль в оптимізації робочих процесів тестування та забезпеченні того, щоб інтеграція інструментів штучного інтелекту, таких як великі мовні моделі, додала відчутну цінність спробам тестування програмного забезпечення. Розглянемо ці критерії більш детально.

Час, необхідний для вирішення завдання (TRT). Цей критерій вимірює час, що минув від моменту запуску тестового завдання до його успішного завершення. Ця метрика є ключовою для розуміння ефективності процесів тестування що використовують допомогу штучного інтелекту.

Низьке значення TRT вказує на більш ефективний процес тестування, що призводить до пришвидшення циклів розробки та зменшення часу виходу програмних продуктів на ринок.

Прискорення процесів тестування за допомогою інтеграції штучного інтелекту надає ключову перевагу при розробці програмного забезпечення [15]. Для оцінки часу, необхідного для вирішення завдання за допомогою великої мовної моделі (LLM) пропонується використовувати наступну шкалу інтервалів (див. табл. 2.1).

Таблиця 2.1 - Шкала інтервалів для критерію TRT (таблиця виконана самостійно)

Оцінка	Обґрунтування
4	0 - 60 Секунди (Дуже добре): надзвичайно швидке вирішення завдань, значно перевищує стандартні людські можливості.
3	1-2 Хвилини (Добре): швидке вирішення завдань, що відображає ефективність LLM у порівнянні з більш тривалими людськими процесами, але існує потенціал для оптимізації.
2	2-5 Хвилин (Задовільно): вирішення завдань у цьому часовому діапазоні може вказувати на певні обмеження LLM або складність завдань, потребує додаткового аналізу.
1	5-10 Хвилин (Погано): тривале вирішення завдань може свідчити про недоліки у функціональності або інтеграції LLM.

Цей критерій має вагу 0.2, що відображає його важливість, але не найвищу серед інших критеріїв. Важливо вимірювати швидкість реакції системи, оскільки це критично впливає на ефективність та швидкість процесу тестування в реальному часі.

Час, необхідний для обробки результатів, що генеруються великою мовною моделлю (TPR). Критерій, що визначає час, необхідний для аналізу та інтерпретації результатів, отриманих після генерації за допомогою штучного інтелекту. Ефективна обробка результатів має важливе значення для підтримки темпу циклів тестування та забезпечення швидкого коригування даних. Таким чином висвітлюється роль оптимізації аналізу результатів генеративного штучного інтелекту, що дає змогу самим покращуючи загальну ефективність тестування [16]. Для оцінки результатів за даним критерієм можна використати подібну шкалу інтервалів із дещо зменшеним діапазоном, адже обробка результатів відбувається вручну, тобто миттєва обробка стає неможливою (див. табл. 2.2)

Таблиця 2.2 - Шкала інтервалів для критерію TPR (таблиця виконана самостійно)

Оцінка	Обґрунтування
3	Хвилини (Дуже добре): швидка ручна обробка результатів, що дозволяє ефективно вносити необхідні корективи без значної затримки, вказуючи на високу ефективність процесу.
2	Десятки хвилин (Добре): результати обробляються в помірні терміни, дозволяючи підтримувати робочий процес без критичних затримок, але з потенціалом для покращення швидкості обробки.
1	Години (Погано): занадто довга обробка результатів вручну вказує малу ефективність результатів, що обробляються. Це може вплинути на швидкість робочого процесу та внести небажаних додаткових викликів для проекту.

Цей критерій отримує трохи більшу вагу - 0.25, оскільки час, який QA інженер витрачає на аналіз результатів, критично впливає на загальну продуктивність процесу тестування. Ефективність процесу не тільки зменшує витрати часу, але і може прямо впливати на вчасне виявлення помилок.

Релевантність отриманих результатів (RGR). Визначення релевантності оцінює, наскільки результати тестування за допомогою штучного інтелекту відповідають фактичним цілям і вимогам тестування у контексті конкретної задачі. Висока релевантність згенерованих результатів гарантує, що зусилля з тестування безпосередньо сприяють виявленню та вирішенню реальних проблем програмного забезпечення, таким чином максимізуючи корисність ресурсів тестування. Це підкреслює важливість застосування інструментів штучного інтелекту з точки зору ефективного доповнення процесу тестування [17].

Щоб створити кількісну шкалу для вимірювання релевантності згенерованих результатів (RGR) на основі підрахунку релевантних і нерелевантних елементів, ми можемо застосувати простий і прямий підхід - підрахування пропорції релевантних та нерелевантних елементів:

$$RGR = \frac{RR}{TR} \quad (2.2)$$

де RR - кількість релевантних результатів (Relevant Results),
TR - загальна кількість результатів (Total Results).

Таким чином утворюється шкала інтервалів, що перетворює підрахунки в пропорцію або відсоток, що відображає рівень релевантності результатів (див. табл. 2.3):.

Таблиця 2.3 - Шкала інтервалів для критерію RGR (таблиця виконана самостійно)

Оцінка	Обґрунтування
0.0 - 0.2	Більшість пунктів не є релевантними, що вказує на значні недоліки в здатності моделі розуміти контекст запиту. Можливо, потрібне доопрацювання моделі або уточнення запитів.
0.21 - 0.4	Чимало пунктів не є релевантними, свідчить про певну розбіжність між потребами тестування та вихідними даними моделі. Варто переглянути критерії відбору результатів.
0.41 - 0.6	Змішані релевантні та нерелевантні пункти. Хоча модель здебільшого відповідає запитам, існують прогалини в точності та деталізації. Рекомендується додаткове налаштування параметрів моделі.
0.61 - 0.8	Більшість пунктів є релевантними, що демонструє високу здатність моделі вирішувати поставлені завдання. Можуть бути незначні пропуски, але в цілому результати задовільні.
0.81 - 1.0	Майже всі пункти є релевантними, свідчить про відмінне розуміння моделлю контексту та вимог. Це ідеальний результат, який вказує на високу ефективність використання LLM в процесах QA тестування.

Для цього критерію встановлена найбільша вага - 0.3, оскільки саме релевантність результатів безпосередньо визначає корисність моделі GPT в QA тестуванні. Якщо результати не є релевантними до запитів тестування, то, незалежно від швидкості обробки чи вирішення завдань, вони не будуть корисними для інженера.

Кількість уточнень (NC). Критерій, що відстежує необхідність додаткової взаємодії, що потрібна для уточнення або вдосконалення отриманих результатів. Мінімізація пояснень є ключем до досягнення більш автономного та спрощеного процесу тестування, так як передові моделі штучного інтелекту можуть зменшити потребу в людському втручанні, підвищуючи ефективність тестування [16]. Необхідність додаткового втручання для уточнення вхідних параметрів може бути ознакою неефективності моделі, а також вказувати на необхідність формулювання більш складних умов для успішної генерації. Тому, відсутність або дуже мала кількість уточнень буде мати позитивний вплив на ефективність (див. табл. 2.4)

Таблиця 2.4 - Шкала інтервалів для критерію *NC* (таблиця виконана самостійно)

Оцінка	Обґрунтування
3	Дуже добре (0 уточнень): свідчить про високу ясність вхідних даних та ефективність моделі, здатної генерувати відповіді без необхідності додаткових уточнень.
2	Задовільно (1-3 уточнень): незначна потреба в уточненнях показує, що модель загалом ефективна, і необхідно переглянути вхідні дані.
1	Погано (більше 3 уточнень): потреба в регулярних уточненнях вказує на проблеми з ясністю вхідних даних та недостатню ефективність моделі, що вимагає значних зусиль для досягнення бажаних результатів.

Цей критерій також має значну вагу - 0.25, адже менша кількість уточнень означає, що штучний інтелект добре розуміє задачі без необхідності додаткових

вхідних даних або виправлень, що свідчить про високу якість взаємодії між користувачем та системою.

Таким чином метрики TRT, TPR, RGR і NC складають основу оцінки процесу тестування за допомогою штучного інтелекту. Вони не лише пропонують уявлення про ефективність і ефективність процесу тестування, але й керують інтеграцією інструментів ШІ таким чином, щоб покращити, а не ускладнити тестування програмного забезпечення. Зосередившись на цих критеріях, організації можуть використовувати весь потенціал ШІ для оптимізації робочих процесів розробки та обслуговування програмного забезпечення.

Показники часу вирішення завдання (TRT), часу обробки результатів (TPR), релевантності згенерованих результатів (RGR) і кількості роз'яснень (NC) є основними при оцінці впливу штучного інтелекту на процес тестування програмного забезпечення. Включення штучного інтелекту в процеси тестування значно полегшує шлях до більш автономного тестування програмного забезпечення. Еволюція від ручного та автоматизованого тестування до автономного тестування на основі штучного інтелекту підвищує ефективність, масштабованість і охоплення тестуванням, тим самим спрощуючи процес контролю якості та покращуючи надійність програмного забезпечення [18].

2.3 Проектування засобів для забезпечення експерименту

Для реалізації експерименту ми розробили код веб-сайту з базовими функціями, який було запущено за допомогою простої конфігурації `docker-compose` на хмарній платформі AWS Cloud та EC2 інстансі [19].

Як можна побачити на рис 2.1, для проведення експерименту ми обрали сторінку, що містить функціонал входу, де користувачу необхідно ввести такі дані, як електронна адреса та пароль. Код цієї веб-сторінки разом з її структурою продемонстровано на рис 2.2.

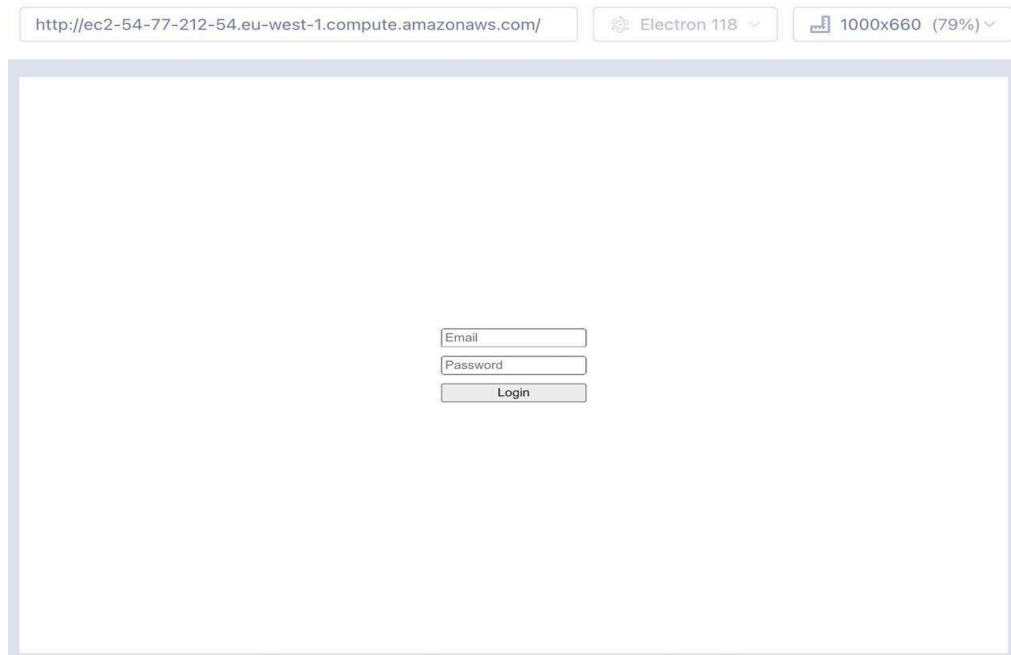


Рисунок 2.1 - Графічний дизайн веб сайту розробленого для проведення експерименту (створено самостійно)

 A screenshot of a code editor interface. On the left, the 'EXPLORER' sidebar shows a project structure for 'WEBSITE-EXAMPLE'. It includes a 'backend' folder with 'package-lock.json', 'package.json', and 'server.js'. A 'frontend' folder contains a 'public' subfolder with 'favicon.ico', 'index.html', 'logo192.png', 'logo512.png', 'manifest.json', and 'robots.txt'. The 'src' folder contains 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'reportWebVitals.js', and 'setupTests.js'. Other files include '.gitignore', 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', and 'docker-compose.yml'. The main editor area shows the code for 'server.js'. The code includes imports for 'express', 'body-parser', and 'cors', sets up an Express app, and defines a '/login' endpoint with a mock validation function. The server is configured to listen on port 5050.

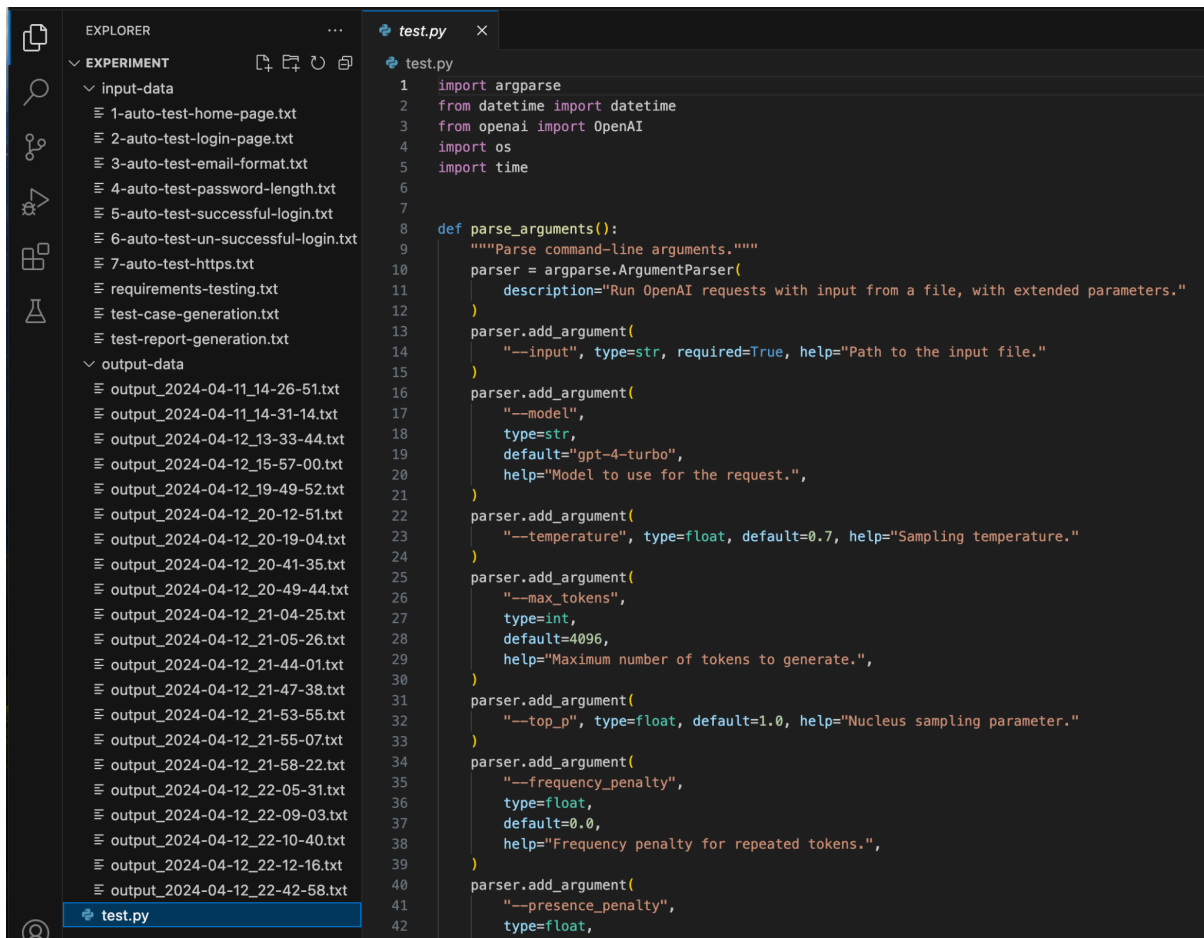

```

backend > JS server.js > app.post('/login') callback
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const cors = require('cors');
4
5  const app = express();
6  const port = process.env.PORT || 5050;
7
8  app.use(cors());
9  app.use(bodyParser.json());
10
11  app.post('/login', (req, res) => {
12    const { email, password } = req.body;
13
14    // Mock user validation
15    if (email === 'user@example.com' && password === 'example') {
16      res.json({ message: 'Login successful' });
17    } else {
18      res.status(401).json({ message: 'Invalid credentials' });
19    }
20  });
21
22  app.listen(port, () => {
23    console.log(`Server is running on port ${port}`);
24  });
25
  
```

Рисунок 2.2 - Код веб-сторінки разом з її структурою (створено самостійно)

Функціонал сайту складається з бекенду та фронтенду. Фронтенд реалізований за допомогою React на JavaScript, а бекенд — за допомогою Node.js та Express. Код веб-сайту представлено у додатку Д.

Для підтримки нашого експерименту ми розробили програму-асистент, використовуючи Python та бібліотеку OpenAI для взаємодії з великою мовною моделлю GPT. Повний код Python програми представлено у додатку Е. Реалізація програми представлена на рис 2.3:



```

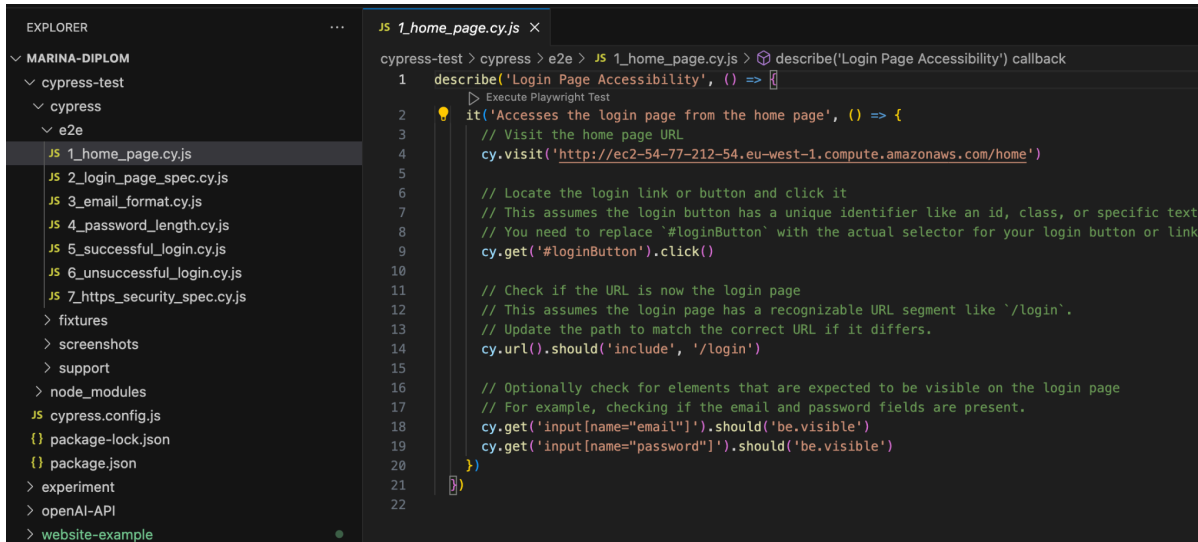
1 import argparse
2 from datetime import datetime
3 from openai import OpenAI
4 import os
5 import time
6
7
8 def parse_arguments():
9     """Parse command-line arguments."""
10    parser = argparse.ArgumentParser(
11        description="Run OpenAI requests with input from a file, with extended parameters."
12    )
13    parser.add_argument(
14        "--input", type=str, required=True, help="Path to the input file."
15    )
16    parser.add_argument(
17        "--model",
18        type=str,
19        default="gpt-4-turbo",
20        help="Model to use for the request.",
21    )
22    parser.add_argument(
23        "--temperature", type=float, default=0.7, help="Sampling temperature."
24    )
25    parser.add_argument(
26        "--max_tokens",
27        type=int,
28        default=4096,
29        help="Maximum number of tokens to generate.",
30    )
31    parser.add_argument(
32        "--top_p", type=float, default=1.0, help="Nucleus sampling parameter."
33    )
34    parser.add_argument(
35        "--frequency_penalty",
36        type=float,
37        default=0.0,
38        help="Frequency penalty for repeated tokens.",
39    )
40    parser.add_argument(
41        "--presence_penalty",
42        type=float,

```

Рисунок 2.3 - Структура файлів для взаємодії з API OpenAI (створено самостійно)

Для ефективної взаємодії з мовною моделлю GPT, було створено папку input-data, яка містить промпти для подальшого використання в автотестах. Під час запуску програми в папці output-data, помічник автоматично генерує відповіді на основі введених промптів. Це дозволяє автоматизувати процес тестування та отримувати релевантні відповіді від штучного інтелекту, що значно спрощує аналіз результатів експерименту.

Для організації та проведення автоматизованих тестів з використанням Cypress фреймворку, заздалегідь було створено проект із спеціалізованою пустою папкою e2e (end-to-end testing). Ця папка призначена для зберігання автоматизованих тестів, що генерує GPT (див рис 2.4)



```

EXPLORER
├── MARINA-DIPLOM
│   ├── cypress-test
│   │   ├── cypress
│   │   │   └── e2e
│   │   │       ├── JS_1_home_page.cy.js
│   │   │       ├── JS_2_login_page_spec.cy.js
│   │   │       ├── JS_3_email_format.cy.js
│   │   │       ├── JS_4_password_length.cy.js
│   │   │       ├── JS_5_successful_login.cy.js
│   │   │       ├── JS_6_unsuccessful_login.cy.js
│   │   │       └── JS_7_https_security_spec.cy.js
│   │   ├── fixtures
│   │   ├── screenshots
│   │   ├── support
│   │   ├── node_modules
│   │   ├── cypress.config.js
│   │   ├── package-lock.json
│   │   └── package.json
│   ├── experiment
│   ├── openAI-API
│   └── website-example
└── JS_1_home_page.cy.js
    1 describe('Login Page Accessibility', () => {
    2     // Execute Playwright Test
    3     it('Accesses the login page from the home page', () => {
    4       // Visit the home page URL
    5       cy.visit('http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com/home')
    6
    7       // Locate the login link or button and click it
    8       // This assumes the login button has a unique identifier like an id, class, or specific text.
    9       // You need to replace '#loginButton' with the actual selector for your login button or link.
   10      cy.get('#loginButton').click()
   11
   12      // Check if the URL is now the login page
   13      // This assumes the login page has a recognizable URL segment like '/login'.
   14      // Update the path to match the correct URL if it differs.
   15      cy.url().should('include', '/login')
   16
   17      // Optionally check for elements that are expected to be visible on the login page
   18      // For example, checking if the email and password fields are present.
   19      cy.get('input[name="email"]').should('be.visible')
   20      cy.get('input[name="password"]').should('be.visible')
   21    })
   22  })
  
```

Рисунок 2.4 - Автоматизовані тести згенеровані за допомогою GPT (створено самостійно)

Отже, в рамках розділу проектування засобів для забезпечення експерименту було здійснено комплексне проектування засобів, необхідних для забезпечення планування та виконання експерименту з автоматизованого тестування веб-додатків за допомогою генеративного штучного інтелекту. Основні зусилля були спрямовані на створення адекватної інфраструктури, що включає розробку програмного забезпечення.

2.4 Планування експериментального дослідження

Планування експериментального дослідження передбачає детальний опис процедур та методик, які будуть використані для оцінки ефективності застосування великих мовних моделей, таких як GPT від OpenAI, у процесах QA тестування сучасних веб-додатків. Основною метою є визначення того, наскільки добре штучний інтелект може асистувати у різних аспектах тестування, від

розуміння вимог до генерації тестових сценаріїв і аналізу результатів тестування [7].

Після планування експерименту сформовано наступні рішення щодо проведення експерименту:

Формування Промптів за Темплейтом. Для кожного аспекту тестування, з яким нам буде допомагати штучний інтелект, будуть сформовані спеціальні промпти - текст природною мовою, що описує завдання, яке повинен виконати штучний інтелект. Ці промпти будуть структуровані за заздалегідь визначеним шаблоном, який включатиме вхідні дані та конкретне завдання для вирішення.

Цей шаблон включатиме:

- вхідні дані (Input Data): конкретні дані, пов'язані з завданням, наприклад, деталі сценарію тестування;
- завдання (Task): чітке формулювання завдання для генерації, включаючи очікувану дію чи результат;
- додаткова Інформація (Additional info):
- контекст: будь-яка додаткова інформація про контекст завдання, яка може вплинути на відповіді моделі;
- вимоги до Відповіді: специфікації щодо формату, деталізації чи структури відповіді;
- обмеження: будь-які особливі обмеження або вимоги, які потрібно враховувати;
- пріоритети: інформація про пріоритетні аспекти завдання або важливість різних частин запиту.

Запуск програми-помічника на Python. За допомогою спеціально розробленої програми на мові Python з використанням OpenAI API, що становить основу експерименту, буде запущено обробку промптів для кожного етапу. Ця програма координуватиме взаємодію з великою мовною моделлю, буде приймати на вхід сформовані промпти у текстових файлах та збиратиме отримані відповіді у вигляді текстових файлів.

Аналіз Отриманих Результатів. Для кожного завдання буде проведено детальний аналіз результатів, які генерує штучний інтелект. Це дозволить оцінити первинну ефективність та точність відповідей створених мовною моделлю.

Уточнення промптів. Уточнення запиту є критерієм, що впливає на кінцеву оцінку ефективності. На основі аналізу результатів та за необхідності, промпти можуть бути модифіковані або уточнені для підвищення якості та релевантності відповідей.

Перегляд та підтвердження кінцевих результатів. Після ітерацій покращень промптів, кінцевий результат буде переглянуто на відповідність запиту та поставлених завдань. Це необхідно для переходу на наступний етап - визначення оцінок за критеріями.

Оцінка за критеріями. Кожен аспект тестування, в якому було використано AI, оцінюється за визначеними критеріями. Це дозволить кількісно визначити ефективність використання великих мовних моделей у процесах тестування.

Такий план експериментального дослідження забезпечує структурований підхід до оцінки вкладу штучного інтелекту в різні етапи процесу QA тестування, від розробки тестових сценаріїв до аналізу результатів тестування, дозволяючи визначити ключові переваги та обмеження застосування великих мовних моделей.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ У ТЕСТУВАННІ ВЕБ-ДОДАТКІВ

3.1 Дослідження тестування вимог (Requirements Testing)

Для тестування вимог до програмного забезпечення (Requirements Testing) необхідно провести аналіз заздалегідь сформованого опису завдання (user story) та отримати чіткі та стислі очікування щодо функціональних можливостей, що розробляються, гарантуючи, що всі члени команди мають спільне розуміння того, що потрібно досягти [20].

Застосувавши шаблон запиту, сформуємо вхідні дані для втілення експерименту:

- вхідні дані (Input data): User story: As a user I want to be able to login at the website using my email and password;
- завдання (Task): Perform Requirements Testing based on the user story for the QA testing;
- додаткова Інформація (Additional info): the result should contain improved requirements; the requirements should be only related to the functionality described in the story.

Створимо файл для запиту “input-data/requirements-testing.txt” та перейдемо до запуску програми (рис 3.1):

```
> py test.py --input input-data/requirements-testing.txt
Request completed in 32.37 seconds.
Output written to output-data/output_2024-04-11_14-26-51.txt
### Original User Story
**As a user, I want to be able to log in to our website using an email and password.**

### Improved Requirements for QA Testing

#### Functional Requirements
1. **User Authentication Interface**
   - The website should have a dedicated login page accessible from the home page.
   - The login page must include fields for the user to enter their email address and password.
```

Рисунок 3.1 – Результат запуску програми для експерименту requirements testing
(створено самостійно)

Переглянемо отриманий результат що сформовано і збережено до файлу
“output-data/output_2024-04-11_14-26-51.txt”:

```
### Original User Story
**As a user, I want to be able to log in to our website using an
email and password.**

### Improved Requirements for QA Testing

#### Functional Requirements
1. **User Authentication Interface**
  - The website should have a dedicated login page accessible from
the home page.
  - The login page must include fields for the user to enter their
email address and password.

2. **Input Validation**
  - The email field must validate input to ensure it conforms to
standard email format (e.g., user@example.com).
  - The password field should accept inputs with a minimum of 8
characters and a maximum of 64 characters to ensure security.
  - An error message should be displayed if the input format for
either the email or password is incorrect.

3. **Authentication Process**
  - Upon entering the email and password, the system must verify
the credentials against the stored user database.
  - Successful login should redirect the user to their dashboard or
a relevant start page.
  - Unsuccessful login attempts should result in an error message
explaining that either the email or password was incorrect.
  - Provide a mechanism to prevent brute force attacks, such as
limiting the number of failed login attempts.

4. **Security Requirements**
  - All password data stored in the database must be encrypted
using a secure hashing algorithm.
  - Login sessions should be managed securely, with session
expiration and renewal handled appropriately.
  - Use HTTPS for all pages that require input or display of user
credentials to protect data in transit.

5. **Accessibility Requirements**
  - Ensure that the login form is accessible according to WCAG 2.1
guidelines, including keyboard navigability and screen reader
support.
  - Provide visible focus indicators for interactive elements like
input fields and submit buttons.

6. **User Feedback**
```

- Display a loading indicator or message while the login request is being processed.
- After a successful login, display a welcome message with the user's name or username.
- Provide a "Forgot password?" link that directs users to a password recovery page.
- Include a "Remember me" checkbox to allow users the option to remain logged in on the device being used.

Testing Scenarios

1. ****Input Format Validation****
 - Test with correct and incorrect email formats.
 - Test password field for minimum and maximum length compliance.

2. ****Authentication Verification****
 - Attempt to log in with correct and incorrect email/password combinations.
 - Test system response to repeated failed login attempts.

3. ****Security Checks****
 - Verify that passwords are not stored in plaintext.
 - Check for HTTPS on login pages.
 - Test session management, especially session expiration.

4. ****Accessibility and Usability****
 - Ensure all form controls are reachable and usable via keyboard.
 - Check that error messages and instructions are clear and helpful.

By enhancing the requirements as specified and conducting detailed testing based on these scenarios, the quality and security of the login functionality can be ensured, aligning with the user's needs and expectations.

Переглянувши результат генерації можемо зробити висновок, що більшість створених елементів необхідних для виконання завдання є релевантними та будуть мати суттєве значення для покращення вимог.

Також, у результаті генерації, мовна модель створила сценарії тестування, що є корисним результатом але не релевантним для конкретного експерименту. Виконання програми зайняло 32.37 секунди, а час на обробку результатів не буде займати більше ніж 5 хвилин.

Результат роботи програми є достатнім і не потребує уточнень. Згідно з заданими критеріями отримаємо наступні оцінки (див табл. 3.1)

Таблиця 3.1 - Результати генерації експерименту (таблиця виконана самостійно)

Оцінка	Критерій
4	Час вирішення завдання (TRT)
3	Час обробки результатів (TPR)
0,66	Релевантність отриманих результатів (RGR)
3	Кількість уточнень (NC)

На підставі результатів експерименту можна зробити висновок, що процес генерації вдосконалених вимог до задачі (user story) був виконаний успішно і продемонстрував задовільні показники за визначеними критеріями.

3.2 Дослідження генерації тестових сценаріїв (Test Case Generation)

Наступним етапом дослідження, необхідно створити тестові сценарії за допомогою систематичного підходу, який забезпечує повне охоплення всіх функціональних можливостей і взаємодії з користувачем [21].

Використовуючи специфікації бізнес-вимог, специфікації вимог до програмного забезпечення і специфікації функціональних вимог тестові сценарії мають охопити очікувані функції та дії користувачів веб-додатку [22].

На основі наявних вимог до завдання (user story) які ми отримали та покращили у результаті попереднього експерименту, створимо новий файл для запиту “input-data/test-case-generation.txt” та перейдемо до запуску програми (рис 3.2):

```

> py test.py --input input-data/test-case-generation.txt
Request completed in 44.70 seconds.
Output written to output-data/output_2024-04-12_13-33-44.txt
### Test Case Scenarios

Below are detailed test case scenarios that map to the functional requirements provided. These t
omated tests.

#### Test Case 1: Access Login Page from Home Page
**Objective**: Verify that the login page is accessible from the home page.
**Steps**:
1. Open the home page of the website.
2. Locate and click on the login link or button.
**Expected Result**:
- The login page should be displayed.

#### Test Case 2: Login Page Fields

```

Рисунок 3.2 - Результат запуску програми для експерименту “Test Case Generation” (створено самостійно)

В результаті генерації, тестові сценарії збережено до файлу “output-data/output_2024-04-12_13-33-44.txt”:

```

### Test Case Scenarios

Below are detailed test case scenarios that map to the functional
requirements provided. These test cases can be used by QA engineers
for manual testing and can be adapted for automated tests.

#### Test Case 1: Access Login Page from Home Page
**Objective**: Verify that the login page is accessible from the
home page.

**Steps**:
1. Open the home page of the website.
2. Locate and click on the login link or button.
**Expected Result**:
- The login page should be displayed.

#### Test Case 2: Login Page Fields
**Objective**: Ensure the login page includes the correct fields.
**Steps**:
1. Access the login page.
**Expected Result**:
- The page should contain an email address input field and a
password input field.

#### Test Case 3: Email Format Validation
**Objective**: Validate that the email field checks for standard
email formats.
**Steps**:
1. Navigate to the login page.

```

2. Enter an invalid email format in the email field (e.g., "user#example.com").

3. Attempt to submit the form.

****Expected Result**:**

- An error message should be displayed indicating an invalid email format.

Test Case 4: Password Length Validation

****Objective**:** Check that the password field enforces length restrictions.

****Steps**:**

1. Navigate to the login page.
2. Enter a password that is less than 8 characters long.
3. Attempt to submit the form.

****Expected Result**:**

- An error message should be displayed indicating the password is too short.

Test Case 5: Successful Login

****Objective**:** Test the successful login flow.

****Steps**:**

1. Navigate to the login page.
2. Enter a valid email and correct password.
3. Submit the form.

****Expected Result**:**

- The user is redirected to their dashboard or relevant home page.
- A welcome message with the user's name or username is displayed.

Test Case 6: Unsuccessful Login

****Objective**:** Verify the system behavior on incorrect credentials.

****Steps**:**

1. Navigate to the login page.
2. Enter a valid email with an incorrect password.
3. Submit the form.

****Expected Result**:**

- An error message should be displayed stating that the email or password is incorrect.

Test Case 7: HTTPS Security

****Objective**:** Confirm that the login page uses HTTPS.

****Steps**:**

1. Access the login page URL.

****Expected Result**:**

- The URL should begin with "https://" indicating a secure connection.

Test Case 8: Loading Indicator During Login

****Objective**:** Ensure that a loading indicator or message is displayed during the processing of a login request.

****Steps**:**

1. Navigate to the login page.
2. Enter valid login credentials.
3. Submit the form.

****Expected Result**:**

- A loading indicator or message should appear while the login request is being processed.

Test Case 9: Forgot Password Link

****Objective****: Verify that the "Forgot password?" link directs to the correct page.

****Steps****:

1. Navigate to the login page.
2. Click on the "Forgot password?" link.

****Expected Result****:

- The user should be redirected to the password recovery page.

Conclusion

These test cases cover basic functional aspects of the user authentication interface, input validation, authentication process, security requirements, and user feedback mechanisms. QA engineers should use these scenarios to ensure that the application meets the specified requirements before it goes live.

Переглянувши результат запиту можемо констатувати, що дані створені штучним інтелектом ефективно перетворюють функціональні вимоги на детальні сценарії тестування, надаючи чіткі та вичерпні інструкції що можуть бути використані як для ручного, так і для автоматизованого тестування. Кожен тестовий приклад добре структурований, має чітко сформульовані цілі, точні кроки, що необхідно виконати для досягнення очікуваних результатів які відповідають вимогам завдання.

Абсолютна більшість сценаріїв також мають підґрунтя для практичного застосування в реальному тестовому середовищі, що робить їх цінним ресурсом для команди розробки та безпосередньо тестувальників. Тестові сценарії створені для всіх критичних компонентів системи у відповідності до вимог, що підкреслює релевантність результату та відсутність необхідності уточнення вхідних даних.

Обробка тестових сценаріїв згенерованих штучним інтелектом, займає не більше ніж 20 хвилин, враховуючи, що інженер який працює з цими даними розуміє контекст вхідних даних та особливості тестування конкретної функціональності.

Для даного експерименту, встановлено такі оцінки за критеріями (див табл. 3.2)

Таблиця 3.2 - Результати генерації експерименту (таблиця виконана самостійно)

Оцінка	Критерій
4	Час вирішення завдання (TRT)
2	Час обробки результатів (TPR)
0.9	Релевантність отриманих результатів (RGR)
3	Кількість уточнень (NC)

На основі результатів робимо висновок, що процес генерації тестових сценаріїв був виконаний успішно і продемонстрував задовільні показники за визначеними критеріями.

3.3 Дослідження генерації авто тестів

Для створення автоматизованих тестів необхідно провести перетворення вимоги та тестових сценаріїв в автоматизовані тестові скрипти. Це включає написання коду тестів з використанням мови програмування, фреймворку та інших технічних засобів якими може користуватися команда тестувальників.

У нашому випадку, для втілення експерименту буде використовуватись Cypress фреймворк, проект якого було створено заздалегідь. У ході виконання експерименту, сформовані тестові сценарії описуються як завдання для генерації авто-тестів. А після генерації, створений код буде запускатися за допомогою Cypress та перевірятися на валідність.

Створимо файл “input-data/auto-test-home-page.txt” для втілення першого тестового сценарію з наступним контентом:

```
### Test Case Scenarios

Вхідні дані (Input data):
  Test Case 1: Access Login Page from Home Page;
Завдання (Task):
  Create an auto test for the functionality described in the test
  case;
  Generate cy.js file that will be used in cypress/e2e/ to run the
  tests;
Додаткова Інформація (Additional info):
```

Our website accessible at:
 http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com;
 Home page accessible at
 http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com/home;
 We already have the cypress project setup.

Перейдемо до запуску програми та генерації авто тесту (рис. 3.3):

```
> py test.py --input input-data/auto-test-home-page.txt
Request completed in 62.88 seconds.
Output written to output-data/output_2024-04-12_15-57-00.txt
To create an automated test using Cypress for the test case provided, you'll need to write a test script that verifies that the login page is displayed correctly. Below is a step-by-step guide to creating the `.cy.js` file.

### Step 1: Setup a New Cypress Test File

Create a new file in your cypress project under the `cypress/e2e/` directory. You might name the file as `login_page_access.cy.js`.

### Step 2: Write Cypress Test

In the `login_page_access.cy.js` file, you'll write a Cypress test that follows the test case steps. Here's a basic script to get you started:

```javascript
// login_page_access.cy.js

describe('Test Case 1: Access Login Page from Home Page', () => {
 it('Verifies that the login page is accessible from the home page', () => {
 // Step 1: Open the home page
 cy.visit('http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com/home');

 // Step 2: Locate and click on the login link or button
 // Assuming the login link has an id or unique selector. Replace '.login-link' with the correct selector.
 cy.get('.login-link').click();
 });
});
```

Рисунок 3.3 - Результат генерації авто тесту сценарію #1 (створено самостійно)

В результаті генерації, дані для автоматичного тесту збережено до файлу “output-data/output\_2024-04-12\_13-33-44.txt”. Розглянемо згенерований текст та особливу увагу звернемо на код, що був створений:

To create an automated test using Cypress for the test case provided, you'll need to write a test script that navigates from the home page to the login page, and verifies that the login page is displayed correctly. Below is a step-by-step guide to creating the `.cy.js` file for this test.

### Step 1: Setup a New Cypress Test File

Create a new file in your cypress project under the `cypress/e2e/` directory. You might name the file as `login\_page\_access.cy.js`.

### Step 2: Write Cypress Test

In the `login\_page\_access.cy.js` file, you'll write a Cypress test that follows the test case steps. Here's a basic script to get you started:

```

```javascript
describe('Login Page Accessibility', () => {
  it('Accesses the login page from the home page', () => {
    // Visit the home page URL

    cy.visit('http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com/home')

    // Locate the login link or button and click it
    // This assumes the login button has a unique identifier like an
    id, class, or specific text.
    // You need to replace `#loginButton` with the actual selector
    for your login button or link.
    cy.get('#loginButton').click()

    // Check if the URL is now the login page
    // This assumes the login page has a recognizable URL segment
    like `/login`.
    // Update the path to match the correct URL if it differs.
    cy.url().should('include', '/login')

    // Optionally check for elements that are expected to be visible
    on the login page
    // For example, checking if the email and password fields are
    present.
    cy.get('input[name="email"]').should('be.visible')
    cy.get('input[name="password"]').should('be.visible')
  })
})
```

```

### ### Step 3: Run the Test

To run your test, you can use the Cypress Test Runner or run it via the command line. Here's how you can run it using the command line:

```

```bash
npx cypress open
```

```

Then, find and select the `login\_page\_access.cy.js` file from the list of test files in the Cypress UI that appears.

### ### Additional Notes

1. **\*\*Selectors\*\***: Ensure the selectors used (like `.login-link` or `form#loginForm`) match those in your actual web application. You might need to inspect the webpage to get the correct selectors.
2. **\*\*Login Page Identification\*\***: The test assumes that the login page can be uniquely identified by the URL. If this is not the case (for example, if the login form is a modal on the same URL), you will need to adapt the test to check for the visibility of the login form or other indicative elements.
3. **\*\*Environment Configurations\*\***: Consider setting the base URL (`http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com`) in the

`cypress.config.js` to avoid repeating it in every test. Use `cy.visit('/home')` instead after configuration.

4. **Error Handling**: Add error handling or assertions as necessary to ensure the test is robust and gives clear feedback on failures.

By following these steps, you will create a simple automated test in Cypress that verifies access to the login page from the home page as described in your test case.

Проаналізувавши код, перейдемо до його запуску щоб переконатися у працездатності та валідності за допомогою команди “`nrx cypress run --spec cypress/e2e/home_page.cy.js`”. Результат запуску можна побачити на рис.3.4:

```

Running: home_page.cy.js (1 of 1)

Login Page Accessibility
 1) Accesses the login page from the home page

0 passing (6s)
1 failing

1) Login Page Accessibility
 Accesses the login page from the home page:
 AssertionError: Timed out retrying after 4000ms: Expected to find element: `#loginButton`, but never found it.
 at Context.eval (webpack://cypress-test/./cypress/e2e/home_page.cy.js:9:9)

(Results)

Tests: 1
Passing: 0
Failing: 1
Pending: 0
Skipped: 0
Screenshots: 1
Video: false
Duration: 5 seconds
Spec Ran: home_page.cy.js

(Screenshots)
- /cypress-test/cypress/screenshots/home_page.cy.js/Login Page Accessibility -- Accesses the login page from the home page (failed).png

(Run Finished)

Spec Tests Passing Failing Pending Skipped
* home_page.cy.js 00:05 1 - 1 - -
* 1 of 1 failed (100%) 00:05 1 - 1 - -

```

Рисунок 3.4 - Запуск авто-тесту для тестового сценарію #1 (створено самостійно)

Також, знімок екрану зроблений тестом Cypress під час запуску авто-тесту для сценарію #1 свідчить про те що, запуск відбувся:

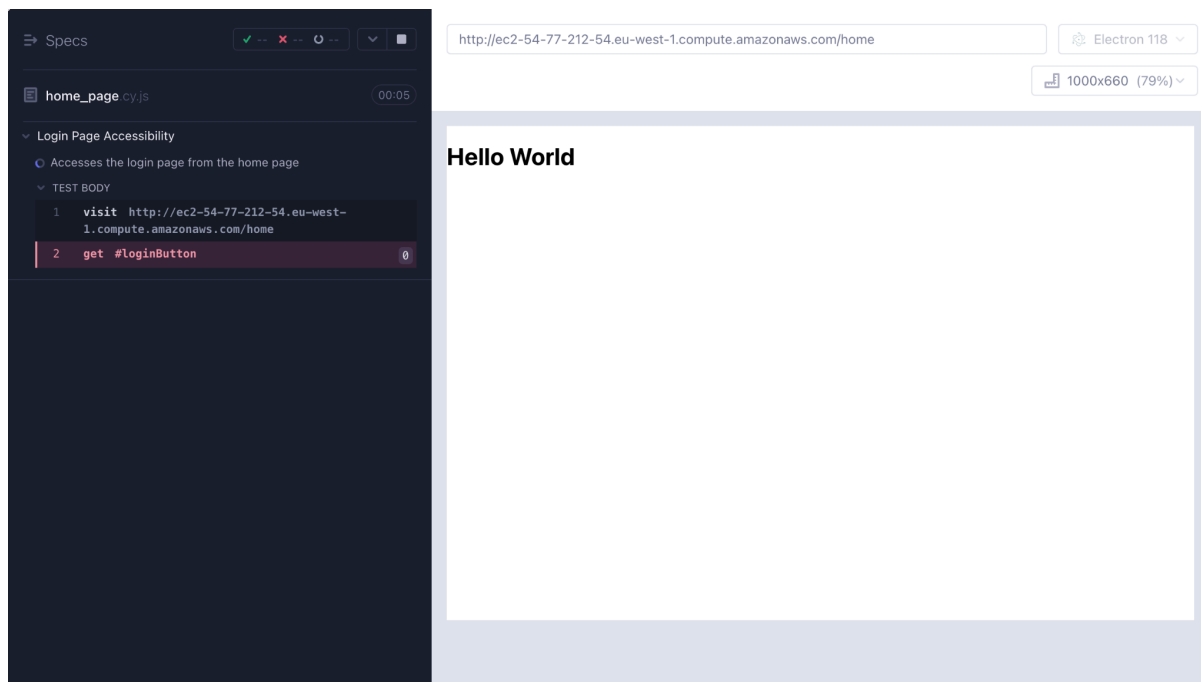


Рисунок 3.5 - Знімок екрану Cypress авто-тесту для сценарію #1 (створено самостійно)

Таким чином ми бачимо, що згенерований код авто тесту не містить помилок, що пов'язані з синтаксисом або іншими функціональними особливостями, тобто код є валідним і робочим.

Загальним висновком, щодо результату є факт того, що код та опис необхідних дій призводить до успішного запуску авто тесту, отже дані мають значну релевантність. Генерація відповіді зайняла трохи більше часу ніж попередні експерименти, через те, що завдання включало в себе створення коду. Обробка результату також вимагає більшої уваги та впливає на час роботи з авто тестом. Після аналізу згенерованого коду та його запуску, стало зрозуміло, що уточнення не потребуються.

Перейдемо до наступної частини експерименту - запуску нашої програми для кожного тестового сценарію та обробки результатів запуску кожного авто тесту.

Результат експерименту для тестового сценарію “Test Case 2: Login Page Fields”:

```

> py test.py --input input-data/auto-test-login-page.txt
Request completed in 29.09 seconds.
Output written to output-data/output_2024-04-12_19-49-52.txt
Below is the Cypress test code (`login_page_spec.cy.js`) that verifies the presence of the email and password input fields and performs
checks that both input fields exist and are correctly labeled or identified.

```javascript
// login_page_spec.cy.js
describe('Login Page Validation', () => {
  it('Ensures the login page includes the correct fields', () => {
    // Step 1: Access the login page
    cy.visit('http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com');

    // Expected Result: The page should contain an email address input field
    cy.get('input[type="email"]').should('be.visible');

    // Expected Result: The page should contain a password input field
    cy.get('input[type="password"]').should('be.visible');
  });
});
```

```

Рисунок 3.6 - Результат генерації авто тесту сценарію #2 (створено самостійно)

```

Running: login_page_spec.cy.js (1 of 1)

Login Page Validation
 ✓ Ensures the login page includes the correct fields (1733ms)

1 passing (2s)

(Results)

Tests: 1
Passing: 1
Failing: 0
Pending: 0
Skipped: 0
Screenshots: 0
Video: false
Duration: 1 second
Spec Ran: login_page_spec.cy.js

====

(Run Finished)

Spec Tests Passing Failing Pending Skipped
✓ login_page_spec.cy.js 00:01 1 1 - - -
✓ All specs passed! 00:01 1 1 - - -

```

Рисунок 3.7 - Запуск авто тесту для тестового сценарію #2 (створено самостійно)

Результат експерименту для тестового сценарію “Test Case 3: Email Format Validation”:

```

> py test.py --input input-data/auto-test-email-format.txt
Request completed in 34.45 seconds.
Output written to output-data/output_2024-04-12_20-12-51.txt
Below is the Cypress test script that can be used to validate the email format as described in the test case.
ld and form submission button have appropriate selectors (like IDs, classes, or attributes) which are commonly
You can save this script in a file named `email_format_validation.cy.js` within your `cypress/e2e/` directory
```javascript
// email_format_validation.cy.js

describe('Email Format Validation', () => {
  it('Validates that the email field checks for standard email formats', () => {
    // Step 1: Navigate to the login page
    cy.visit('http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com');

    // Step 2: Enter an invalid email format in the email field
    cy.get('input[type="email"]').type('user@example.com');

    // Step 3: Attempt to submit the form
    cy.get('form').submit();

    // Expected Result: An error message should be displayed indicating an invalid email format
    cy.contains('Invalid email format').should('be.visible');
  });
});
```

Notes:
1. Replace `input[type="email"]` with the actual selector for the email input field on your login page.
2. Replace `form` with the actual selector used for the form on your login page, if it has a specific ID or class.
3. Replace the error message `Invalid email format` with the actual error message text that your application displays.
4. Ensure that your test environment and application URL (`http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com`) are correctly specified.

This script will open the specified URL, input an incorrectly formatted email, attempt to submit the form, and
omating the manual test process described in your test case.

```

Рисунок 3.8 - Результат генерації авто тесту сценарію #3 (створено самостійно)

```

Running: email_format.cy.js (1 of 1)

Email Format Validation
 1) Validates that the email field checks for standard email formats

0 passing (7s)
1 failing

1) Email Format Validation
 Validates that the email field checks for standard email formats:
 AssertionError: Timed out retrying after 4000ms: Expected to find content: 'Please' but never did.
 at Context.eval (webpack://cypress-test/./cypress/e2e/email_format.cy.js:15:28)

(Results)

Tests: 1
Passing: 0
Failing: 1
Pending: 0
Skipped: 0
Screenshots: 1
Video: false
Duration: 7 seconds
Spec Ran: email_format.cy.js

(Screenshots)
- [redacted]/cypress-test/cypress/screenshots/email_format.cy.js/Email Format Validation -- Validates that the email field checks for standard email formats (failed).png (2560x1440)

=====

(Run Finished)

Spec Tests Passing Failing Pending Skipped
* email_format.cy.js 00:07 1 - 1 - -
* 1 of 1 failed (100%) 00:07 1 - 1 - -

```

Рисунок 3.9 - Запуск авто тесту для тестового сценарію #3 (створено самостійно)

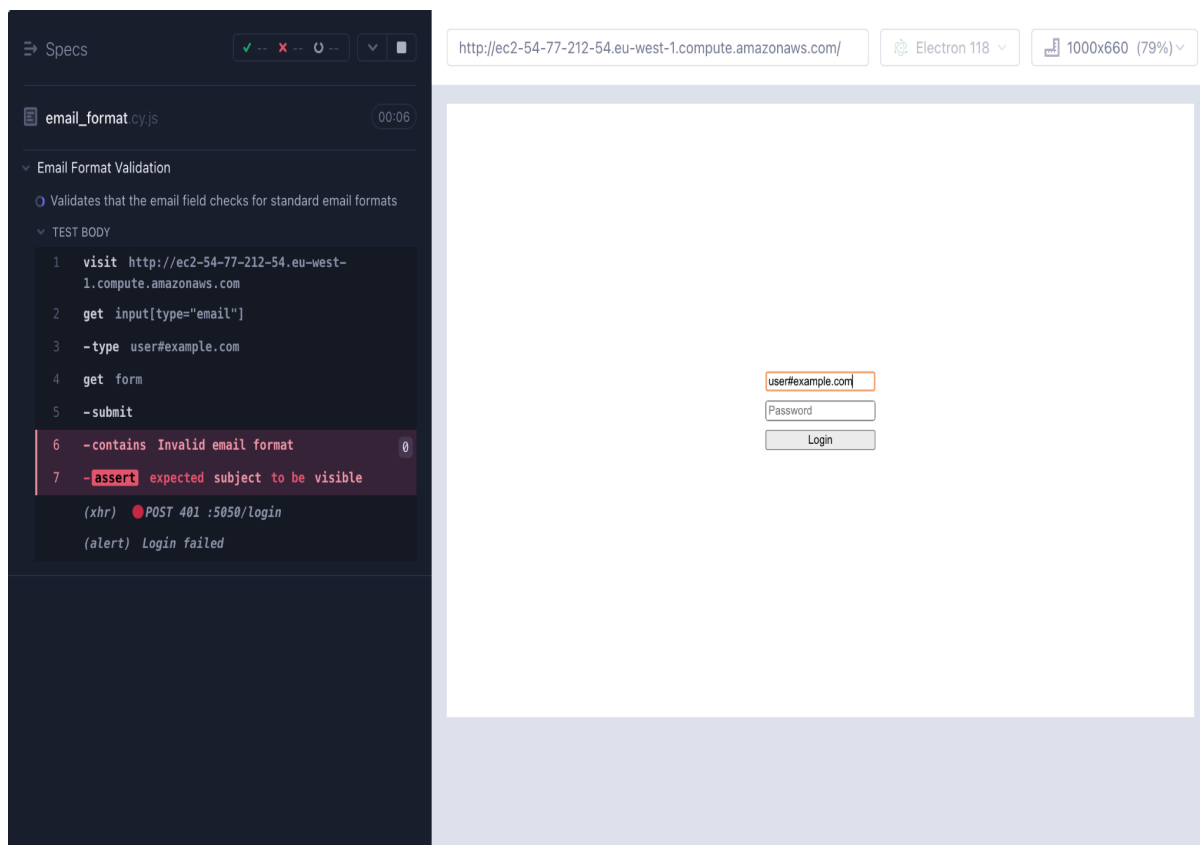


Рисунок 3.9 -Знімок екрану Сурпресс авто тесту для сценарію #3 (створено самостійно)

Результат експерименту для тестового сценарію “Test Case 4: Password Length Validation”:

```

> py test.py --input input-data/4-auto-test-password-length.txt
Request completed in 15.52 seconds.
Output written to output-data/output_2024-04-12_20-41-35.txt
Below is the Cypress test code which you can place in a `.cy.js` file within your `cypress/e2e/` directory

```javascript
// password-length-validation.cy.js

describe('Test Case 4: Password Length Validation', () => {
  it('checks that the password field enforces length restrictions', () => {
    // Step 1: Navigate to the login page.
    cy.visit('/');

    // Step 2: Enter a password that is less than 8 characters long.
    cy.get('#password').type('short');

    // Step 3: Attempt to submit the form
    cy.get('#login-form').submit();

    // Expected Result: An error message should be displayed indicating the password is too short.
    cy.contains('Password is too short').should('be.visible');
  });
});
```

Notes:
- Ensure the selectors like `#password`, `#login-form` are correct as per your application's HTML structure.
- Adjust the error message 'Password is too short' to whatever specific message your application displays.
- This script assumes that the base URL (`http://ec2-54-77-212-54.eu-west-1.compute.amazonaws.com`) is `isit('/')`; to navigate directly to the login page. If the login page is not the root URL, you may need `cy.visit('/login')`.

```

Рисунок 3.10 - результат генерації авто тесту сценарію #4 (створено самостійно)

```
Running: 4_password_length.cy.js (1 of 1)

Password Length Validation
 1) displays an error for passwords that are too short

0 passing (7s)
1 failing

1) Password Length Validation
 displays an error for passwords that are too short:
 AssertionError: Timed out retrying after 4000ms: Expected to find content: 'Password is too short' but never did.
 at Context.eval (webpack://cypress-test/./cypress/e2e/4_password_length.cy.js:15:41)

(Results)

Tests: 1
Passing: 0
Failing: 1
Pending: 0
Skipped: 0
Screenshots: 1
Video: false
Duration: 6 seconds
Spec Ran: 4_password_length.cy.js

(Screenshots)
- [redacted]/cypress-test/cypress/screenshots/4_password_length.cy.js/Password Length Validation -- displays an error for passwords that are too short (failed).png

=====

(Run Finished)

Spec Tests Passing Failing Pending Skipped
* 4_password_length.cy.js 00:06 1 - 1 - -
* 1 of 1 failed (100%) 00:06 1 - 1 - -
```

Рисунок 3.11 - Запуск авто тесту для тестового сценарію #4 (створено самостійно)

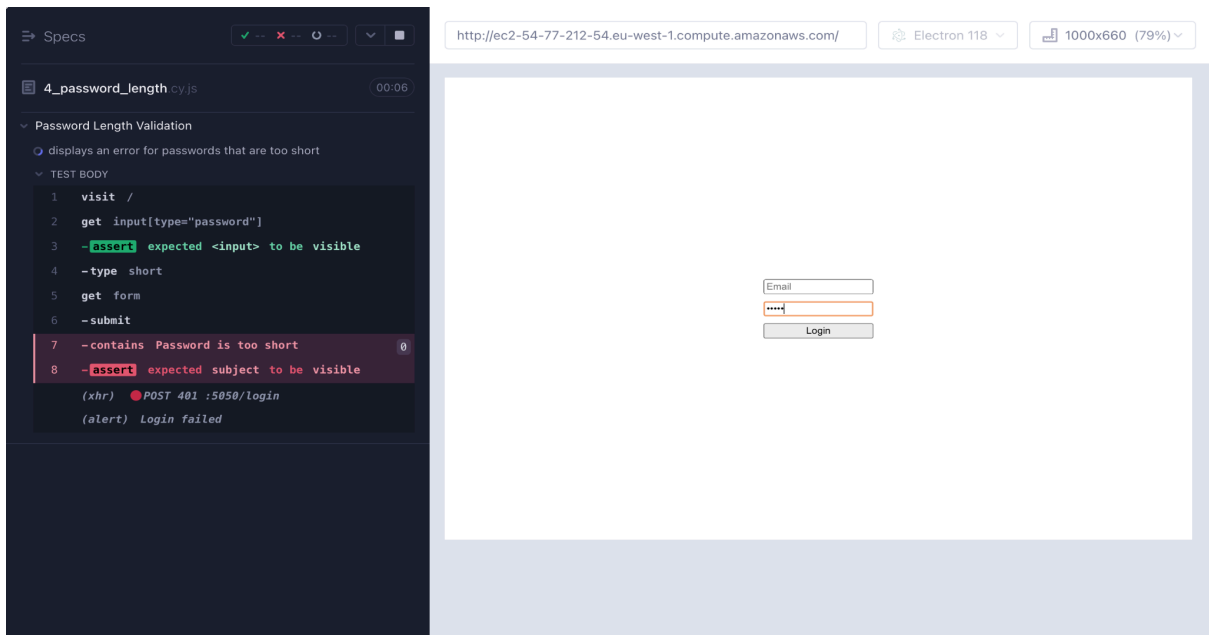


Рисунок 3.12 - Знімок екрану Cypress авто тесту для сценарію #4 (створено самостійно)

Результат експерименту для тестового сценарію “Test Case 5: Successful Login”:

```
> py test.py --input input-data/5-auto-test-password-length.txt
Request completed in 11.23 seconds.
Output written to output-data/output_2024-04-12_21-04-25.txt
```javascript
// successful-login.spec.js

describe('Test Case 5: Successful Login', () => {
  it('Objective: Test the successful login flow', () => {
    // Step 1: Navigate to the login page
    cy.visit('/login');

    // Step 2: Enter a valid email and correct password
    cy.get('input[name="email"]').type('user@example.com');
    cy.get('input[name="password"]').type('example');

    // Step 3: Submit the form
    cy.get('form').submit();

    // Expected Result: The user is redirected to their dashboard or relevant home page
    cy.url().should('include', '/dashboard');

    // Expected Result: A welcome message with the user's name or username is displayed
    cy.contains('Welcome user@example.com').should('be.visible');
  });
});
```
```

Рисунок 3.13 - Результат генерації авто тесту сценарію #5 (створено самостійно)

```
Running: 5_successful_login.cy.js (1 of 1)

Test Case 5: Successful Login
 ✓ successfully logs in and redirects to the dashboard with a welcome message (4200ms)

1 passing (4s)

(Results)

Tests: 1
Passing: 1
Failing: 0
Pending: 0
Skipped: 0
Screenshots: 0
Video: false
Duration: 4 seconds
Spec Ran: 5_successful_login.cy.js

====

(Run Finished)

Spec Tests Passing Failing Pending Skipped
✓ 5_successful_login.cy.js 00:04 1 1 - - -
✓ All specs passed! 00:04 1 1 - - -
```

Рисунок 3.14 - Запуск авто тесту для тестового сценарію #5 (створено самостійно)

Результат експерименту для тестового сценарію “Test Case 6: Unsuccessful Login”:

```

> py test.py --input input-data/6-auto-test-un-successful-login.txt
Request completed in 9.82 seconds.
Output written to output-data/output_2024-04-12_21-58-22.txt
```javascript
// Filename: unsuccessful_login.cy.js
// Location: cypress/e2e/

describe('Test Case 6: Unsuccessful Login', () => {
  it('Verify the system behavior on incorrect credentials', () => {
    // Step 1: Navigate to the login page
    cy.visit('/');

    // Step 2: Enter a valid email with an incorrect password
    cy.get('input[name="email"]').type('user@example.com');
    cy.get('input[name="password"]').type('wrongpassword');

    // Prepare to intercept the login request
    cy.intercept('POST', '/login').as('loginRequest');

    // Step 3: Submit the form
    cy.get('form').submit();

    // Check the response of the intercepted POST request to '/login'
    cy.wait('@loginRequest').then((interception) => {
      expect(interception.response.statusCode).to.eq(401);
      expect(interception.response.body).to.have.property('message', 'Invalid credentials');
    });
  });
});
```

```

Рисунок 3.15 - Результат генерації авто тесту сценарію #6 (створено самостійно)

```

Running: 6_unsuccessful_login.cy.js (1 of 1)

Test Case 6: Unsuccessful Login
 ✓ Verify the system behavior on incorrect credentials (2771ms)

1 passing (3s)

(Results)

Tests: 1
Passing: 1
Failing: 0
Pending: 0
Skipped: 0
Screenshots: 0
Video: false
Duration: 2 seconds
Spec Ran: 6_unsuccessful_login.cy.js

=====

(Run Finished)

Spec Tests Passing Failing Pending Skipped
✓ 6_unsuccessful_login.cy.js 00:02 1 1 - - -
✓ All specs passed! 00:02 1 1 - - -

```

Рисунок 3.16 - Запуск авто тесту для тестового сценарію #6 (створено самостійно)

Результат експерименту для тестового сценарію “Test Case 7: HTTPS Security”:

```

> py test.py --input input-data/7-auto-test-https.txt
Request completed in 4.33 seconds.
Output written to output-data/output_2024-04-12_22-12-16.txt
```javascript
// e2e/specs/https_security_spec.cy.js

describe('HTTPS Security', () => {
  it('confirms that the login page uses HTTPS', () => {
    cy.visit('/')
    .location('protocol').should('eq', 'https:');
  });
});
```

```

Рисунок 3.17 - Результат генерації авто тесту сценарію #7 (створено самостійно)

```

Running: 7_https_security_spec.cy.js (1 of 1)

Test Case 7: HTTPS Security
 1) confirms that the login page uses HTTPS

0 passing (6s)
1 failing

1) Test Case 7: HTTPS Security
 confirms that the login page uses HTTPS:

 Timed out retrying after 4000ms
 + expected - actual

 -'http:'
 +'https:'

 at Context.eval (webpack://cypress-test/./cypress/e2e/7_https_security_spec.cy.js:4:28)

(Results)

Tests: 1
Passing: 0
Failing: 1
Pending: 0
Skipped: 0
Screenshots: 1
Video: false
Duration: 6 seconds
Spec Ran: 7_https_security_spec.cy.js

(Screenshots)

- [redacted]/cypress-test/cypress/screenshots/7_h (2560x1440)
 ttps_security_spec.cy.js/Test Case 7 HTTPS Security -- confirms that the login p
 age uses HTTPS (failed).png

=====

(Run Finished)

Spec Tests Passing Failing Pending Skipped

* 7_https_security_spec.cy.js 00:06 1 - 1 - -
* 1 of 1 failed (100%) 00:06 1 - 1 - -

```

Рисунок 3.18 - Запуск авто тесту для тестового сценарію #7 (створено самостійно)

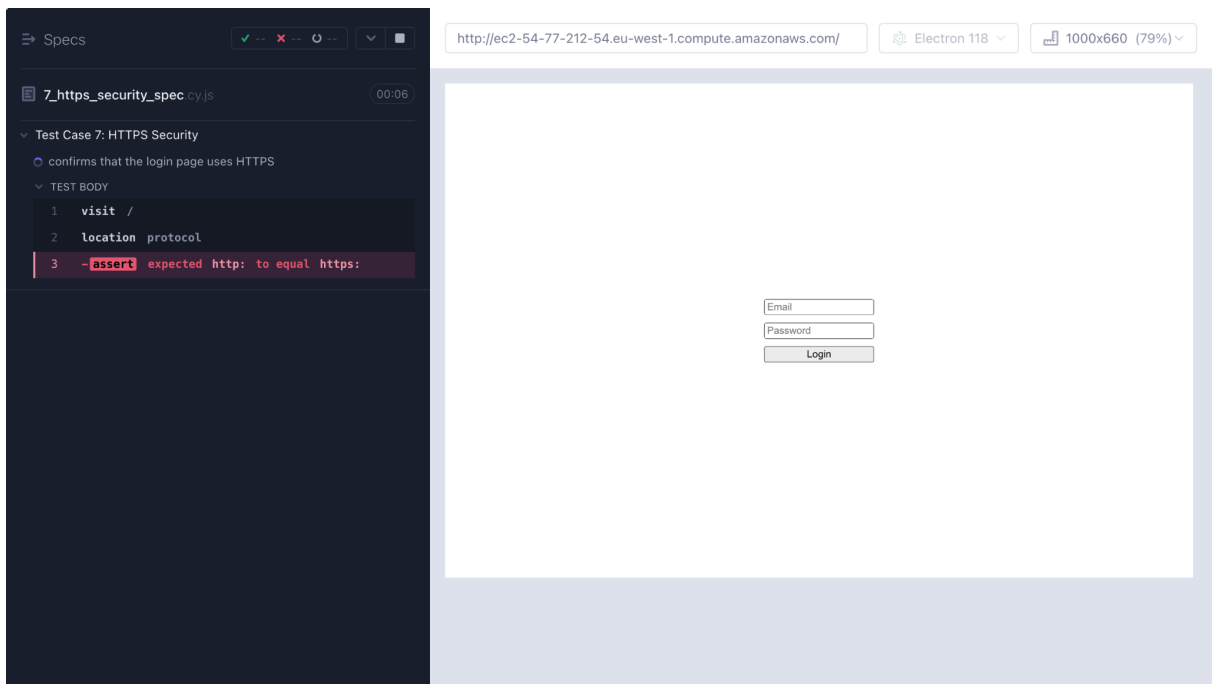


Рисунок 3.19 - Знімок екрану Cypress авто тесту для сценарію #7 (створено самостійно)

Після успішного проведення експерименту для кожного з тестових сценаріїв, можемо зробити висновки, щодо кожної частини цього експерименту, оцінивши результати відповідно до встановлених раніше критеріїв:

Таблиця 3.3 - Оцінки для всіх тестових сценаріїв (таблиця виконана самостійно)

| Номер тестового сценарію/Оцінка |     |     |     |     |     |     | Критерії |
|---------------------------------|-----|-----|-----|-----|-----|-----|----------|
| #1                              | #2  | #3  | #4  | #5  | #6  | #7  |          |
| 3                               | 3   | 4   | 3   | 4   | 3   | 4   | TRT      |
| 2                               | 3   | 2   | 2   | 2   | 2   | 3   | TPR      |
| 0.8                             | 0.9 | 0.7 | 0.6 | 0.9 | 0.7 | 0.8 | RGR      |
| 3                               | 3   | 2   | 2   | 2   | 1   | 2   | NC       |

### 3.4 Дослідження створення звіту з тестування (Test Report Creation)

Щоб створити хороший звіт після тестування, важливо зосередитися на ясності, точності та вичерпності. Добре структурований звіт про тестування має стисло підсумовувати цілі тестування, методологію та використане середовище, а також чітко описувати результати тестування. Він має включати розділи про виявлені дефекти, їх серйозність, наслідки та рекомендовані дії [23]. Ефективне використання візуальних елементів, таких як діаграми та таблиці, може покращити читабельність звіту та допомогти зацікавленим сторонам швидко зрозуміти результати тестування. Загалом хороший звіт про випробування повинен надавати всю необхідну інформацію для прийняття рішень щодо подальших випробувань і готовності продукту до випуску [24].

Тому, для втілення експерименту та отримання валідного результату, необхідно використати результати попередніх експериментів - створені тестові сценарії, результат запуску усіх авто тестів, та недоліки або помилки у програмному забезпеченні, які були знайдені під час ручного та автоматичного тестування. Загальний результат за авто тестами виглядає наступним чином:

| Spec                          | Tests | Passing | Failing | Pending | Skipped |
|-------------------------------|-------|---------|---------|---------|---------|
| * 1_home_page.cy.js           | 00:09 | 1       | –       | 1       | –       |
| ✓ 2_login_page_spec.cy.js     | 00:01 | 1       | 1       | –       | –       |
| * 3_email_format.cy.js        | 00:06 | 1       | –       | 1       | –       |
| * 4_password_length.cy.js     | 00:06 | 1       | –       | 1       | –       |
| ✓ 5_successful_login.cy.js    | 00:02 | 1       | 1       | –       | –       |
| ✓ 6_unsuccessful_login.cy.js  | 00:02 | 1       | 1       | –       | –       |
| * 7_https_security_spec.cy.js | 00:05 | 1       | –       | 1       | –       |
| * 4 of 7 failed (57%)         | 00:35 | 7       | 3       | 4       | –       |

Рисунок 3.20 – Результат запуску всіх створених авто тестів (створено самостійно)

Створимо файл “input-data/test-report-generation.txt” для подальшої реалізації експерименту та генерації безпосередньо звіту з тестування:

Input Data:

User story: As a user I want to be able to login at our website using email and password.

```
Test Case 1: Access Login Page from Home Page
```

```
...
```

```
Test Case 7: HTTPS Security
```

```
...
```

Results of Cypress run for all the automated tests related to each test case.

```
✘ 1_home_page.cy.js
✔ 2_login_page_spec.cy.js
✘ 3_email_format.cy.js
✘ 4_password_length.cy.js
✔ 5_successful_login.cy.js
✔ 6_unsuccessful_login.cy.js
✘ 7_https_security_spec.cy.js
✘ 4 of 7 failed (57%)
```

Task:

Create QA test report

Additional info:

- The report should be relevant - shortly describing the issues found.
- The report should not contain irrelevant text that talks about the report itself or how to use it

Проведемо запуск програми для вхідних даних та отримаємо згенерований звіт з тестування:

```
> py test.py --input input-data/test-report-generation.txt
Request completed in 24.55 seconds.
Output written to output-data/output_2024-04-12_22-42-58.txt
QA Test Report: Website Login Functionality

Date: [Insert Date]

Test Environment: [Specify Environment, e.g., Staging]

Overview: A series of seven automated test cases were executed to validate the login function and security protocols.

Test Results Summary:
- Total Tests: 7
- Passed: 3
- Failed: 4

Detailed Test Outcomes:

1. **Access Login Page from Home Page**
 - **Result**: Failed
```

Рисунок 3.21 - Результат запуску програми для експерименту “Test Report Creation” (створено самостійно)

## Переглянемо та проаналізуємо згенерований звіт:

**QA Test Report: Website Login Functionality**

**Date:** [Insert Date]

**Test Environment:** [Specify Environment, e.g., Staging]

**Overview:** A series of seven automated test cases were executed to validate the login functionality of the website. The tests ranged from page accessibility to input validation and security protocols.

**Test Results Summary:**

- Total Tests: 7
- Passed: 3
- Failed: 4

**Detailed Test Outcomes:**

1. **Access Login Page from Home Page**

- **Result:** Failed
- **Issue:** The login page is not accessible from the home page. The link or button intended to navigate to the login page may be missing or not functioning as expected.

2. **Login Page Fields**

- **Result:** Passed
- **Details:** The login page correctly includes both an email address input field and a password input field.

3. **Email Format Validation**

- **Result:** Failed
- **Issue:** The email field does not validate for standard email formats. Inputs such as "user#example.com" are accepted without error, which should not be the case.

4. **Password Length Validation**

- **Result:** Failed
- **Issue:** The password field does not enforce the minimum length restriction. Passwords shorter than 8 characters are accepted, which contradicts the expected behavior.

5. **Successful Login**

- **Result:** Passed
- **Details:** The login process with correct credentials operates as expected, successfully redirecting the user to their dashboard with a welcome message.

6. **Unsuccessful Login**

- **Result:** Passed
- **Details:** The system appropriately handles incorrect credentials, displaying an error message about the wrong email or password.

7. **HTTPS Security**

- **Result:** Failed
- **Issue:** The login page is not secured with HTTPS. The URL does not begin with "https://", thus not ensuring a secure connection.

**\*\*Recommendations\*\*:**

- **\*\*Urgency for Fixing Access to Login Page\*\*:** The inability to access the login page from the home page is critical and needs immediate attention.
- **\*\*Email and Password Validation\*\*:** Implement strict validation rules for email formats and enforce password length to enhance security and user experience.
- **\*\*Secure the Login Page with HTTPS\*\*:** Transition to HTTPS to protect user data and comply with best security practices.

**\*\*Next Steps\*\*:**

- The development team should address the identified issues promptly.
- Retest the failed cases after corrections have been implemented to ensure all functionalities are working as intended.

**\*\*Conclusion\*\*:**

This test cycle has highlighted crucial areas for improvement in the login functionality, which are essential for both user experience and security. Immediate action is recommended to resolve these issues.

Проведемо оцінювання за критеріями, позначивши відповідні оцінки у таблиці:

Таблиця 3.4 - Оцінювання за критеріями (таблиця виконана самостійно)

| Оцінка | Критерій                                  |
|--------|-------------------------------------------|
| 4      | Час вирішення завдання (TRT)              |
| 3      | Час обробки результатів (TPR)             |
| 0.9    | Релевантність отриманих результатів (RGR) |
| 3      | Кількість уточнень (NC)                   |

На основі представлених результатів тестування можна зробити висновок, що звіт, створений за допомогою генеративної мовної моделі, є адекватним і високоякісним. Модель ефективно ідентифікувала ключові проблеми в процесах входу на сайт, включаючи критичні питання безпеки та функціональності, такі як відсутність доступу до сторінки логіну з головної сторінки, проблеми з валідацією електронної пошти та пароля, та необхідність захисту за допомогою HTTPS. Результати чітко окреслили потребу в імплементації жорстких правил валідації та забезпечення захисту даних, підкреслюючи їх значущість для збільшення безпеки та надійності веб-додатку.



## 4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

### 4.1 Аналіз ефективності за критеріями

За отриманими результатами вимірів можна зробити порівняння отриманих даних для кожного експерименту. Спочатку створимо зведену таблицю із оцінками за критеріями для кожного експерименту (див табл. 4.1)

Таблиця 4.1 - Результати всіх експериментів (таблиця виконана самостійно)

| Критерій/Оцінка   |     |     |      |    |
|-------------------|-----|-----|------|----|
| #<br>Експерименту | TRT | TPR | RGR  | NC |
| 1                 | 4   | 3   | 0,66 | 3  |
| 2                 | 4   | 2   | 0,91 | 3  |
| 3.1               | 3   | 2   | 0,82 | 3  |
| 3.2               | 3   | 3   | 0,91 | 3  |
| 3.3               | 4   | 2   | 0,69 | 2  |
| 3.4               | 3   | 2   | 0,61 | 2  |
| 3.5               | 4   | 2   | 0,91 | 2  |
| 3.6               | 3   | 2   | 0,71 | 1  |
| 3.7               | 4   | 3   | 0,83 | 2  |
| 4                 | 4   | 3   | 0,91 | 3  |

Для того, щоб забезпечити узгодженість даних за оцінками, що мають різні одиниці виміру необхідно провести нормалізацію оцінок за критеріями. Цей процес допомагає уникнути перекосів вагомості різних критеріїв та математично коректно комбінувати різні критерії у єдиний показник, забезпечуючи більшу об'єктивність і точність в оцінках.

Для нормування оцінок за шкалами використаємо нормування “за еталоном”. Отримаємо нову таблицю (див табл. 4.2) значень де усі критерії мають оцінку від 0 до 1:

Таблиця 4.2 - Результати всіх експериментів з оцінками після нормування  
(таблиця виконана самостійно)

| #<br>Експерименту | Критерій/Оцінка |      |      |      |
|-------------------|-----------------|------|------|------|
|                   | TRT             | TPR  | RGR  | NC   |
| 1                 | 1,00            | 1,00 | 0,66 | 1,00 |
| 2                 | 1,00            | 0,67 | 0,91 | 1,00 |
| 3.1               | 0,75            | 0,67 | 0,82 | 1,00 |
| 3.2               | 0,75            | 1,00 | 0,91 | 1,00 |
| 3.3               | 1,00            | 0,67 | 0,69 | 0,67 |
| 3.4               | 0,75            | 0,67 | 0,61 | 0,67 |
| 3.5               | 1,00            | 0,67 | 0,91 | 0,67 |
| 3.6               | 0,75            | 0,67 | 0,71 | 0,33 |
| 3.7               | 1,00            | 1,00 | 0,83 | 0,67 |
| 4                 | 1,00            | 1,00 | 0,91 | 1,00 |

Створимо діаграму оцінок за усіма критеріями для більш наглядного візуального аналізу отриманих результатів:

Результат дослідження за критеріями

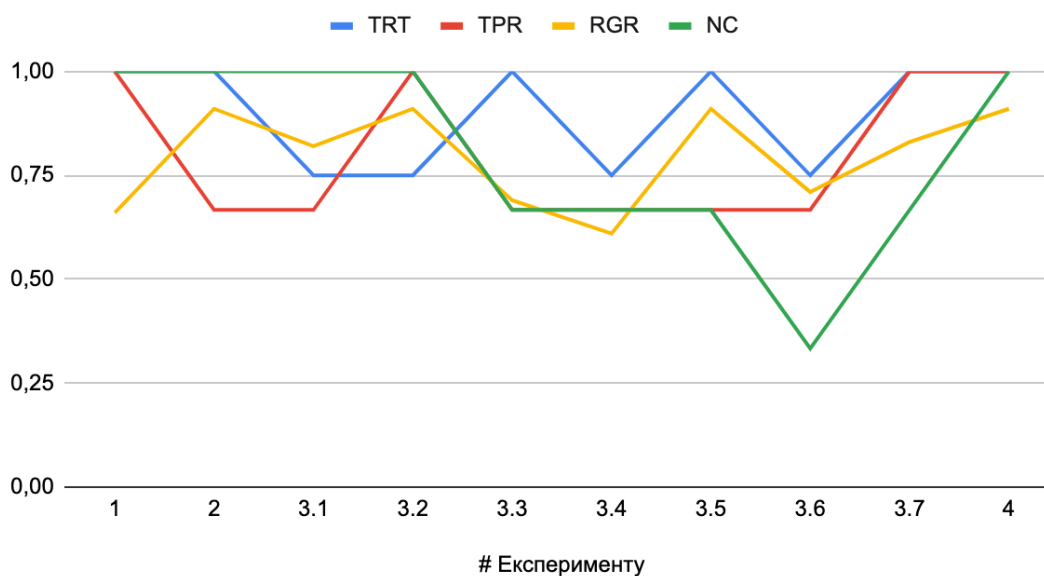


Рисунок 4.1 - Діаграма результатів дослідження за критеріями (створено самостійно)

Можемо помітити, що всі оцінки за всіма критеріями мають значення, що перевищує 0,66 за одним виключенням для експерименту 3.6. Враховуючи, що критерії мають шкали що сформовані за принципом оптимальності “за максимумом” це свідчить про загальний високий рівень ефективності результатів.

Час вирішення завдання (TRT) не опускається нижче рівня 0,75 - це означає, що даний критерій з одного боку має незначний вплив на результат експерименту, проте, з іншого боку це також свідчить про те, що вибрана велика мовна модель має високу швидкість вирішення задач незалежно від їх змісту. Для нашого дослідження це є позитивним фактором.

Час обробки результатів (TPR) у більшості випадків становить 0,67 або вище. Це вказує на те, що час витрачений на обробку переважної більшості згенерованих результатів становив 10-20 хвилин. Враховуючи той факт, що обробка результатів відбувається у ручному режимі, цей результат є досить задовільним, проте може бути покращений.

Релевантність отриманих результатів (RGR) є найбільш вагомим критерієм. І за результатом досліджень, можемо бачити, що так як цей показник не падає за межі 0,6 і навіть становить 0,8-0,9 у багатьох випадках - згенеровані результати мають досить високий показник релевантності, що є ключовим фактором при прийнятті рішення про застосування штучного інтелекту у процесах QA тестування веб-додатків.

Кількість уточнень (NC) залежить від якості сформованих вхідних даних, а також від здібностей великої мовної моделі, що обробляє ці дані. В нашому випадку, сформовані вхідні дані не мали значних недоліків, а результат експерименту підтверджує те, що здібності штучного інтелекту в обробці вхідних даних мають високий показник ефективності. Значення для кожного експерименту перевищує 0,66 лише за одним виключенням для експерименту 3.6, що може свідчити про додаткову складність цього експерименту або вхідних даних.

Проведемо розрахунки відповідно до обраної математичної моделі визначення ефективності застосування штучного інтелекту у процесах QA тестування веб-додатків та переглянемо результати у табл. 4.3.

Таблиця 4.3 - Результати ефективності (таблиця виконана самостійно)

| Критерій/Оцінка       |      |      |      |      |                     |
|-----------------------|------|------|------|------|---------------------|
| #<br>Експеримент<br>у | TRT  | TPR  | RGR  | NC   | Ефективність<br>(%) |
| 1                     | 1,00 | 1,00 | 0,66 | 1,00 | 89,80%              |
| 2                     | 1,00 | 0,67 | 0,91 | 1,00 | 88,97%              |
| 3.1                   | 0,75 | 0,67 | 0,82 | 1,00 | 81,27%              |
| 3.2                   | 0,75 | 1,00 | 0,91 | 1,00 | 92,30%              |
| 3.3                   | 1,00 | 0,67 | 0,69 | 0,67 | 74,03%              |
| 3.4                   | 0,75 | 0,67 | 0,61 | 0,67 | 66,63%              |
| 3.5                   | 1,00 | 0,67 | 0,91 | 0,67 | 80,63%              |
| 3.6                   | 0,75 | 0,67 | 0,71 | 0,33 | 61,30%              |
| 3.7                   | 1,00 | 1,00 | 0,83 | 0,67 | 86,57%              |
| 4                     | 1,00 | 1,00 | 0,91 | 1,00 | 97,30%              |
| Вагові<br>коефіцієнти | 0,2  | 0,25 | 0,3  | 0,25 |                     |

Створимо діаграму за таблицею для візуалізації значень ефективності результатів за кожним експериментом:

## Діаграма ефективності

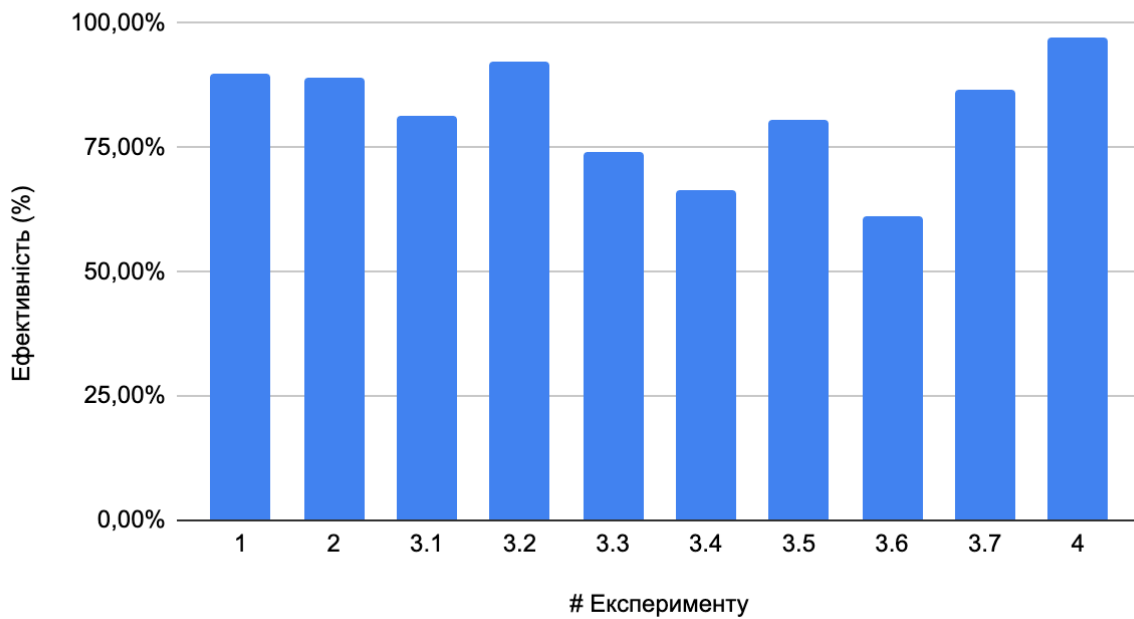


Рисунок 4.2 - Діаграма ефективності (створено самостійно)

Результати дослідження демонструють високий рівень ефективності великої мовної моделі GPT у вирішенні задач QA тестування, з більшістю оцінок за експериментами значно вище 0,66. Особливо вражає, що для багатьох експериментів показники ефективності склали від 80% до 97% із загальним середнім результатом у 81,88%, що свідчить про високу адаптивність і потенціал технології.

Проте, деякі результати, як наприклад у експерименті 3.6 з показником 61,30%, потребують уваги і можуть вказувати на необхідність покращення в алгоритмах обробки або підготовки вхідних даних, щоб забезпечити більшу стабільність і надійність результатів. Це допоможе досягти не тільки високих показників ефективності, але й їх однорідності на всіх рівнях тестування.

### 4.2 Рекомендації щодо подальшого застосування

Наше дослідження підтвердило високу ефективність великих мовних моделей, таких як GPT, у процесах QA тестування веб-додатків. Їх використання значно спрощує вирішення комплексних задач і дозволяє автоматизувати багато

аспектів тестування, що традиційно потребували значного вкладу часу та ресурсів.

Завдяки автоматизації процесів, значно зменшується час від виявлення до усунення дефектів. Високий рівень релевантності згенерованих відповідей підтверджує ефективність моделі у виконанні специфічних завдань QA.

Для подальшого покращення результатів необхідно провести роботу їх покращення та оптимізації вхідних даних. Це дозволить зменшити кількість уточнень і забезпечить більш точне та цілеспрямоване використання моделі. Встановлення чітких процедур для обробки результатів може допомогти уніфікувати процес тестування і забезпечити більшу консистенцію результатів.

Не зайвим стане проведення тренінгів для команди QA з метою кращого розуміння принципів роботи великих мовних моделей і оптимізації їх використання у повсякденній роботі.

Ці рекомендації допоможуть не тільки оптимізувати використання штучного інтелекту в QA процесах, але й забезпечити більш високий рівень загальної продуктивності тестування. Впровадження цих заходів має стати ключовим кроком у подальшому розвитку тестування веб-додатків.

## ВИСНОВКИ

Використання великих мовних моделей у процесах QA тестування сучасних веб-додатків показало значний потенціал для підвищення ефективності і якості робочих процесів. Моделі, такі як GPT, ефективно справляються з автоматизацією складних і неявних аспектів генерації та аналізу тестів, що може значно зменшити часові та трудові витрати.

Релевантність генерації тестових сценаріїв висока, що засвідчує потенціал штучного інтелекту для забезпечення критичних аспектів QA тестування. Виявлення дефектів і повнота охоплення тестами також показали хороші результати, що підкреслює високу ефективність у цих напрямках.

Інтеграція великих мовних моделей демонструє потенціал для скорочення загального часу, необхідного для тестування, через швидшу генерацію тестів і аналіз результатів.

Попри значні переваги, є кілька обмежень, які потребують уваги: необхідність подальшої оптимізації вхідних даних та налаштувань алгоритмів для уникнення помилок і забезпечення стабільності результатів; потреба в додаткових ресурсах для тренінгу та адаптації персоналу до нових технологій.

На основі аналізу рекомендується:

- розширити використання великих мовних моделей у тестуванні для більшого охоплення тестами і кращої автоматизації
- провести додаткові дослідження для визначення оптимальних параметрів моделей в різних сценаріях використання
- розробити комплексні тренінгові програми для інженерів та розробників для підвищення їх компетенцій у роботі з новітніми технологіями

Це дослідження відкриває нові перспективи для використання штучного інтелекту в QA тестуванні і може сприяти значному прогресу в цій області, дозволяючи розробникам і тестувальникам знаходити нові шляхи для покращення ефективності та точності при вирішенні задач створення програмного забезпечення.

Використання великих мовних моделей допомагає не тільки оптимізувати існуючі методики, але й відкриває можливості для створення зовсім нових підходів до аналізу якості програмного забезпечення.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Akmayulinari A. I. F. Multi Criteria Decision Making | PDF | Scenario P Guijarro R. Testing strategies for modern web applications. Medium. URL: <https://medium.com/scopedev/testing-strategies-for-modern-web-applications-71836e480cc6> (date of access: 16.04.2024).
2. Levels of Testing: A Complete Approach to Quality Assurance. Testsigma Blog. URL: <https://testsigma.com/blog/levels-of-testing/> (date of access: 16.04.2024).
3. Levels of Testing in Software Testing. Guru99. URL: <https://www.guru99.com/levels-of-testing.html> (date of access: 16.04.2024).
4. System Testing - GeeksforGeeks. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/system-testing/> (date of access: 16.04.2024)
5. Key Difference between Functional and Non Functional Testing. Testsigma Blog. URL: <https://testsigma.com/blog/functional-and-non-functional-testing/> (date of access: 16.04.2024).
6. Differences Between Functional and Non-functional Testing | BrowserStack. BrowserStack. URL: <https://www.browserstack.com/guide/functional-vs-non-functional-testing> (date of access: 16.04.2024).
7. Colantonio J. 5 great ways to use AI in your test automation | TechBeacon. TechBeacon. URL: <https://techbeacon.com/app-dev-testing/how-ai-changing-test-automation-5-examples> (date of access: 16.04.2024).
8. Chattopadhyay T. Revolutionizing Software Testing: Unleashing the Power of AI in Automated Testing - DZone. dzone.com. URL: <https://dzone.com/articles/revolutionizing-software-testing> (date of access: 16.04.2024).
9. The AI Testing Revolution: Innovations and Insights for 2024. TMS Outsource. URL: <https://tms-outsource.com/blog/posts/how-ai-is-revolutionizing-automated-testing-in-2024> (date of access: 16.04.2024).

10. AI-Based Software Testing: The Future of Test Automation. AI-driven E2E automation with code-like flexibility for your most resilient tests. URL: <https://www.testim.io/resources/ai-automated-testing-future/> (date of access: 16.04.2024).
11. OpenAI's ChatGPT vs. Google's Bard AI: A Comparative Analysis | Deepchecks. Deepchecks. URL: <https://deepchecks.com/openais-chatgpt-vs-googles-bard-ai-a-comparative-analysis/> (date of access: 16.04.2024).
12. Google Bard vs ChatGPT: A Comparative Analysis. Simform - Product Engineering Company. URL: <https://www.simform.com/blog/google-bard-vs-chatgpt/> (date of access: 16.04.2024)
13. Akmayulinari A. I. F. Multi Criteria Decision Making | PDF | Scenario Planning | Strategic Management. Scribd. URL: <https://www.scribd.com/document/204395979/Multi-Criteria-Decision-Making> (date of access: 16.04.2024)
14. Devi S. Multi-Criteria Decision Making | PDF | Sensitivity Analysis | Analysis. Scribd. URL: <https://www.scribd.com/document/154949202/Multi-Criteria-Decision-Making> (date of access: 16.04.2024).
15. 3 AI, automation, and testing · AI-Assisted Testing. liveBook · Manning. URL: <https://livebook.manning.com/book/ai-assisted-testing/chapter-3/v-3/49> (date of access: 16.04.2024).
16. Generative AI in Software Testing: Reshaping the QA Landscape - testRigor. testRigor AI-Based Automated Testing Tool. URL: <https://testrigor.com/generative-ai-in-software-testing/> (date of access: 16.04.2024).
17. AI-Assisted Testing. Manning Publications. URL: <https://www.manning.com/books/ai-assisted-testing> (date of access: 16.04.2024).
18. Katalon. Autonomous Testing: A Complete Guide | Katalon. katalon.com. URL: <https://katalon.com/resources-center/blog/autonomous-testing> (date of access: 16.04.2024).

19. Груздо І.В., Мічурін І.Є. Особливості використання хмарних технологій для розміщення програмного забезпечення. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : тези доп. 12-ї міжнар. наук.-техн. конф., 27-28 квітня 2022 р., Баку–Харків–Жиліна : [у 2 т.]. Т. 2 : секція 5 / Військ. акад. збройних сил Азербайджанської Республіки
20. Evaluating Large Language Models: A Comprehensive Survey. arXiv.org. URL: <https://arxiv.org/abs/2310.19736> (date of access: 16.04.2024).
21. Writing Test Cases from User Stories & Acceptance Criteria. TestLodge Blog. URL: <https://blog.testlodge.com/writing-test-cases-from-user-stories-acceptance-criteria/> (date of access: 16.04.2024).
22. How to create Test Scenarios? (with Examples) | BrowserStack. BrowserStack. URL: <https://www.browserstack.com/guide/how-to-create-test-scenarios> (date of access: 16.04.2024).
23. Why should Companies focus on Automated Software Testing? | BrowserStack. BrowserStack. URL: <https://www.browserstack.com/guide/why-should-companies-focus-on-automated-testing#toc1> (date of access: 16.04.2024)
24. How to Write A Good Test Summary Report (Template + Example) - TestRail. TestRail | The Quality OS for QA Teams. URL: <https://www.testrail.com/blog/test-summary-report/> (date of access: 16.04.2024).