НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Виконав студент IV курсу, групи ІП-01
(шифр групи)

Москаленко Владислав Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник д.т.н., проф., засл.діяч, Павлов О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент професор кафедри ІСТ, д.т.н., проф. Корнага Я. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Москаленку Владиславу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Плагін браузера Google Chrome для автоматизації тестування веб застосунків

керівник проєкту Павлов Олександр Анатолійович, д.т.н., проф., засл.діяч
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27»квітня 2024 р. № 2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання, аналіз та розроблення вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, засоби розробки, конструювання програмного забезпечення.
4). Аналіз якості та тестування програмного забезпечення: аналіз якості, опис тестів, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна компонентів програмного забезпечення _____

3) Креслення вигляду екранних форм _____

_____

_____

_____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	05.04.2024	
3	Постановка та формалізація задачі	15.04.2024	
4	Розробка інформаційного забезпечення	20.04.2024	
5	Алгоритмізація задачі	24.04.2024	
6	Обґрунтування вибору використаних технічних засобів	24.04.2024	
7	Розробка програмного забезпечення	08.05.2024	
8	Налагодження програми	12.05.2024	
9	Виконання графічних документів	27.05.2024	
10	Оформлення пояснювальної записки	25.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Москаленко МОСКАЛЕНКО

(підпис) (ініціали, прізвище)

Керівник _____ Олександр ПАВЛОВ

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 35 таблиць, 20 рисунків та 22 джерел – загалом 74 сторінки.

Дипломний проєкт присвячений розробці плагіна для Google Chrome, який спрощує процес автоматизації тестування веб застосунків.

Метою роботи є спрощення можливості створення, підтримки та виконання тестових сценаріїв за рахунок створення Google Chrome плагіна.

Об'єкт дослідження: процес автоматизації тестування веб застосунків.

Предмет дослідження: методи і засоби автоматизації тестування веб застосунків за допомогою плагінів для браузерів.

У першому розділі проаналізовано предметну область, існуючі рішення та описано бізнес процеси і постановку задачі.

У другому розділі описано варіанти використання системи, а також розроблено функціональні та нефункціональні вимоги.

Третій розділ присвячений проєктуванню архітектури та конструюванню програмного забезпечення.

У четвертому розділі проаналізовано якість програмного забезпечення та описано процеси тестування.

П’ятий розділ присвячено розгортанню та супроводу програмного забезпечення.

КЛЮЧОВІ СЛОВА: GOOGLE CHROME ПЛАГІН, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, AI СЕРВІС, ГЕНЕРАЦІЯ КОДУ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 35 tables, 20 figures, and 22 sources – in total 74 pages.

The purpose of the diploma project is to simplify the creation, maintenance, and execution of test scenarios by developing a Google Chrome extension.

The diploma project is dedicated to the development of a Google Chrome plugin, which simplifies the process of automating the testing of web applications.

Object of research: The process of automating the testing of web applications.

Subject of research: Methods and tools for automating the testing of web applications using browser extensions.

In the first section, the subject area is analyzed, existing solutions are reviewed, and business processes and task statements are described.

The second section describes the use cases of the system and develops functional and non-functional requirements.

The third section is devoted to the design of the architecture and construction of the software.

The fourth section analyzes the quality of the software and describes the testing processes.

The fifth section is dedicated to the deployment and maintenance of the software.

KEYWORDS: GOOGLE CHROME EXTENSION, TEST AUTOMATION, AI SERVICE, CODE GENERATION.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0120.045440.01.91	Технічне завдання	17	
3	А4	КПІ.ІП-0120.045440.02.81	Пояснювальна записка	76	
4	А4	КПІ.ІП-0120.045440.03.12	Текст програми	29	
5	А4	КПІ.ІП-0120.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0120.045440.05.34	Керівництво користувача	16	
7	А3	КПІ.ІП-0120.045440.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-0120.045440.00.90		
Розроб.		Москаленко В.Ю.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Павлов О.А.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-01	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Технічне завдання

КПІ.ІП-0120.045440.01.91

“ПОГОДЖЕНО”
Керівник роботи:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____Владислав МОСКАЛЕНКО

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: плагін браузера Google Chrome для автоматизації тестування веб застосунків.

Галузь застосування:

наведене технічне завдання поширюється на розробку плагіну Google Chrome, котрий використовується для автоматизації тестування інтерфейсу веб застосунків та призначений для спрощення створення та виконання автоматизованих тестів для широкого кола тестувальників.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки плагіну Google Chrome є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для забезпечення можливості створення та виконання тестових сценаріїв для широкого кола тестувальників, а також для генерації коду виконання тестів.

Метою розробки є спрощення можливості створення, підтримки та виконання тестових сценаріїв за рахунок створення Google Chrome плагіну.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1. Користувацького інтерфейсу

– Відображення тестових наборів (рис. 4.1)

– Відображення списку тестів (рис. 4.1)

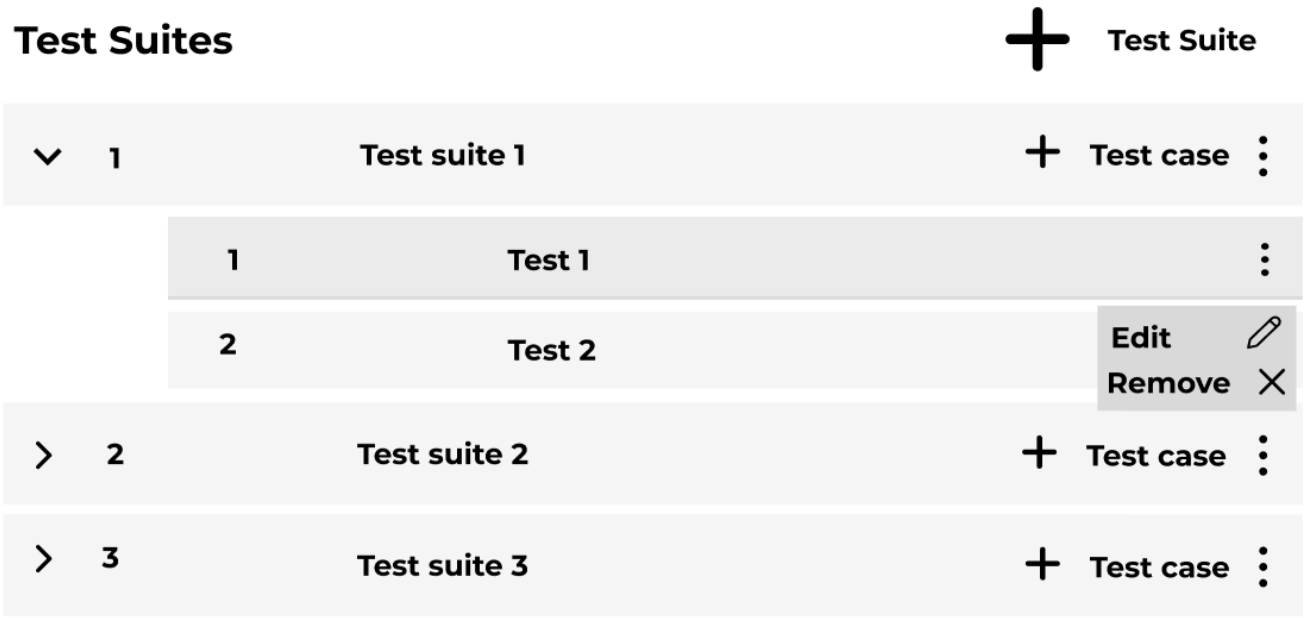


Рисунок 4.1 – Список тестових наборів та тестів

– Відображення тестового сценарію (рис. 4.2)

– Відображення результату виконання тесту (рис. 4.3)

– Відображення вікна генерації коду для автоматизованого тесту (рис. 4.4)

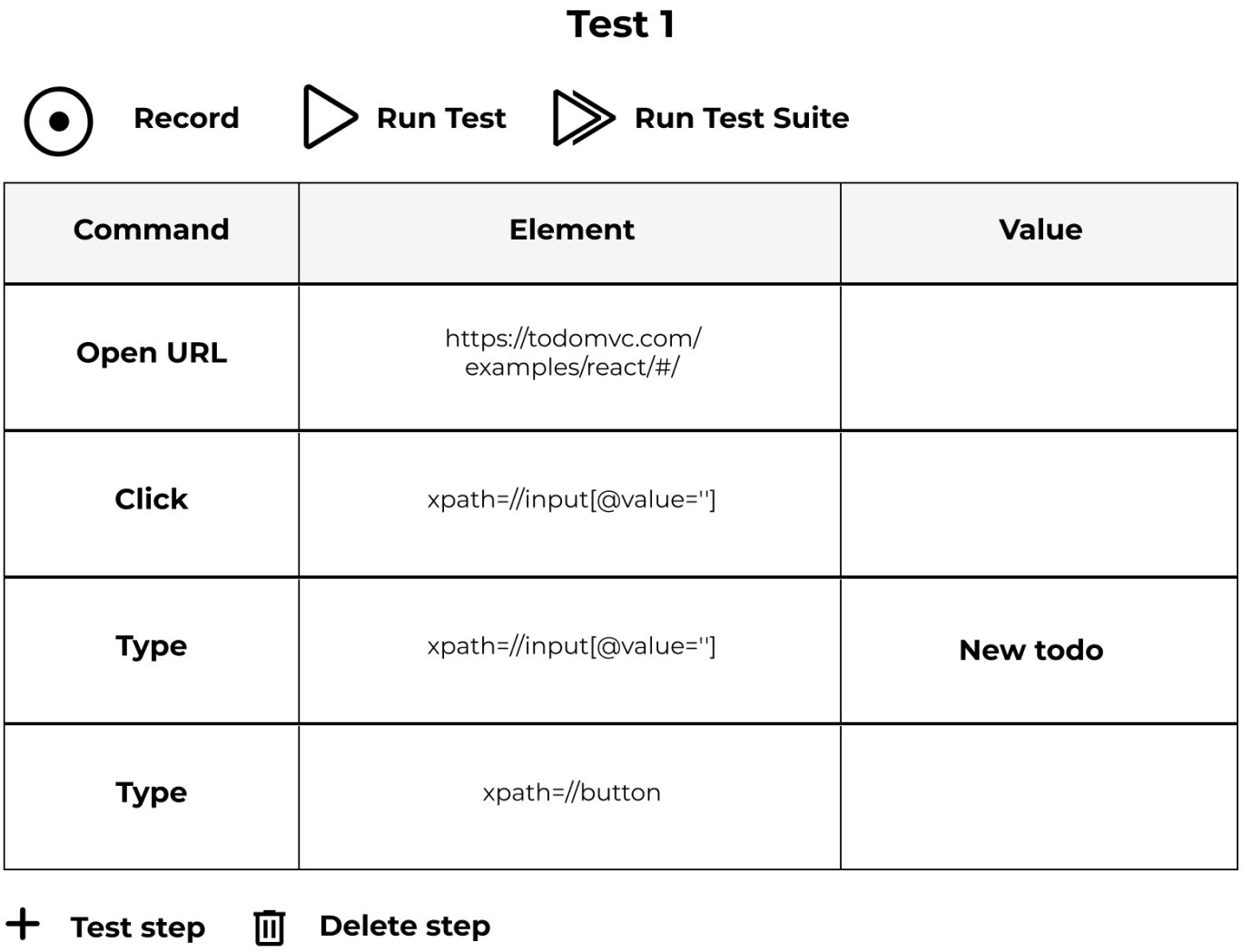


Рисунок 4.2 – Кроки тестового сценарію

Command	Element	Value	Status
Open URL	https://todomvc.com/examples/react/#/		✓
Click	xpath=//input[@value='']		✓
Type	xpath=//input[@value='']	**New todo**	✓
Type	xpath=//button		✗

Logs

[info] Execute action: Open URL - https://todomvc.com/examples/react/#/

[info] Execute action: Click - xpath=//input[@value='']

[info] Execute action: Type - xpath=//input[@value=''] - value='New todo'

[info] Execute action: Click - xpath=//input[@value='']

[error] Execute action: Type - xpath=//button

[error] Cannot find element: xpath=//button

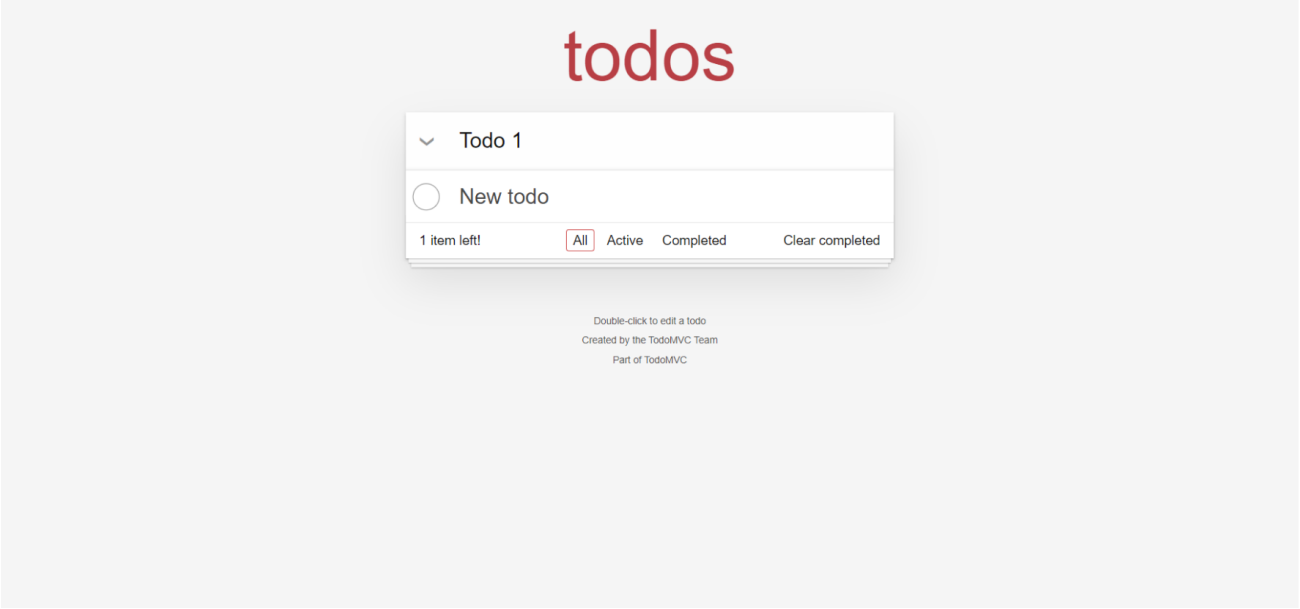


Рисунок 4.3 – Звіт виконання тесту

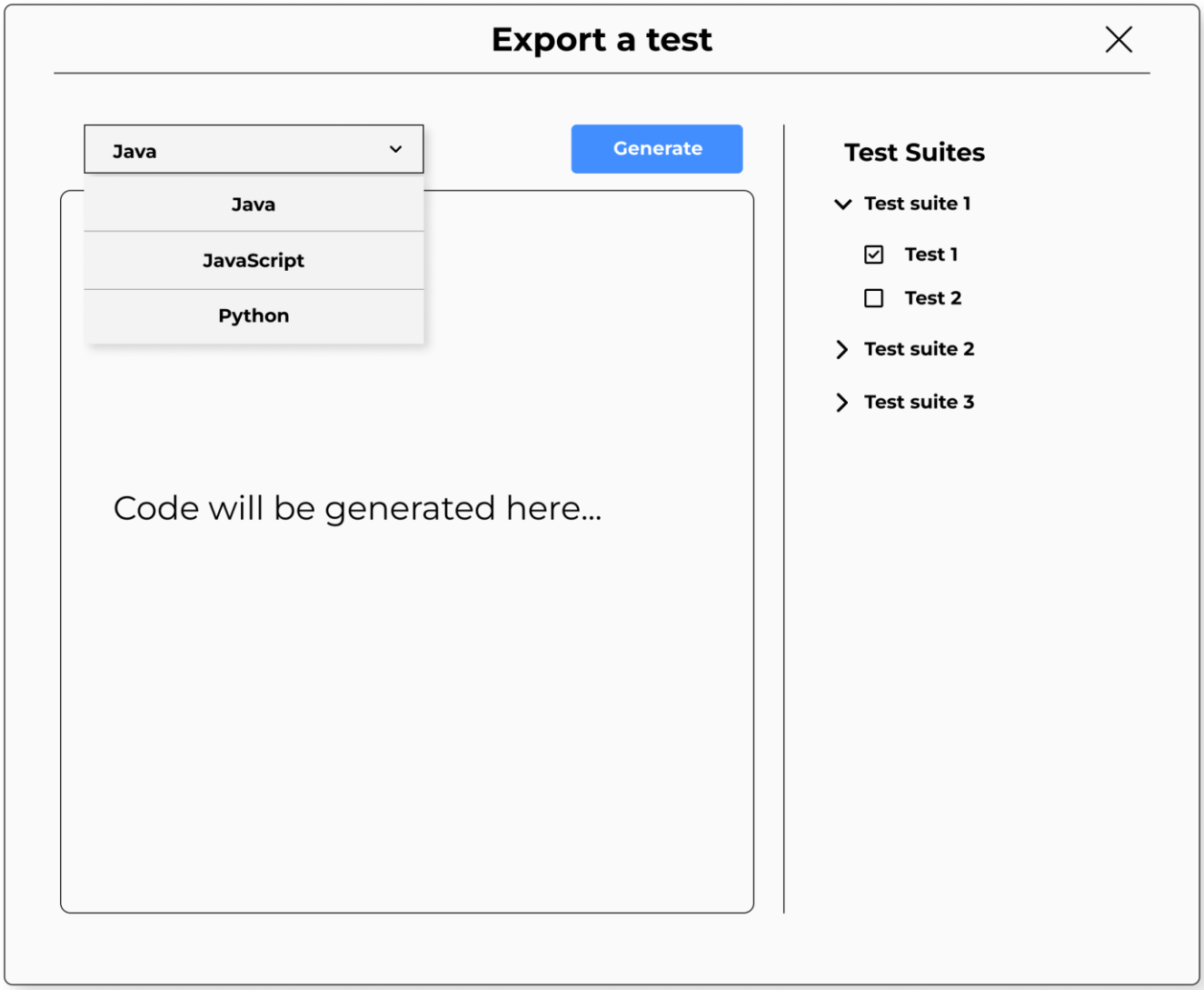


Рисунок 4.4 – Генерація коду для тестів

4.1.2. Для користувача:

4.1.2.1. Можливість створити тестовий набір зазначивши його назву

4.1.2.1.1. Буде створено пустий тестовий набір із заданою назвою

4.1.2.2. Можливість додати тест в тестовий набір зазначивши його назву

4.1.2.2.1. Буде додано пустий тест із заданою назвою

4.1.2.3. Можливість записати тестовий сценарій виконуючи послідовно набір із доступних операцій на сторінці браузера:

4.1.2.3.1. Відкриття сторінки браузера по URL

4.1.2.3.2. Клік миші

4.1.2.3.3. Введення тексту

4.1.2.3.4. Вибір опції у селекторі

4.1.2.3.5. Натискання клавіші

4.1.2.3.6. Перевірка текстового значення елемента

4.1.2.3.7. Перевірка заголовку елемента

4.1.2.3.8. Перевірка видимості елемента

4.1.2.3.9. Перевірка можливості редагування елемента

4.1.2.3.10. Перевірка фокусування елемента

4.1.2.3.11. Перевірка елемента на належність класу через атрибут клас

4.1.2.4. Можливість редагувати створений тестовий сценарій

4.1.2.4.1. Можливість видалити тестовий крок

4.1.2.4.2. Можливість додати новий тестовий крок

4.1.2.4.3. Можливість змінити порядок тестових кроків

4.1.2.4.4. Можливість редагувати тестовий крок

4.1.2.4.4.1. Зміна типу операції

4.1.2.4.4.2. Переобрання елементу для даного кроку

4.1.2.5. Забезпечити можливість покрокового виконання тестового сценарію

4.1.2.5.1. Тестові кроки починають послідовно виконуватися

4.1.2.6. Можливість переглянути звіт виконання тестового сценарію

4.1.2.6.1. Звіт включає логування виконаних кроків та опис виявлених помилок

4.1.2.6.1.1. Помилки можуть бути наступними:

4.1.2.6.1.1.1. Не вдалося відкрити сторінку

4.1.2.6.1.1.2. Не вдалося знайти елемент на сторінці

4.1.2.6.1.1.3. Не виконана умова (текст не співпадає, елемент не видимий, елемент недоступний до редагування, кнопка не активна)

4.1.2.6.2. При виникненні помилки при виконанні тестового сценарію, звіт має містити скріншот сторінки браузера

4.1.2.7. Можливість запустити набір тестів на виконання

4.1.2.7.1. Послідовно запускає тести тестового набору

4.1.2.8. Можливість перегляду статистики виконання тестів

4.1.2.8.1. Статистика включає кількість запусків тестів, кількість успішних та кількість невдалих спроб

4.1.2.8.2. Можливість перегляду логів виконання тестів для кожного запуску

4.1.2.9. Можливість експортувати набір тестів у вигляді файлу з розширенням JSON

4.1.2.10. Можливість імпортувати набір тестів за допомогою файлу з розширенням JSON

4.1.2.11. Можливість згенерувати код для виконання тестових сценаріїв на мові JavaScript використовуючи можливості великої мовної моделі

4.1.3. Для адміністратора системи (якщо він передбачений):

– Не передбачений в системі

4.1.4. Додаткові вимоги:

4.1.4.1. Забезпечити можливість установки плагіна в браузері Google Chrome наступним чином:

4.1.4.1.1. Користувач переходить у Chrome Web Store

4.1.4.1.2. У пошуку вводить його назву Test Automation

4.1.4.1.3. У списку знайдених плагінів обирає коректний

4.1.4.1.4. Далі натискає завантажити плагін, після чого він автоматично додається у браузер Google Chrome

4.1.4.2. Плагін сумісний із браузером Google Chrome

4.2. Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

Програмне забезпечення має задовільняти наступним вимогам:

– не повинно порушувати безпеку браузера або комп'ютера користувача;
– має працювати стабільно, не спричиняючи падінь браузера;
– не повинно сповільнювати роботу браузера або завантаження веб-сторінок;
– має використовувати тільки надійні API через захищений протокол;
– повинно працювати коректно з версіями браузера після 124.0.6367.60.

4.3. Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1. Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2. Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4. Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати у браузері Google Chrome.

Підключення до мережі Інтернет повинно мати швидкість від 2 мегабіт.

4.5. Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати в середовищі браузера Google Chrome 124.0.6367.60.

4.5.1. Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2. Вимоги до вихідних даних

Вимоги до вхідних даних не висуваються.

4.5.3. Вимоги до мови розробки

Розробку виконати на мовах програмування JavaScript (TypeScript)

4.5.4. Вимоги до середовища розробки

Розробку виконати на за допомогою середовища розробки Web Storm.

4.5.5. Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді GitHub репозиторію.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7. Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8. Спеціальні вимоги

Версія плагіну, яка готова до інтеграції в Google Chrome – 1.0.0 і більше.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1. Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– технічне завдання;

– пояснювальна записка;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема структурна компонент;

– креслення вигляду екранних форм.

5.2. Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою роботи	15.03.2024	
2.	Розробка технічного завдання	14.04.2024	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	25.04.2024	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04.2024	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	15.05.2024	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	20.05.2024	Тести, результати тестування
7.	Розробка матеріалів текстової частини роботи	30.05.2024	Пояснювальна записка
8.	Розробка матеріалів графічної частини роботи	03.06.2024	Графічний матеріал проекту
9.	Оформлення технічної документації роботи	05.06.2024	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: плагін браузера Google Chrome для автоматизації тестування веб застосунків

КПІ.ІП-0120.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
IT	–	Інформаційні технології
LLM	–	Large language model, велика мовна модель
NLP	–	Natural language processing, обробка природної мови
ШІ	–	Штучний інтелект
UI	–	User interface, інтерфейс користувача
ПЗ	–	Програмне забезпечення
DOM	–	Document Object Model, об'єктна модель документа
AI	–	Artificial intelligence, штучний інтелект

ВСТУП

У сучасному технологічному світі якість програмного забезпечення грає важливу роль для підтримки конкурентоспроможності, сприяння довіри клієнтів та підтримки гарної репутації IT компанії. Зараз веб-застосунки стають неодмінною складовою повсякденного життя і швидкість впровадження нових релізів напряму впливає на їх успіх та розвиток, тому виникає велика потреба у вдосконаленні методів тестування для забезпечення надійності та ефективності цих застосунків.

Актуальність розробки пояснюється зростанням потреби автоматизації тестування веб-застосунків через швидкий темп розвитку інтернет-технологій та зростаючу кількість веб застосунків. Одним із типів тестування, який вимагає багато часу та людських ресурсів є мануальне тестування. Крім того, під час виходу нової версії продукту потрібно переконатися, що нові зміни не порушили існуючу функціональність. Це виконується в рамках регресійного тестування, де існуючі тести проходяться знову та знову. Тому виникає необхідність автоматизувати існуючі сценарії ручного тестування користувацького інтерфейсу застосунку для значного зменшення витрат часу та ресурсів.

На даний час складність автоматизації тестових сценаріїв полягає в тому, що вона вимагає широкого спектру знань. Потрібно розуміти мови програмування, тестові бібліотеки та фреймворки, а також технології веб-розробки. Тому спеціаліст, який володіє такими знаннями та навичками коштує досить дорого.

Сприяючи цьому, дана дипломна робота присвячена розробці плагіну для браузера Google Chrome, який спрощує створення автоматизованих сценаріїв для тестування веб-застосунків. Однак, особливість цього плагіну полягає у використанні великої мовної моделі ChatGPT, яка останнім часом вийшла на новий рівень розвитку та впроваджуються у широкому спектрі галузей. Дана модель буде використовуватися для генерації програмного коду для автоматизації тестових сценаріїв мовою програмування.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз предметної області

Google Chrome плагіни (розширення) – це невеликі програмні продукти, які кастомізують користувацький досвід під час використання веб-застосунків. Вони дозволяють користувачам налаштовувати функціонал та поведінку Chrome згідно зі своїми потребами або вподобаннями. Вони створюються на базі основних, широко відомих веб-технологій, таких як HTML, JavaScript та CSS [1].

Головні характеристики плагінів [1]:

– виконують конкретну задачу або функцію;

– можуть мати мінімалістичний інтерфейс або повноцінну сторінку;

– розповсюджуються через Chrome Web Store та встановлюються як архівний файл;

– не залежать від контенту з Інтернету та не потребують постійного підключення до мережі.

Chrome плагіни надають можливість змінювати роботу браузера, використовуючи різноманітні API [2]:

– змінювати сторінки і налаштування браузера Chrome;

– розширювати інструменти розробника;

– відображати сповіщення;

– керувати історією браузера;

– керувати вкладками та вікнами;

– керувати завантаженнями.

Ці можливості дозволяють розробникам створювати розширення, які відповідають конкретним потребам користувачів та забезпечують персоналізований та розширений функціонал браузера Chrome.

Велика мовна модель (LLM) — це статистична мовна модель, навчена на величезній кількості даних, яку можна використовувати для створення та

перекладу тексту та іншого вмісту, а також виконання інших завдань обробки природної мови (NLP) [3].

Великі мовні моделі також відомі як нейромережі, імітують структуру людського мозку із шарованими вузлами, схожими на нейрони. Вони також мають велику кількість параметрів, які схожі на пам'ять, яку модель збирає під час навчання. Ці параметри можна розглядати як сховище знань моделі. Як і людський мозок, великі мовні моделі повинні бути передньо навчені, а потім докладно налаштовані, щоб вони могли розв'язувати проблеми класифікації тексту, відповіді на питання, резюмування документів та генерації тексту. Їхні можливості вирішення завдань можуть бути застосовані в галузях, таких як охорона здоров'я, фінанси та розваги, де великі мовні моделі обслуговують різні завдання обробки природної мови, такі як переклад, чат-боти, ШІ-асистенти і так далі [4].

Обробка природної мови (NLP) відноситься до галузі штучного інтелекту, яка займається наданням комп’ютерам здатності розуміти текст і вимовлені слова майже так само, як це можуть зробити люди [5].

Обробка природної мови (NLP) об'єднує обчислювальну лінгвістику - моделювання людської мови на основі правил, зі статистичними, машинними та глибинними моделями навчання. Ці технології дозволяють комп'ютерам обробляти людську мову у вигляді тексту чи голосових даних та 'розуміти' її повне значення, включаючи наміри та настрій написаного.

NLP вирішує завдання, пов'язані з складнощами людської мови, такі як наступні [5]:

– розпізнавання мови: перетворення голосових даних в текст;

– визначення частин мови: визначення частин мови для слів у тексті на основі їхнього контексту;

– роз'яснення значень слів: вибір значення слова з кількох можливих в контексті;

– визначення іменованих сутностей: виділення слів чи фраз як корисних сутностей;

– аналіз настрою: Спроба витягти суб'єктивні якості (емоції, настрій, сарказм) з тексту;

– генерація природної мови: задача створення структурованої інформації в текстовій формі.

Тестування – це важливий етап в розробці програмного забезпечення, під час якого важливі частини ПЗ перевіряються на коректність, якість та швидкодію. Тестування програмного забезпечення має різні типи [6]:

– модульні тести: низькорівневі тести, які перевіряють окремі методи або функції класів, компонентів чи модулів;

– інтеграційні тести: перевіряють взаємодію різних модулів чи сервісів у межах додатка;

– функціональні тести: спрямовані на перевірку бізнес-вимог додатка;

– приймальні тести: формальні тести, які перевіряють, чи система відповідає бізнес-вимогам;

– тести продуктивності: оцінюють, як система працює при конкретному робочому навантаженні.

Тестування ПЗ може бути або ручним, або автоматизованим процесом.

Ручне тестування — це тип тестування програмного забезпечення, у якому тестові приклади виконуються тестувальником вручну без використання будь-яких автоматизованих інструментів. Метою тестування вручну є виявлення помилок, проблем і дефектів програмного забезпечення. Тестування програмного забезпечення вручну є найпримітивнішою технікою з усіх типів тестування, яка допомагає знайти критичні помилки в програмному додатку [7].

Тестування користувацького інтерфейсу, або UI-тестування, є важливим етапом у розробці та підтримці веб-сайтів. Це включає оцінку графічного інтерфейсу та користувацького досвіду для того, щоб переконатися, що він відповідає заданому дизайну та функціональності. UI тестування спрямоване

на перевірку того, що елементи інтерфейсу користувача, такі як кнопки, меню, форми та навігація, працюють так, як очікується, і забезпечують коректну взаємодію з користувачами [8].

Основна мета UI-тестування - виявлення та усунення будь-яких проблем, пов'язаних з візуальним представленням та взаємодією з користувачем на веб-сайті. Тести відтворюють взаємодію користувача з інтерфейсом, щоб переконатися, що веб-сайт поводиться так, як очікується, забезпечуючи безперебійний та задовільний досвід користувача [8].

На сучасному етапі розвитку IT-технологій, процес тестування веб-застосунків здійснюється переважно вручну або за допомогою традиційних автоматизованих тестових інструментів, які потребують знання мови програмування та тестових бібліотек або фреймворків. Тому створення автоматизованих тестів – це процес який потребує достатньої кваліфікації та навичок.

Для покращення процесу тестування вживаються заходи щодо покращення процесів аналізу предметної області, створення тестових планів та менеджменту процесу тестування [9].

Недоліками поточних підходів до тестування веб-застосунків є складність процесу виконання тестів через трудомісткість і значні витрати часу у випадку ручного тестування та високий поріг входу та високий рівень кваліфікації у випадку написання автоматизованих тестів.

Для подолання цих недоліків розглянемо методи спрощення процесу створення автоматизованих тестових сценаріїв. Одним із методів спрощення – є запис дій тестувальника на сторінці веб-застосунку для подальшого їх автоматизованого відтворення. Інший метод включає використання великої мовної моделі для генерації коду автоматизації для записаного тестового сценарію на відповідній мові програмування. Дані методи якраз і будуть використані в даній роботі.

Отже, в результаті даного дипломного проєкту буде створенно Google Chrome плагіну, який спростить автоматизацію тестування користувацького

інтерфейсу різних веб-застосунків використовуючи і дозволить генерувати код тестового сценарію використовуючи можливості LLM.

1.2. Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації плагіну браузера Google Chrome для автоматизації тестування веб-застосунків. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.2.1. Аналіз відомих програмних продуктів

Розглянемо готові програмні продукти, які відносяться до даної предметної області та фокусуються на автоматизації тестування користувацького інтерфейсу веб-застосунків. Хоча існуючих інструментів не багато, але все-таки деякі можна розглянути.

UI.Vision RPA, плагін для Google Chrome, надає зручні інструменти для автоматизації веб-процесів. За допомогою цього плагіну користувачі можуть записувати та відтворювати макроси, автоматизуючи повторювані дії на веб-сторінках. Функціонал включає взаємодію з веб-елементами, такими як натискання кнопок, заповнення форм, а також використання змінних та умов для створення більш складних сценаріїв. Плагін дозволяє використовувати цикли та ітерації для повторення частин скрипту та впровадження JavaScript-коду для розширеної автоматизації та обробки даних. Змінні можуть використовуватися для зберігання та передачі даних між різними етапами макросу. Крім того, плагін дозволяє інтегрувати автоматизацію з іншими сервісами, що допомагає створювати комплексні рішення для вирішення рутинних завдань у веб-браузері [10].

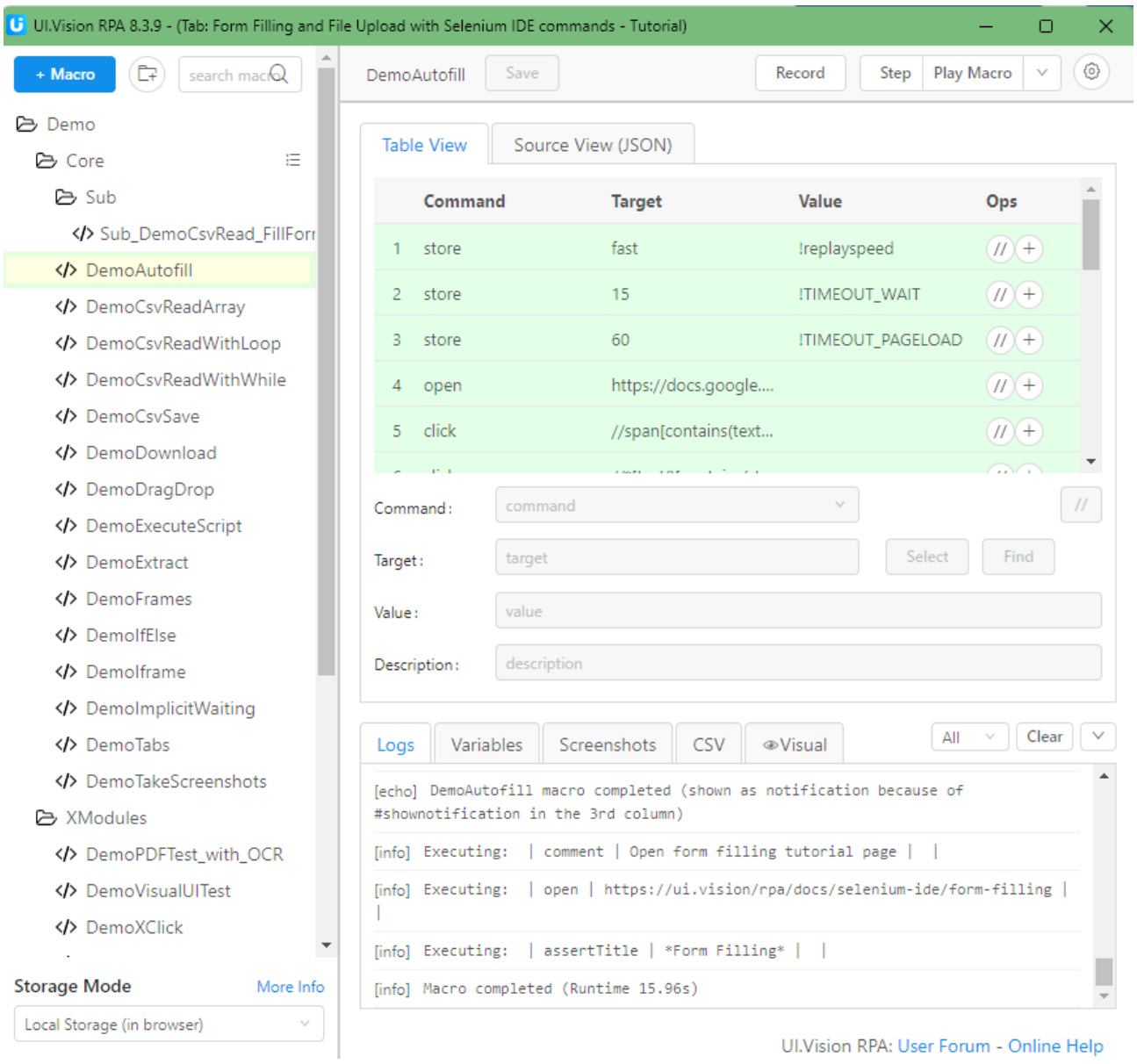


Рисунок 1.1 – Вигляд плагіну UI.Vision RPA

Selenium IDE представляє собою інтегроване середовище розробки, створене для автоматизації тестування веб-додатків і сумісне з веб-браузерами, такими як Google Chrome та Mozilla Firefox. Плагін надає зручний інтерфейс для запису, редагування та відтворення тестових сценаріїв, спрощуючи процес автоматизації. Основні можливості включають підтримку різних мов програмування, таких як Java, C#, Python, Ruby, що дозволяє створювати більш складні та гнучкі тестові скрипти. Плагін дозволяє редагувати та налаштовувати тестові сценарії в інтерактивному режимі, уточнюючи та оптимізуючи процес автоматизації. Також надає підтримку

асинхронних операцій, що робить Selenium IDE більш гнучким для автоматизації сучасних веб-додатків. Використання змінних та зберігання даних дозволяє передавати інформацію між різними етапами тестового сценарію [11]. На рисунку 1.2 можна побачити як виглядає плагін Selenium IDE.

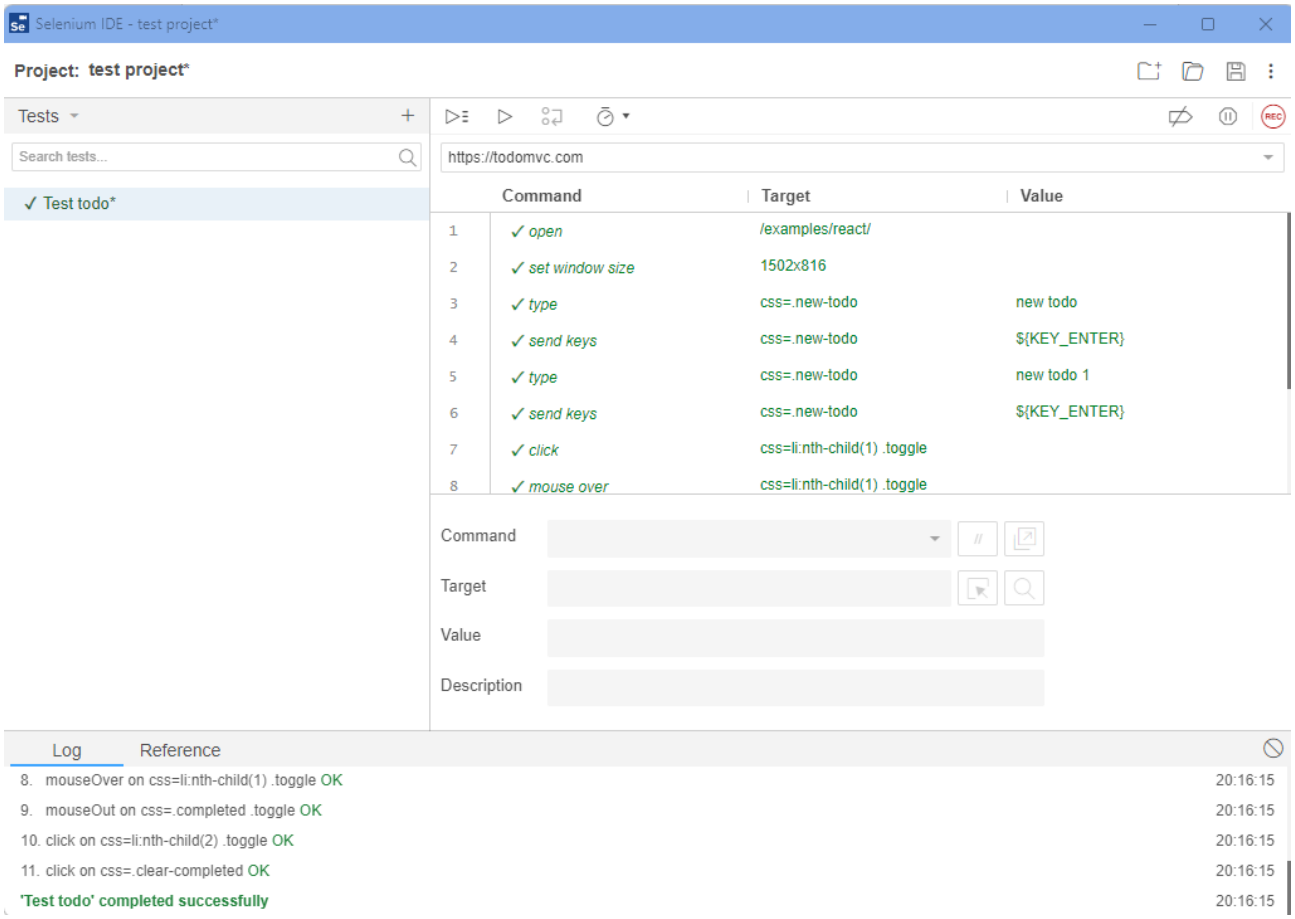


Рисунок 1.2 – Вигляд плагіну Selenium IDE

Для порівняння дипломної роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал / характеристика	Дипломний проєкт	UI.Vision RPA	Selenium IDE	Пояснення
Можливість запису автоматизованого тестового сценарію	Надається	Надається	Надається	Користувач може записати тест у вигляді послідовності дій
Можливість виконання автоматизованих тестових сценаріїв	Надається	Надається	Надається	Записаний авто тест можна запустити на виконання
Можливість редагування	Надається	Надається	Надається	Кроки в тесті можна редагувати
Можливість генерувати код для тестових сценаріїв	Надається	Не надається	Надається	Чи надається можливість експортувати створений тест у вигляді готового скрипту на мові програмування

Продовження таблиці 1.1

Функціонал / характеристика	Дипломний проєкт	UI.Vision RPA	Selenium IDE	Пояснення
Складність користування	Простий та інтуїтивний	Високий поріг входу	Достатньо простий у користуванні	Наскільки складно використовувати та скільки часу потрібно витратити на вивчення функціональності
Наявність AI можливостей	Використання AI для генерації коду	Використовується AI для можливості «бачення» інтерфейсу користувача	Не надані	Чи використовується функціональність пов’язана з AI
Наявність підтримки	Підтримується	Підтримується	Не підтримується	Чи виходять оновлення даних плагінів

1.2.2. Аналіз відомих алгоритмічних та технічних рішень

Стосовно алгоритмічних рішень дана розробка не передбачає застосування особливих обчислювальних алгоритів.

В контексті технічних рішень, одним із ключових елементів є ідентифікація та знаходження елементів на веб-сторінці, що є важливим для автоматизації тестування. Для виконання даних задач можна використовувати вбудовані можливості мови JavaScript або використовувати веб драйвер.

Використання вбудованих можливостей мови JavaScript для взаємодії зі сторінкою браузера має деякі вагомі переваги. JavaScript є стандартною мовою для розробки веб-сторінок, що робить його легким у використанні для взаємодії з елементами сторінки. Він може безпосередньо маніпулювати DOM (Document Object Model), що дозволяє взаємодіяти з елементами сторінки без необхідності використання додаткових інструментів чи фреймворків. Завдяки прямому доступу до DOM та вбудованим функціям браузера, JavaScript може працювати дуже ефективно, забезпечуючи швидке виконання коду. Крім того, JavaScript підтримується у всіх сучасних браузерах і є стандартом для розробки веб-додатків, що робить його універсальним рішенням для взаємодії зі сторінками.

WebDriver є інтерфейсом для віддаленого управління, який надає можливість дослідження та керування агентами користувача. Він надає протокол передачі даних, що є нейтральним до платформи та мови програмування, як засіб віддаленого управління браузером. WebDriver надає набір інтерфейсів для виявлення та маніпулювання елементами DOM в веб-сторінках та управління поведінкою агента користувача [12].

Одним із найбільш поширених веб драйверів є Selenium. Selenium WebDriver — це інтерфейс програмування, який можна використовувати для створення та виконання тестів. Цей інструмент використовується для автоматизації тестування веб-застосунків. Він може працювати на різних платформах і з різними мовами програмування [13].

Тепер виберемо з двох варіантів кращий. В даній роботі доцільно буде використати вбудовані можливості мови JavaScript, бо це дає перевагу у швидкості роботи, простоті реалізації і також даний підхід не потребує спеціального середовища роботи та може бути інтегрований у будь-яку веб-сторінку браузера, що є важливим для гнучкості та простоти використання плагіну.

Для взаємодії із веб-застосунками працюючими у браузері будемо використовувати Chrome Extension API. За допомогою цього API можна

взаємодіяти з браузером, змінювати його функціональність та надавати користувачам нові можливості. Наступні можливості Chrome Extension API знадобляться при розробці браузерного плагіну [14]:

– можливість вбудування власного коду в веб-сторінки, які відкриті в браузері. Це дозволяє взаємодіяти з DOM сторінки та змінювати її вміст;

– можливість створення фонових процесів, які можуть виконувати задачі, не пов'язані з безпосередньо відкритими сторінками;

– додавання додатковиї елементів у панелі інструментів браузера або у контекстне меню;

– збереження даних в локальному сховищі браузера;

– управління відкритими вкладками браузера: відкриття, закриття, зміна URL, отримання інформації про активну вкладку тощо.

Для генерації програмного коду можна використовувати сучасні AI системи на основі LLM. AI-інструменти для генерації коду, такі як ті, що надаються OpenAI, можуть автоматично створювати фрагменти коду або навіть цілі програми на основі текстових описів. Це позбавляє необхідності створювати алгоритми генерації коду вручну та допомагає знизити кількість помилок у коді. AI може генерувати код, який відповідає найкращим практикам програмування, та пропонує оптимальні рішення на основі аналізу великого обсягу даних. Це забезпечує високу якість коду і спрощує процес його підтримки та розвитку. Одним з ключових переваг AI-генерації коду є її здатність швидко адаптуватися до змін в вимогах та середовищі розробки. Наприклад, якщо змінюються вимоги до програмного забезпечення, AI-інструменти можуть оперативно згенерувати новий код, який відповідає новим умовам, зменшуючи час на його розробку та тестування [15].

Можливість генерації коду без використання LLM також існує. Існують різні алгоритми, які можуть бути використані для автоматизації написання коду, такі як шаблонні генератори, що використовують заздалегідь визначені шаблони коду, або системи з правилами, що генерують код на основі заданих правил та умов. Наприклад, для автоматизації тестових сценаріїв можна

використовувати комбінацію регулярних виразів та скриптів на JavaScript, що генерують необхідний код на основі вхідних даних. Однак, ці підходи мають свої обмеження. Вони часто потребують значних зусиль для налаштування та підтримки, а також не мають гнучкості у врахуванні складних контекстів і варіантів використання. Також для підтримки інших мов програмування потрібно буде реалізовувати новий алгоритм, що вимагає значних зусиль і тому не є добре масштабованим рішенням.

Стосовно використання LLM є два підходи: розгорнути LLM локально або використовувати існуючих провайдерів LLM через API. Оскільки LLM має велику модель для її локального розгортання потрібен значний обсяг пам’яті. Також для забезпечення належної швидкодії LLM потрібна значна кількість обчислювальних ресурсів. Використання існуючих сервіс провайдерів через API дозволить уникнути складності локального розгортання LLM. І хоч такі послуги платні, оплата залежить від використання та від розміру запитів, тому для розробки такий підхід краще і не вимагає багатьох витрат [16].

1.3. Опис бізнес-процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 1.3).

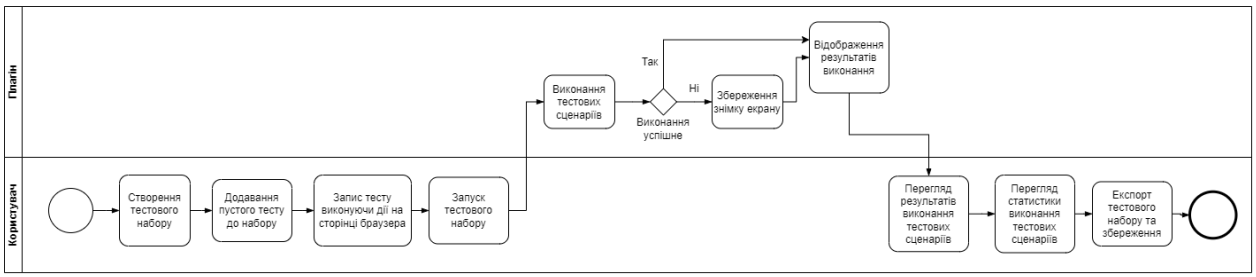


Рисунок 1.3 – BPMN модель бізнес процесу програмного забезпечення

Далі буде подано опис послідовності взаємодії користувача з плагіном для тестування веб-застосунків, спрямованого на створення та відтворення тестових сценаріїв для веб-застосунків:

1.3.1. Створення тестового набору:

1.3.1.1. Користувач відкриває браузер Google Chrome та активує плагін для тестування веб-застосунків.

1.3.1.2. Користувач обирає опцію "Створити тестовий набір".

1.3.2. Додавання тесту до набору:

1.3.2.1. Користувач обирає створений тестовий набір та обирає опцію "Додати тестовий сценарій" та вказує його назву.

1.3.2.2. Після цього створюється пустий новий тестовий сценарій.

1.3.3. Запис тестового сценарію виконуючи дії на сторінці браузера:

1.3.3.1. Користувач у створеному тесті обирає опцію "Записати тест" та переходить на сторінку браузера виконує послідовні дії, які потрібні для виконання тесту.

1.3.3.2. Плагін автоматично записує виконані дії в поточний тест, генеруючи тестовий сценарій.

1.3.4. Редагування тестового сценарію

1.3.4.1. Користувач видаляє помилково створені кроки сценарію.

1.3.4.2. Користувач обирає кроки сценарію, які потрібно змінити та редагує їх.

1.3.4.3. Також користувач може доповнити тестовий сценарій новими кроками натискаючи на кнопку "Додати тестовий крок".

1.3.5. Запуск тестового набору на виконання:

1.3.5.1. Користувач обирає опцію "Запустити тестовий набір" в поточному тестовому наборі.

1.3.5.2. Плагін послідовно запускає тестові сценарії в наборі на виконання та відтворює їх відповідно до записаних кроків.

1.3.5.3. По мірі виконання тестів результати автоматично відображаються у полі результатів. Також відображаються логи виконання тесту.

1.3.5.4. У випадку неуспішного виконання тесту автоматично робиться знімок екрану та зберігається в звіті виконання.

1.3.6. Перегляд статистики виконання тестів:

1.3.6.1. Користувач обирає тест на обирає опцію "Переглянути статистику", яка включає час виконання та результати.

1.3.7. Експорт тестового набору:

1.3.7.1. Користувач обирає тестовий набір на обирає опцію "Експортувати у вигляді файлу".

1.3.7.2. Після цього генерується файл, який включає вміст тестів в наборі та завантажується користувачу.

1.4. Постановка задачі

Розробити плагін браузера Google Chrome, який надаватиме можливість створення, виконання та підтримки тестових сценаріїв для автоматизації тестування веб-застосунків. Плагін повинен надавати можливість запису дій користувача на сторінці браузера для формування тестового сценарію. При виконанні тестового сценарію плагін повинен відтворити записані дії. Також має виконуватися збір статистики і звітів виконання тестових сценаріїв.

Метою розробки є спрощення можливості створення, підтримки та виконання тестових сценаріїв за рахунок створення Google Chrome плагіну.

В результаті розробки даного плагіну мають бути реалізовані наступні функціональні задачі:

– створення тестових сценаріїв;
– виконання тестових сценаріїв;
– збір статистики виконання тестових сценаріїв;
– генерація коду для створеного тестового сценарію;
– підтримка тестових сценаріїв, що включає можливість редагування, а також перенесення тестових наборів шляхом експортування та імпортування.

Висновки до розділу

В даному розділі було перш за все описано предметну область та основні поняття, якими вона оперує. Було проаналізовано поточний стан речей в області автоматизації тестування користувацького інтерфейсу веб-застосунків та було знайдено, що дана область потребує покращення.

Потім було розглянуто два існуючих продукти, які в деякій мірі покривають поставлену задачу та допомагають автоматизувати процес тестування інтерфейсу користувача. Було також проведено їх порівняльний аналіз.

Далі було проаналізовано існуючі технології та технічні рішення, які дозволяють вирішувати поставлену задачу. Було обрано мову програмування JavaScript, за допомогою якої є можлива інтеграція з браузерним середовищем. Було визначено використовувати можливості Chrome Extension API, що є основою реалізації будь-якого Google Chrome плагіну. Також було обрано використання вбудованих можливостей мови програмування JavaScript для взаємодії із веб-сторінками, бо веб драйвер вимагає спеціального середовища, де забезпечуються умови для його роботи.

Потім було визначено і описано бізнес процеси та побудовано BPMN модель. Також було детально описано в таблицях усі події, що відбуваються в рамках бізнес процесу, який покривається у даній розробці.

В кінці на основі проведеного аналізу було визначено функціональні задачі та сформульовано постановку задачі.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є спрощення процесу створення, підтримки та виконання автоматизованих тестових сценаріїв за рахунок створення плагіну Google Chrome, більше функцій можна побачити на графічному матеріалі 1.

В таблицях 2.1 - 2.11 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Створення тестового набору
Use case ID	UC-01
Goals	Створення нового тестового набору для групування тестів
Actors	Користувач
Trigger	Користувач бажає створити новий тестовий набір
Pre-conditions	-
Flow of Events	Користувач натискає на кнопку створення нового тестового набору, після чого відкривається спливаюче вікно із полем для введення назви тестового набору. Далі користувач підтверджує створення
Extension	-
Post-Condition	Створення нового тестового набору виконано успішно

Таблиця 2.2 - Варіант використання UC-2

Use case name	Додавання тестового сценарію в тестовий набір
Use case ID	UC-02
Goals	Додавання тестового сценарію до існуючого тестового набору
Actors	Користувач
Trigger	Користувач бажає додати тестовий сценарій до тестового набору

Продовження таблиці 2.2

Pre-conditions	Наявність створеного тестового набору
Flow of Events	Користувач вибирає тестовий набір та натискає кнопку додати тестовий сценарій, після чого відкривається спливаюче вікно із полем для введення назви сценарію. Далі користувач підтверджує створення
Extension	-
Post-Condition	Тестовий сценарій додано до тестового набору

Таблиця 2.3 - Варіант використання UC-3

Use case name	Запис тестового сценарію на сторінці браузера
Use case ID	UC-03
Goals	Створення тестового сценарію на основі виконаних дій користувача на сторінці браузера
Actors	Користувач
Trigger	Користувач бажає записати тестовий сценарій
Pre-conditions	Створено новий тестовий сценарій
Flow of Events	Користувач натискає кнопку почати запис тестового сценарію, переходить на сторінку в браузері та починає послідовно виконувати дії необхідні для сценарію і вони автоматично додаються у тест як послідовні кроки
Extension	-
Post-Condition	В тестовому сценарії додано кроки на основі виконаних дій на сторінці браузера

Таблиця 2.4 - Варіант використання UC-4

Use case name	Редагування створеного тестового сценарію
Use case ID	UC-04
Goals	Внесення змін до існуючого тестового сценарію, виправлення помилок
Actors	Користувач
Trigger	Користувач бажає внести зміни до тестового сценарію

Продовження таблиці 2.4

Pre-conditions	Наявність створеного тестового сценарію
Flow of Events	Користувач відкриває існуючий тестовий сценарій, обирає той крок, який хоче змінити, та редагує його. Також користувач може додати нові кроки в сценарій або видалити існуючі.
Extension	-
Post-Condition	Зміни успішно внесено до тестового сценарію

Таблиця 2.5 - Варіант використання UC-5

Use case name	Виконання тестового набору
Use case ID	UC-05
Goals	Запуск тестового набору на виконання
Actors	Користувач
Trigger	Користувач бажає виконати тестовий набір
Pre-conditions	Наявність створеного тестового набору
Flow of Events	Користувач відкриває існуючий тестовий набір та натискає кнопку почати виконання. Після цього всі тестові сценарії починають послідовно виконуватися. Для кожного тестового сценарію його кроки автоматично виконуються у веб-сторінці браузера.
Extension	-
Post-Condition	Тестовий набір виконано та збережено результати виконання кожного тестового сценарію

Таблиця 2.6 - Варіант використання UC-6

Use case name	Виконання тестового сценарію
Use case ID	UC-06
Goals	Запуск тестового сценарію на виконання
Actors	Користувач
Trigger	Користувач бажає виконати тестовий сценарій

Продовження таблиці 2.6

Pre-conditions	Наявність створеного тестового сценарію
Flow of Events	Користувач обирає тестовий сценарій та запускає його на виконання, після чого усі його кроки починають послідовно виконуватися у веб-сторінці браузера.
Extension	-
Post-Condition	Тестовий сценарій виконано та збережено результати його виконання

Таблиця 2.7 - Варіант використання UC-7

Use case name	Знімок екрану
Use case ID	UC-07
Goals	Запис знімку екрана при неуспішному виконанні тестового сценарію
Actors	Плагін
Trigger	Виконання тестового сценарію завершилось невдачею
Pre-conditions	Неуспішне виконання тестового сценарію
Flow of Events	Плагін робить знімок екрану після невдалого виконання тестового сценарію та зберігає його у звіті виконання, для полегшення розуміння причини падіння тесту.
Extension	-
Post-Condition	Знімок екрану успішно збережено у звіті виконання тестового сценарію

Таблиця 2.8 - Варіант використання UC-8

Use case name	Перегляд статистики виконання тестових сценаріїв
Use case ID	UC-08
Goals	Перегляд статистики виконання тестових сценаріїв, що включає в себе час виконання та кількість успішних та неуспішних запусків
Actors	Користувач
Trigger	Користувач бажає переглянути статистику виконання тестових сценаріїв

Продовження таблиці 2.8

Pre-conditions	Було виконано принаймні один тестовий сценарій
Flow of Events	Користувач відкриває існуючий тестовий сценарій і натискає кнопку переглянути статистику запусків, після чого вікривається вікно із інформацією про запуски тестового сценарію, де вказується їх успішність або неуспішність та час виконання
Extension	-
Post-Condition	Статистику запусків тестового сценарію відображено

Таблиця 2.9 - Варіант використання UC-9

Use case name	Експорт тестового набору у вигляді JSON файлу
Use case ID	UC-09
Goals	Збереження тестового набору у вигляді JSON файлу для подальшого використання або обміну
Actors	Користувач
Trigger	Користувач бажає експортувати тестовий набір
Pre-conditions	Наявність створеного тестового набору
Flow of Events	Користувач обирає існуючий тестовий набір та натискає кнопку зберегти його у вигляді JSON файлу. Після цього формується файл та завантажується користувачу в систему.
Extension	-
Post-Condition	Тестовий набір успішно збережено у вигляді JSON файлу

Таблиця 2.10 - Варіант використання UC-10

Use case name	Імпорт тестового набору за допомогою JSON файлу
Use case ID	UC-10
Goals	Відновлення або обмін тестового набору за допомогою імпорту з JSON файлу
Actors	Користувач
Trigger	Користувач бажає імпортувати тестовий набір

Продовження таблиці 2.10

Pre-conditions	Наявність JSON файлу з тестовим набором
Flow of Events	Користувач натискає кнопку імпортувати тестовий набір, після цього відкривається вікно вибору файлу, користувач обирає відповідний JSON файл, який містить інформацію про тестовий набір. Далі тестовий набір завантажується у плагін.
Extension	-
Post-Condition	Тестовий набір успішно створено із завантаженого JSON файлу

Таблиця 2.11 - Варіант використання UC-11

Use case name	Генерація коду тестового сценарію на мові JavaScript
Use case ID	UC-11
Goals	Збереження тестового сценарію у вигляді коду на мові JavaScript для інтеграції з платформами автоматизованого виконання тестових сценаріїв
Actors	Користувач
Trigger	Користувач бажає згенерувати код для тестового сценарію
Pre-conditions	Наявність створеного тестового сценарію
Flow of Events	Користувач відкриває існуючий тестовий сценарій та натискає кнопку згенерувати код на мові JavaScript, після цього відкривається вікно із згенерованим кодом, який користувач може скопіювати.
Extension	-
Post-Condition	Тестовий сценарій подано у вигляді коду на мові JavaScript, який готовий для подальшого використання

2.2. Аналіз системних вимог

Оскільки Google Chrome плагін не є відокремленим та самостійним ПЗ, а працює тільки в середовищі браузера, то він повинен мати сумісність з останньою стабільною версією браузера 124.0.6367.60. Також плагін має функціональність генерації коду залежну від сторонніх сервісів, які надають

доступ до великої мовної моделі, тому має бути підключення до мережі Інтернет зі швидкістю не менше 2 мегабіт в секунду.

2.3. Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.12 наведено загальну модель вимог, а в таблиці 2.13 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.12 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Створення тестових наборів та сценаріїв.	FR-1	Високий	Середній
1.1	Створення тестового набору.	FR-2	Високий	Середній
1.2	Створення тестового сценарію на основі дій користувача.	FR-3	Високий	Високий
2	Виконання тестових наборів та сценаріїв.	FR-4	Високий	Високий
2.1	Виконання тестового набору.	FR-5	Високий	Високий
2.2	Виконання тестового сценарію	FR-6	Високий	Середній
2.3	Збереження знімку екрану	FR-7	Середній	Середній
3	Підтримка тестових наборів та сценаріїв	FR-8	Середній	Середній
3.1	Редагування тестових сценаріїв	FR-9	Середній	Середній
3.2	Експорт тестового набору у вигляді JSON файлу	FR-10	Середній	Низький

Продовження таблиці 2.12

№	Назва	ID вимоги	Пріоритети	Ризики
3.3	Імпорт тестового набору із JSON файлу	FR-11	Середній	Низький
4	Збір статистики виконання	FR-12	Високий	Низький
4.1	Збір результатів виконання тестових сценаріїв	FR-13	Високий	Низький
4.2	Збір статистичних даних таких як середня тривалість виконання та кількість успішних і неуспішних запусків	FR-14	Середній	Низький
5	Генерація коду	FR-15	Середній	Високий
5.1	Генерація коду для автоматизованого тестового сценарію на мові JavaScript	FR-16	Середній	Високий

Таблиця 2.13 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Створення тестових наборів та сценаріїв. Користувач повинен мати можливість створювати тестові набори та тестові сценарії в системі.
FR-2	Створення тестового набору. Користувач повинен мати можливість створити пустий тестовий набір зазначивши його назву.

Продовження таблиці 2.13

ID вимоги	Назва та опис
FR-3	Створення тестового сценарію на основі дій користувача. Користувач повинен мати можливість додати пустий тестовий сценарій у тестовий набір зазначивши його назву і записати послідовні кроки сценарію виконуючи послідовні дії на сторінці браузера.
FR-4	Виконання тестових наборів та сценаріїв. Користувач повинен мати можливість запускати тестові набори та тестові сценарії на виконання.
FR-5	Виконання тестового набору. Користувач повинен мати можливість виконати тестовий набір. Кожен тестовий сценарій у даному тестовому наборі має виконатися послідовно.
FR-6	Виконання тестового сценарію. Користувач повинен мати можливість виконати тестовий сценарій. Усі кроки тестового сценарію мають виконатися послідовно.
FR-7	Збереження знімку екрану Плагін має зробити знімок екрану у випадку невдалого виконання тестового сценарію і зберегти його у звіті виконання тестового сценарію.
FR-8	Підтримка тестових наборів та сценаріїв. Користувач повинен мати можливості вносити зміни до існуючих тестових наборів та сценаріїв, а також мати можливість зберігати їх у файлах.

Продовження таблиці 2.13

ID вимоги	Назва та опис
FR-9	Редагування тестових сценаріїв. Користувач повинен мати можливість редагувати кроки тестових сценаріїв, додавати нові, видаляти існуючі і також змінювати порядок їх виконання.
FR-10	Експорт тестового набору у вигляді JSON файлу. Користувач повинен мати можливість зберегти існуючий тестовий набір, експортувавши його у вигляді JSON файлу.
FR-11	Імпорт тестового набору із JSON файлу. Користувач повинен мати можливість додати тестовий набір за допомогою імпорту його із JSON файлу.
FR-12	Збір статистики виконання. Система повинна зберігати результати виконання тестових наборів та сценаріїв і збирати статистику.
FR-13	Збір результатів виконання тестових сценаріїв. Система повинна зберігати результати виконання кожного кроку тестового сценарію. У разі виникнення помилки має бути збереженим скріншот екрану веб-застосунку. Також система повинна збирати логи виконання.
FR-14	Збір статистичних даних таких як середня тривалість виконання та кількість успішних і неуспішних запусків. Після кожного запуску тестового набору чи сценарію система повинна оновлювати статистичні дані для кожного тестового сценарію такі, як середня тривалість виконання та кількість успішних і неуспішних запусків.
FR-15	Генерація коду. Користувач повинен мати можливість згенерувати код для існуючих тестових сценаріїв.

Кінець таблиці 2.13

ID вимоги	Назва та опис
FR-16	Генерація коду для автоматизованого тестового сценарію на мові JavaScript. Користувач повинен мати можливість згенерувати код для обраного тестового сценарію на мові JavaScript.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16
UC-1	+	+														
UC-2	+															
UC-3			+													
UC-4								+	+							
UC-5				+	+											
UC-6				+		+						+	+			
UC-7				+			+									
UC-8												+		+		
UC-9								+		+						
UC-10								+			+					
UC-11															+	+

Рисунок 2.1 – Матриця трасування вимог

2.4. Розроблення нефункціональних вимог

Програмне забезпечення має задовільняти наступним нефункціональним вимогам:

– не повинно порушувати безпеку браузера або комп'ютера користувача;
– має працювати стабільно, не спричиняючи падінь браузера;
– не повинно сповільнювати роботу браузера або завантаження веб-сторінок;
– має використовувати тільки надійні API через захищений протокол;
– повинно працювати коректно в середовищі браузера Google Chrome з версіями після 124.0.6367.60.

Висновки до розділу

В даному розділі було проведено детальний аналіз вимог до функціональності системи. Була побудована діаграма варіантів використання для даного ПЗ. Кожен варіант використання було детально представлено в окремій таблиці, де було описано які дії доступні для кожного актору та за яких умов.

При аналізі системних вимог було виявлено, що для роботи плагіна, користувачу необхідно і достатньо мати встановлений браузер Google Chrome, оскільки він не є самостійним і може працювати тільки в середовищі браузера.

Далі було проаналізовано функціональність створюваного ПЗ і розроблено функціональні вимоги до програмного забезпечення. В таблиці було наведено загальну модель вимог, де вказувалися назва та ідентифікатор вимоги, а також пріоритет вимогі і ризики пов’язані з нею. Потім було детально описано кожну вимогу. Після цього було побудовано матрицю трасування вимог, яка показує, яка вимога покриває який варіант використання.

Потім було розроблено нефункціональні вимоги до ПЗ такі як швидкодія, безпека користувачів та підтримка новими версіями браузера.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура програмного забезпечення

Архітектура плагіну матиме типову архітектуру для Google Chrome плагінів. На графічному матеріалі 2 зображено основні компоненти архітектури розроблюваного плагіну.

Тепер детально розглянемо кожен компонент архітектури та зв’язки між ними.

Service worker – це компонент, який грає ключову роль у плагіні. Він складається із скриптів, які працюють у фоновому режимі плагіну, незалежно від активності користувача. Вони призначені для реагування на події, які виникають в системі. При активації плагіну, даний компонент відповідає за відкриття головного вікна (Plugin window) з користувацьким інтерфейсом. Крім того, він грає роль як посередник в комунікації між основними компонентами системи: головним вікном плагіну (Plugin window) та сторінками браузера, оскільки немає можливості взаємодіяти напряму. Це використовується для забезпечення можливості запису дій користувача на сторінці браузера, а також для виконання записаного тестового сценарію.

Plugin window – це головне вікно плагіну, яке відкривається при його активації. Даний компонент позначає графічний інтерфейс, в якому надаються користувачам різні функції даного плагіну, такі як створення тестових сценаріїв, управління тестовими наборами, їх виконання, збір і відображення статистики та інше. Тут відбувається безпосередня взаємодія з користувачем. При надходженні запиту від користувача, цей компонент відправляє повідомлення через Chrome Extension API, яке оброблюється фоновими скриптами (service worker). Також даний компонент включає в себе комунікацію із сторонніми сервісами, наприклад OpenAI API, що використовується для генерацій коду для тестових сценаріїв. Крім того, тут

відбується взаємодія із сховищем даних для збереження усіх необхідних даних, як тестові набори, запуски тестів та статистика.

Chrome storage - це компонент системи, який являє собою локальне сховище для зберігання даних в об'єктному вигляді. Тут зберігаються дані, такі як створені тестові сценарії, набори тестів, статистика виконання тестових сценаріїв та додаткові налаштування плагіну. Взаємодія з даним компонентом відбувається через запити по Chrome Storage API.

Content script - це компонент, який відповідає за впровадження додаткової функціональності та обробників подій у сторінки браузера. При запису тестового сценарію на сторінку браузеру додаються скрипти, які призначені для обробки подій, які прослуховуються на даній сторінці. Також при запуску виконання автоматизованого тестового сценарію, на сторінці браузера починають виконуватися дії через взаємодію даних скриптів та веб-сторінки браузера. Даний компонент спілкується за допомогою повідомлень із фоновим скриптом, який надсилає йому повідомлення з командами для виконання та очікує результат.

Web page - це компонент, з яким взаємодіє плагін і позначає веб-сторінку, яку користувач відкриває у своєму браузері. На цій сторінці може бути вбудований скрипт контенту, який взаємодіє з плагіном та прослуховує дії користувача або виконує дії, пов'язані з виконанням тестового сценарію.

OpenAI API - це зовнішній сервіс, який забезпечує доступ до функціоналу великої мовної моделі для генерації коду для тестових сценаріїв. Плагін взаємодіє з ним за допомогою захищених HTTPS запитів, та передає йому опис тестового сценарію з його кроками, пояснює, що потрібно виконати та очікує на результат. LLM в свою чергу оброблює запит за надає у відповіді згенерований код для наданого тестового сценарію.

В результаті система працює наступним чином:

– фонові скрипти координують комунікацію між інтерфейсом плагіну та веб-сторінками браузера;

– вікно плагіну служить інтерфейсом для взаємодіє із користувачами плагіну та комунікує із іншими компонентами системи через відповідні API, а також через фонові скрипти;

– браузерне сховище даних забезпечує доступ до збережених в плагіні даних;

– скрипти контенту взаємодіють з елементами сторінки та збирають необхідні дані для створення і виконання тестових сценаріїв;

– сторінка браузера є платформою для виконання скриптів контенту та збору даних для тестування;

– OpenAI API використовується для генерації коду для тестових сценаріїв.

3.2. Обґрунтування вибору засобів розробки

Розглянемо допоміжні програмні засоби, фреймворки, мови програмування та інтегровані середовища розробки (IDE), які використовуються для створення плагіну Google Chrome для автоматизації тестування веб-застосунків. Проаналізуємо кожен інструмент та вибиремо на основі їх переваг та відповідності вимогам проєкту.

Для розробки було обрано WebStorm через його численні переваги. WebStorm забезпечує потужне автозаповнення коду для JavaScript, TypeScript, HTML і CSS, що значно прискорює процес розробки. Інтегровані інструменти для роботи з системою контролю версій, підтримка Git, GitHub та інших систем дозволяють легко керувати версіями коду. Також для даного ПЗ надається студентська ліцензія, що дозволяє використовувати його безкоштовно. Альтернативою WebStorm могли б бути такі IDE, як Visual Studio Code або Atom. Проте, вони поступаються WebStorm за рівнем інтеграції та підтримки специфічних для JavaScript та TypeScript функцій, що впливає на зручність та швидкість розробки [17].

Основною мовою програмування для проєкту обрано TypeScript завдяки його перевагам над JavaScript. TypeScript додає до JavaScript підтримку

об'єктно-орієнтованого програмування, що дозволяє створювати більш структурований і зрозумілий код. Статична типізація знижує ризики помилок, пов'язаних з типами даних, що спрощує налагодження та підтримку коду. TypeScript також інтегрується з багатьма інструментами розробки, забезпечуючи кращу підтримку автозаповнення, рефакторингу та інших функцій IDE. Сумісність з JavaScript дозволяє використовувати TypeScript в будь-якому середовищі, де підтримується JavaScript. Використання чистого JavaScript могло б бути альтернативою, але воно позбавлене переваг статичної типізації та підтримки ООП, що ускладнює розробку та підтримку коду [18].

Для розробки інтерфейсу плагіну обрано бібліотеку React через його компонентну архітектуру, високу продуктивність та широку підтримку спільноти. React дозволяє створення інтерфейсів користувача, використовуючи компоненти, що спрощує розробку, тестування та повторне використання коду. Також React підтримує автоматичне оновлення користувацького інтерфейсу при зміні стану компоненту, що робить його дуже зручним для використання. Завдяки використанню віртуального DOM, React забезпечує високу продуктивність та швидку реакцію інтерфейсу. Альтернативою могло б бути використання звичайних HTML, CSS та JavaScript. Однак, такий підхід позбавлений компонентної архітектури, що ускладнює повторне використання та тестування коду, а також може призвести до проблем з продуктивністю при обробці великих обсягів даних [19].

Для управління станом плагіну було обрано Redux завдяки його централізованому управлінню станом, передбачуваності поведінки додатку та наявності зручної інтеграції з бібліотекою React, що спрощує його використання. Альтернативою могло б бути керування станом в кожному окремому компоненті та передавання передавання його каскадно з одного компоненту в інший, але це ускладнює відстеження змін стану та налагодження, особливо при великій кількості компонентів, що взаємодіють між собою [20].

Для розробки інтерфейсу користувача у цьому проєкті була обрана бібліотека MUI (Material-UI) у комбінації з React. MUI надає набір готових компонентів, які відповідають принципам Material Design, розробленим компанією Google. Цей вибір базується на кількох ключових перевагах порівняно з іншими бібліотеками, такими як Bootstrap.

MUI розроблена спеціально для React, що забезпечує тісну інтеграцію з цією бібліотекою. Компоненти MUI легко налаштовуються і можуть бути використані безпосередньо у React-додатках, знижуючи складність інтеграції та підвищуючи продуктивність. Крім того, MUI надає сучасний та уніфікований дизайн завдяки відповідності принципам Material Design, що забезпечує інтуїтивно зрозумілий користувацький інтерфейс.

Порівняно з Bootstrap, MUI забезпечує кращу інтеграцію з React, оскільки Bootstrap не є спеціалізованою бібліотекою для цієї бібліотеки і часто вимагає додаткових обгорток, таких як React-Bootstrap. MUI також пропонує більш потужні та гнучкі механізми налаштування стилів і тем, що дозволяє створювати унікальні дизайни, тоді як налаштування стилів у Bootstrap може бути менш інтуїтивним та вимагати більше зусиль [21].

Також у проєкті використовується велика мовна модель (LLM) через OpenAI API для генерації коду для тестових сценаріїв. Моделі OpenAI, зокрема GPT-4, здатні ефективно генерувати код на мовах програмування, таких як JavaScript, для автоматизації тестових сценаріїв. Використання навченої моделі для цієї задачі має кілька переваг. По-перше, LLM може швидко генерувати великий обсяг коду на основі наданих описів сценаріїв, що значно прискорює процес розробки. По-друге, моделі OpenAI навчені на великому обсязі даних і здатні пропонувати оптимальні рішення та враховувати найкращі практики програмування.

Існує кілька провайдерів для LLM, таких як OpenAI, Google (з моделями на основі BERT та інших технологій), Microsoft (з моделями на основі GPT) та інші. Проте, вибір OpenAI API обумовлений його високою точністю, доступністю та широкою підтримкою спільноти. Моделі OpenAI є одними з

найпотужніших на ринку, вони добре документовані та мають зрозумілий API, що спрощує інтеграцію в проєкт [22].

Перевага використання LLM полягає в його здатності швидко і ефективно адаптуватися до різних сценаріїв і контекстів, пропонуючи більш точні та оптимальні рішення. Моделі OpenAI навчені на великому обсязі даних і здатні генерувати код, що відповідає найкращим практикам програмування, що значно спрощує процес розробки та знижує ймовірність помилок. Таким чином, вибір LLM для генерації коду обґрунтований його високою ефективністю, точністю та здатністю забезпечити якісний результат у короткий термін.

Таким чином, вибір WebStorm, TypeScript, React, Redux, Material UI та OpenAI API обґрунтований їх перевагами та відповідністю вимогам проєкту. Використання цих технологій забезпечує ефективну, надійну та підтримувану розробку плагіну для Google Chrome, що автоматизує тестування веб-застосунків. Альтернативні можливості, хоча й мають свої переваги, не забезпечують такого рівня зручності, продуктивності та надійності, необхідного для даного проєкту.

3.3. Конструювання програмного забезпечення

В якості сховища даних буде використовуватися сховище даних браузера, де дані зберігаються у вигляді об’єктів. У таблицях 3.1 – 3.6 описані основні об’єкти та їх поля, які зберігаються у сховищі: тестовий набір, виконання тестового набору, тест, виконання тесту та крок тесту.

Таблиця 3.1 – Опис кореневого об’єкту даних

Назва поля	Тип даних	Опис
testSuites	Array of TestSuite	Масив тестових наборів

Таблиця 3.2 – Опис об’єкту TestSuite

Назва поля	Тип даних	Опис
id	string	Ідентифікатор тестового набору
title	string	Назва тестового набору
testCases	Array of TestCase	Масив об’єктів тестових сценаріїв, які містяться в даному наборі

Таблиця 3.3 – Опис об’єкту TestCase

Назва поля	Тип даних	Опис
id	string	Ідентифікатор тестового сценарію
title	string	Назва тестового сценарію
steps	Array of TestStep	Масив об’єктів тестових кроків, які містяться в даному тестовому сценарії
runs	Array of TestRun	Масив об’єктів виконання даного тестового сценарію

Таблиця 3.4 – Опис об’єкту TestStep

Назва поля	Тип даних	Опис
id	string	Ідентифікатор кроку тестового сценарію

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
Name	string	Назва команди
element	string	Шлях до елемента сторінки у форматі XPath
value	string	Опціональне поле - значення, яке буде використане при виконанні команди

Таблиця 3.5 – Опис об’єкту TestRun

Назва поля	Тип даних	Опис
id	string	Ідентифікатор запуску тестового сценарію
steps	Array of TestRunStep	Масив об’єктів виконання кроків тестового сценарію
status	TestRunStatus – перелічувальний тип із значеннями: Running, passed, failed, unknown	Статус виконання тестового сценарію
start	Date	Дата початку виконання тестового сценарію
duration	number	Тривалість виконання тестового сценарію в мілісекундах

Продовження таблиці 3.5

Назва поля	Тип даних	Опис
logs	Array of string	Масив рядків, які позначають логи виконання тестового сценарію
screenshot	string	Опціональний параметр, який подає скіншот екрану у вигляді символьного рядку

Таблиця 3.6 – Опис об’єкту TestRunStep

Назва поля	Тип даних	Опис
id	string	Ідентифікатор кроку тестового сценарію
name	string	Назва команди
element	string	Шлях до елемента сторінки у форматі XPath
value	string	Опціональне поле - значення, яке буде використане при виконанні команди
status	TestRunStatus	Статус виконання тестового кроку

На рисунку 3.1 показана діаграма об’єктів, де вказані основні об’єкти системи, які зберігаються у сховищі даних, їх поля об’єктів та зв’язки між ними.

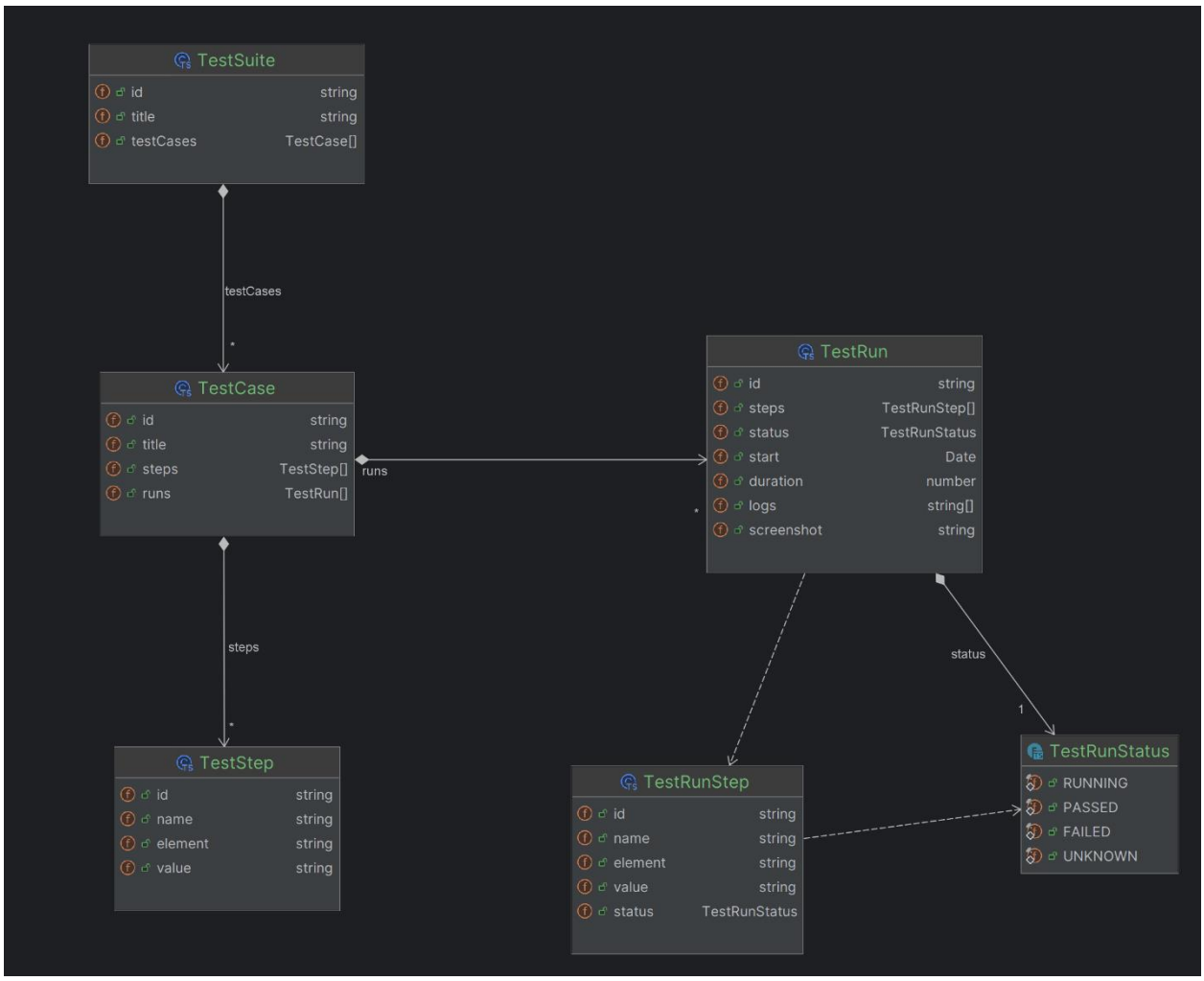


Рисунок 3.1 – Діаграма об’єктів

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.7.

Таблиця 3.7 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ WebStorm	Інтегроване середовище розробки (IDE), що забезпечує потужне автозаповнення коду, інтеграцію з системами контролю версій та інструменти для налагодження, що значно прискорює процес розробки. Використовується для розробки на JavaScript, TypeScript, HTML і CSS.

Продовження таблиці 3.7

№ п/п	Назва утиліти	Опис застосування
2	TypeScript	Мова програмування, що додає підтримку об'єктно-орієнтованого програмування та статичну типізацію до JavaScript, знижуючи ризики помилок і спрощуючи налагодження та підтримку коду.
3	React	Бібліотека для розробки інтерфейсів користувача з компонентною архітектурою, що забезпечує високу продуктивність та повторне використання коду. Використовується для створення інтерфейсу плагіну.
4	Redux	Бібліотека для управління станом додатка, що забезпечує централізоване управління станом, передбачуваність поведінки додатка та потужні інструменти для налагодження, такі як Redux DevTools.
5	Material UI	Бібліотека компонентів для React, що відповідає принципам Material Design. Забезпечує сучасний та уніфікований дизайн, глибоку інтеграцію з React, високу продуктивність та гнучкість налаштування стилів і тем.
6	OpenAI API	API для використання великих мовних моделей (LLM) для генерації коду для тестових сценаріїв. Забезпечує швидку генерацію коду на основі описів сценаріїв та врахування найкращих практик програмування.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4. Аналіз безпеки даних

Забезпечення безпеки даних у даному плагіні для тестування веб-застосунків є важливою складовою. Перш за все, важливо відзначити, що плагін працює з тестовими даними, а не особистою інформацією користувачів. Однак існують деякі особливості та ризики, які можуть виникнути в процесі розробки та використання плагіну.

Однією з потенційних проблем є взаємодія з OpenAI API, який надає доступ до LLM, куди передаються дані про систему, з якою взаємодіє користувач. При такій інтеграції слід дотримуватися принципів безпеки, зокрема використання захищеного протоколу HTTPS, який забезпечить шифрування переданих даних. Також система не має надавати інформацію LLM про особливості авторизації та тестові приклади пов’язані із введенням паролів у веб-застосунки.

Для забезпечення належного рівня безпеки в плагіні рекомендується уникати створення тестових сценаріїв, які передбачають введення чутливих даних користувачів. Відповідальність про уникнення проблем такого типу покладається на користувача. Також дані мають зберігатися у сховищі, до якого не можуть отримати доступ браузерні веб-застосунки та інші плагіни. Для цього буде використане Chrome Storage API, яке надає такі можливості.

Враховуючи вищезазначені принципи та заходи безпеки, можна буде мінімізувати ризики витоку даних та несанкціонованого використання інформації у плагіні для тестування веб-застосунків.

Висновки до розділу

У даному розділі було виконано ряд важливих завдань. Було розроблено архітектуру програмного забезпечення, яка є типовою для плагінів Google Chrome, визначено основні компоненти системи, їх взаємодію та інтеграцію. Було детально розглянуто кожен компонент архітектури, його роль та призначення, а також описано тип комунікації із іншими компонентами.

Потім було проведено аналіз можливих технологій та інструментів для реалізації проєкту, обрано найбільш підходящі з них з точки зору функціональних можливостей, продуктивності та зручності використання. Було визначено мову програмування, середовище розробки та бібліотеки, які спрощують розробку користувацького інтерфейсу плагіну. Зокрема, було вирішено використовувати TypeScript – розширення мови JavaScript, бібліотеки React, Material UI та Redux для використання компонентного підходу розробки UI плагіну та централізованим керуванням станом, Chrome Extension API, який надає можливість забезпечити комунікацію між компонентами плагіну та отримати доступ до сховища даних, а також OpenAI API для доступу до LLM.

Далі було детально описано процес конструювання основних модулів ПЗ, здійснено їх проектування з урахуванням вимог до функціональності. Було детально розглянуто модель сховища даних та описано об’єкти, які зберігаються в ній, їх структуру та ієрархію. Також було наведено таблицю з описом основних утиліт, які використовуються при створенні даного ПЗ.

Також було проведено аналіз безпеки даних, що використовуються та обробляються плагіном. Було визначено, що плагін не зберігає дані користувача, що значно зменшує загрози витоку чутливих даних, але також було розроблено заходи для забезпечення безпечної комунікації із сторонніми сервісами.

Таким чином, даний розділ надав всебічне уявлення про процес розробки програмного забезпечення, починаючи від архітектурного проєктування до реалізації функціональних можливостей та забезпечення безпеки даних.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Аналіз якості ПЗ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності плагіну з останніми версіями браузеру Chrome;

– знаходження дефектів, проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу;

– оцінка швидкості та ефективності роботи плагіну;

– перевірка коректності обробки виняткових ситуацій;

– перевірка інтеграції зі сторонніми сервісами та API.

Метриками для оцінки якості ПЗ обрано наступні:

– безпека;

– цикломатична складність коду;

– підтримуваність коду;

– відсоток дублювань;

– читабельність коду.

Використавши статичний аналізатор коду Sonar Cloud перевіримо розроблений плагін на відповідність даним метрикам. На рисунку 4.1 можна побачити, що немає проблем з безпекою коду, низький відсоток дублювань, хороша читабельність коду, але існують місця, де потрібно виконати рефакторинг коду для покращення його підтримуваності.

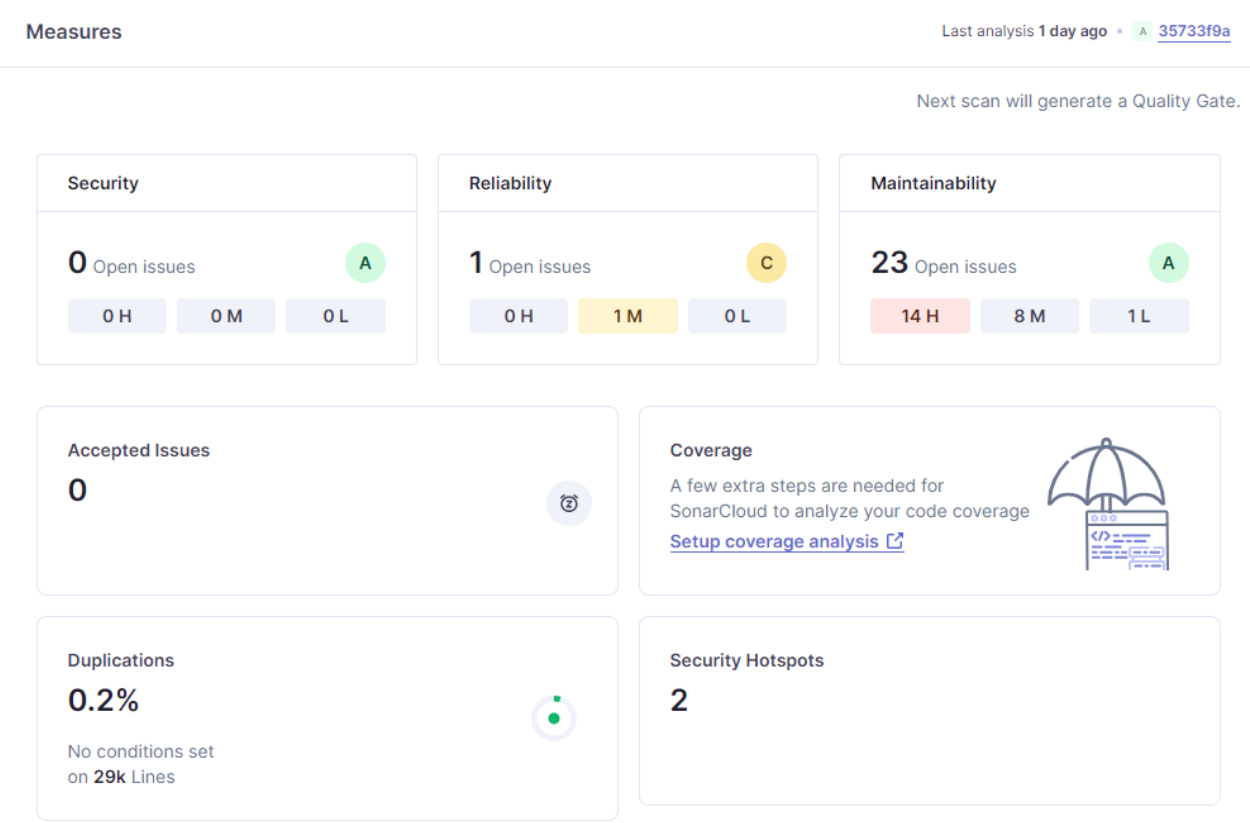


Рисунок 4.1 – Статичний аналіз коду

4.2. Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.14.

Таблиця 4.1 – Тест 1.1 Створення тестового набору

Початковий стан системи	Користувач знаходиться на головній сторінці плагіну
Вхідні дані	Назва тестового набору
Опис проведення тесту	Користувач натискає кнопку створити тестовий набір, після відкриття діалогу вводить назву тестового набору і натискає кнопку додати для підтвердження створення.
Очікуваний результат	Новий тестовий набір успішно створено із заданою назвою.

Продовження таблиці 4.1

Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Створення тестового сценарію

Початковий стан системи	Користувач знаходиться на головній сторінці плагіну і створений тестовий набір відкрито
Вхідні дані	Назва тестового сценарію
Опис проведення тесту	Користувач натискає кнопку додати тестовий сценарій всередині відкритого тестового набору, після відкриття діалогу вводить назву тестового сценарію і після цього натискає кнопку додати для підтвердження створення.
Очікуваний результат	Новий тестовий сценарій успішно створено із заданою назвою і додано у відкритий тестовий набір.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Запис тестового сценарію на основі дій користувача

Початковий стан системи	Користувач відкрив створений пустий тестовий сценарій
Вхідні дані	Послідовність дій виконаних на веб-сторінці браузера
Опис проведення тесту	Користувач натискає кнопку запису тесту у відкритому тестовому сценарії, переходить на сторінку браузера і починає виконувати послідовно дії для формування тестового сценарію. Дії можуть включати в себе відкриття посилання, натискання на кнопки, введення тексту, перевірки вмісту елементів сторінки і т. д. Після цього користувач натискає кнопку завершення запису у вікні плагіну.

Продовження таблиці 4.3

Очікуваний результат	Дії виконані на сторінці браузера записані та успішно додані в тестовий сценарій як послідовні кроки виконання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Ручне створення тестового сценарію

Початковий стан системи	Користувач відкрив створений пустий тестовий сценарій
Вхідні дані	Послідовність кроків тестового сценарію
Опис проведення тесту	Користувач натискає кнопку додати тестовий крок у відкритому тестовому сценарії і вибирає тип операції із списку доступних. Тип операції може включати відкриття посилання, натискання миші, введення тесту, натискання клавіші, перевірка тексту елементу веб-сторінки і т. д. Далі користувач натискає на кнопку обрати елемент на сторінці, переходить на сторінку браузера і натискає на обраний елемент, який автоматично виділяється червоною границею, після чого шлях до даного елемента автоматично обраховується і додається у поле елемента. Якщо операція включає перевірку, то користувач додає у поле значення очікуваний текст, який очікується в елементі. Таким самим чином додаються всі інші кроки тестового сценарію.
Очікуваний результат	Тестовий сценарії містить усі додані тестові кроки із заповненими полями: тип операції, елемент і значення.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Редагування тестового сценарію

Початковий стан системи	Користувач відкрив існуючий тестовий сценарій
Вхідні дані	Зміни, які користувач має внести до тестового сценарію
Опис проведення тесту	Користувач може змінити порядок виконання тестових кроків шляхом перетягування їх у інше місце. Для зміни існуючого тестового крок потрібно виконати подвійний клік на відповідний крок у таблиці, після чого він стає доступним для редагування. Далі можна змінити тип операції, переобрати елемент на сторінці браузера і змінити очікуване значення вмісту. Також можна видалити непотрібний тестовий крок.
Очікуваний результат	Дії виконані на сторінці браузера записані та успішно додані в тестовий сценарій як послідовні кроки виконання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Виконання тестового сценарію

Початковий стан системи	Користувач відкрив існуючий тестовий сценарій
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає кнопку запустити тестовий сценарій і переходить на веб-сторінку браузера для перегляду ходу виконання.

Продовження таблиці 4.6

Очікуваний результат	Тестовий сценарій виконано відповідно до кроків. Кожен успішно виконаний крок помічено зеленим кольором. У випадку неуспішного виконання тестового кроку, даний тестовий крок має бути помічений червоним кольором і виконання тестового сценарію має припинитися. Також для кожного кроку мають збиратися логи про успішність виконання кроків і відображатися у відповідному полі. При неуспішному виконанні тестового сценарію, має бути збережено знімок останнього стану веб-сторінки для полегшення розуміння причини падіння. Усі результати мають бути збереженими у відповідному тестового запуску.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Виконання тестового набору

Початковий стан системи	Користувач відкрив тестовий набір, який включає в себе декілька тестових сценаріїв, готових до виконання
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає кнопку запустити тестовий набір і переходить на веб-сторінку браузера для перегляду ходу виконання кожного тестового сценарію в даному тестовому наборі.
Очікуваний результат	Усі тестові сценарії мають бути виконаними послідовно відповідно до їх порядку у тестовому наборі. Для кожного запуску тестового сценарію мають бути збережені результати виконання і логи.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Перегляд статистики виконання тестових сценаріїв

Початковий стан системи	Користувач відкрив сторінку статистики виконання тестового сценарію, який було виконано декілька разів.
Вхідні дані	Відсутні
Опис проведення тесту	На сторінці статистики виконання мають бути відображені такі дані: таблиця із усіма запусками даного тестового сценарію, кількість успішних і неуспішних запусків, а також гістограма, на якій візуально відображено кількість успішний і неуспішних запусків і діаграма де показано тривалість виконання кожного запуску в мілісекундах і середню тривалість.
Очікуваний результат	Усі тестові сценарії мають бути виконаними послідовно відповідно до їх порядку у тестовому наборі. Для кожного запуску тестового сценарію мають бути збережені результати виконання і логи.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Генерація коду для тестового сценарію

Початковий стан системи	Користувач відкрив існуючий тестовий сценарій.
Вхідні дані	Відсутні

Продовження таблиці 4.9

Опис проведення тесту	Користувач натискає кнопку згенерувати код, після чого відкривається діалог на якому вказано, що код буде згенеровано на мові JavaScript і також він містить поле для відображення коду. Далі користувач натискає на кнопку згенерувати і чекає на завершення генерації. Після цього користувач натискає на кнопку скопіювати код у буфер обміну.
Очікуваний результат	Після початку генерації, код має почати поступово з’являтися у відповідному полі в режимі реального часу відповідно до взаємодії із сервісом штучного інтелекту. Після копіювання код має бути успішно скопійований у буфер обміну для його подальшого використання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Імпортування тестового набору

Початковий стан системи	Користувач знаходиться на головній сторінці плагіну.
Вхідні дані	Тестовий набір у вигляді файлу у форматі JSON
Опис проведення тесту	Користувач натискає кнопку імпортувати тестовий набір, обирає відповідний JSON файл із тестовим набором з відкритого вікна провідника. Після цього натискає кнопку завантажити.
Очікуваний результат	Новий тестовий набір успішно додано із усіма тестовими сценаріями збереженими у файлі. Всі назви тестових сценаріїв мають бути незмінені відповідно до назв у файлі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Експортування тестового набору

Початковий стан системи	Користувач відкрив існуючий тестовий набір.
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає кнопку експортувати тестовий набір, після чого відкривається провідник, де користувач може вибрати місце збереження файлу та вказати його назву.
Очікуваний результат	Тестовий набір має бути успішно збережено у JSON файлі у вказаному користувачем місці. Мають бути збережені дані про всі сценарії даного тестового набору, але усі статистичні дані та результатів запусків мають бути опущені.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Редагування назви тестового сценарію

Початковий стан системи	Користувач відкрив тестовий набір зі списком тестових сценаріїв, які він містить.
Вхідні дані	Нова назва тестового сценарію
Опис проведення тесту	Користувач обирає тестовий сценарій, назву якого потрібно змінити, натискає кнопку з трьома крапками і обирає опцію редагувати назву. Після цього з’являється діалог із назвою тестового сценарію, куди користувач вводить змінену назву та натискає кнопку зберегти.
Очікуваний результат	Назва тестового сценарію успішно змінена і збережена.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Видалення тестового набору

Початковий стан системи	Користувач відкрив існуючий тестовий набір.
Вхідні дані	Відсутні
Опис проведення тесту	Для відображеного тестового набору користувач натискає кнопку з трьома крапками і обирає опцію видалити тестовий набір.
Очікуваний результат	Тестовий набір має бути успішно видаленим із списку існуючих наборів.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.14 Видалення тестового сценарію

Початковий стан системи	Користувач відкрив тестовий набір, що містить тестовий сценарій, який потрібно видалити.
Вхідні дані	Відсутні
Опис проведення тесту	Для відображеного тестового сценарію користувач натискає кнопку з трьома крапками і обирає опцію видалити тестовий сценарій.
Очікуваний результат	Тестовий сценарій має бути успішно видаленим із списку сценаріїв даного тестового набору.
Фактичний результат	Збігається з очікуваним.

4.3. Опис контрольного прикладу

В даному контрольному прикладі буде описано тестовий сценарій, який має наступні кроки:

- створення тестового набору;
- створення тестового сценарію;

– запис простого тестового сценарію на існуючому веб-застосунку;
– виконання тестового сценарію;
– перевірка статистики.

Спочатку створимо тестовий набір із назвою «Test suite 1». Для цього натискаємо на кнопку «+ Test suite». У відкритому діалогу вводимо назву та натискаємо кнопку «Save» (рис. 4.2).

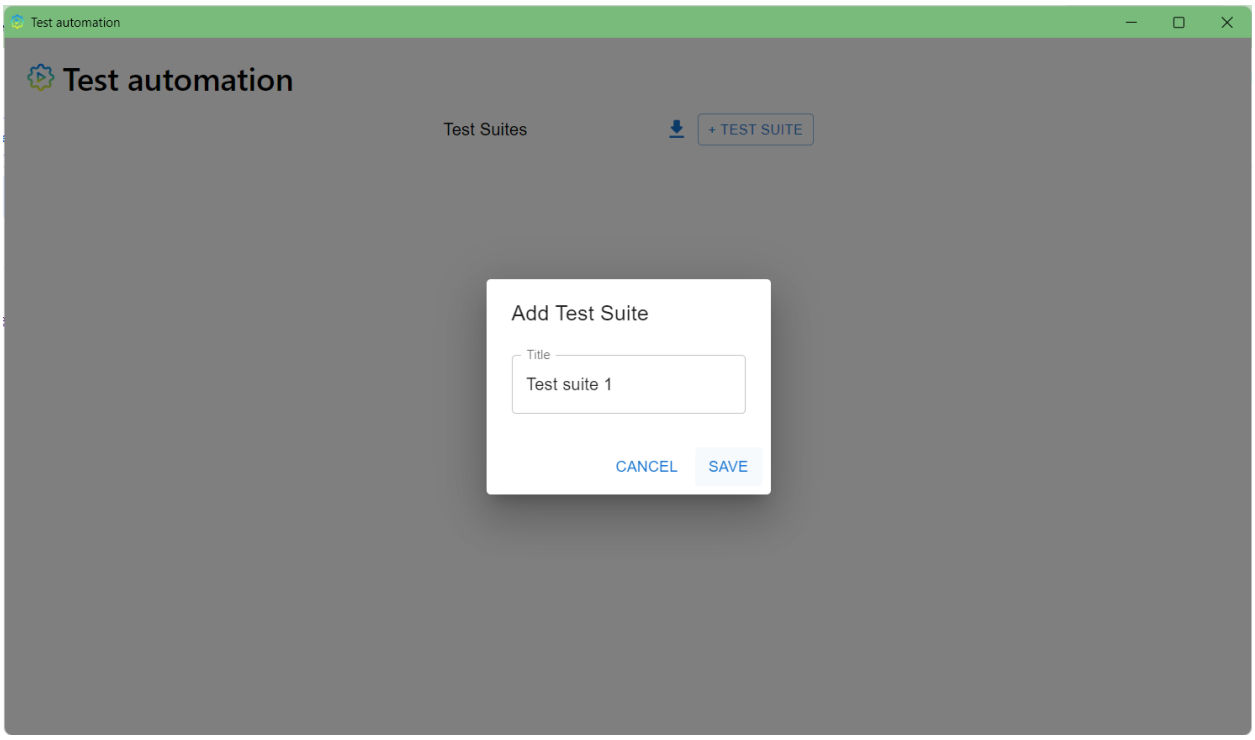


Рисунок 4.2 – Додавання нового тестового набору

Для створення тестового сценарію натискаємо на кнопку «+ Test case» і у відкритому діалогу вводимо назву тестового сценарію «Add test case». Після цього натискаємо на кнопку «Save» (рис. 4.3).

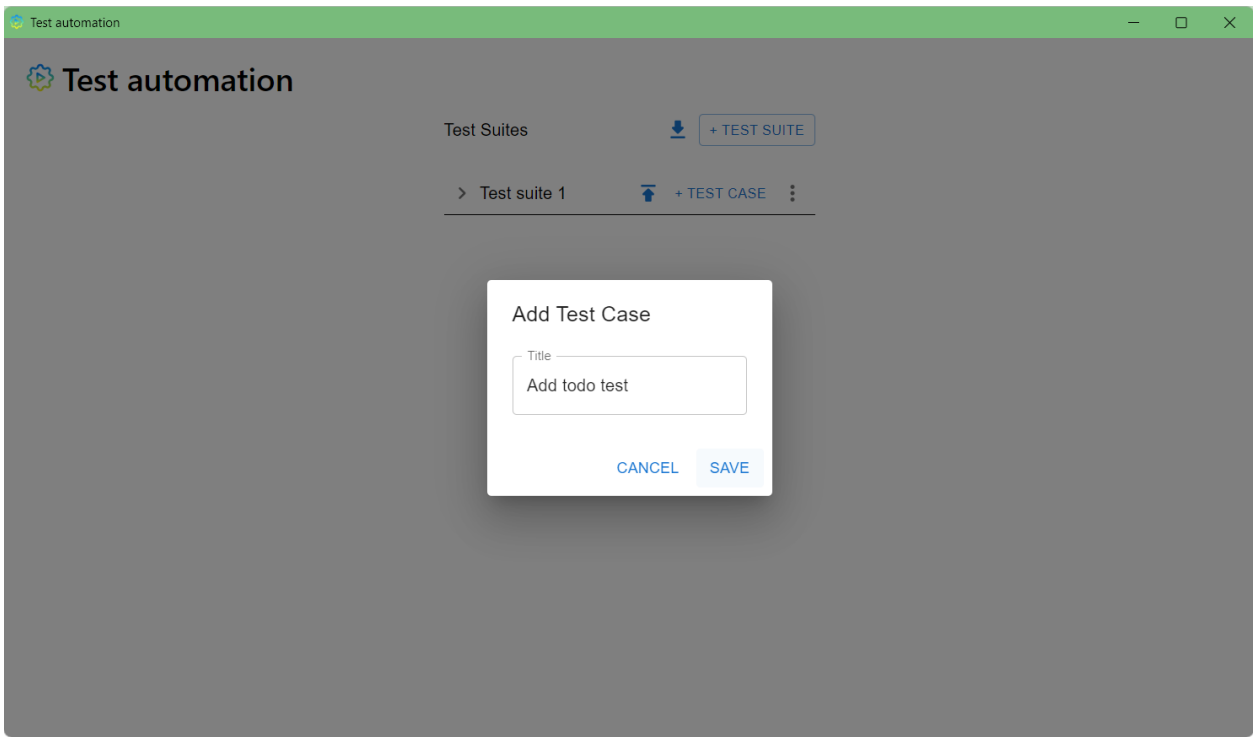


Рисунок 4.3 – Додавання нового тестового сценарію

Тепер запишемо тестовий сценарій. Для цього переходимо на новостворений сценарій і натискаємо кнопку «Record» і переходимо на сторінку веб-застосунку. Далі додамо дві задачі і перевіримо, що вони дійсно створені із введеним вмістом використовуючи функції контекстного меню (рис 4.4).

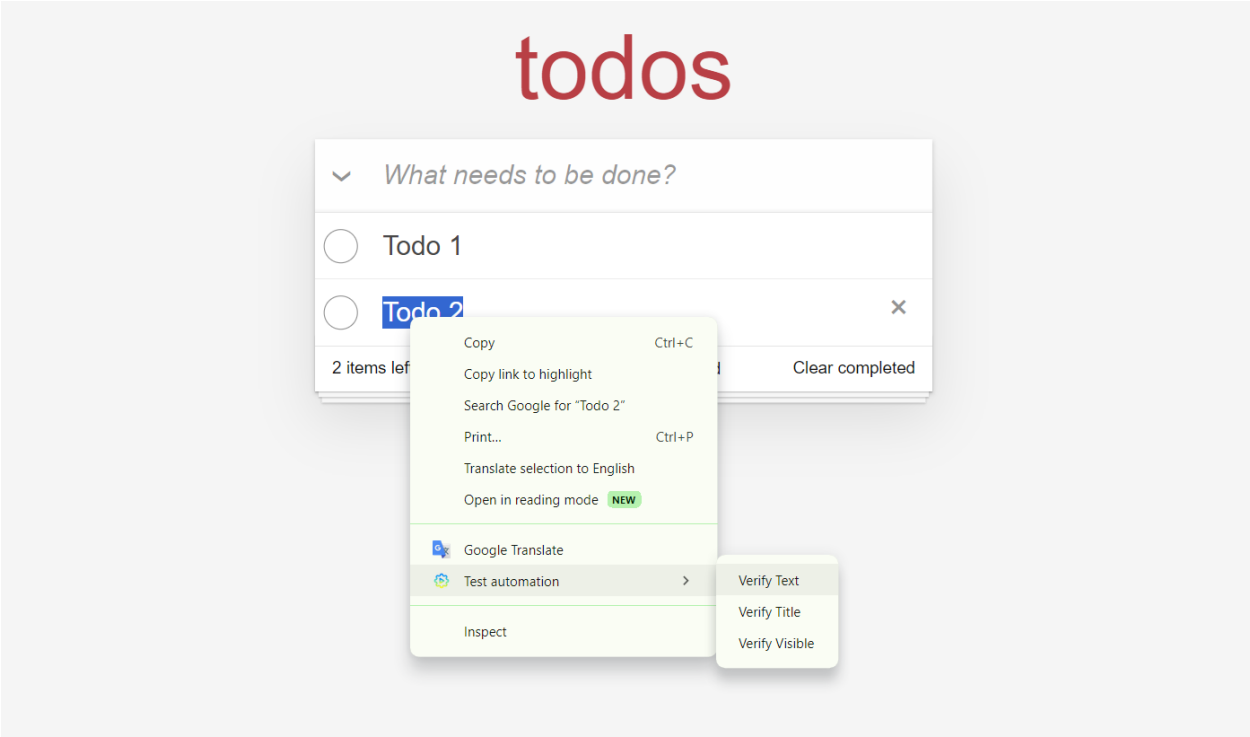


Рисунок 4.4 – Додавання нового тестового сценарію

Переконаємося, що всі виконані дії у застосунку записані у тестовий сценарій. Перша дія – це відриття веб застосунку по URL-посиланню, потім слідують введення назви задачі у поле вводу і натискання клавіші Enter. В кінці мають бути перевірки текстового вмісту елементів для створених задач (рис. 4.5). Потім закінчуємо запис тестового сценарію натискаючи кнопку «Stop recording».

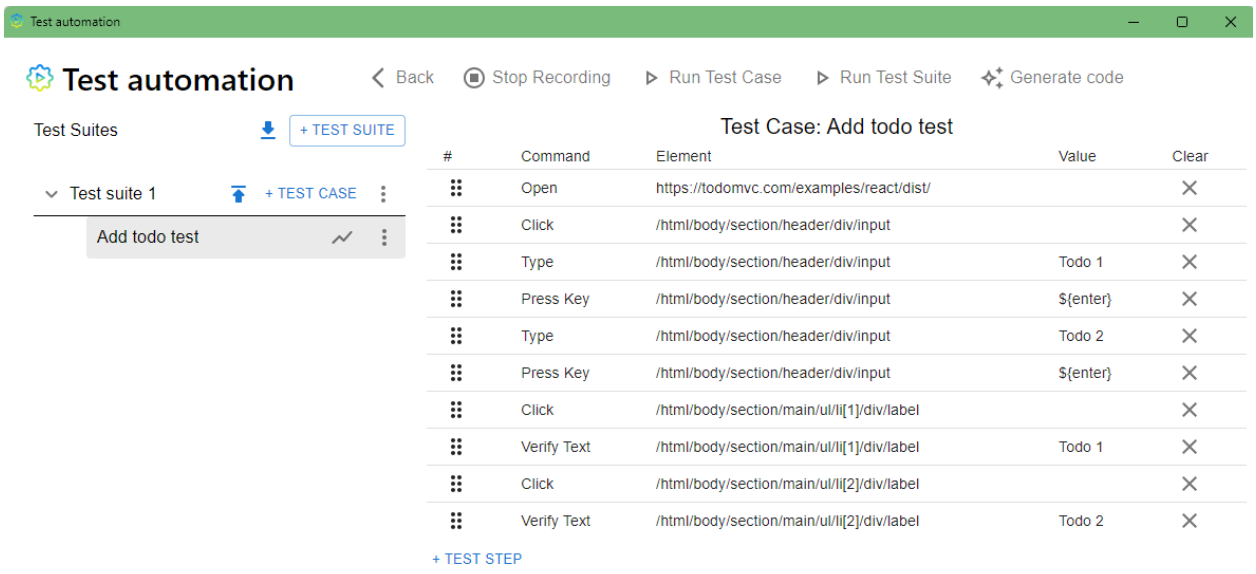


Рисунок 4.5 – Записані кроки тестового сценарію

Тепер запустимо даний тестовий сценарій на виконання, натискаючи кнопку «Run test case». Переконаємося, що усі кроки тестового сценарію виконані успішно і по закінченню на сторінці веб застосунку було автоматично створено дві задачі. На рисунку 4.6 ми можемо побачити, що усі сценарію кроки пройдені успішно, а також зібрано логи виконання. На рисунку 4.7 можна побачити, що дві задачі було створено у веб застосунку успішно із заданими назвами.

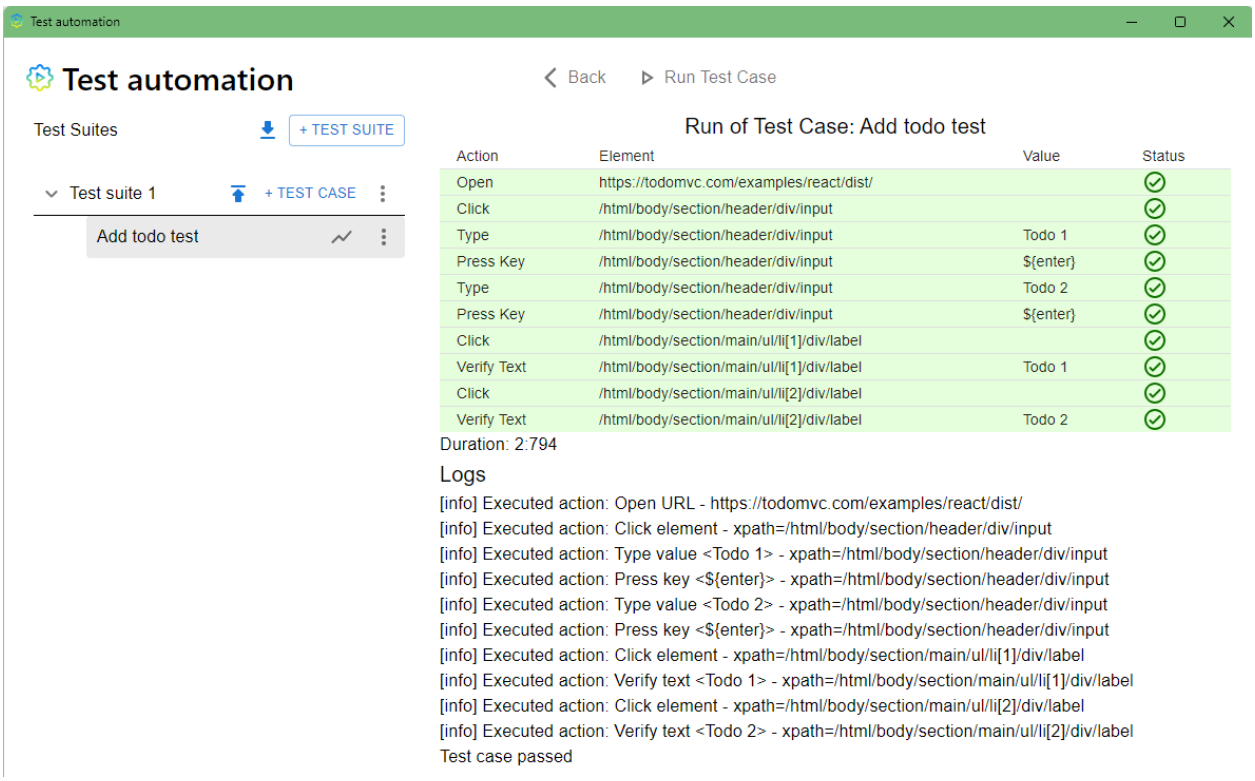


Рисунок 4.6 – Результати виконання тестового сценарію

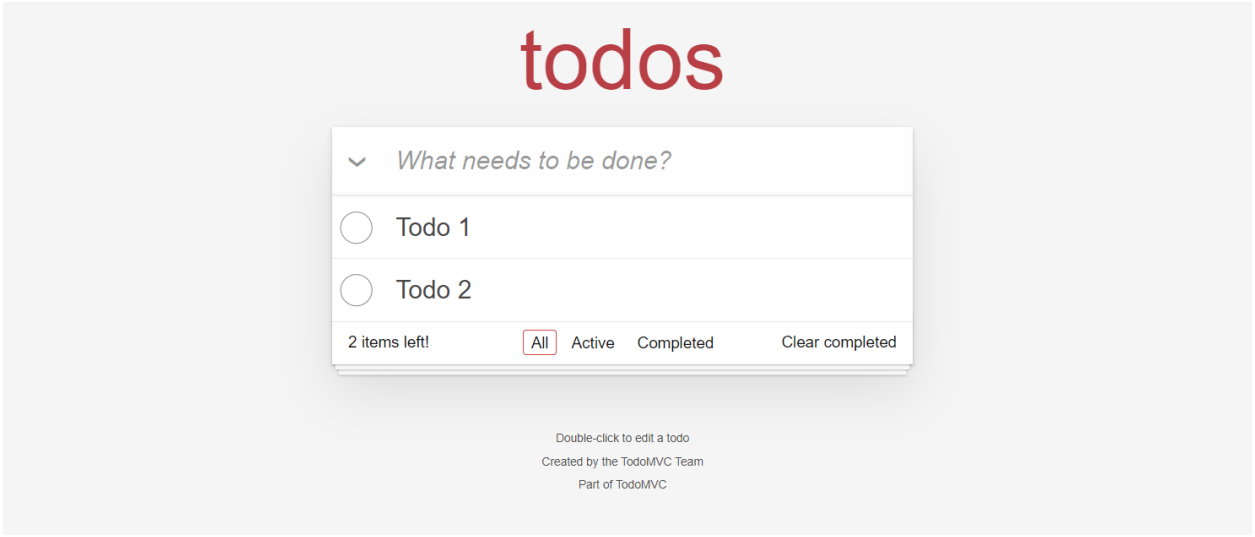


Рисунок 4.7 – Стан веб застосунку після виконання тестового сценарію

Тепер спробуємо створити штучну помилку у тестового сценарії, щоб перевірити коректність обробки падінь. Для цього введемо у крок перевірки текстового вмісту елементу некоректне очікуване значення (рис. 4.8).

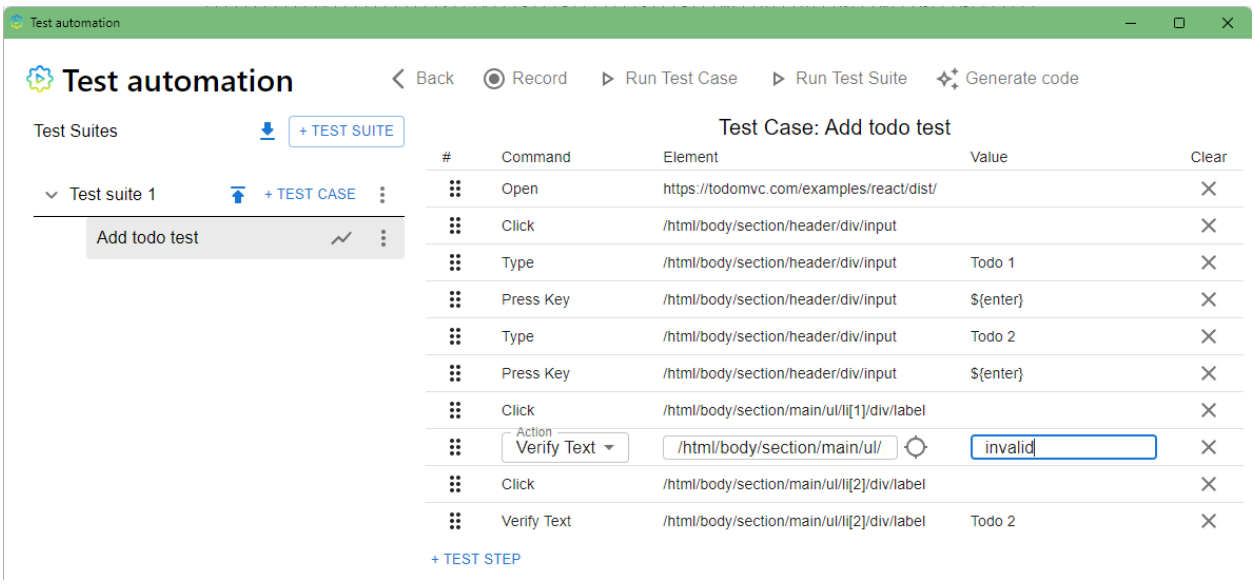


Рисунок 4.8 – Створення ситуації для виникнення помилки виконання тестового сценарію

Запустимо даний тестовий сценарій на виконання ще раз. Очікуємо, що він має завершитися невдачею і в логах має бути пояснення, що очікуваний текст елементу веб-сторінки відсутній. Також має бути наведений знімок екрану на момент виникнення помилки. На рисунку 4.9 можемо побачити, що помилкова ситуація оброблена коректно і містить пояснення помилки і знімок екрану.

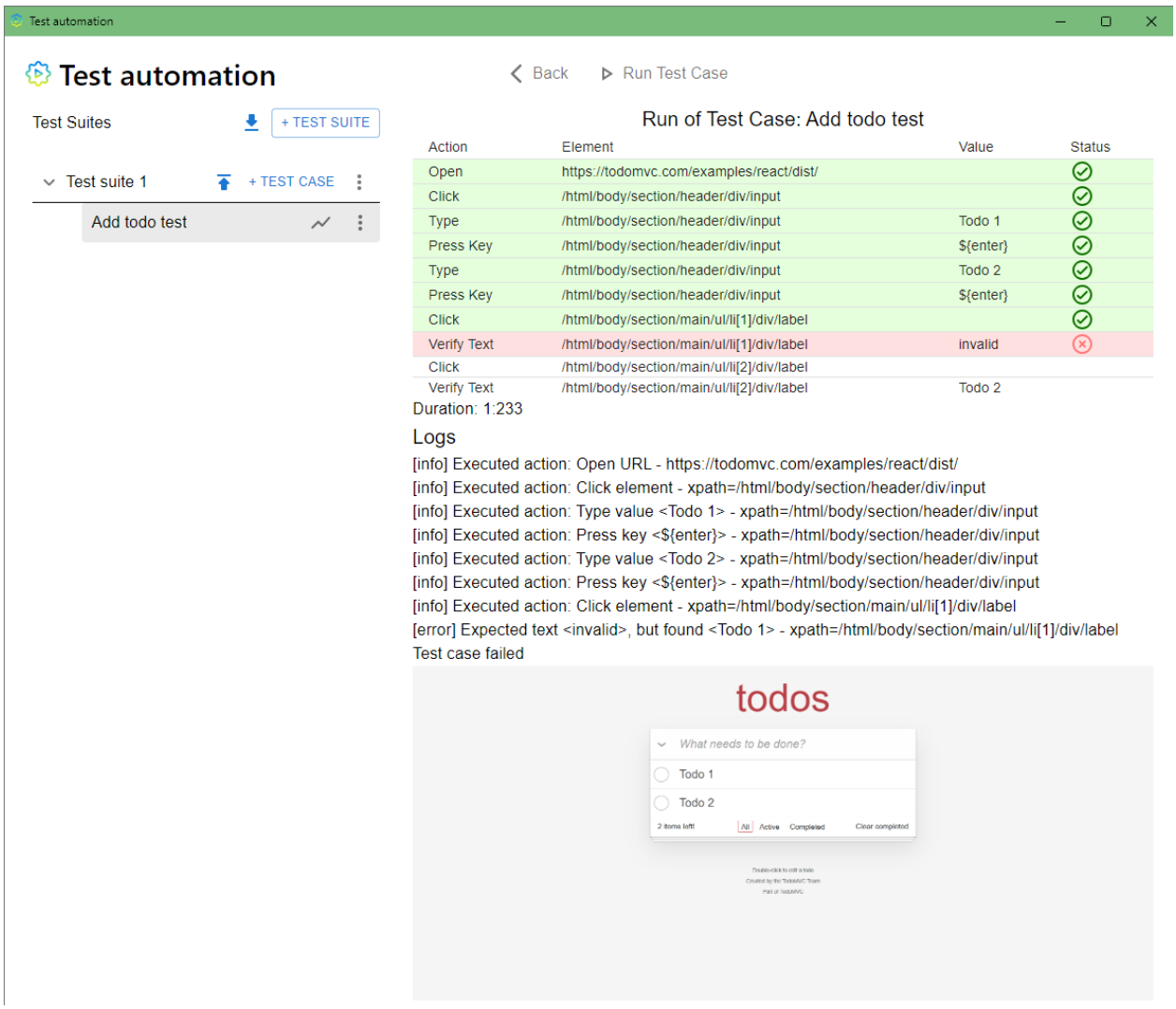


Рисунок 4.9 – Повторний запуск тестового сценарію

Останнім кроком є перевірка формування статистики. Статистика має включати усі запуски тестового сценарію, де вказана успішність виконання та час виконання в мілісекундах. Також має бути відображена гістограма, де зображено скільки успішних і неуспішних запусків. Крім того має бути зображена діаграма, де на графіку показано тривалість виконання кожного запуску тестового сценарію і показано середню тривалість. Усі ці очікувані дані можна побачити на рисунку 4.10.

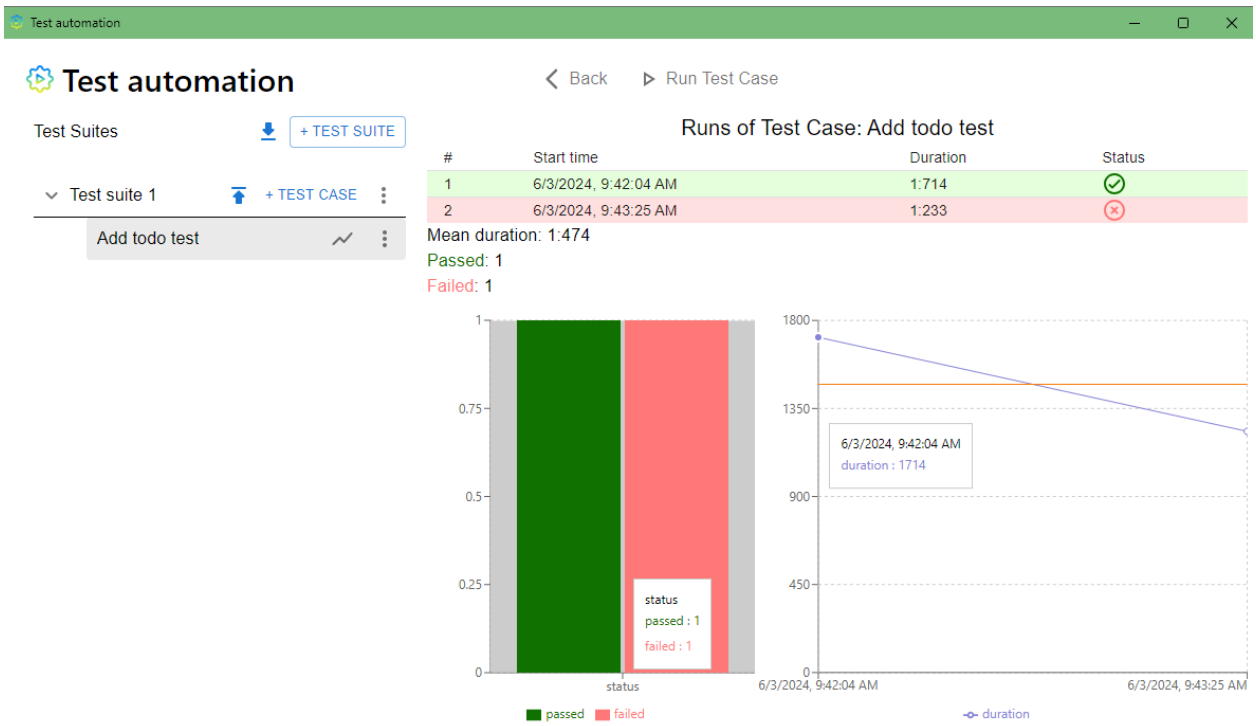


Рисунок 4.10 – Статистика виконання тестового сценарію

Висновки до розділу

У даному розділі було проведено комплексний аналіз якості та тестування програмного забезпечення, розробленого в рамках цього проєкту. Спочатку визначено ключові показники якості ПЗ, зокрема надійність, продуктивність, зручність використання, сумісність, мобільність, а також безпека. Проведено оцінку відповідності ПЗ цим показникам, що дозволило виявити сильні та слабкі сторони розробленого продукту.

Далі було описано процеси тестування, що включає розробку тестів для перевірки виконання функціональних вимог. Тестові сценарії включаються в себе опис початкового стану системи, вхідних даних, опис тестових кроків, очікуваний результат і фактичний результат. В результаті проведеного тестування було виявлено, що функціонал плагіну працює коректно.

Також було розглянуто конкретний приклад тестування, який ілюструє практичне застосування описаних методів і підходів. У цьому прикладі було описано сценарії тестування, включаючи очікувані результати та фактичні дані, отримані під час тестування.

У цьому розділі надано узагальнення та ключові результати, отримані в процесі тестування, проведеного відповідно до зазначеної програми. Тут продемонстровано комплексний підхід до забезпечення якості та тестування розробленого ПЗ. Було проведено всебічний аналіз та детальне тестування, що дозволило виявити критичні аспекти та забезпечити необхідний рівень якості для успішного впровадження та подальшого використання програмного продукту.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. Розгортання програмного забезпечення

Плагіни Google Chrome зазвичай розгортають в Chrome Web Store. Для цього спочатку потрібно зареєструвати обліковий запис розробника. Потім потрібно перейти на інформаційну панель розробника Chrome Web Store.

У кореневій директорії вихідного коду плагіну потрібно відкрити термінал і виконати команду: npm run build. По завершенню виконання даної команди буде сформовано zip-архів, готовий до завантаження у Chrome Web Store. Далі повертаємося в панель розробника і натискаємо кнопку «+ New Item». Після відкриття діалогового вікна завантажуємо сформований zip-архів (рис. 5.1).

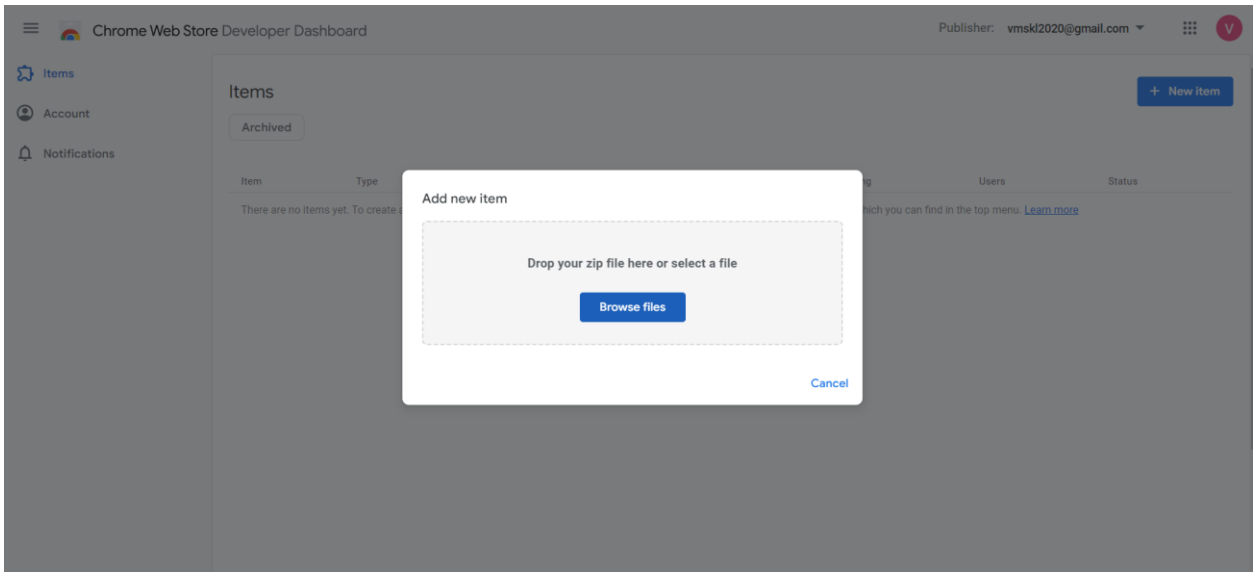


Рисунок 5.1 – Завантаження архіву з вихідним кодом

Потім заповнюємо усі необхідні поля: опис плагіну, категорія, мова, завантажуємо іконку і скріншоти (рис. 5.2). Далі зберігаємо введені значення, натискаючи кнопку «Save draft».

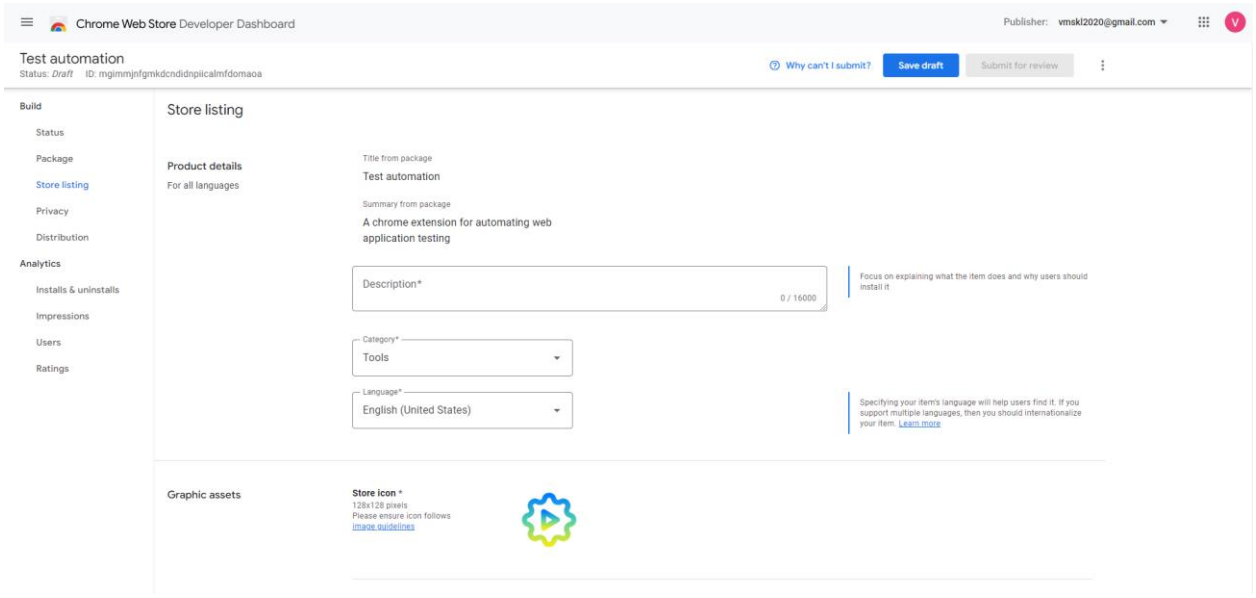


Рисунок 5.2 – Заповнення необхідних полів

Після цього переходимо на вкладку «Prіvacy» і заповнюємо обов’язкові поля: призначення плагіну, обґрунтування запрошених дозволів доступу до вкладок, контекстного меню, сховища даних та можливості завантаження файлів (рис. 5.3). Після заповнення зберігаємо введені значення.

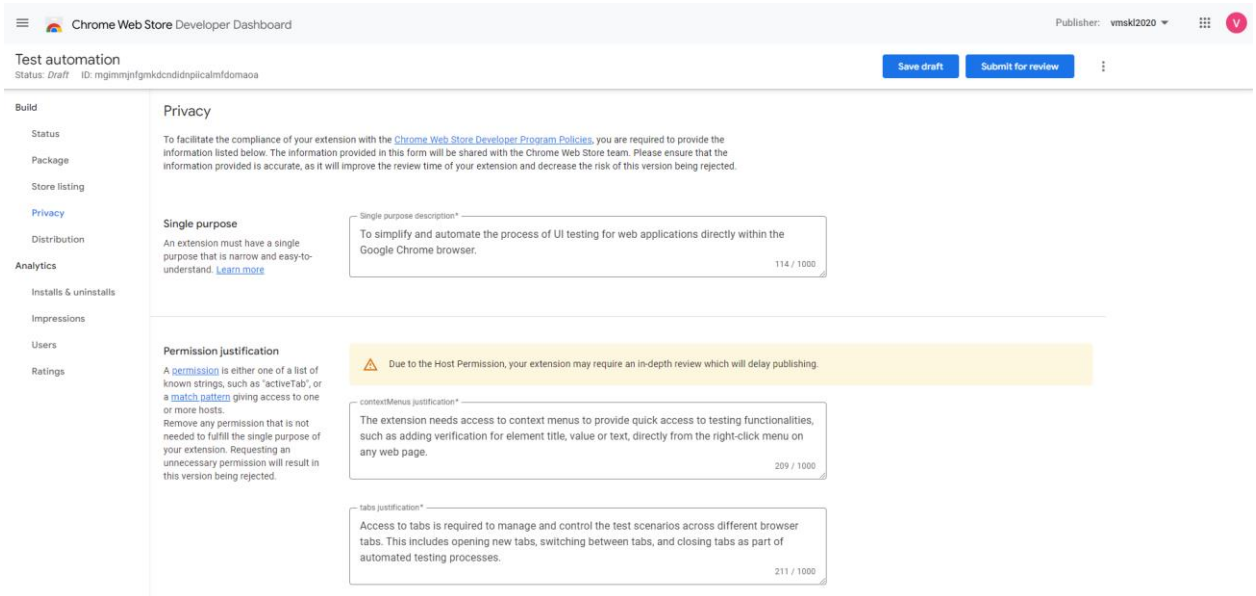


Рисунок 5.3 – Надання обґрунтування запрошених дозволів

Далі переходимо у вкладку «Distribution» і налаштовуємо доступність даного плагіну. Для цього вказуємо, що його можна використовувати безкоштовно, використовувати його можна за прямим посиланням, а також зазначаємо, що він доступний для використання в усіх регіонах (рис. 5.4).

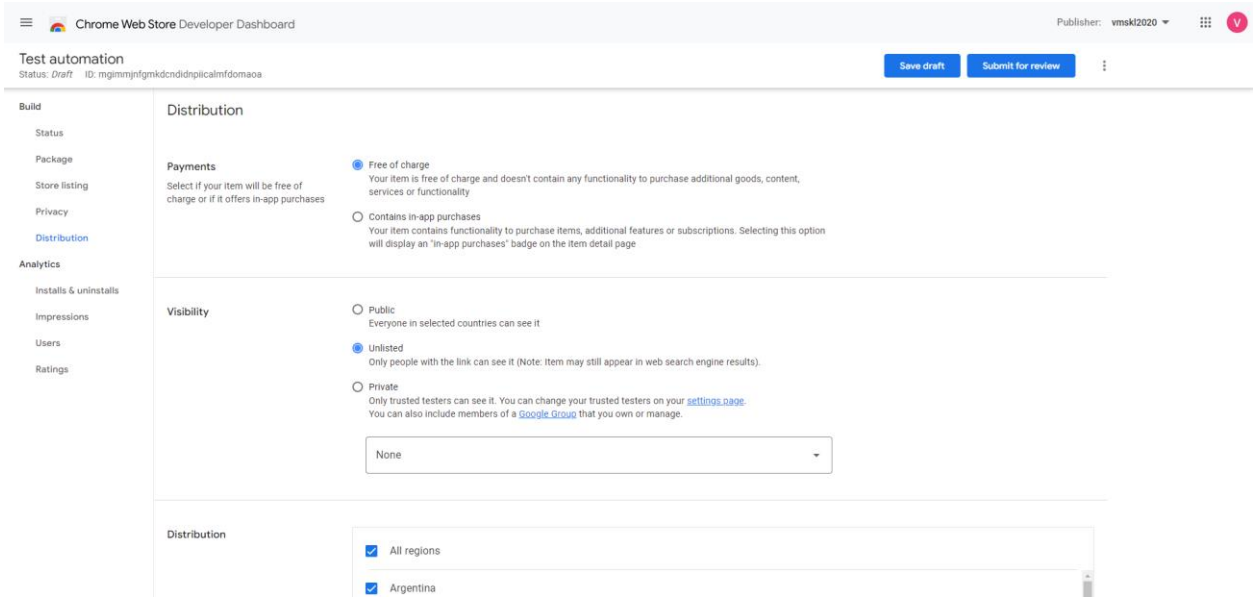


Рисунок 5.4 – Налаштування доступності плагіну

В кінці натискаємо кнопку «Submit for review» і надсилаємо запит на перевірку даного плагіну. Після успішної перевірки, даний плагін буде опубліковано.

Побудуємо діаграму розгортання для даного плагіну. Вона буде включати такі компоненти як браузер, сам плагін з його компонентами, а також сторонній AI сервіс, з яким взаємодіє плагін (рис. 5.5).

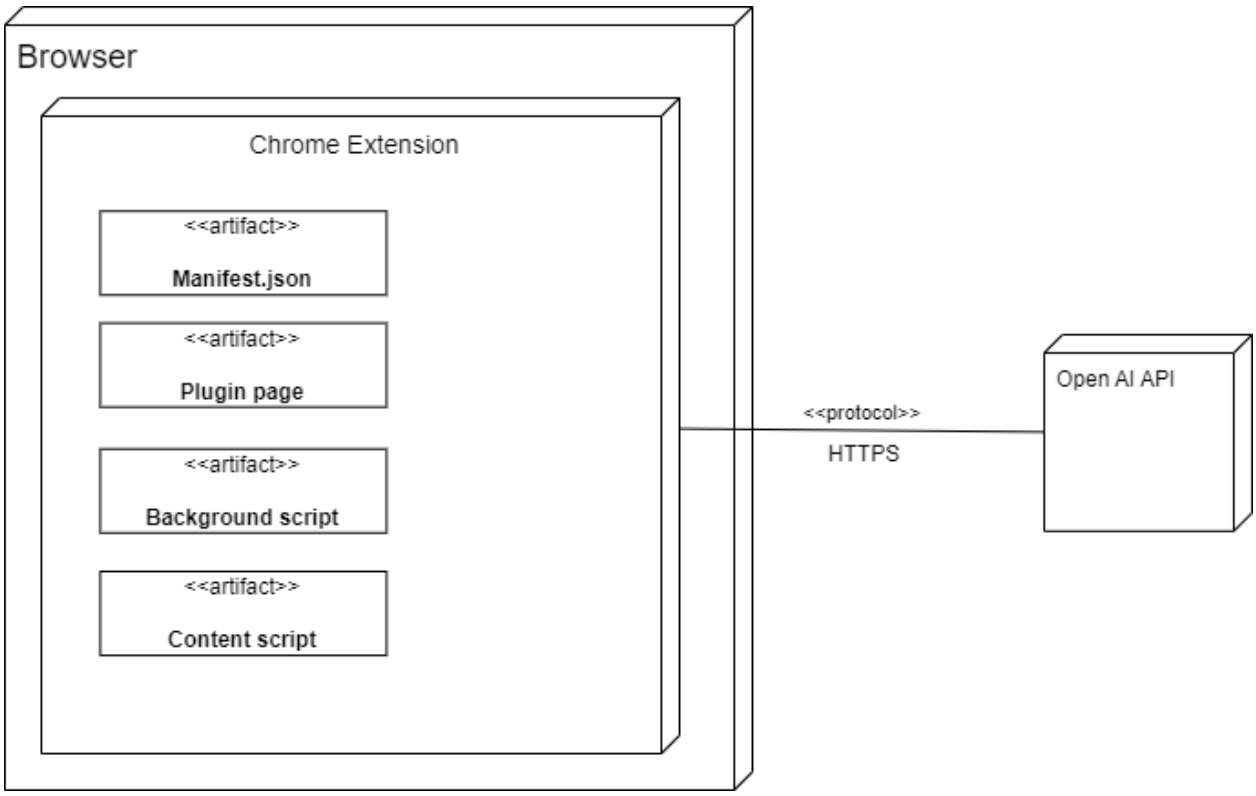


Рисунок 5.5 – Діаграма розгортання

5.2. Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Користувачі повинні мати можливість отримати нову версію плагіну, яка включає оновлення. Для цього спочатку оновимо версію плагіну у файлі manifest.json. Далі знову перезберемо архів з вихідним кодом за допомогою команди npm run build. Потім перейдемо в панель розробника Google Chrome і перейдемо на сторінку опублікованого плагіну. Далі переходимо на вкладку «Package» і натискаємо кнопку «Upload new package». Після цього завантажуємо новий архів з кодом плагіну, у відкрите діалогове вікно. При необхідності можна оновити опис плагіну, завантажити нові знімки екрану або додати відео огляду плагіну.

Коли вся інформація про плагін успішно оновлена, натискаємо кнопку «Submit for review» для відправки оновленого коду на перегляд. Після успішної перевірки завантаженого оновлення, новий випуск стане доступним для користувачів.

Висновки до розділу

У даному розділі було докладно розглянуто процес розгортання та супроводу ПЗ, зокрема плагіна для браузера Google Chrome.

Процес розгортання плагіна для Google Chrome починається з реєстрації облікового запису розробника в Chrome Web Store. Після цього розробник готує проєкт до завантаження, створюючи архів готового плагіна, який потім завантажується в панель розробника Chrome Web Store. На наступному етапі заповнюються всі необхідні поля, додаються опис, категорія, мова, іконка та скріншоти. Важливим кроком є вказання призначення плагіна та обґрунтування запрошених дозволів доступу. Після цього плагін зберігається як чернетка для подальшого опублікування.

Супровід плагіна передбачає регулярні оновлення, виправлення помилок та вдосконалення функціоналу. Це включає періодичне оновлення коду для підтримки сумісності з новими версіями браузера та врахування зворотного зв'язку від користувачів. Важливою частиною супроводу є перевірка та тестування нових версій плагіна перед їх опублікуванням для забезпечення стабільності та безпеки.

Процес супроводу також включає моніторинг продуктивності плагіна та аналіз використання, що дозволяє виявляти та оперативно виправляти потенційні проблеми. Це допомагає підтримувати високу якість продукту та задовольняти потреби користувачів.

Таким чином, у розділі детально розглянуто всі ключові аспекти розгортання та супроводу програмного забезпечення, що є важливими етапами у життєвому циклі будь-якого програмного продукту. Виконані заходи забезпечують успішне впровадження, стабільну роботу та постійний розвиток плагіна для браузера Google Chrome, що сприяє покращенню користувацького досвіду та підвищенню рівня задоволеності користувачів.

ВИСНОВКИ

У результаті виконання дипломного проєкту було спроектовано та реалізовано Google Chrome плагін для тестування веб застосунків.

Мету дипломного проєктування було успішно досягнуто, тому що розроблений плагін дав можливість спростити процес створення, підтримки та виконання тестових сценаріїв. Плагін був спроектований та реалізований відповідно до передових методів і технологій розробки програмного забезпечення та відповідає сучасному стану речей у області тестування веб застосунків.

Усі поставлені функціональні задачі були виконані в повному обсязі. Плагін дає можливість створювати, змінювати та виконувати тестові сценарії, а також збирати статистику їх виконання. Крім того, забезпечується можливість генерувати код для створених тестових сценаріїв.

Також усі задачі дипломного проєктування виконані в повному обсязі, зокрема аналіз предметної області, розроблення функціональних вимог, проєктування архітектури та конструювання програмного забезпечення. Крім того, було визначено ключові показники якості і описано процеси тестування, а також розгортання та супровід ПЗ.

Проєкт демонструє високу практичну значущість, оскільки створений плагін може бути використаний для автоматизації тестування різних типів веб застосунків, що забезпечує підвищення якості програмного забезпечення і зменшення витрат на його розробку та підтримку.

Даний проєкт відкритий до розширення та покращень для підтримки більшої кількості типів тестових кроків та більш складних тестових сценаріїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) What are extensions? | Manifest V2 | Chrome for Developers. *Chrome for Developers*. URL: https://developer.chrome.com/docs/extensions/mv2/overview (дата звернення: 11.06.2024).

2) Extensions / develop | chrome for developers. *Chrome for Developers*. URL: https://developer.chrome.com/docs/extensions/develop (дата звернення: 11.06.2024).

3) Large language models (llms) with google AI. *Google Cloud*. URL: https://cloud.google.com/ai/llms (дата звернення: 11.06.2024).

4) What is a large language model? | A comprehensive llms guide. *Elastic — The Search AI Company | Elastic*. URL: https://www.elastic.co/what-is/large-language-models (дата звернення: 11.06.2024).

5) What is NLP (natural language processing)? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/natural-language-processing (дата звернення: 11.06.2024).

6) The different types of testing in software | Atlassian. *Atlassian*. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (дата звернення: 11.06.2024).

7) Manual testing tutorial. *Guru99*. URL: https://www.guru99.com/manual-testing.html (дата звернення: 11.06.2024).

8) What is web UI testing? . *Sencha.com*. URL: https://www.sencha.com/blog/what-is-web-ui-testing/ (дата звернення: 11.06.2024).

9) 6 innovative ways to improve your software testing process. *Maruti Techlabs*. URL: https://marutitech.com/software-testing-improvement-ideas/ (дата звернення: 11.06.2024).

10) UI.Vision RPA. *Chrome Web Store*. URL: https://chromewebstore.google.com/detail/uivision-rpa/gcbalfbdmfieckjlnblleoemohcganoc (дата звернення: 11.06.2024).

11) Getting started · selenium IDE. *Selenium*. URL: https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started (дата звернення: 11.06.2024).

12) WebDriver. *W3C*. URL: https://www.w3.org/TR/webdriver2/ (дата звернення: 11.06.2024).

13) Selenium webdriver tutorial in java with examples | browserstack. *BrowserStack*. URL: https://www.browserstack.com/guide/selenium-webdriver-tutorial (дата звернення: 11.06.2024).

14) Chrome.extension | API | chrome for developers. *Chrome for Developers*. URL: https://developer.chrome.com/docs/extensions/reference/api/extension (дата звернення: 11.06.2024).

15) What is AI code generation? Benefits, tools & challenges. *Better Code & Better Software | Ultimate Security and Quality | Sonar*. URL: https://www.sonarsource.com/learn/ai-code-generation/ (дата звернення: 11.06.2024).

16) Archana V. OpenAI vs self hosted llms: a cost analysis. *ScaleDown | Vaidheeswaran Archana | Substack*. URL: https://tinyml.substack.com/p/openai-vs-self-hosted-llms-a-cost (дата звернення: 11.06.2024).

17) Getting started with WebStorm | WebStorm. *WebStorm Help*. URL: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html (дата звернення: 11.06.2024).

18) Advantages of TypeScript over JavaScript 2024 : aalpha. *Aalpha*. URL: https://www.aalpha.net/articles/advantages-of-typescript-over-javascript (дата звернення: 11.06.2024).

19) Frogh A. F. React vs Vanilla JavaScript. *Medium*. URL: https://blog.stackademic.com/react-vs-vanilla-javascript-e23e0f03e4c0 (дата звернення: 11.06.2024).

20) What are the advantages of using Redux with ReactJS ? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/what-are-the-advantages-of-using-redux-with-reactjs/ (дата звернення: 11.06.2024).

21) Bootstrap vs Material UI: a battle of two popular frameworks. *Material Tailwind - Easy-to-use Tailwind CSS components library with React and Material Design*. URL: https://www.material-tailwind.com/blog/bootstrap-vs-material-ui (дата звернення: 11.06.2024).

22) Best large language model apis in 2024 | eden AI. *Eden AI | Worfklow Builder | Combine Multiple AI Models for Your Business*. URL: https://www.edenai.co/post/best-large-language-model-apis (дата звернення: 11.06.2024).

ДОДАТКИ

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016348500

Дата перевірки:
12.06.2024 00:54:25 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
12.06.2024 07:03:13 EEST

ID користувача:
76913

Назва документа: **ІП-01_Москаленко_ПЗ**

Кількість сторінок: **67** Кількість слів: **10222** Кількість символів: **82143** Розмір файлу: **1.70 MB** ID файлу: **1016151417**

9.02%
Схожість

Найбільша схожість: **3.76%** з джерелом з Бібліотеки (ID файлу: **1016114009**)

2.68% Джерела з Інтернету	118	Сторінка 69
9.02% Джерела з Бібліотеки	280	Сторінка 70

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Текст програми

КПІ.ІП-0120.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав МОСКАЛЕНКО

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/Vladyslav2020/graduate-project

Файл TestSuites\index.tsx
Реалізація функціональної вимоги створення тестових сценаріїв.

```
import React, {useRef, useState} from "react";
import {TestSuite} from "../../interfaces/TestSuite";
import ChevronRightRoundedIcon from '@mui/icons-
material/ChevronRightRounded';
import KeyboardArrowDownRoundedIcon from '@mui/icons-
material/KeyboardArrowDownRounded';
import MoreVertRoundedIcon from '@mui/icons-material/MoreVertRounded';
import ShowChartRoundedIcon from '@mui/icons-material/ShowChartRounded';
import PublishRoundedIcon from '@mui/icons-material/PublishRounded';
import GetAppRoundedIcon from '@mui/icons-material/GetAppRounded';
import {
    Button,
    Dialog,
    DialogActions,
    DialogContent,
    DialogTitle,
    IconButton,
    Menu,
    MenuItem,
    TextField,
    Typography
} from "@mui/material";
import {TestCase} from "../../interfaces/TestCase";
import {ADD_TEST_SUITE, RootState, SET_ACTIVE_TEST_CASE, SET_TEST_SUITES,
SHOW_RUNS} from "../../redux/Reducers";
import {useDispatch, useSelector} from "react-redux";
import {generateUniqueId} from "../../utils";

type CommandType = {
    id: string;
    dialogTitle: string;
    handler: any;
}

export const TestSuites = () => {
    const testSuites = useSelector((state: RootState) =>
state.root.testSuites);
    const dispatch = useDispatch();
    const activeTestCase = useSelector((state: RootState) =>
state.root.activeTestCase);
    const [expandedTestSuites, setExpandedTestSuites] =
useState<string[]>(testSuites.map(suite => suite.id));
    const [dialogOpen, setDialogOpen] = useState(false);
    const [inputValue, setInputValue] = useState('');

    const [activeItem, setActiveItem] = useState<TestSuite | TestCase |
null>(null);

    const [anchorEl, setAnchorEl] = useState(null);

    const [command, setCommand] = useState<CommandType | null>(null);
    const fileInputRef = useRef(null);
```

```
    const handleMoreTestSuiteOptionsClick = (event, testSuite) => {
        setAnchorEl(event.currentTarget);
        setActiveItem(testSuite);
    }

    const handleMoreTestCaseOptionsClick = (event, testCase) => {
        event.stopPropagation();
        setAnchorEl(event.currentTarget);
        setActiveItem(testCase);
    };

    const handleShowRuns = (event, testSuite, testCase) => {
        event.stopPropagation();
        dispatch({
            type: SET_ACTIVE_TEST_CASE,
            testSuite: testSuite,
            testCase: testCase,
        });
        dispatch({type: SHOW_RUNS});
    }

    const handleAddTestSuiteClick = () => {
        setInputValue('');
        setCommand({id: 'add-test-suite', dialogTitle: 'Add Test Suite',
handler: handleAddTestSuite});
        setDialogOpen(true);
    }

    const handleAddTestCaseClick = (testSuite) => {
        setActiveItem(testSuite);
        setInputValue('');
        setCommand({id: 'add-test-case', dialogTitle: 'Add Test Case',
handler: handleAddTestCase});
        setDialogOpen(true);
    };

    const handleAddTestSuite = () => {
        dispatch({
            type: ADD_TEST_SUITE, testSuite: {
                id: generateUniqueId(),
                title: inputValue,
                testCases: []
            }
        });
        setDialogOpen(false);
    }

    const handleAddTestCase = () => {
        let activeTestSuite;
        const newTestCase: TestCase = {
            id: generateUniqueId(),
            title: inputValue,
            steps: [],
            runs: [],
        };
        const newTestSuites = testSuites.map(testSuite => {
            if (testSuite.id === activeItem?.id) {
                activeTestSuite = testSuite;
                return {
                    ...testSuite,
```

```
                    testCases: [...testSuite.testCases, newTestCase]
                };
            }
            return testSuite;
        });
        dispatch({
            type: SET_TEST_SUITES,
            testSuites: newTestSuites,
        });
        dispatch({
            type: SET_ACTIVE_TEST_CASE,
            testSuite: activeTestSuite,
            testCase: newTestCase,
        });
        setDialogOpen(false);
    }

    const handleEditTestSuite = () => {
        if (!activeItem) {
            return;
        }
        const newTestSuites = testSuites.map(testSuite => {
            if (testSuite.id === activeItem.id) {
                return {...testSuite, title: inputValue};
            }
            return testSuite;
        });
        dispatch({
            type: SET_TEST_SUITES,
            testSuites: newTestSuites,
        });
        setDialogOpen(false);
        setActiveItem(null);
    }

    const handleEditTestCase = () => {
        const newTestSuites = testSuites.map(testSuite => {
            return {
                ...testSuite, testCases: testSuite.testCases.map(testCase
=> {
                    if (testCase.id === activeItem?.id) {
                        return {...testCase, title: inputValue};
                    }
                    return testCase;
                })
            };
        });
        dispatch({
            type: SET_TEST_SUITES,
            testSuites: newTestSuites,
        });
        setDialogOpen(false);
        setActiveItem(null);
    };

    const handleEditClick = () => {
        if (!activeItem) {
            return;
        }
        if (activeItem.hasOwnProperty('testCases')) {
            setCommand({id: 'edit-test-suite', dialogTitle: 'Edit Test
```

```
Suite', handler: handleEditTestSuite});
        } else {
            setCommand({id: 'edit-test-case', dialogTitle: 'Edit Test
Case', handler: handleEditTestCase});
        }
        setDialogOpen(true);
        setInputValue(activeItem.title);
        setAnchorEl(null);
    };

    const handleRemoveTestCase = () => {
        const newTestSuites = testSuites.map(testSuite => {
            return {...testSuite, testCases:
testSuite.testCases.filter(testCase => testCase.id !== activeItem?.id)};
        });
        dispatch({
            type: SET_TEST_SUITES,
            testSuites: newTestSuites,
        });
        setDialogOpen(false);
        setActiveItem(null);
    };

    const handleRemoveTestSuite = () => {
        const newTestSuites = testSuites.filter(testSuite => testSuite.id
!== activeItem?.id);
        dispatch({
            type: SET_TEST_SUITES,
            testSuites: newTestSuites,
        });
    }

    const handleClose = () => {
        setAnchorEl(null);
    };

    const handleRemoveClick = () => {
        if (activeItem?.hasOwnProperty('testCases')) {
            handleRemoveTestSuite();
        } else {
            handleRemoveTestCase();
        }
        setAnchorEl(null);
    };

    const toggleTestSuite = (id: string) => {
        setExpandedTestSuites(prevState =>
            prevState.includes(id) ? prevState.filter(suiteId => suiteId
!== id) : [...prevState, id]
        );
    };

    const handleCloseDialog = () => {
        setDialogOpen(false);
    }

    const handleCommand = () => {
        switch (command?.id) {
            case 'add-test-suite':
                handleAddTestSuite();
                break;
```

```
            case 'add-test-case':
                handleAddTestCase();
                break;
            case 'edit-test-suite':
                handleEditTestSuite();
                break;
            case 'edit-test-case':
                handleEditTestCase();
                break;
        }
    }

    const setActiveTestCase = (testSuite, testCase) => {
        dispatch({
            type: SET_ACTIVE_TEST_CASE,
            testSuite: testSuite,
            testCase: testCase,
        });
    }

    const exportTestSuite = (testSuite: TestSuite) => {
        const jsonData = JSON.stringify({
            ...testSuite,
            testCases: testSuite.testCases.map(testCase => ({
                id: testCase.id,
                title: testCase.title,
                steps: testCase.steps
            }))
        });

        const blob = new Blob([jsonData], {type: 'application/json'});

        chrome.downloads.download({
            url: URL.createObjectURL(blob),
            filename: 'test-suite.json',
            saveAs: true
        }, (downloadId) => {
            if (chrome.runtime.lastError) {
                console.error('Error exporting test suite:',
chrome.runtime.lastError.message);
            } else {
                console.log('Test suite exported to test-suite.json');
            }
        });
    }

    const importTestSuite = () => {
        if (!fileInputRef.current) {
            return;
        }
        (fileInputRef.current as HTMLInputElement).click();
    }

    const handleImportFile = (event) => {
        const file = event.target.files[0];
        const reader = new FileReader();
        reader.onload = (e: any) => {
            if (!e.target.result) {
                return;
            }
            const jsonData = JSON.parse(e.target.result as string);
```

```
            if (!jsonData) {
                return;
            }
            const newTestSuite: TestSuite = {
                id: generateUniqueId(),
                title: jsonData.title,
                testCases: jsonData.testCases.map(testCase => ({
                    id: generateUniqueId(),
                    title: testCase.title,
                    steps: testCase.steps,
                    runs: []
                }))
            };
            dispatch({
                type: SET_TEST_SUITES,
                testSuites: [...testSuites, newTestSuite],
            });
        };
        reader.readAsText(file);
    }

    return (
        <div style={{padding: '0 20px', minWidth: '352px'}}>
            <div>
                <div style={{
                    display: 'flex',
                    justifyContent: 'space-between',
                    alignItems: 'center',
                    marginBottom: '20px'
                }}>
                    <Typography>Test Suites</Typography>
                    <div>
                        <input
                            ref={fileInputRef}
                            type="file"
                            accept="application/json"
                            hidden
                            onChange={handleImportFile}
                        />
                        <IconButton title='Import test suite'
color='primary' onClick={importTestSuite}>
                            <GetAppRoundedIcon/>
                        </IconButton>
                        <Button variant="outlined" size='small'
onClick={handleAddTestSuiteClick}>+ Test Suite</Button>
                    </div>
                </div>
                <Dialog open={dialogOpen} onClose={handleCloseDialog}>
                    <DialogTitle>{command?.dialogTitle}</DialogTitle>
                    <DialogContent>
                        <TextField autoFocus margin="dense" label="Title"
type="text" fullWidth value={inputValue}
                                   onChange={e =>
setInputValue(e.target.value)}/>
                    </DialogContent>
                    <DialogActions>
                        <Button
onClick={handleCloseDialog}>Cancel</Button>
                        <Button onClick={handleCommand}>Save</Button>
                    </DialogActions>
                </Dialog>
```

```
                <div>
                    {testSuites.map((testSuite) => (
                        <div key={testSuite.id} style={{marginBottom:
'10px'}}>
                            <div style={{
                                display: 'flex',
                                alignItems: 'center',
                                justifyContent: 'space-between',
                                borderBottom: '1px solid black'
                            }}>
                                <div style={{display: 'flex', alignItems:
'center'}}>
                                    <IconButton size='small' onClick={()
=> toggleTestSuite(testSuite.id)}>

{expandedTestSuites.includes(testSuite.id) ?
<KeyboardArrowDownRoundedIcon/> :
                                                <ChevronRightRoundedIcon/>}
                                    </IconButton>
                                    <Typography title={testSuite.title}
noWrap sx={{
                                        overflow: 'hidden',
                                        textOverflow: 'ellipsis',
                                        maxWidth: '140px',
                                    }}>{testSuite.title}</Typography>
                                </div>
                                <div style={{width: '178px'}}>
                                    <IconButton title='Export test suite'
                                                onClick={() =>
exportTestSuite(testSuite)}

color='primary'><PublishRoundedIcon/></IconButton>
                                    <Button size='small' variant="text"
                                            onClick={() =>
handleAddTestCaseClick(testSuite)}>+ Test Case</Button>
                                    <IconButton title='More actions'
                                                onClick={(event) =>
handleMoreTestSuiteOptionsClick(event, testSuite)}>
                                        <MoreVertRoundedIcon/>
                                    </IconButton>
                                </div>
                            </div>
                            {expandedTestSuites.includes(testSuite.id) &&
(
                                <div style={{marginLeft: '50px'}}>
                                    {testSuite.testCases.map((testCase) =>
(
                                        <Button key={testCase.id}
component='div'
                                                onClick={() =>
setActiveTestCase(testSuite, testCase)} color={'inherit'}
                                                variant='text' sx={{
                                            display: 'flex',
                                            padding: '0 0 0 10px',
                                            width: '100%',
                                            justifyContent: 'space-
between',
                                            alignItems: 'center',
                                            textTransform: 'none',
                                            backgroundColor:
activeTestCase?.id === testCase.id ? '#eaeaea' : 'inherit',
```

```
                                        }}>
                                            <Typography
title={testCase.title} noWrap
                                                        sx={{maxWidth:
'200px'}}>{testCase.title}</Typography>
                                            <div style={{width: '80px'}}>
                                                <IconButton title='View
statistics'

onClick={(event) => handleShowRuns(event, testSuite,
testCase)}><ShowChartRoundedIcon/></IconButton>
                                                <IconButton title='More
actions'

onClick={(event) => handleMoreTestCaseOptionsClick(event,
testCase)}><MoreVertRoundedIcon/></IconButton>
                                            </div>
                                        </Button>
                                    ))}
                                    <Menu
                                        id="simple-menu"
                                        anchorEl={anchorEl}
                                        keepMounted
                                        open={Boolean(anchorEl)}
                                        onClose={handleClose}
                                    >
                                        <MenuItem
onClick={handleEditClick}>Edit</MenuItem>
                                        <MenuItem
onClick={handleRemoveClick}>Remove</MenuItem>
                                    </Menu>
                                </div>
                            )}
                        </div>
                    ))}
                </div>
            </div>
        </div>
    );
};
```

Файл TestRun\index.tsx

Реалізація функціональної вимоги виконання тестових сценаріїв.

```
import {Box, Table, TableBody, TableHead, TableRow, Typography} from
"@mui/material"
import React, {useEffect} from "react"
import {useDispatch, useSelector} from "react-redux";
import {
    ADD_TEST_STEP_EXECUTION_RESULT,
    FINISH_TEST_CASE,
    RootState,
    START_TEST_STEP_EXECUTION
} from "../../redux/Reducers";
import {formatDuration, getActionDescriptor, getTestRunBackgroundColor}
from "../../utils";
import {TestCase} from "../../interfaces/TestCase";
import {CustomTableCell} from "../CustomTableCell";
import {TestRunStatusIcon} from "../TestRunStatusIcon";
```

```
type TestRunProps = {
    testCase: TestCase;
}

export const TestRunComponent = ({testCase}: TestRunProps) => {
    const testRun = useSelector((state: RootState) =>
state.root.activeTestRun);
    const dispatch = useDispatch();

    useEffect(() => {
        const handleMessage = (message: any) => {
            if (message.type === 'start-test-step-execution') {
                dispatch({
                    type: START_TEST_STEP_EXECUTION,
                    stepId: message.stepId,
                });
            }
            if (message.type === 'finish-test-step-execution') {
                dispatch({
                    type: ADD_TEST_STEP_EXECUTION_RESULT,
                    stepId: message.stepId,
                    status: message.status,
                    logs: message.logs,
                });
            }
            if (message.type === 'finish-test-case-execution') {
                dispatch({
                    type: FINISH_TEST_CASE,
                    testCaseId: testCase.id,
                    status: message.status,
                    logs: message.logs,
                    screenshot: message.screenshot,
                });
            }
        }

        chrome.runtime.onMessage.addListener(handleMessage);

        return () => {
            chrome.runtime.onMessage.removeListener(handleMessage);
        }
    }, [testCase]);

    return (
        <Box
            sx={{
                display: 'flex',
                flexDirection: 'column',
                m: 'auto',
                maxWidth: '750px',
            }}
        >
            <Typography variant='h6' align='center'>Run of Test Case:
{testCase.title}</Typography>
            <Table sx={{minWidth: 650}} aria-label="test run steps">
                <TableHead>
                    <TableRow>
                        <CustomTableCell>Action</CustomTableCell>
                        <CustomTableCell>Element</CustomTableCell>
                        <CustomTableCell>Value</CustomTableCell>
```

```
                        <CustomTableCell>Status</CustomTableCell>
                    </TableRow>
                </TableHead>
                <TableBody>
                    {testRun?.steps.map((step) =>
                        <TableRow
                            key={step.id}
                            sx={{
                                "&:last-child td, &:last-child th":
{border: 0},
                                backgroundColor:
getTestRunBackgroundColor(step),
                            }}
                        >

<CustomTableCell>{getActionDescriptor(step.name)?.label}</CustomTableCell>
                            <CustomTableCell>
                                {step.element}
                            </CustomTableCell>

<CustomTableCell>{step.value}</CustomTableCell>
                            <CustomTableCell>
                                <div style={{
                                    display: 'flex',
                                    alignItems: 'center'
                                }}><TestRunStatusIcon run={step}/></div>
                            </CustomTableCell>
                        </TableRow>
                    )}
                </TableBody>
            </Table>
            {testRun?.duration &&
                <Typography variant="body1">Duration:
{formatDuration(testRun.duration)}</Typography>}
            <Typography variant="h6">Logs</Typography>
            {testRun?.logs.map((log, index) =>
                <Typography key={index}>{log}</Typography>
            )}
            {testRun?.screenshot &&
                <img style={{maxWidth: '750px', height: 'auto'}}
src={testRun.screenshot}
                     alt="screenshot"/>}
        </Box>
    );
}
```

Файл TestRunner.ts

Реалізація функціональної вимоги виконання тестових сценаріїв.

```
import {TestCase} from "../../../Popup/interfaces/TestCase";
import {
    ActionExecutor,
    clickActionExecutor, doubleClickActionExecutor,
    ExecutionResult,
    openActionExecutor,
    pressKeyActionExecutor,
    typeActionExecutor, verifyEditableActionExecutor,
    verifyTextActionExecutor,
    verifyTitleActionExecutor,
```

```
    verifyValueActionExecutor, verifyVisibleActionExecutor
} from "./ActionExecutor";
import {TestStep} from "../../../Popup/interfaces/TestStep";
import {TestRunStatus} from "../../../Popup/interfaces/TestRun";

class TestRunner {
    private actionExecutors: { [key: string]: ActionExecutor } = {};

    public async runTestCase(testCase: TestCase, startTestStep?,
stepExecutionResult?) {
        try {
            let executionStarted = !startTestStep;
            for (const testStep of testCase.steps) {
                let executionResult = stepExecutionResult;
                if (!executionStarted && testStep.id !==
startTestStep?.id) {
                    continue;
                } else if (!executionStarted && testStep.id ===
startTestStep?.id) {
                    executionStarted = true;
                } else {
                    chrome.runtime.sendMessage({type: 'start-test-step-
execution', stepId: testStep.id});
                    if (testStep.name === 'open') {
                        chrome.storage.local.set({executionData:
{testCase: testCase, testStep: testStep}});
                    }
                    executionResult = await
this.actionExecutors[testStep.name].execute(testStep);
                }
                chrome.runtime.sendMessage({
                    type: 'finish-test-step-execution',
                    stepId: testStep.id,
                    status: executionResult.status,
                    logs: this.getLogs(executionResult, testStep)
                });
                if (executionResult.status === TestRunStatus.FAILED) {
                    const screenshot = await this.captureScreenshot();
                    chrome.runtime.sendMessage({
                        type: 'finish-test-case-execution',
                        status: TestRunStatus.FAILED,
                        logs: 'Test case failed',
                        screenshot,
                    });
                    return;
                }
            }
            chrome.runtime.sendMessage({
                type: 'finish-test-case-execution',
                status: TestRunStatus.PASSED,
                logs: 'Test case passed',
            });
        } catch (e: any) {
            chrome.runtime.sendMessage({
                type: 'finish-test-case-execution',
                status: TestRunStatus.FAILED,
                logs: 'Test case failed' + e.message ? `: ${e.message}` :
'',
            });
        }
    }
```

```
    private getLogs(executionResult: ExecutionResult, testStep: TestStep)
{
        return (executionResult.status === TestRunStatus.PASSED ? '[info]
' : '[error] ') + executionResult.message +
            (testStep.name === 'open' ? ' - ' + testStep.element : ' -
xpath=' + testStep.element);
    }

    public addActionExecutor(actionExecutor: ActionExecutor) {
        this.actionExecutors[actionExecutor.name] = actionExecutor;
    }

    private async captureScreenshot() {
        return new Promise((resolve) => {
            chrome.runtime.sendMessage({type: 'captureScreenshot'});
            const callback = (message, sender, sendResponse) => {
                if (message.type === 'capturedScreenshot') {
                    resolve(message.screenshot);
                    chrome.runtime.onMessage.removeListener(callback);
                }
            };
            chrome.runtime.onMessage.addListener(callback);
            setTimeout(() => {
                resolve(null)
            }, 1000);
        });
    }
}

export const testRunner = new TestRunner();
testRunner.addActionExecutor(openActionExecutor);
testRunner.addActionExecutor(clickActionExecutor);
testRunner.addActionExecutor(doubleClickActionExecutor);
testRunner.addActionExecutor(typeActionExecutor);
testRunner.addActionExecutor(pressKeyActionExecutor);
testRunner.addActionExecutor(verifyValueActionExecutor);
testRunner.addActionExecutor(verifyTextActionExecutor);
testRunner.addActionExecutor(verifyTitleActionExecutor);
testRunner.addActionExecutor(verifyEditableActionExecutor);
testRunner.addActionExecutor(verifyVisibleActionExecutor);
```

Файл ActionExecutor.ts

Реалізація функціональної вимоги виконання тестових сценаріїв.

```
import {TestStep} from "../../../Popup/interfaces/TestStep";
import {waitForElementByXPath} from "../utils";
import {TestRunStatus} from "../../../Popup/interfaces/TestRun";

const EDITABLE_TAGS = ['input', 'textarea', 'select'];

export interface ExecutionResult {
    status: TestRunStatus;
    message: string;
}

export interface ActionExecutor {
    name: string;
```

```
    execute(testStep: TestStep): Promise<ExecutionResult>;
}

class OpenActionExecutor implements ActionExecutor {
    name = 'open';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        const url = testStep.element;

        return new Promise(async (resolve) => {
            function handleError() {
                resolve({status: TestRunStatus.FAILED, message: 'Failed to
load the URL'});
                setTimeout(() => window.location.href, 500);
            }

            try {
                const response = await fetch(url, { method: 'HEAD' });
                if (!response.ok) {
                    handleError();
                }
                window.location.href = url;
            } catch (error) {
                handleError();
            }
        });
    }
}

class ClickActionExecutor implements ActionExecutor {
    name = 'click';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element);
            this.clickElement(element);
            return {status: TestRunStatus.PASSED, message: 'Executed
action: Click element'};
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }

    private clickElement(element): void {
        const clickEvent = new MouseEvent('click', {
            bubbles: true,
            cancelable: true,
            view: window,
        });

        element.dispatchEvent(clickEvent);
    }
}

class DoubleClickActionExecutor implements ActionExecutor {
    name = 'doubleClick';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element);
```

```
            this.doubleClickElement(element);
            return {status: TestRunStatus.PASSED, message: 'Executed
action: Double click element'};
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }

    private doubleClickElement(element): void {
        const clickEvent = new MouseEvent('dblclick', {
            bubbles: true,
            cancelable: true,
            view: window,
        });

        element.dispatchEvent(clickEvent);
    }
}

class TypeActionExecutor implements ActionExecutor {
    name = 'type';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element);
            if (element.tagName !== 'INPUT') {
                throw new Error('Element not found');
            }

            this.typeIntoInput(element, testStep.value);
            return {status: TestRunStatus.PASSED, message: `Executed
action: Type value <${testStep.value}>`};
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }

    private typeIntoInput(inputElement, text): void {
        inputElement.value = text;

        const inputEvent = new Event('input', {
            bubbles: true,
            cancelable: true,
        });
        inputElement.dispatchEvent(inputEvent);
    }
}

class PressKeyActionExecutor implements ActionExecutor {
    name = 'pressKey';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element);
            if (element.tagName !== 'INPUT') {
                throw new Error('Element not found');
            }

            this.pressKey(element, testStep.value);
```

```
            return {status: TestRunStatus.PASSED, message: `Executed
action: Press key <${testStep.value}>`};
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }

    private pressKey(inputElement, key): void {
        const eventInitDict = {
            bubbles: true,
            cancelable: true,
            code: key === '${enter}' ? 'Enter' : 'Tab',
            key: key === '${enter}' ? 'Enter' : 'Tab',
            keyCode: key === '${enter}' ? 13 : 9,
        };
        const keyDownEvent = new KeyboardEvent('keydown', eventInitDict);
        const keyPressEvent = new KeyboardEvent('keypress',
eventInitDict);

        inputElement.focus();
        inputElement.dispatchEvent(keyDownEvent);
        inputElement.dispatchEvent(keyPressEvent);
    }
}

class VerifyValueActionExecutor implements ActionExecutor {
    name = 'verifyValue';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element);
            if (element.tagName !== 'INPUT') {
                throw new Error('Element not found');
            }

            const value = (element as HTMLInputElement).value;
            if (value === testStep.value) {
                return {status: TestRunStatus.PASSED, message: `Executed
action: Verify value <${value}>`};
            } else {
                return {
                    status: TestRunStatus.FAILED,
                    message: `Expected value <${testStep.value}>, but
found <${value}>`
                };
            }
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }
}

class VerifyTextActionExecutor implements ActionExecutor {
    name = 'verifyText';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element)
as HTMLElement;
```

```
            const text = element.innerText;
            if (text === testStep.value) {
                return {status: TestRunStatus.PASSED, message: `Executed
action: Verify text <${text}>`};
            } else {
                return {
                    status: TestRunStatus.FAILED,
                    message: `Expected text <${testStep.value}>, but found
<${text}>`
                };
            }
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }
}

class VerifyTitleActionExecutor implements ActionExecutor {
    name = 'verifyTitle';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element)
as HTMLElement;
            const title = element.title;
            if (title === testStep.value) {
                return {status: TestRunStatus.PASSED, message: `Executed
action: Verify title <${title}>`};
            } else {
                return {
                    status: TestRunStatus.FAILED,
                    message: `Expected title <${testStep.value}>, but
found <${title}>`
                };
            }
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }
}

class VerifyEditableActionExecutor implements ActionExecutor {
    name = 'verifyEditable';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element)
as HTMLInputElement;
            const editable =
EDITABLE_TAGS.includes(element.tagName.toLowerCase()) && !element.disabled
&& !element.readOnly || element.contentEditable === 'true';
            if (editable) {
                return {status: TestRunStatus.PASSED, message: `Executed
action: Verify editable <${editable}>`};
            } else {
                return {
                    status: TestRunStatus.FAILED,
                    message: 'Expected editable, but found not editable'
                };
```

```
            }
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }
}

class VerifyVisibleActionExecutor implements ActionExecutor {
    name = 'verifyVisible';

    async execute(testStep: TestStep): Promise<ExecutionResult> {
        try {
            const element = await waitForElementByXPath(testStep.element)
as HTMLElement;
            // check visibility hidden
            const visible = element.style.display !== 'none' &&
element.style.visibility !== 'hidden';
            if (visible) {
                return {status: TestRunStatus.PASSED, message: `Executed
action: Verify visible <${visible}>`};
            } else {
                return {
                    status: TestRunStatus.FAILED,
                    message: 'Expected visible, but found not visible'
                };
            }
        } catch (e) {
            return {status: TestRunStatus.FAILED, message: 'Element not
found'};
        }
    }
}

export const openActionExecutor = new OpenActionExecutor();
export const clickActionExecutor = new ClickActionExecutor();
export const doubleClickActionExecutor = new DoubleClickActionExecutor();
export const typeActionExecutor = new TypeActionExecutor();
export const pressKeyActionExecutor = new PressKeyActionExecutor();
export const verifyValueActionExecutor = new VerifyValueActionExecutor();
export const verifyTextActionExecutor = new VerifyTextActionExecutor();
export const verifyTitleActionExecutor = new VerifyTitleActionExecutor();
export const verifyEditableActionExecutor = new
VerifyEditableActionExecutor();
export const verifyVisibleActionExecutor = new
VerifyVisibleActionExecutor();
```

Файл TestRuns\index.tsx

Реалізація функціональної вимоги збору статистики виконання тестових сценаріїв.

```
import React from "react";
import {CustomTableCell} from "../CustomTableCell";
import {capitalizeFirstLetter, formatDuration, getTestRunBackgroundColor,
getTestStepIconColor} from "../../utils";
import {Box, Table, TableBody, TableHead, TableRow, Typography} from
"@mui/material";
import {TestCase} from "../../interfaces/TestCase";
```

```
import {TestRunStatusIcon} from "../TestRunStatusIcon";
import {
    Bar,
    BarChart,
    CartesianGrid,
    Legend,
    Line,
    LineChart,
    ReferenceLine,
    ResponsiveContainer,
    Tooltip,
    XAxis,
    YAxis
} from "recharts";
import {CustomTableRow} from "../CustomTableRow";
import {TestRun} from "../../interfaces/TestRun";
import {useDispatch} from "react-redux";
import {SET_TEST_RUN} from "../../redux/Reducers";

type TestRunProps = {
    testCase: TestCase;
}

export const TestRuns = ({testCase}: TestRunProps) => {
    const dispatch = useDispatch();

    const runStatus2Count = testCase.runs.reduce((acc, run) => {
        const group = acc[run.status] || 0;
        acc[run.status] = group + 1;
        return acc;
    }, {});

    const statusData = [{name: 'status', ...runStatus2Count}];

    const runDurationData = testCase.runs.filter(run => run.duration)
        .map(run => ({start: run.start.toLocaleString(), duration:
run.duration}));

    const calculateMeanDuration = (runs) => {
        const sum = runs.reduce((acc, run) => acc + (run.duration ?
run.duration : 0), 0);
        return sum / runs.length;
    };

    const meanDuration = testCase.runs.length > 1 ?
calculateMeanDuration(testCase.runs) : 0;

    const openTestRun = (testRun: TestRun) => {
        dispatch({type: SET_TEST_RUN, testRun});
    }

    return (
        <Box>
            <Typography variant='h6' align='center'>Runs of Test Case:
{testCase.title}</Typography>
            {testCase.runs.length === 0 &&
                <Typography variant='body1' align='center' sx={{color:
'gray'}}>No runs yet</Typography>}
            {testCase.runs.length > 0 && <Table sx={{minWidth: '650px'}}
aria-label="test runs">
                <TableHead>
```

```
                    <TableRow>
                        <CustomTableCell>#</CustomTableCell>
                        <CustomTableCell>Start time</CustomTableCell>
                        <CustomTableCell>Duration</CustomTableCell>
                        <CustomTableCell>Status</CustomTableCell>
                    </TableRow>
                </TableHead>
                <TableBody>
                    {testCase.runs.map((run, index) =>
                        <CustomTableRow
                            key={run.id}
                            sx={{
                                "&:last-child td, &:last-child th":
{border: 0},
                                backgroundColor:
getTestRunBackgroundColor(run),
                            }}
                            onClick={() => openTestRun(run)}
                        >
                            <CustomTableCell>{index + 1}</CustomTableCell>

<CustomTableCell>{run.start.toLocaleString()}</CustomTableCell>
                            <CustomTableCell>{!run.duration ? 'unknown' :
formatDuration(run.duration)}</CustomTableCell>
                            <CustomTableCell>
                                <div style={{
                                    display: 'flex',
                                    alignItems: 'center'
                                }}><TestRunStatusIcon run={run}/></div>
                            </CustomTableCell>
                        </CustomTableRow>
                    )}
                </TableBody>
            </Table>}
            {testCase.runs.length > 1 && <Typography>Mean duration:
{formatDuration(meanDuration)}</Typography>}
            {Object.getOwnPropertyNames(runStatus2Count).map((status,
index) => <div key={index}><Typography
                sx={{color: getTestStepIconColor({status: status}),
display: 'inline'}}>{capitalizeFirstLetter(status)}</Typography>
                <Typography sx={{display: 'inline'}}>:
{runStatus2Count[status]}</Typography>
            </div>)}
            {testCase.runs.length > 0 && <Box sx={{display: 'flex'}}>
                <ResponsiveContainer width='40%' height={400}>
                    <BarChart margin={{top: 20}} data={statusData}>
                        <CartesianGrid strokeDasharray="3 3"/>
                        <XAxis dataKey="name"/>
                        <YAxis/>
                        <Tooltip/>
                        <Legend/>

{Object.getOwnPropertyNames(runStatus2Count).map((name, index) => (
                            <Bar key={index} dataKey={name}
                                 fill={getTestStepIconColor({status:
name})}/>
                        ))}
                    </BarChart>
                </ResponsiveContainer>
                <ResponsiveContainer width='60%' height={400}>
                    <LineChart margin={{top: 20}} data={runDurationData}>
```

```
                        <CartesianGrid strokeDasharray="3 3"/>
                        <XAxis dataKey="start"/>
                        <YAxis/>
                        <Tooltip/>
                        <Legend/>
                        <Line type="monotone" dataKey="duration"
stroke="#8884d8"/>
                        {testCase.runs.length > 1 &&
                            <ReferenceLine y={meanDuration}
stroke="#f57d05"/>}
                    </LineChart>
                </ResponsiveContainer>
            </Box>}
        </Box>
    );
}
```

Файл CodeGenerationDialog\index.tsx

Реалізація функціональної вимоги генерації коду для створеного тестового сценарію.

```
import React, {useEffect, useRef, useState} from 'react';
import ContentCopyRoundedIcon from '@mui/icons-
material/ContentCopyRounded';
import {Box, Button, Dialog, DialogActions, DialogContent, DialogTitle,
IconButton, TextField} from '@mui/material';
import {CopyToClipboard} from 'react-copy-to-clipboard';
import {TestCase} from "../../interfaces/TestCase";
import {aiGenerationService} from "../../services/AIGenerationService";
import ReactMarkdown from 'react-markdown';
import hljs from 'highlight.js/lib/core';
import javascript from 'highlight.js/lib/languages/javascript';
import remarkHighlight from 'remark-highlight.js';
import remarkGfm from 'remark-gfm';
import 'highlight.js/styles/default.css';
import './index.css';

hljs.registerLanguage('javascript', javascript);

type CodeGenerationDialogProps = {
    testCase: TestCase;
    open: boolean;
    setOpen: (open: boolean) => void;
};

export const CodeGenerationDialog = ({testCase, open, setOpen}:
CodeGenerationDialogProps) => {
    const [generatedCode, setGeneratedCode] = useState(`\`\`\`javascript
// code will be generated here
\`\`\``);
    const [codeGenerating, setCodeGenerating] = useState(false);
    const [language, setLanguage] = useState('JavaScript');
    const codeFieldRef = useRef<HTMLDivElement>(null);

    const handleClose = () => {
        setOpen(false);
    };
```

```
    const handleGenerate = async () => {
        setCodeGenerating(true);
        await aiGenerationService.getCodeForTestCase(testCase,
setGeneratedCode);
        setCodeGenerating(false);
    };

    useEffect(() => {
        if (codeFieldRef.current) {
            const codeSnippet = codeFieldRef.current.querySelector('.code-
snippet') as HTMLDivElement;
            codeSnippet.scrollTop = codeSnippet.scrollHeight;
        }
    }, [generatedCode]);

    const components: any = {
        code: ({node, inline, className, children, ...props}) => {
            const match = /language-(\w+)/.exec(className || '');
            return !inline && match ? (
                <pre className={className} {...props}>
                  <code className={`hljs ${match[0]}`}>{children}</code>
                </pre>
            ) : (
                <code className={className} {...props}>
                    {children}
                </code>
            );
        }
    };

    useEffect(() => {
        document.querySelectorAll('pre code').forEach((block) => {
            hljs.highlightBlock(block as HTMLElement);
        });
    }, [generatedCode]);

    return (
        <Dialog open={open} onClose={handleClose} maxWidth="md" fullWidth>
            <DialogTitle>Code Generation for Test Case:
{testCase.title}</DialogTitle>
            <DialogContent>
                <Box mt={2}>
                    <TextField
                        label="Language"
                        value={language}
                        InputProps={{
                            readOnly: true,
                        }}
                    />
                </Box>
                <Box ref={codeFieldRef} sx={{backgroundColor: '#F3F3F3'}}
                     mt={2} position="relative">
                    <ReactMarkdown className='code-snippet'
remarkPlugins={[remarkGfm, remarkHighlight as any]}

components={components}>{generatedCode}</ReactMarkdown>
                    <CopyToClipboard text={generatedCode}>
                        <IconButton title='Copy to Clipboard' sx={{
                            position: 'absolute',
                            top: '8px',
                            right: '8px',
```

```
                        }}>
                            <ContentCopyRoundedIcon/>
                        </IconButton>
                    </CopyToClipboard>
                </Box>
            </DialogContent>
            <DialogActions>
                <Button variant="contained" disabled={codeGenerating}
onClick={handleGenerate}>
                    Generate
                </Button>
                <Button variant="outlined" onClick={handleClose}>
                    Close
                </Button>
            </DialogActions>
        </Dialog>
    );
};
```

Файл AIGenerationService.ts

Реалізація функціональної вимоги генерації коду для створеного тестового сценарію.

```
import OpenAI from 'openai';
import {TestCase} from "../interfaces/TestCase";
import {ChatCompletionMessageParam} from 'openai/resources';

const openai = new OpenAI({apiKey: process.env.OPENAI_API_KEY,
dangerouslyAllowBrowser: true});

class AIGenerationService {
    private readonly systemMessage: ChatCompletionMessageParam = {
        role: 'system',
        content: 'User will give you steps for UI test case containing
consequent steps like open url, click element by xpath and so on. Your
task is to automate this test case with JavaScript (Selenium) code
including test method for testing in Chrome browser environment. The
result should only include code without additional instructions.'
    };

    async getCodeForTestCase(testCase: TestCase, callback):
Promise<string> {
        try {
            return this.streamChatGPTResponse(JSON.stringify({
                title: testCase.title,
                steps: testCase.steps
            }), callback);
        } catch (error) {
            console.error('Error fetching ChatGPT response:', error);
            return '';
        }
    }

    private async streamChatGPTResponse(userPrompt: string, callback):
Promise<string> {
        const stream: any = await openai.chat.completions.create({
```

```
            model: 'gpt-3.5-turbo',
            messages: [this.systemMessage, {role: 'user', content:
userPrompt}],
            stream: true,
        });

        let partialResponse = '';
        for await (const chunk of stream) {
            partialResponse += chunk.choices[0]?.delta?.content || '';
            callback(partialResponse);
        }

        return partialResponse;
    }
}

export const aiGenerationService = new AIGenerationService();
```

Файл TestCaseSteps\index.tsx
Реалізація функціональної вимоги підтримки тестових сценаріїв.

```
import React, {useEffect, useState} from "react";
import {closestCorners, DndContext, PointerSensor, useSensor, useSensors}
from "@dnd-kit/core";
import {arrayMove, SortableContext, verticalListSortingStrategy} from
"@dnd-kit/sortable";
import {TestCaseStep} from "../TestCaseStep";
import {Button, Table, TableBody, TableHead, TableRow, Typography} from
"@mui/material";
import {TestStep} from "../../interfaces/TestStep";
import {RootState, SET_TEST_STEPS} from "../../redux/Reducers";
import {useDispatch, useSelector} from "react-redux";
import {actionsDescriptors, generateUniqueId} from "../../utils";
import {CustomTableCell} from "../CustomTableCell";

type TestCaseStepsProps = {
    recordingEnabled: boolean;
    recordingTab: any;
    setRecordingTab: (arg: any) => void;
}

export const TestCaseSteps = ({recordingEnabled, recordingTab,
setRecordingTab}: TestCaseStepsProps) => {
    const testCase = useSelector((state: RootState) =>
state.root.activeTestCase);
    const dispatch = useDispatch();
    const [editingStep, setEditingStep] = useState(null);
    const [locatorEnabled, setLocatorEnabled] = useState(false);

    const sensors = useSensors(useSensor(PointerSensor, {
        activationConstraint: {
            distance: 5
        }
    }));

    const setTestStep = (testStep: TestStep) => {
        dispatch({
            type: SET_TEST_STEPS,
            steps: testCase?.steps.map(step => step.id === testStep.id ?
```

```
testStep : step)
        });
    }

    const handleDragEnd = (e) => {
        const {active, over} = e;

        if (!active || !over || active.id === over.id) {
            return;
        }

        const getStepIndex = (id) => testCase?.steps.findIndex((step) =>
step.id === id);

        const originalPos = getStepIndex(active.id) as number;
        const newPos = getStepIndex(over.id) as number;

        dispatch({
            type: SET_TEST_STEPS,
            steps: arrayMove(testCase?.steps as TestStep[], originalPos,
newPos)
        });
    };

    const handleStepEditing = (id) => {
        if (editingStep === id) {
            setEditingStep(null);
        } else {
            setEditingStep(id);
        }
        setLocatorEnabled(false);
        chrome.runtime.sendMessage({command: 'disable-locator-
selection'});
    }

    const handleLocatorEnable = () => {
        if (locatorEnabled) {
            setLocatorEnabled(false);
            chrome.runtime.sendMessage({command: 'disable-locator-
selection'});
        } else {
            setLocatorEnabled(true);
            chrome.runtime.sendMessage({command: 'enable-locator-
selection'});
        }
    }

    const clearTestStep = (id) => {
        dispatch({
            type: SET_TEST_STEPS,
            steps: testCase?.steps.filter(step => step.id !== id)
        });
    }

    useEffect(() => {
        const handleMessage = (message, sender, sendResponse) => {
            if (!sender.url || (!sender.tab &&
!sender.url.includes('background'))) {
                return;
            }
            const steps = testCase?.steps as TestStep[];
```

```
            if (locatorEnabled && message.locator) {
                dispatch({
                    type: SET_TEST_STEPS,
                    steps: steps?.map(step => step.id === editingStep ? {
                        ...step,
                        element: message.locator
                    } : step)
                });
                setLocatorEnabled(false);
                chrome.runtime.sendMessage({command: 'disable-locator-
selection'});
            }
            if (recordingEnabled && message.type === 'captureTestStep') {
                const newStep = message.step;
                if (actionsDescriptors.some(actionDescriptor =>
actionDescriptor.name === newStep.action)) {
                    const newTestSteps: any[] = [];
                    if (sender.tab && sender.tab.id !== recordingTab?.id)
{
                        newTestSteps.push({
                            id: generateUniqueId(),
                            name: 'open',
                            element: sender.tab.url,
                            value: '',
                        });
                        setRecordingTab(sender.tab);
                    }
                    newTestSteps.push({
                        id: generateUniqueId(),
                        name: newStep.action,
                        element: newStep.element,
                        value: newStep.value
                    });
                    dispatch({
                        type: SET_TEST_STEPS,
                        steps: [...steps, ...newTestSteps],
                    });
                }
            }
        }

        chrome.runtime.onMessage.addListener(handleMessage);

        return () => {
            chrome.runtime.onMessage.removeListener(handleMessage);
        }
    }, [editingStep, locatorEnabled, setLocatorEnabled, testCase,
recordingEnabled, recordingTab, setRecordingTab]);

    const addTestStep = () => {
        const steps = testCase?.steps as TestStep[];
        dispatch({
            type: SET_TEST_STEPS,
            steps: [...steps, {
                id: generateUniqueId(),
                name: 'click',
                element: '/html/body'
            }]
        });
    }
```

```
    return (
        <div>
            <Typography variant='h6' align='center'>Test Case:
{testCase?.title}</Typography>
            <DndContext sensors={sensors}
collisionDetection={closestCorners} onDragEnd={handleDragEnd}>
                <Table sx={{minWidth: 650}} aria-label="commands table">
                    <TableHead>
                        <TableRow>
                            <CustomTableCell>#</CustomTableCell>
                            <CustomTableCell>Command</CustomTableCell>
                            <CustomTableCell>Element</CustomTableCell>
                            <CustomTableCell>Value</CustomTableCell>
                            <CustomTableCell>Clear</CustomTableCell>
                        </TableRow>
                    </TableHead>
                    <TableBody>
                        <SortableContext items={testCase?.steps as
TestStep[]} strategy={verticalListSortingStrategy}>
                            {testCase?.steps.map((step) => (
                                <TestCaseStep key={step.id} id={step.id}
testStep={step} setTestStep={setTestStep}
                                              locatorEnabled={step.id ===
editingStep && locatorEnabled}
                                              onRemove={clearTestStep}
handleStepEditing={handleStepEditing}

handleLocatorEnable={handleLocatorEnable}
                                              activated={step.id ===
editingStep}/>
                            ))}
                        </SortableContext>
                    </TableBody>
                </Table>
            </DndContext>
            {testCase?.steps?.length === 0 &&
                <Typography variant='body1' align='center' sx={{color:
'gray'}}>No steps yet</Typography>}
            <Button size='small' variant='text' sx={{marginTop: '5px'}}
onClick={addTestStep}>+ Test Step</Button>
        </div>
    );
}
```

Файл TestCaseStep\index.tsx
Реалізація функціональної вимоги підтримки тестових сценаріїв.

```
import DragIndicatorIcon from "@mui/icons-material/DragIndicator";
import {TestStep} from "../../interfaces/TestStep";
import {useSortable} from "@dnd-kit/sortable";
import {CSS} from "@dnd-kit/utilities";
import React from "react";
import {FormControl, IconButton, InputLabel, MenuItem} from
"@mui/material";
import ClearOutlinedIcon from '@mui/icons-material/ClearOutlined';
import LocationSearchingIcon from '@mui/icons-material/LocationSearching';
import {actionsDescriptors, getActionDescriptor} from "../../utils";
import {CustomTableRow} from "../CustomTableRow";
import {CustomTableCell} from "../CustomTableCell";
```

```
import {CustomTextField} from "../CustomTextField";
import {CustomSelect} from "../CustomSelect";


type TestStepProps = {
    id: string;
    testStep: TestStep;
    setTestStep: (TestStep) => void;
    handleStepEditing: (id: string) => void;
    handleLocatorEnable: () => void;
    onRemove: (id: string) => void;
    locatorEnabled: boolean;
    activated: boolean;
}

export const TestCaseStep = ({
                                 id,
                                 testStep,
                                 setTestStep,
                                 onRemove,
                                 locatorEnabled,
                                 activated,
                                 handleStepEditing,
                                 handleLocatorEnable
                             }: TestStepProps) => {
    const {attributes, listeners, setNodeRef, transform, transition} =
useSortable({id});

    const style = {
        transition,
        ...(transform ? {transform: CSS.Transform.toString({...transform,
x: 0})} : {})
    };

    const handleRemove = () => {
        onRemove(id);
    }

    const handleActionChange = (event) => {
        setTestStep({...testStep, name: event.target.value});
    }

    const handleElementChange = (event) => {
        setTestStep({...testStep, element: event.target.value});
    }

    const handleValueChange = (event) => {
        setTestStep({...testStep, value: event.target.value});
    }

    const handleActivation = () => {
        handleStepEditing(testStep.id);
    }

    return (
        <CustomTableRow
            sx={{"&:last-child td, &:last-child th": {border: 0}}}
            style={style}
            {...attributes}
            {...listeners}
            onDoubleClick={handleActivation}
```

```
        >
            <CustomTableCell
                component="th"
                scope="row"
                ref={setNodeRef}
            >
                <div style={{display: 'flex', alignItems: 'center'}}>
                    <DragIndicatorIcon/>
                </div>
            </CustomTableCell>
            <CustomTableCell>{!activated ?
getActionDescriptor(testStep.name)?.label :
                <FormControl fullWidth>
                    <InputLabel id="select-action-
label">Action</InputLabel>
                    <CustomSelect
                        labelId="select-action-label"
                        id="select-action"
                        value={testStep.name}
                        label="Action"
                        onChange={handleActionChange}
                    >
                        {actionsDescriptors.map(actionDescriptor =>
<MenuItem key={actionDescriptor.name}

value={actionDescriptor.name}>{actionDescriptor.label}</MenuItem>)}
                    </CustomSelect>
                </FormControl>
            }</CustomTableCell>
            <CustomTableCell>
                {!activated ? testStep.element : <div style={{display:
'flex', alignItems: 'center'}}>
                    <CustomTextField value={testStep.element}
onChange={handleElementChange}/>
                    {getActionDescriptor(testStep.name)?.elementType ===
'html' &&
                        <IconButton title='Locate HTML element'
onClick={handleLocatorEnable}
                                    color={locatorEnabled ? 'primary' :
'default'}

size={'small'}><LocationSearchingIcon/></IconButton>}
                </div>}
            </CustomTableCell>
            <CustomTableCell>{!activated ? testStep.value :
                <CustomTextField value={testStep.value}
onChange={handleValueChange}/>}</CustomTableCell>
            <CustomTableCell onClick={handleRemove}>
                <IconButton title='Remove test step' size='small'
onClick={handleRemove}><ClearOutlinedIcon/></IconButton>
            </CustomTableCell>
        </CustomTableRow>
    )
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Програма та методика тестування

КПІ.ІП-0120.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав МОСКАЛЕНКО

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є плагін браузера Google Chrome, який надає можливість створення автоматизованих тестів для веб застосунків. Оскільки плагін не може функціонувати самостійно, то тестування має проводитися в контексті браузера.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів Google Chrome;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування (End-to-end) з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на сумісність із браузером Google Chrome;

– тестування на різних версіях браузера Google Chrome;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Керівництво користувача

КПІ.ІП-0120.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав МОСКАЛЕНКО

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Test automation – це плагін браузера Google Chrome, який було розроблено для автоматизації тестування веб застосунків. Даний плагін було завантажено та опубліковано в Chrome Web Store.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність браузера Google Chrome зі стабільною версією 124.0.6367.60 і вище;

– наявність доступу до Інтернету.

2.2 Завантаження застосунку

На даний момент плагін можна встановити використовуючи Chrome Web Store. Потім потрібно натиснути кнопку «Add to Chrome» і потім у спливаючому вікні погодитися із запрошеними дозволами, натиснувши кнопку «Add extension». Далі потрібно почекати до завершення завантаження. Після цього цей плагін буде додано у списку плагінів браузера.

2.3 Перевірка коректної роботи

По завершенню встановлення плагіну у верхньому правому кутку браузера Google Chrome у списку плагінів має додатися новий плагін «Test automation». У разі, якщо даний плагін не з’явився у списку, то встановлення відбулось не успішно. Інакше користувач має змогу запустити плагін, натиснувши на відповідний елемент списку. Після натискання повинне відкритися вікно із початковою сторінкою плагіну.

3 ВИКОНАННЯ ПРОГРАМИ

Для використання плагіну потрібно натиснути на його іконку у правому верхньому кутку браузера де відображено список встановлених плагінів. Після натискання буде відкрито головне вікно плагіну (рис. 3.1).

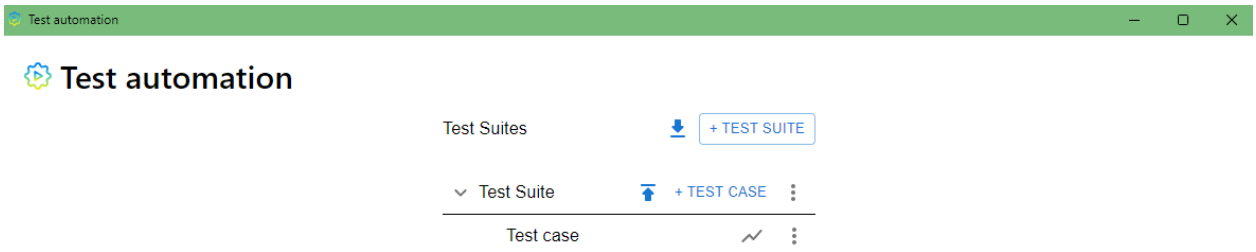


Рисунок 3.1 – Початкова сторінка плагіну

Далі можна додати новий тестовий набір натискаючи на кнопку «+ Test suite». У відкритому діалоговому вікні потрібно ввести його назву і натиснути кнопку «Save» (рис. 3.2). Після цього він відобразиться у списку тестових наборів.

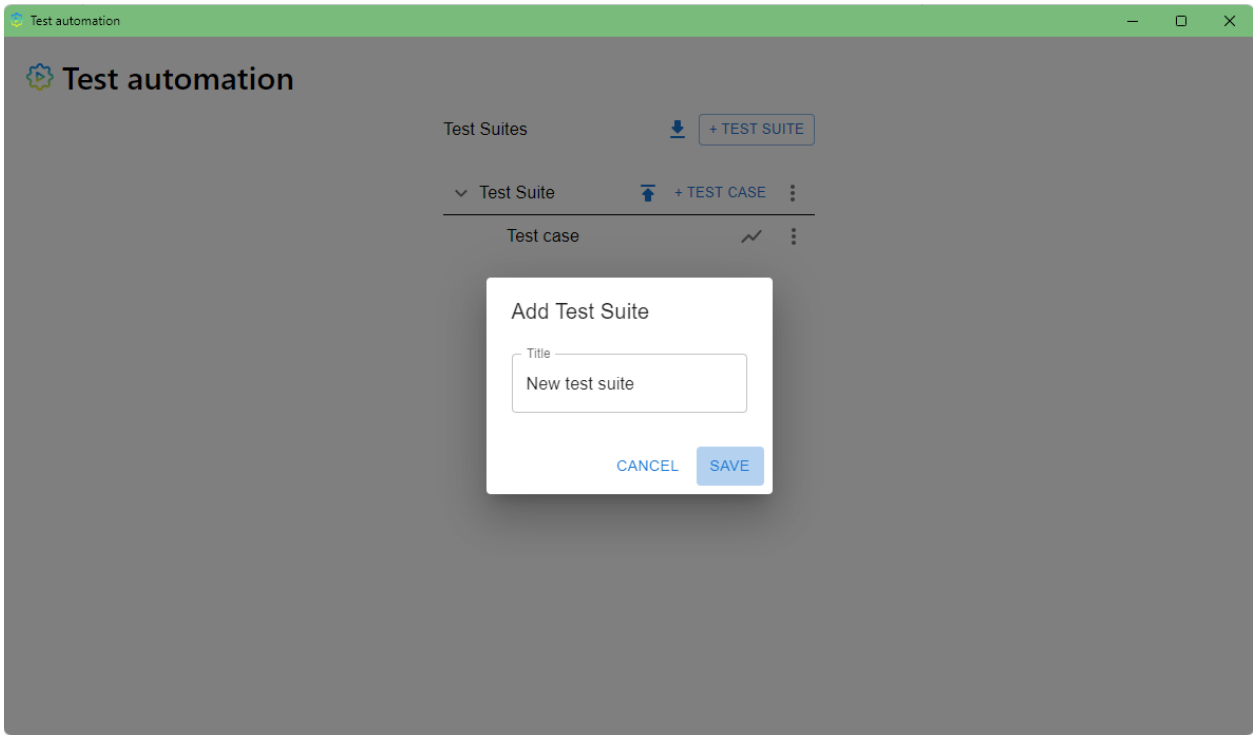


Рисунок 3.2 – Створення нового тестового набору

Тепер можна додати новий тестовий сценарій у створений набір. Для цього потрібно натиснути кнопку «+ Test case». У відкритому діалогу потрібно ввести його назву і підтвердити створення, натиснувши кнопку «Save» (рис. 3.3). Після цього створений тестовий сценарій буде додано у тестовий набір і на даний момент сценарій не містить жодного кроку.

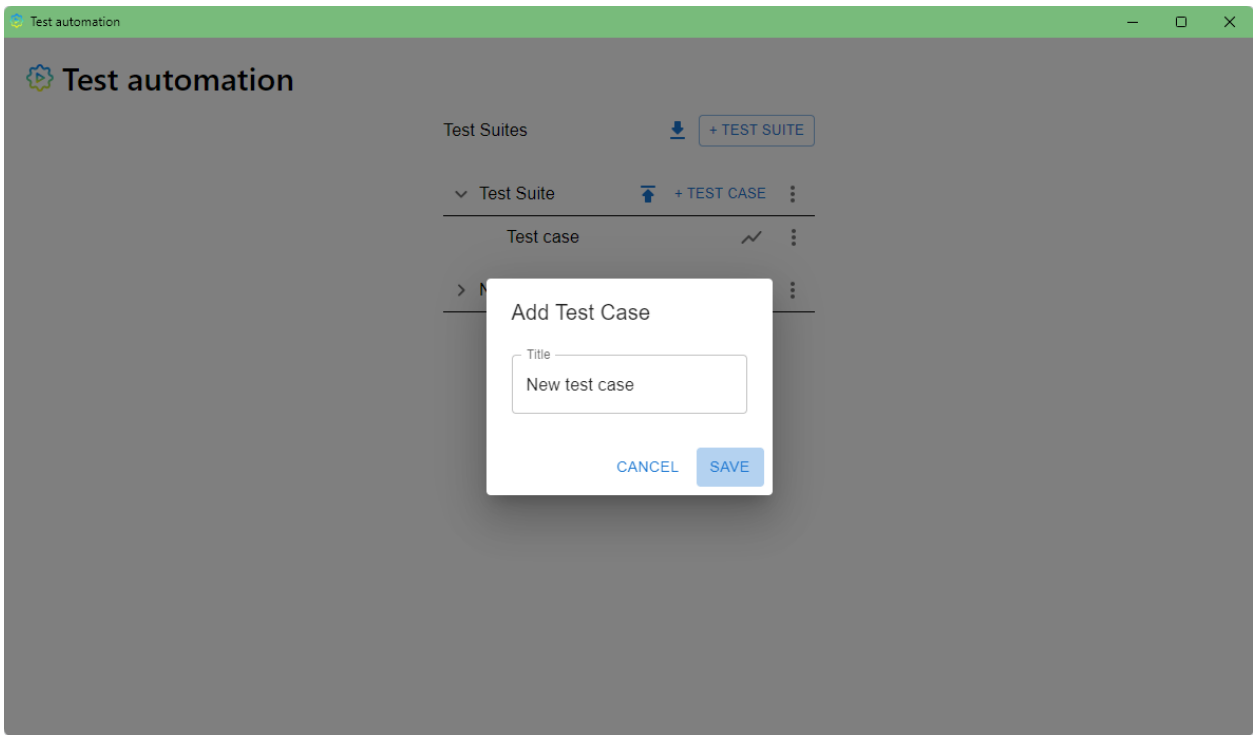


Рисунок 3.3 – Створення нового тестового сценарію

Додавати кроки в тестовий сценарій можна двома способами: вручну і автоматизовано. Щоб додати новий тестовий крок вручну потрібно натиснути кнопку «+ Test step». Після цього буде додано стандартний тестовий крок, який можна буде відредагувати під потреби користувача, виконавши подвійний клік по рядку із новим кроком (рис. 3.4).

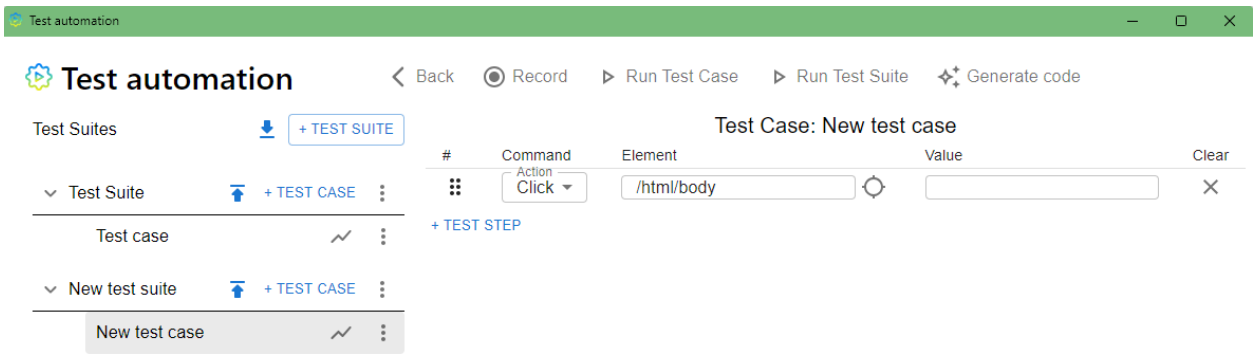


Рисунок 3.4 – Створення нового тестового сценарію

Далі можна змінити тип команди, вибравши потрібну з випадаючого списку. Також можна змінити елемент веб сторінки, над яким виконується дія, вводячи шлях у поле елементу або натиснувши кропку локатора (рис. 3.5).

Рисунок 3.5 – Натискання кнопки локатора

Після натискання кнопки локатора потрібно перейти на веб застосунок відкритий у браузері та натиснути на елемент, який буде виділено червоним кольором (рис. 3.6).

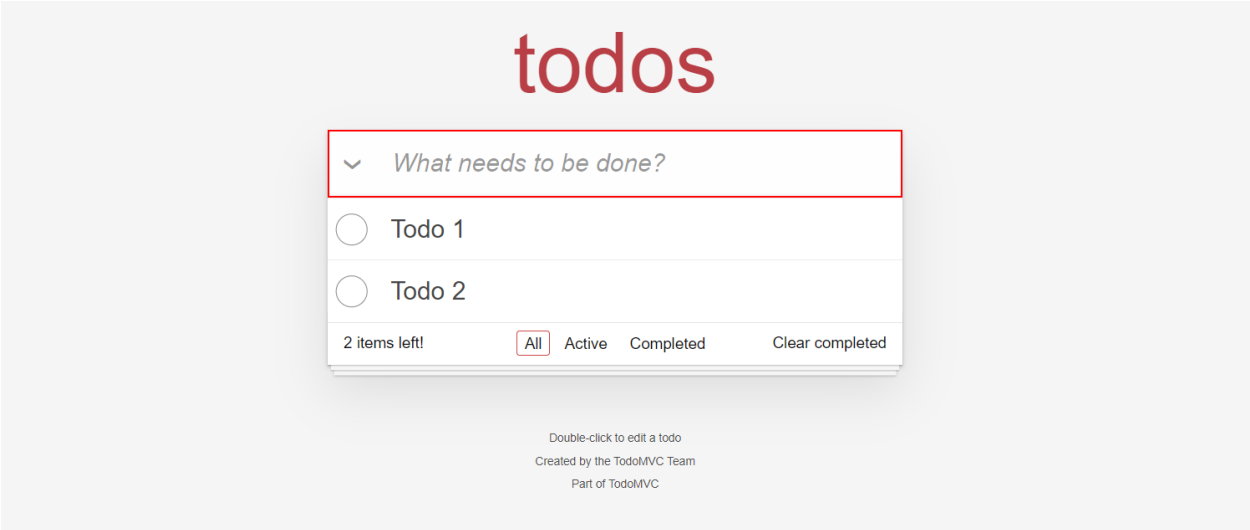


Рисунок 3.6 – Обрання елементу на сторінці, над яким буде виконуватися дія

Для зміни порядку кроків в тестовому сценарії можна використовувати можливість перетягування (drag&drop) кроків. Для видалення непотрібного кроку тестового сценарію потрібно натиснути на хрестик у відповідному рядку таблиці.

Для спрощення створення тестового сценарію можна використати запис дій користувача на сторінці браузера, натиснувши кнопку «Record» для вибраного тестового сценарію (рис 3.7).

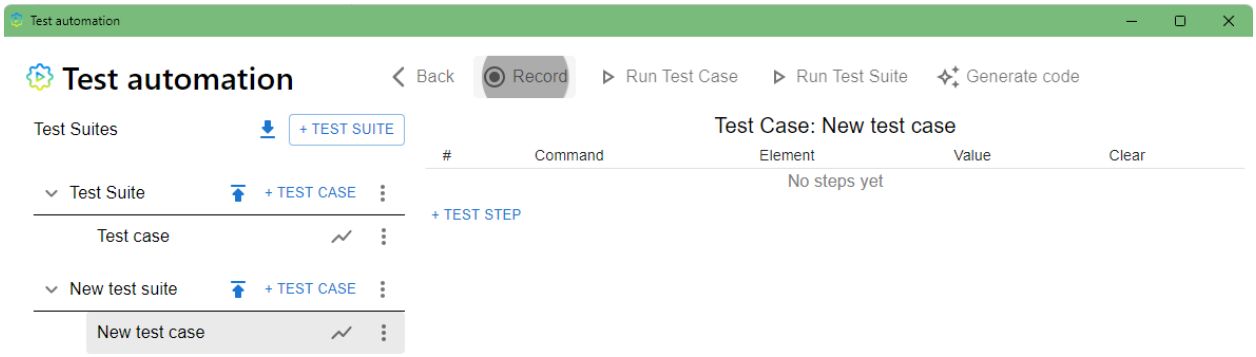


Рисунок 3.7 – Увімкнення запису тестового сценарію

Після цього перейшовши на сторінку браузера, де відкрито застосунко, який потрібно протестувати потрібно виконати послідовність команд, які необхідні для формування тестового сценарію (рис. 3.8). Команди можуть включати кліки, введення тексту, натискання клавіш, виконання перевірок і т. д.

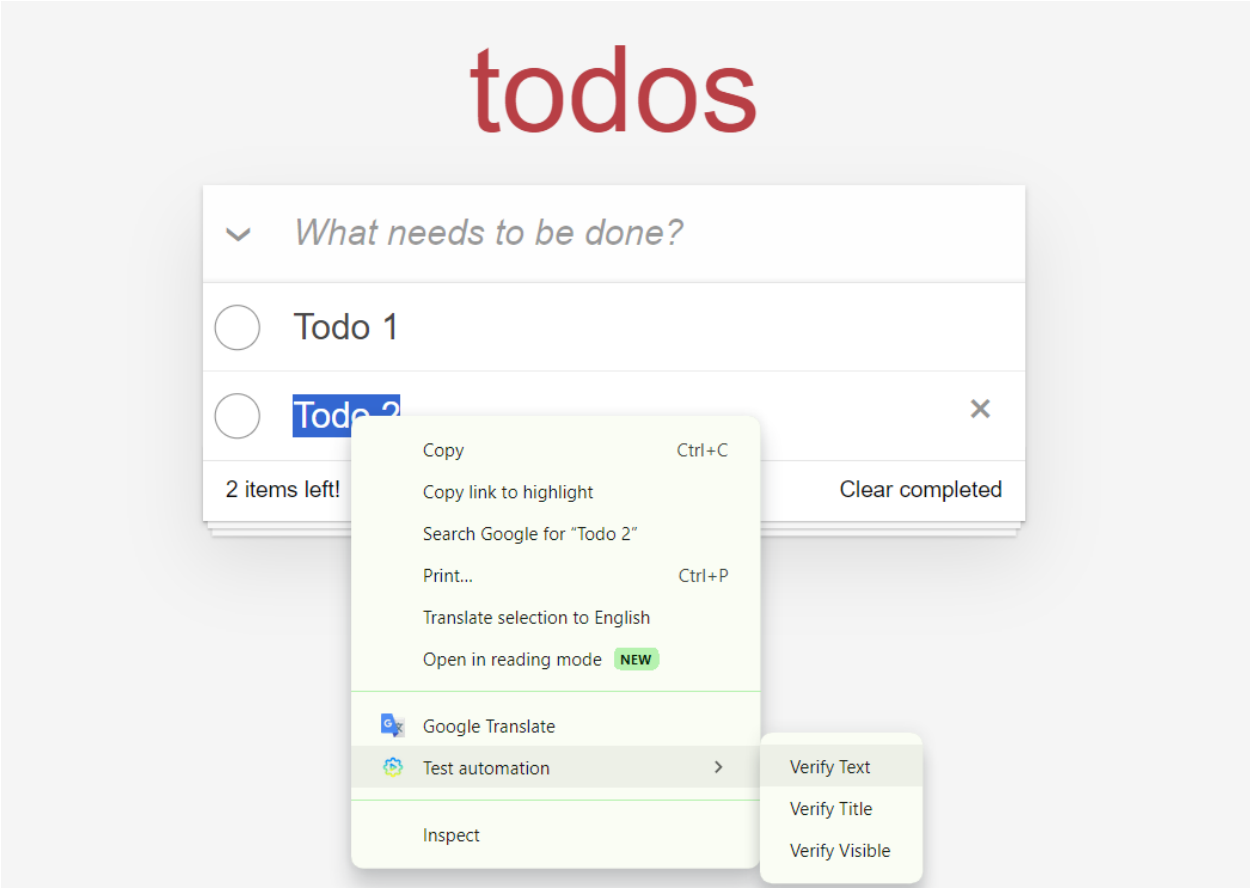


Рисунок 3.8 – Виконання дій на сторінці браузера

Після закінчення формування тестового сценарію, можна побачити, що виконані дії успішно записані до тестового сценарію. Після цього потрібно натиснути кнопку «Stop recording» для завершення запису (рис. 3.9).

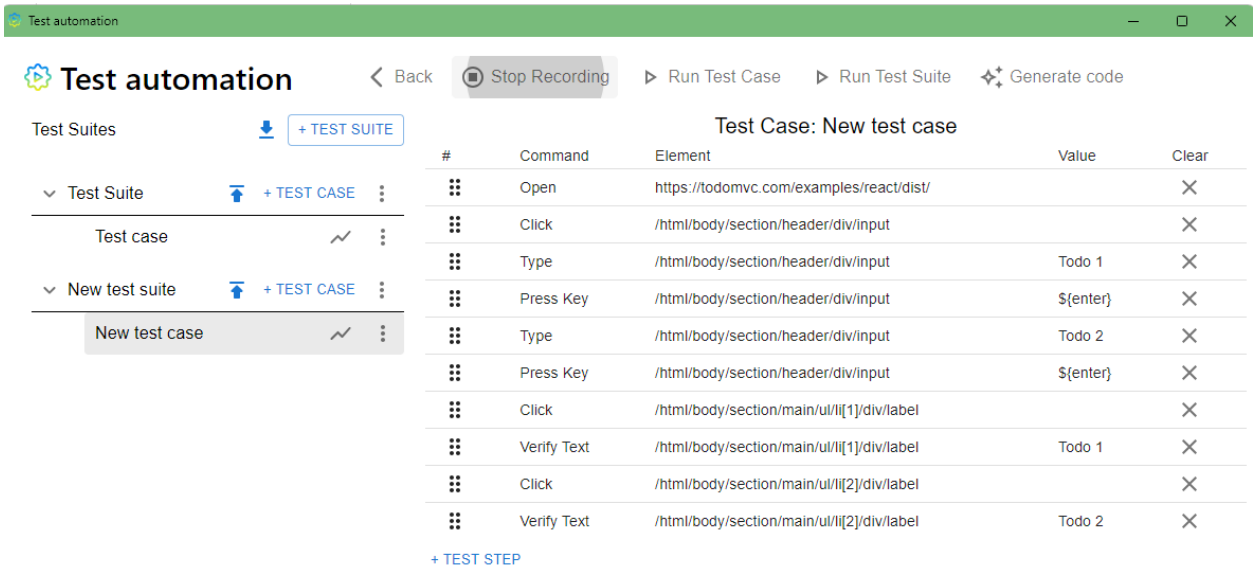


Рисунок 3.9 – Завершення запису тестового сценарію

Тепер можна запустити даний тестовий сценарій на виконання. Для цього потрібно натиснути кнопку «Run Test Case» (рис. 3.10).

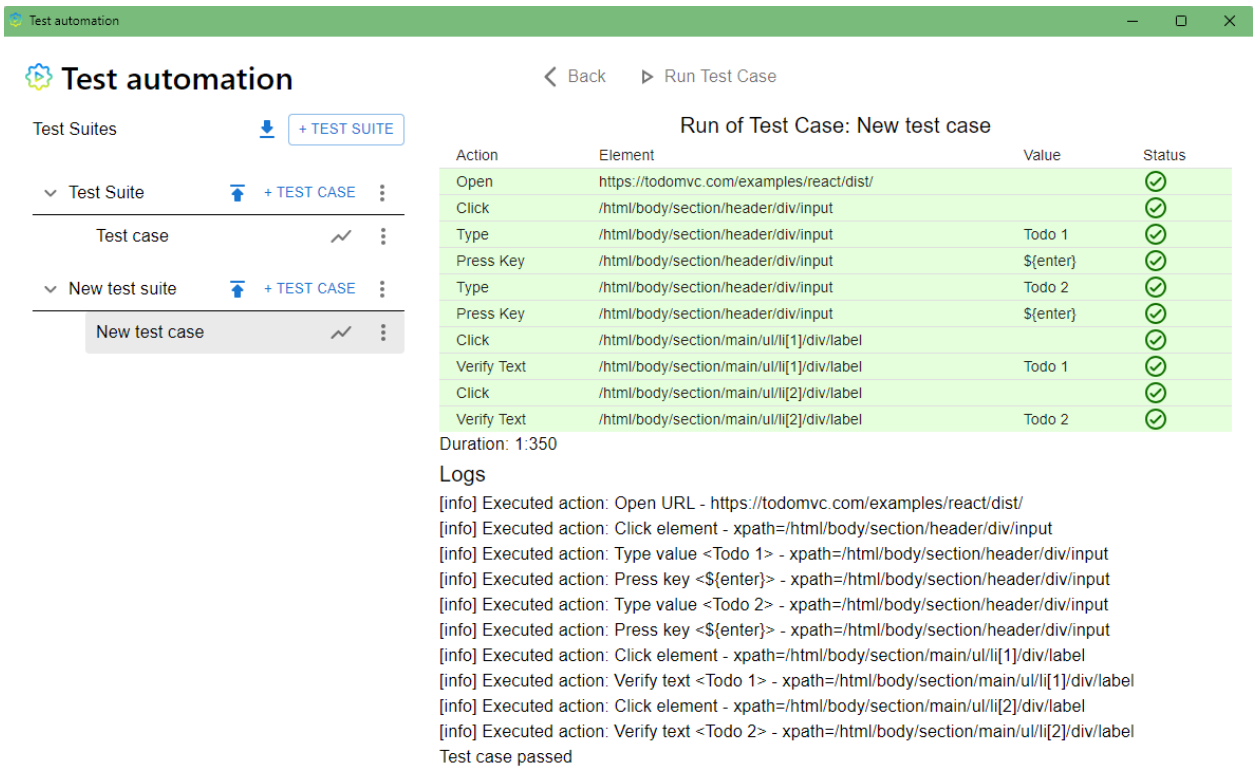


Рисунок 3.10 – Виконання тестового сценарію

Після завершення виконання можна побачити які кроки виконалися успішно, а які неуспішно. Також можна переглянути логи виконання до кожного тествого кроку.

Крім того існує можливість запуску усього тестового набору на виконання. Усі тестові сценарії даного набору будуть виконані послідовно. Для цього потрібно натиснути кнопку «Run Test Suite». Після завершення виконання для кожного запуску тестових сценаріїв буде зібрано статистику виконання та логи.

Для перегляду статистики потрібно натиснути кнопку, яка позначена іконкою з графіком. Після цього відкриється сторінка із статистикою виконання даного тестового сценарію. У таблиці буде відображено усі запуски даного сценарію, де натискаючи на кожен з них можна переглянути результати виконання. Крім того буде відображено гістограму, на якій зображено успішність виконання тестового сценарію, а також і графік, на якому зображено тривалість виконання кожного запуску і середня тривалість виконання у мілісекундах (рис. 3.11).

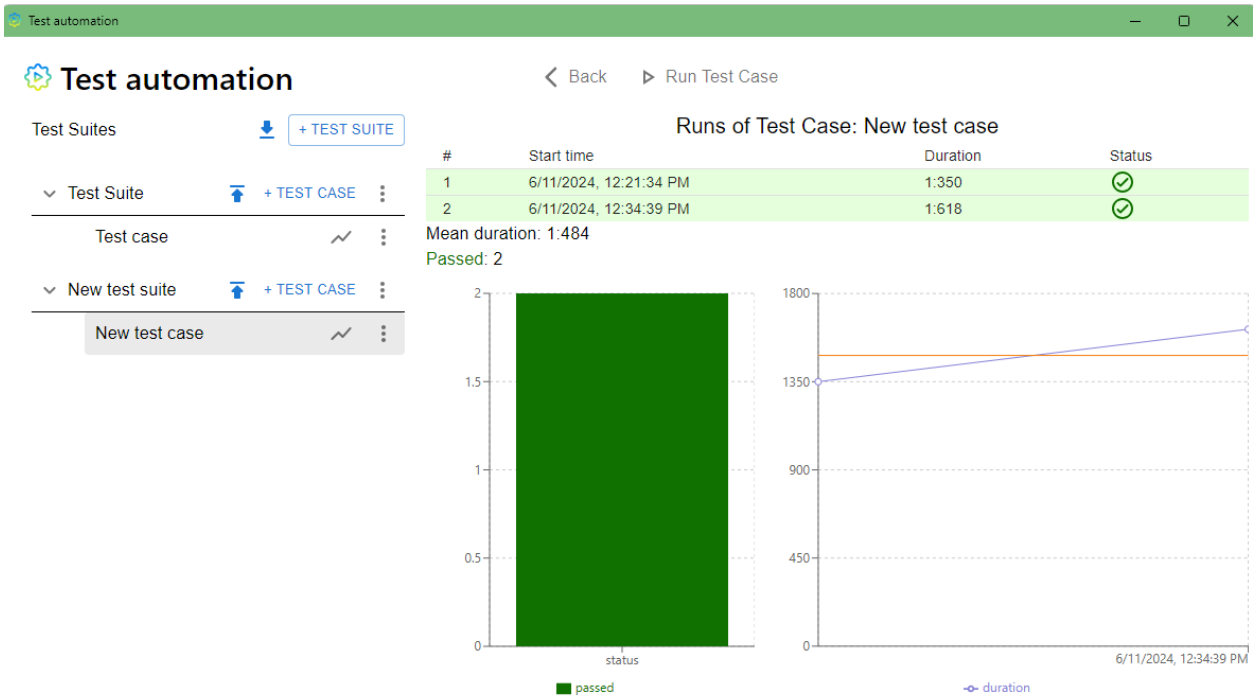


Рисунок 3.11 – Статистика виконання тестового сценарію

Для створеного тестового сценарію можна згенерувати код на мові JavaScript. Для цього у відкритому тестовому сценарії натискаємо кнопку «Generate code». У відкритому діалозі підтверджуємо натискаючи на кнопку «Generate» (рис. 3.12). Після закінчення генерації можна натиснути на кнопку скопіювати код до буферу обміну.

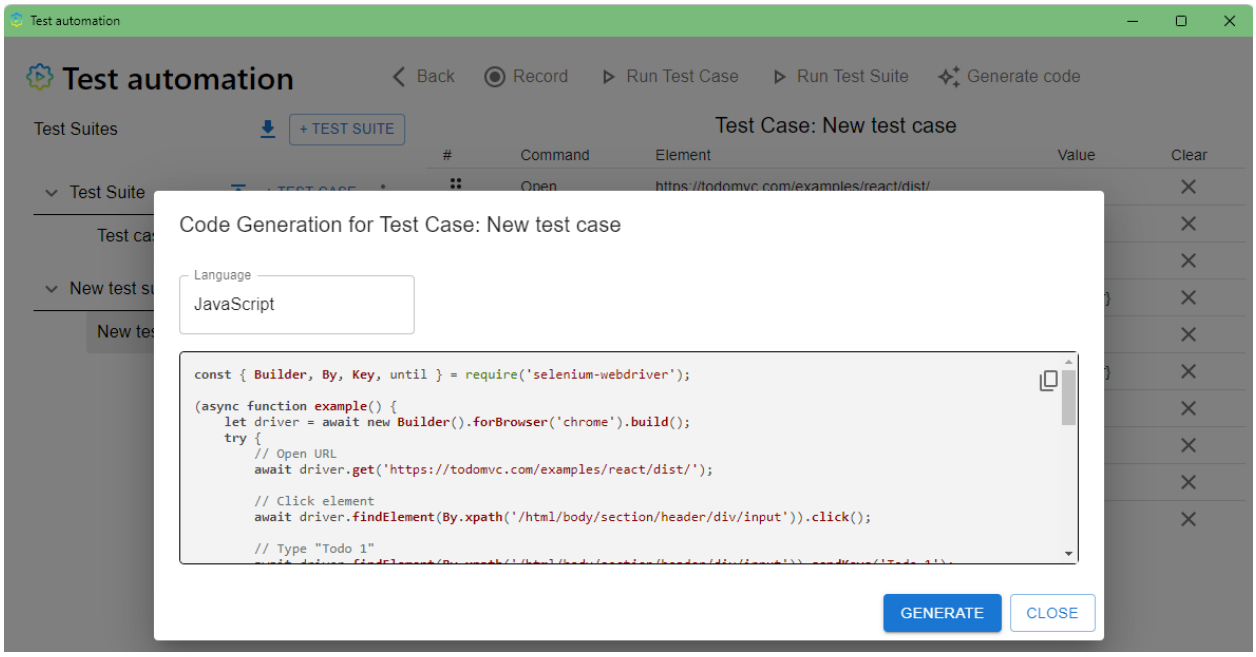


Рисунок 3.12 – Генерація коду тестового сценарію

Також можна експортувати створений тестовий набір, натиснувши відповідну кнопку у рядку де відображено його назву. Далі буде відкрито провідник, де можна зазначити назву файлу і натиснути кнопку зберегти (рис. 3.13). Після цього тестовий набір буде успішно експортовано у вигляді JSON файлу.

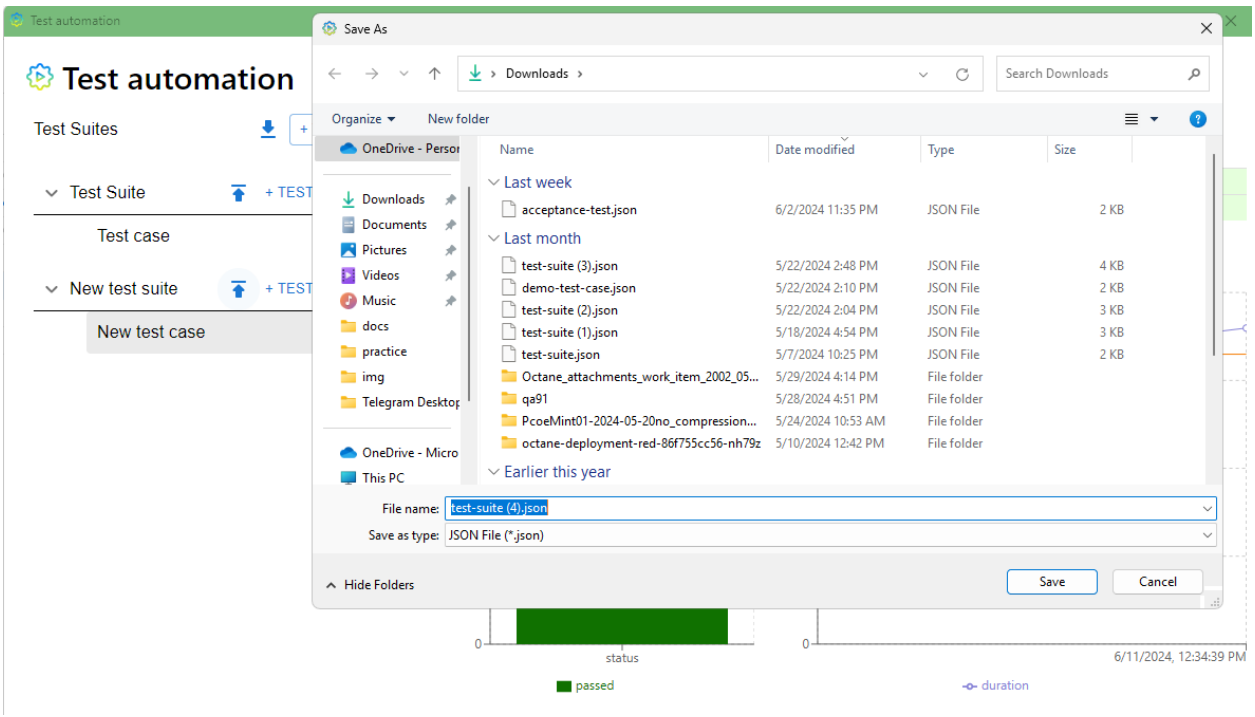


Рисунок 3.13 – Експортування тестового набору

Для імпорту тестового сценарію потрібно натиснути відповідну кнопку, позначену іконкою завантаження. Далі потрібно вибрати JSON файл із тестовим набором та натиснути відкрити (рис. 3.14). Після цього обраний тестовий набір успішно завантажено в систему.

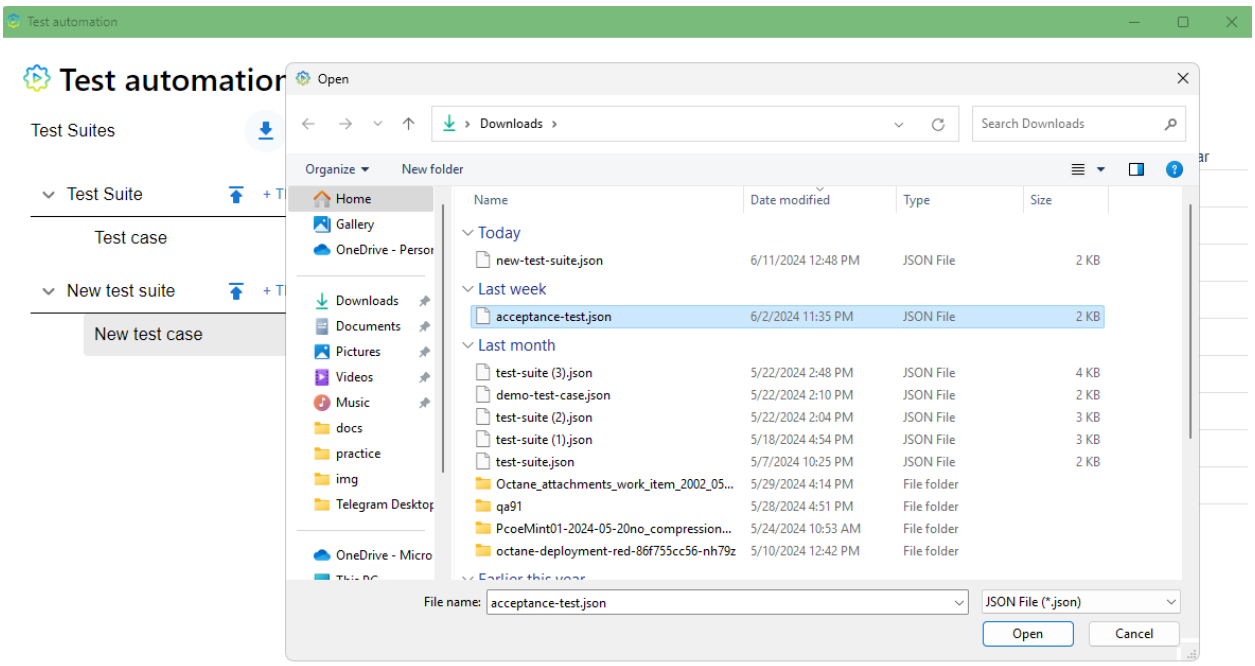


Рисунок 3.14 – Імпортування тестового набору

Для видалення тестового набору потрібно натиснути кнопку з трьома крапками і далі натиснути кнопку «Remove» (рис. 3.15). Після цього тестовий набір буде успішно видалено із системи.

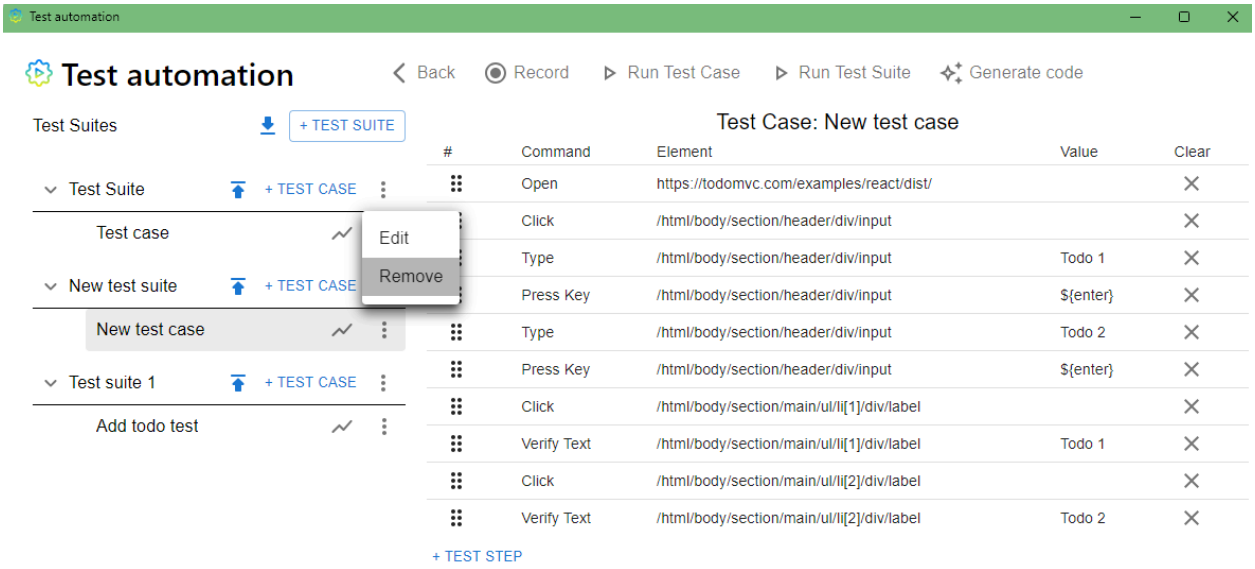


Рисунок 3.15 – Імпортування тестового набору

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Плагін браузера Google Chrome для автоматизації тестування веб застосунків

Графічний матеріал

КПІ.ІП-0120.045440.06.99

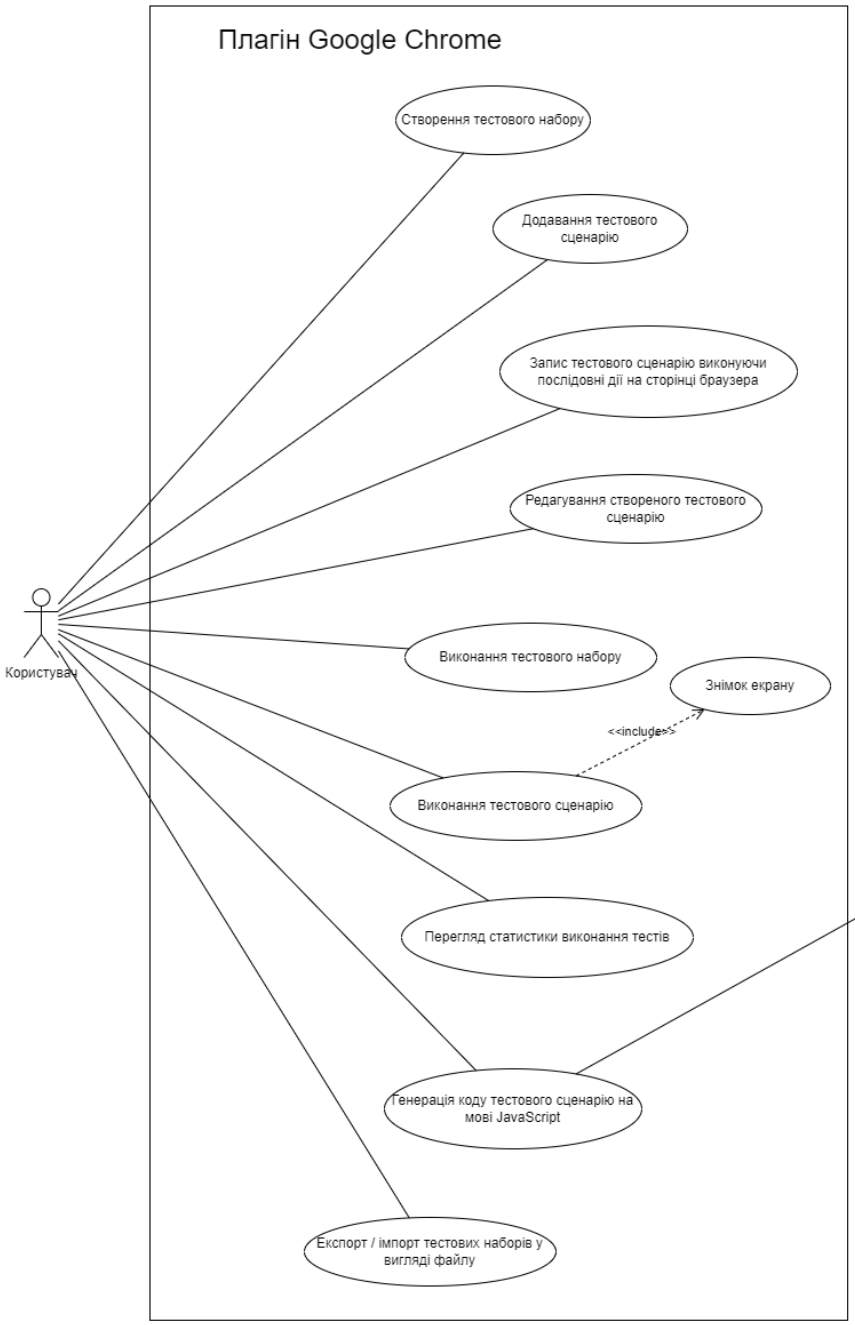
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав МОСКАЛЕНКО

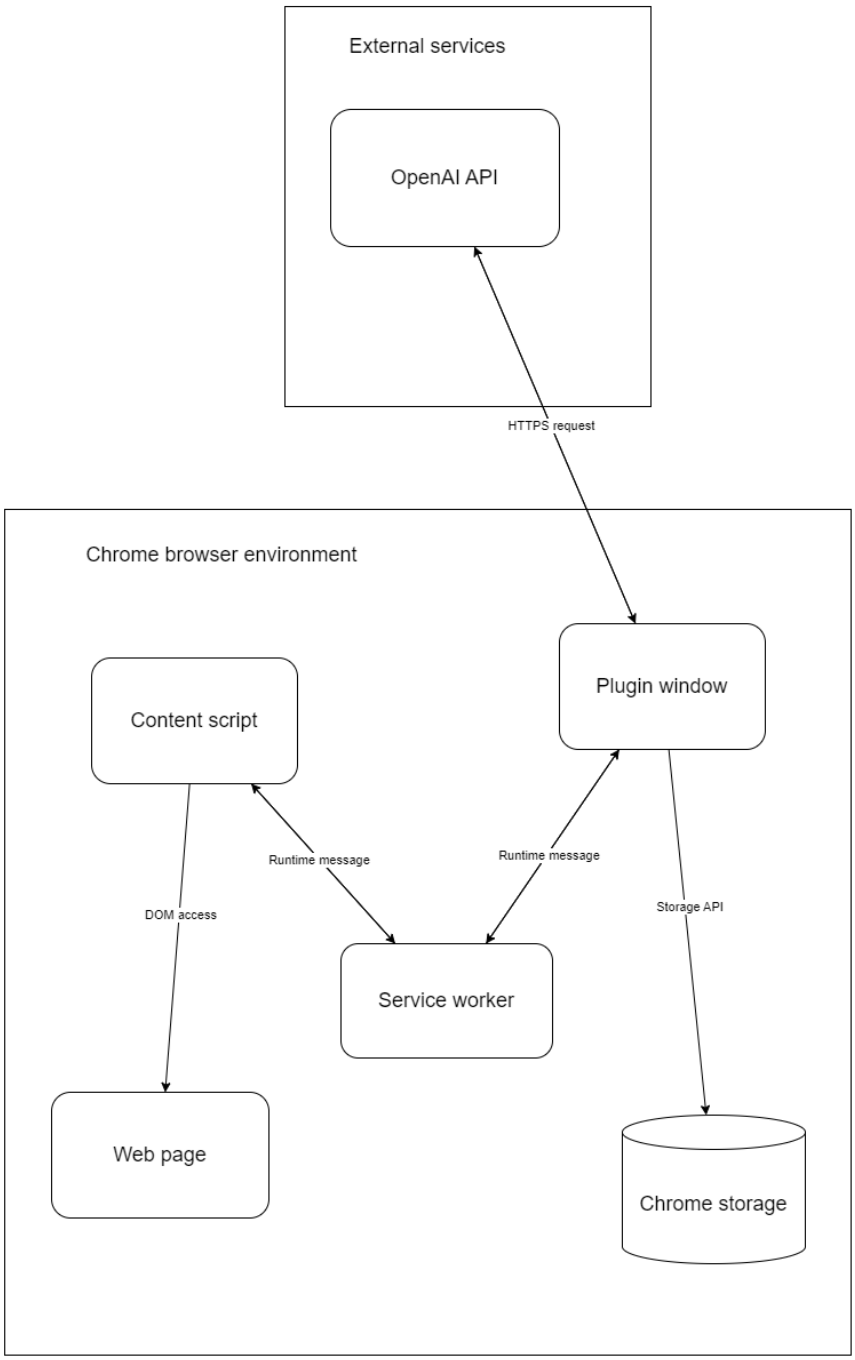
Київ – 2024

КПІ.ІП-0120.045440.06.99.ССВ

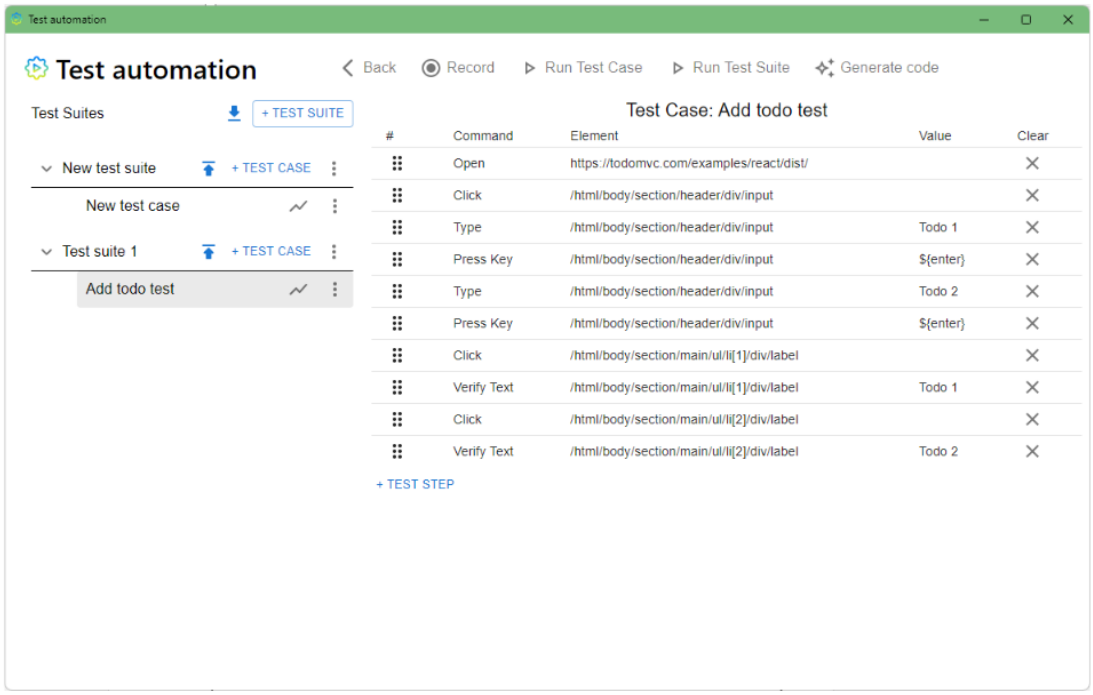


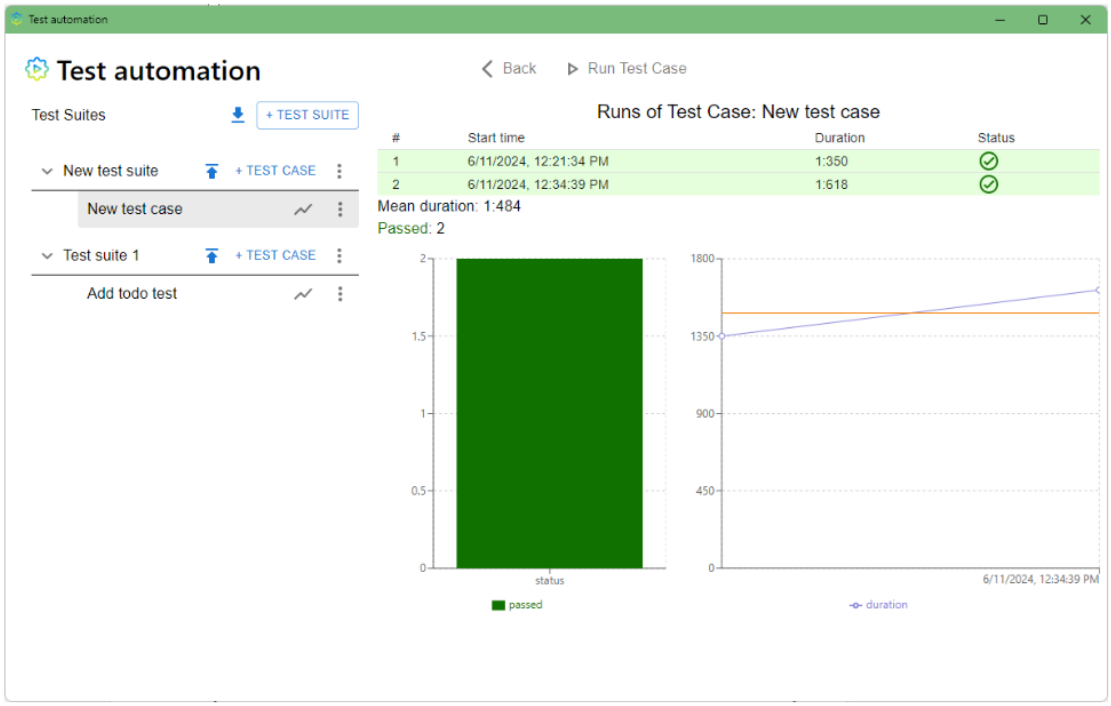
					КПІ.ІП-0120.045440.06.99.ССВ			
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна варіантів використань	Лит.	Маса	Масштаб
Розробив		Москаленко В.Ю						
Перевірив		Павлов О.А.				Аркуш	Аркушів	
Т. контр.					Плагін браузера Google Chrome для автоматизації тестування веб застосунків	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		
Н. контр.		Головченко М.М.						
Затвердив		Жаріков Е.В.						

КПІ.ІП-0120.045440.06.99.ССМ



					КПІ.ІП-0120.045440.06.99.ССМ			
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна компонентів програмного забезпечення	Лит.	Маса	Масштаб
Розробив		Москаленко В.Ю						
Перевірив		Павлов О.А.				Аркуш	Аркушів	
Т. контр.								
Н. контр.		Головченко М.М.			Плагін браузера Google Chrome для автоматизації тестування веб застосунків	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		
Затвердив		Жаріков Е.В.						





					КПІ.ІП-0120.045440.06.99.КЕ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Москаленко В.Ю.			Креслення вигляду екранних форм
Перевірив		Павлов О.А.			
Т. контр.					
Н. контр.		Головченко М.М.			Плагін браузера Google Chrome для автоматизації тестування веб застосунків
Затвердив		Жаріков Е.В.			

Лит.	Маса	Масштаб
Аркуш	Аркушів	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-01