Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія тестування WEB застосувань»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Вовчарук П. Ю.
(прізвище та ініціали)

Керівник: к.т.н., ст. викладач каф. КН
_____ Озеранський В. С.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., доц. каф. КСУ
_____ Юхимчук М.С.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
14. 09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Вовчарук Павло Юрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія тестування WEB застосувань
керівник роботи к.т.н., старший викладач кафедри КН Озеранський В. С.
затверджені наказом вищого навчального закладу від "14" 09 2022 року №203
2. Строк подання студентом роботи 18.11. 2022 року

3. Вихідні дані до роботи:
вхідні дані –посилання на WEB застосування та локатори, кількість елементів тестування: не більше 100шт., кількість тестових сценаріїв – не менше 50од., кросбраузерність - підтримка не менше 3 браузерів, мова програмування – об'єктно-орієнтована.
4. Зміст текстової частини:
Вступ, аналіз сучасних технологій тестування web застосувань, моделювання інформаційної технології тестування web застосувань, програмна реалізація інформаційної технології тестування web застосувань, економічна частина, висновки, перелік використаних джерел, додатки
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Структурна схема інформаційної технології тестування WEB застосувань, UML діаграма послідовностей, Usecase-діаграма доступних дій користувача, алгоритм функціонування інформаційної технології тестування WEB застосувань, робочі вікна програми, програми-аналоги, розрахунок ефективності, принцип роботи Selenium Webdriver

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконав прийняв
1-3	Озеранський В. С., к.т.н., ст. викл. каф. КН	14.09.22	14.11.22
4	Бурєннікова Н.В., д. е. н., проф. каф ЕПВМ	19.09.22	12.12.22

7. Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасних технологій тестування WEB застосувань	14.09.22 01.10.22
2	Моделювання інформаційної технології тестування WEB застосувань	02.10.22 7.11.22
3	Програмна реалізація та оцінка ефективності розробленої інформаційної технології	8.11.22 21.11.22
4	Підготовка економічної частини	22.11.22 - 01.12.22
5	Апробація та/або впровадження результатів дослідження	02.12.22 09.12.22
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	10.12.22 14.12.22

Студент _____ (підпис) Вовчарук П. Ю.

Керівник роботи _____ (підпис) Озеранський В. С.

АНОТАЦІЯ

УДК 621.374.415

Вовчарук П. Ю. Інформаційна технологія тестування WEB застосувань. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - комп'ютерні науки. Вінниця: ВНТУ, 2022. 140 с.

На укр. мові. Бібліогр.: 33 назв; рис.: 40; табл. 13.

Дана магістерська кваліфікаційна робота присвячена удосконаленню інформаційної технологію тестування WEB застосувань. В роботі було проведено аналіз предметної області тестування WEB застосувань, сучасних методів та інструментів автоматизованого тестування, здійснено порівняльний аналіз програм-аналогів. Обгрунтовано вибір бібліотеки Selenium драйвера браузера WebDriver для вирішення задачі автоматизованого тестування. Удосконалено математичну модель тестування застосувань. Розроблено алгоритм функціонування інфрмаційної технології тестування WEB застосувань. Здійснено програмну реалізацію інформаційної технології тестування WEB застосувань мовою прогрмування Python. Проведено тестування розробленого програмного забезпечення, що підтверджує коректкність його роботи та оцінку ефективності роботи, що вища на 3,5% в середньому від програм-аналогів.

Графічна частина складається з 5 плакатів.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 608297,124 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0,69 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: інформаційна технологія, тестування, автоматизація

ABSTRACT

Vovcharuk P. I. Information technology of of web applacation testing. Master's thesis in the specialty 122 - computer sciences, educational program - computer science. Vinnytsia: VNTU, 2022. 140 p.

In Ukrainian language. Bibliographer: 33 titles; fig .: 40; tables 13.

This master's thesis is devoted to improving the information technology of testing WEB applications. In the work, an analysis of the subject area of testing WEB applications, modern methods and tools of automated testing was carried out, a comparative analysis of similar programs was carried out. The choice of the Selenium library of the WebDriver browser driver to solve the problem of automated testing is justified. The mathematical model of application testing has been improved. An algorithm for the functioning of information technology for testing WEB applications has been developed. Software implementation of the information technology for testing WEB applications using the Python programming language was carried out. Testing of the developed software was carried out, which confirms the correctness of its operation and the evaluation of the efficiency of work, which is 3.5% higher on average than similar programs.The graphic part consists of 5 posters.

The economic section calculates the amount of costs for the development and manufacture of a new technical solution, which is 608297,124 hryvni, predicted the estimated cost of each of the cost items, calculated net profit, payback period for the manufacturer 0.69 years and the economic effect for consumers using this development.

Keywords: information technology, testing, automation

ЗМІСТ

ВСТУП

Актуальність Сучасне програмне забезпечення є складним багатофункціональним об'єктом. Його ручна перевірка вимагає значних трудових і тимчасових витрат. На допомогу приходять засоби автоматизації тестування, які підвищують якість, забезпечують повторне використання тестів при коригуванні ПЗ. Основна задача автоматизації тестування веб додатків - скорочення витрат на випробування програми після її модернізації та позбавлення людського фактору при перевірках. Однією з головних проблем автоматизованого тестування є його трудомісткість: попри те, що воно дозволяє усунути частину рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на створення самих тестів. Це відноситься до обох видів автоматизації. При рефакторінгу часто буває необхідно оновити і модульні тести, і зміна коду тестів може зайняти стільки ж часу, скільки і зміна основного коду. З іншого боку, при зміні інтерфейсу програми необхідно заново переписати всі тести, які пов'язані з оновленими вікнами, що при великій кількості тестів може відняти значні ресурси.

Існує два основних підходи до автоматизації тестування: тестування на рівні коду і GUI-тестування. До першого типу належить, зокрема, модульне тестування. До другого — імітація дій користувача за допомогою спеціальних тестових фреймворків. Найпоширенішою формою автоматизації є тестування додатків через графічний інтерфейс користувача. Популярність такого виду тестування пояснюється двома факторами: по-перше, додаток тестується тим же способом, яким його буде використовувати людина, по-друге, можна тестувати додаток, не маючи при цьому доступу до вихідного коду.

Зараз на ринку існує величезна різноманітність різних продуктів для автоматизованого тестування, деякі компанії-розробники програмного забезпечення створюють власні інструменти, пристосовані для тестування розроблюваних ними додатків. Причинами цього є висока вартість засобів автоматизованого тестування та унікальність програмного забезпечення, що

тестується, яка не дозволяє використовувати стандартні засоби автоматизації тестування. Крім засобів тестування існують так звані засоби підтримки процесу тестування, що дозволяють вести облік вимоги і тест-кейси, проводити аналіз покриття вимоги тестами, керувати ходом виконання тестування, вести облік виявлених дефектів і тому подібне.

Оскільки мова йде про тестування саме функціоналу веб застосувань, то варто зазначити, що потрібно брати до уваги саме ті існуючі інформаційні технології, які призначені для вирішення цієї проблеми. Тобто йдеться про тестування саме додатків, що написані на HTML, і відповідно для них не підійдуть інструменти, фреймворки та бібліотеки, які призначені для тестування додатків, написаних наприклад на Java чи .NET, а оскільки в роботі розглядається тестування саме веб застосувань, то важливо враховувати кроссбраузерність обраної технології, для її використання.

Автоматичні тести не можуть повністю замінити ручне тестування, оскільки автоматизація всіх процесів — дуже дорогий процес, і тому автоматичне тестування є скоріше лише доповненням ручного тестування, тому варто зазначити, що удосконалення інформаційної технології тестування веб-застосувань, яка полягатиме у створенні програмного продукту, який спрощуватиме цей процес, за рахунок роботи універсальних шаблонів, які можна буде застосувати до більшості стандартних веб-застосувань і таким чином підвищити ефективність зменшивши часові витрати на написання нових тестових сценаріїв під кожен окремий об’єкт тестування є актуальним.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання досліджень. Метою магістерського дослідженння є підвищення ефективності процесу автоматизованого тестування веб-застосувань. Для досягнення мети роботи потрібно виконати такі задачі:

- здійснити аналіз сучасних технологій тестування WEB застосувань;
- розглянути існуючі методи та засоби вирішення задачі автоматизованного тестування WEB застосувань і обґрунтувати вибір методу, що задовольняє мету даної магістерської роботи;
- здійснити моделювання інформаційної технології тестування WEB застосувань;
- удосконалити математичну модель тестування WEB застосувань;
- розробити алгоритм функціонування інформаційної технології тестування WEB застосувань;
- здійснити програмну реалізацію запропонованої інформаційної технології;
- провести тестування програмного засобу та проаналізувати отримані результати.

Об’єкт дослідження – процес автоматизованого тестування WEB застосувань.

Предмет дослідження – інформаційні технології автоматизованого тестування WEB застосувань.

Методи дослідження. У роботі використані такі методи наукових досліджень: методи та моделі подання знань, методи математичного моделювання, методи автоматизованого тестування програмних засобів, зокрема, функціональне тесування та тестування методом «чорного ящика», методи та техніки тест-дизайну, методи об’єктно-орієнтованого програмування.

Наукова новизна одержаних результатів.

Набула подальшого розвитку інформаційна технологія тестування WEB застосувань, яка відрізняється від існуючих використанням удосконаленої математичної моделі тестування WEB застосувань , що дозволило підвищити ефективність процесу автоматизованого тестування WEB застосувань..

Практичне значення одержаних результатів полягає в тому, що на основі проведених досліджень розроблено програмне забезпечення для автоматизованого тестування WEB застосувань.

Запропонована інформаційна технологія сприяє підвищенню ефективності процесу автоматизованого тестування функціоналу WEB застосувань:

- удосконалено математичну модель тестування WEB застосувань;
- розроблено алгоритм функціонування інформаційної технології тестування WEB застосувань;
- розроблено програмні засоби для автоматизованого тестування WEB застосувань.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології тестування WEB застосувань. Адекватність розроблених математичних моделей підтверджується результатами експериментальних досліджень.

Особистий внесок здобувача. Усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно. У працях, які написано у співавторстві, здобувачу належать: особливості тестування функціоналу веб-застосункців[1], особливості автоматизованого тестування веб-застосувань[2].

Апробація результатів роботи. Результати роботи були апробовані на конференціях «Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (НКТ ФІІТА 2022)» (м. Вінниця, Україна, 2022 р.)[1], «Науково-технічна конференція факультету інформаційних технологій та комп’ютерної інженерії (НКТ ФІТКІ 2021)» (м. Вінниця, Україна, 2021 р.)[2]. Подано заяву на авторське право на твір.

Публікації. За результатами досліджень опубліковано дві тези доповіді на науково-технічній конференції.

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ТЕСТУВАННЯ WEB ЗАСТОСУВАНЬ

1.1 Аналіз технологій автоматизованного тестування програмних засобів

Тестуванням програмного забезпечення є процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових із метою оцінки. Тобто це процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники;
- правильна відповідь для усіх можливих вхідних даних;
- виконання функцій за прийнятний час;
- практичність;
- сумісність із програмним забезпеченням та операційними системами;
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів) [3].

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників. Тестування може проводитись, як тільки створено виконуваний код (навіть частково

завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення й забезпечення якості програмного забезпечення, до якого належать всі складові ділового процесу, а не тільки тестування [4].

Тестування, яке виходячи з усього вище зазначеного, можна назвати по суті контролем якості, включає в себе:

- планування робіт;
- проектування тестів;
- виконання тестування;
- аналіз отриманих результатів.

Верифікація – це процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази. Валідація – це визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи. План тестування – це документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення. Тест дизайн – це етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування. Тестовий випадком або тест кейсом є документ, що описує сукупність кроків,

конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини [5].

Баг/Дефект репорт є документом, що описує ситуацію або послідовність дій, що призвела до некоректної роботи об'єкта тестування, із зазначенням причин та очікуваного результату. Тестове покриття одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується. Деталізація тест кейсів є рівенем деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття. Час проходження тест кейса є часом від початку проходження кроків тест кейса до отримання результату тесту [6].

Розглянемо методи тестування. Спершу розглянемо статичне та динамічне тестування. Тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Тестова діяльність, що передбачає експлуатацію програмного продукту, називається динамічним тестуванням. Динамічне та статичне тестування доповнюють одне одного. На етапі статичного тестування перевіряється вся документація, отримана як результат життєвого циклу програми. Це і технічне завдання, і специфікація, і вихідний текст програми на мові програмування. Вся документація аналізується на предмет дотримання стандартів програмування. У результаті статичної перевірки встановлюється, наскільки програма відповідає заданим критеріям та вимогам замовника. Усунення неточностей та помилок у документації – запорука того, що створюваний програмний засіб має високу якість. Динамічні методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми [7].

Тестування методом «білої скриньки» – відома внутрішня структура програми, досліджуються внутрішні елементи програми і зв'язки між ними. Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше – інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми. Зазвичай тестування «білої скриньки» засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління[8]. У цьому випадку формуються тестові варіанти, в яких:

- гарантується перевірка всіх незалежних маршрутів програми;
- знаходяться гілки true, false для всіх логічних рішень;
- виконуються всі цикли (у межах їхніх кордонів та діапазонів);
- аналізується правильність внутрішніх структур даних.

Недоліки тестування методом «білої скриньки»:

- кількість незалежних маршрутів може бути дуже велика;
- повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї;
- у програмі можуть бути пропущені деякі маршрути;
- не можна виявити помилки, поява яких залежить від даних.

Переваги тестування «білої скриньки» пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

- кількість помилок мінімально в «центрі» і максимально на «периферії» програми;
- попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. у результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо;

- при записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних);
- деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми;

Кожна з цих причин є аргументом для проведення тестування за методом «білої скриньки». Тести методом «чорної скриньки» не зможуть реагувати на помилки таких типів .

Тестування методом «чорної скриньки» Відомі функції програми. Досліджується робота кожної функції на всій області визначення. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Ці тести демонструють:

- як виконуються функції програми;
- як приймаються вихідні дані;
- як виробляються результати;
- як зберігається цілісність зовнішньої інформації.

При тестуванні методом «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 1010. Тестування методом «чорної скриньки» не реагує на багато особливостей програмних помилок. Тестування методом «чорної скриньки» дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

- набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його it);
- набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ot).

Будь-який спосіб тестування методом «чорної скриньки» повинен:

- виявити такі вхідні дані, які з високою ймовірністю належать набору ІТ;
- сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок. Тестування методом «чорної скриньки» забезпечує пошук наступних категорій помилок:

- некоректних чи відсутніх функцій;
- помилок інтерфейсу;
- помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;
- помилок характеристик (необхідна ємність пам'яті і т. д.);
- помилок ініціалізації та завершення.

Подібні категорії помилок способами «білої скриньки» не виявляються.

Тестування інтерфейсу користувача – це процес тестування продукту інтерфейсу користувача, для забезпечення його відповідності до даної специфікації [9].

Мета функціонального тестування – виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. Функціональні тести повинні охоплювати всі реалізовані функції з урахуванням найбільш ймовірних типів помилок. Тестові сценарії, що поєднують окремі тести, орієнтовані на перевірку якості розв'язку функціональних задач. Функціональні тести створюються за зовнішніми специфікаціями функцій, проєктної інформації і за текстом на МП, що стосуються його функціональних характеристик і застосовуються на процесі комплексного тестування й іспитів для визначення повноти реалізації функціональних задач і їхньої відповідності вхідним вимогам. До задач функціонального тестування належать:

- ідентифікація множини функціональних вимог;

- ідентифікація зовнішніх функцій і побудова послідовностей функцій відповідно до їхнього використання в ПС;
- ідентифікація множини вхідних даних кожної функції і визначення областей їхньої зміни;
- побудова тестових наборів і сценаріїв тестування функцій;
- виявлення і подання усіх функціональних вимог за допомогою тестових наборів і проведення тестування помилок у програмі і при взаємодії із середовище.

Тести, створювані за проєктною інформацією, пов'язані зі структурами даних, алгоритмами, інтерфейсами між окремими компонентами і застосовуються для тестування компонентів і їхніх інтерфейсів. Основна мета – забезпечення повноти і погодженості реалізованих функцій і інтерфейсів між ними [10].

В основу комбінованого методу «чорної скриньки» і «білої скриньки» покладено розбивку вхідної області функції на підобласті виявлення помилок. Підобласть містить у собі однорідні елементи, які обробляються коректно або некоректно. Для тестування підобласті застосовується виконання програми на одному з елементів цієї області.

Отже, враховуючи усе вище зазначене, можна сказати, що для даної розробки є актуальним використання методу тестування «чорної скриньки», оскільки буде перевірятись саме працездатність елементів веб-застосування, а очікування їх поведінки є як для звичайного користувача, тобто без безпосередньої вимоги знань та доступу до програмного коду, а оскільки перевірятиметься саме відповідність роботи саме «клієнтської частини», тобто інтерфейсу, отже здійснюватиметься тестування інтерфейсу користувача, в якому повинен бути наявний певний функціонал, отже буде здійснюватись функціональне тестування.

1.2 Аналіз методів та засобів вирішення задачі автоматизованного тестування WEB застосувань

Для тестування веб-застосунку необхідно імітувати роботу браузера. Для цього є різні підходи. Є прості інструменти, які лише вміють відправляти HTTP-запити до сервера і аналізувати отриманий HTML-код, і більш просунуті, які або використовують справжній браузерний «движок» в режимі без виведення віконця зі сторінкою на екран , а найбільш просунуті надають драйвери, за допомогою яких можна контролювати реальний браузер [11]. Саме таким інструментом, що дозволяє досягти процесу автоматизації веб-застосувань є Selenium.

Selenium – це набір бібліотек для різних мов програмування. Ці бібліотеки використовуються для відправки HTTP запитів драйверу за допомогою протоколу JsonWireProtocol, в яких зазначено дію, яку повинен зробити браузер в рамках поточної сесії. Прикладами таких команд можуть бути команди знаходження елементів по локатору, перехід по посиланнях, парсинг тексту сторінки / елемента, натискання кнопок або перехід по посиланнях на сторінці веб-сайту [12].

Selenium дозволяє писати сценарії практично на будь-якій мові програмування. Він є ключовим компонентом безлічі відкритих інструментів автоматизації. Selenium дозволяє управляти браузером віддалено, завдяки чому можна створювати розподілені стенди, що складаються з безлічі машин з різними операційними системами і браузерами, і навіть запускати браузери в хмарах. Розглянемо які існують компоненти Selenium:

- Selenium IDE – інтегроване середовище розробки у вигляді Firefox-додатка, який дозволяє записувати та відтворювати тести в Firefox 2+;
- Selenium Client API – набір API, що дозволяє писати тести на Java, C#, Ruby, JavaScript та Python;
- Selenium Remote Control – це клієнт / серверна система, яка дозволяє керувати веб-браузерами локально або на іншому комп'ютері,

використовуючи практично будь-яку мову програмування та тестування системи;

- Selenium WebDriver – драйвер що дозволяє керувати веб-браузером за допомогою Selenese або API;
- Selenium Grid – дозволяє одночасно запускати тести на кількох серверах та типах веб-браузерів зменшуючи час на тестування.

Використання набору бібліотек Selenium для тестування веб-застосувань можливе при використанні драйвера для керування браузером [13].

Webdriver – популярний інструмент для управління реальним браузером, який можна використовувати як для автоматизації тестування веб-додатків, так і для виконання інших рутинних завдань, пов'язаних з роботою в інтернеті. Загалом можна сказати, що це драйвер браузера, тобто не має призначеного для користувача інтерфейсу програмна бібліотека, яка дозволяє різним іншим програмам взаємодіяти з браузером, управляти його поведінкою, отримувати від браузера якісь дані і змушувати браузер виконувати якісь команди [14].

За своєю сутністю Selenium WebDriver є :

- специфікацію програмного інтерфейсу для управління браузером;
- референсні реалізації цього інтерфейсу для декількох браузерів;
- набір клієнтських бібліотек для цього інтерфейсу на декількох мовах програмування .

Крім того, Webdriver – проект з відкритим вихідним кодом, підтримує безліч мов програмування і має велике співтовариство користувачів. Тобто можна зробити висновок, що в даному випадку для розробки потрібно використати саме Selenium WebDriver. Selenium WebDriver повністю реалізований та підтримується в Python , Ruby , Java та C #.

Розглянемо переваги використання Webdriver:

- інтерфейс Webdriver був спроектований більш простим і виразним;
- Webdriver володіє більш компактним і об'єктно-орієнтованим API;

- Webdriver управляє браузером більш ефективно, а також справляється з деякими обмеженнями, характерними для Selenium RC, як завантаження та відправлення файлів, попап і дилогії.

Проектом Selenium і співтовариством підтримується робота з браузерами Microsoft Internet Explorer , Google Chrome , Mozilla Suite і Mozilla Firefox під управлінням операційних систем Microsoft Windows , Linux і Apple Macintosh .

Принципу роботи Selenium Webdriver зображено на рисунку 2.1

Рисунок 1.1 – Принцип роботи Selenium Webdriver

Враховуючи все, що було зазначено вище підсумуємо переваги Selenium:

- Selenium - інструмент з відкритим вихідним кодом;
- може бути розширений для різних технологій, які надають DOM;
- має можливості виконувати скрипти в різних браузерах. може запускати тести в певних версіях Firefox, Chrome;
- може виконувати скрипти в різних операційних системах;
- підтримує мобільні пристрої;
- виконує тести в браузері, тому фокус не потрібно під час виконання сценарію;

- може виконувати тести паралельно з використанням селенових сіток;
- архітектура Webdriver простіше, ніж в інших інструментах, оскільки вона керує браузером з рівня ОС;
- Webdriver взаємодіє безпосередньо з браузером і використовує механізм браузера для управління ним;
- Webdriver працює швидше, так як він безпосередньо взаємодіє з браузером;
- Webdriver може підтримувати безмовне виконання;
- складний і трохи великий API в порівнянні з RC;
- чисто об'єктно-орієнтована API;
- драйвер емулює дії користувача, тобто генерує він все ті ж самі події в браузері, які виникають при роботі справжнього користувача;
- можна працювати з діалоговими вікнами (alert, prompt);
- можна працювати з «нативними» вікнами (діалог завантаження файлів);
- можна працювати з https-протоколами і сертифікатами.

Для роботи з Selenium Webdriver необхідно 3 основних програмних компонента. Розглянемо кожен із них. По-перше це браузер, роботу якого користувач хоче автоматизувати. Це реальний браузер певної версії, встановлений на певній ОС і має свої налаштування (за замовчуванням або кастомниє). Для управління браузером абсолютно необхідний driver браузера. Driver насправді є веб сервером, який запускає браузер і відправляє йому команди, а також закриває його. У кожного браузера свій driver. Пов'язано це з тим, що у кожного браузера свої відмінні команди управління і реалізовані вони по-своєму. Знайти список доступних драйверів і посилання для скачування можна на офіційному сайті Selenium проекту. Скрипт / тест, який містить набір команд певною мовою програмування для драйвера браузера. Такі скрипти використовують Selenium Webdriver bindings (готові бібліотеки), які доступні користувачам на різних мовах.

Розглянемо основні методи та поняття Selenium Webdriver. Webdriver - найважливіша сутність, відповідальна за управління браузером. Основний хід скрипта / тесту будується саме навколо екземпляра цієї сутності. Webelement - друга важлива сутність, що представляє собою абстракцію над веб елементом (кнопки, посилання, інпут і ін.). Webelement інкапсулює методи для взаємодії користувача з елементами і отримання їх теперішнього статусу. By - абстракція над локатором веб елемента. Цей клас інкапсулює інформацію про селекторі (наприклад, CSS), а також сам локатор елемента, тобто всю інформацію, необхідну для знаходження потрібного елемента на сторінці [15].

Також в Selenium WebDriver можна виділити три типи основних команд. Дії - функціональне дію над елементами веб-сторінки або браузером. Наприклад, заповнення полів, натискання на кнопку і інші. Перевірки - виконання перевірок на тестованої сторінці. Наприклад, перевірка того, що певне поле форми має вказане значення, або перевірка заголовка вікна і т.д. .Очікування - організація очікування настання певних подій на сторінці або зі сторінкою (наприклад, очікування завантаження сторінки або очікування завантаження ajax запитів, поява на сторінці певного елемента і т.д.).

Розглянемо також вигідність інформаційної технології тестування web застосувань у порівнянні з русним тестуванням. При стандартних затратах часу на написання, відлагодження та підтримку тестових сценаріїв, коефіцієнт вигоди буде зростати прямо пропорційно до кількості білдів застосунку при використанні інформаційної технології і відповідно зменшуватись при використанні ручного тестування[16].

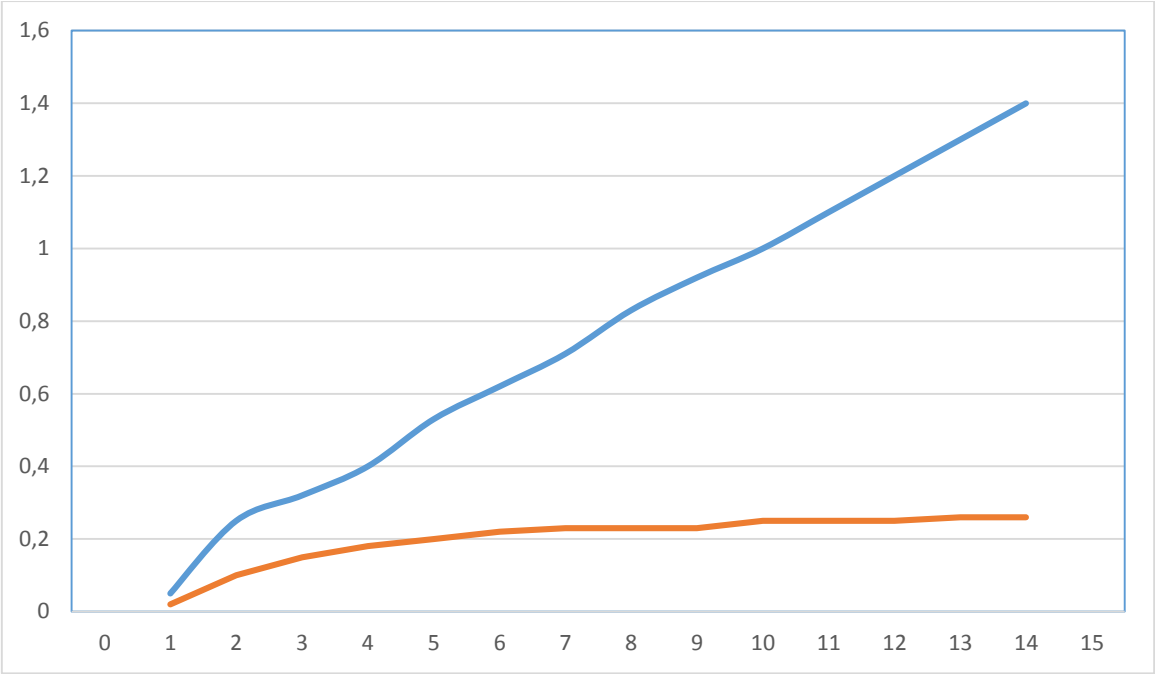


Рисунок 1.2 – Порівняння вигідності автоматизованого та ручного тесування

1.3 Порівняльний аналіз існуючих програм-аналогів

Розглянемо програми, за допомогою яких можна здійснювати тестування веб-застосувань.

Katalon Studio (рисунок 1.3) – це ефективний інструмент для автоматизації процесу тестування веб-додатків, мобільних додатків і веб-сервісів. Katalon Studio є нащадком таких фреймворків, як Selenium і Appium. Він перейняв у останніх безліч переваг, пов'язаних з інтегрованою автоматизацією тестування ПО. Для початку роботи з даним інструментом ви можете як володіти початковими знаннями в тестуванні ПО, так і бути справжнім гуру своєї справи. Люди, далекі від програмування, можуть з легкістю запустити свій проект по автоматизації тестування (наприклад, запустивши функцію Object Spy для запису тестових скриптів), а для програмістів і досвідчених тестувальників Katalon Studio виявиться корисним з точки зору економії часу при написанні нових бібліотек і підтримки існуючих скриптів. Katalon Studio може бути інтегрований в CI / CD, він прекрасно працює в зв'язці з популярними інструментами під час тестування ПО: qTest, JIRA, Jenkins і Git. Для нього

передбачена приємна функція - Katalon Analytics, завдяки якій користувачі отримують повне уявлення про процес тестування. Для цього передбачені спеціальні звіти, які виводяться на екран користувачів у вигляді метрики, діаграм і графіків [17].

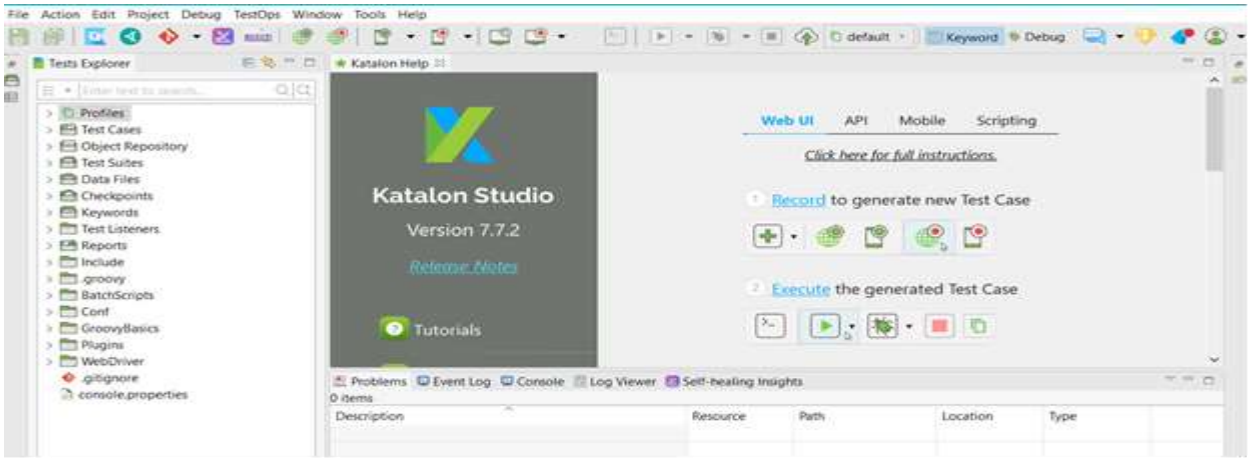


Рисунок 1.3 – Головне вікно програми Katalon Studio

Unified Functional Testing (з англ. Комплексне функціональне рішення для тестування ПО) або UFT (рисунок 1.4) - це популярний комерційний інструмент для функціонального тестування. Він надає повний набір функцій для тестування API, веб-сервісів, а також для тестування графічного інтерфейсу десктопних, мобільних і веб-додатків на всіх існуючих платформах. Для даного інструменту передбачена розширена функція розпізнавання об'єктів на основі зображень, багаторазові тестові компоненти і документація по автоматичному тестування. UFT використовує Visual Basic Scripting Edition, який може стати в нагоді для запису інформації про виконане тестуванні, а також для управління об'єктами. UFT інтегрований з Mercury Business Process Testing і Mercury Quality Center. Інструмент підтримує CI за допомогою інтеграції з інструментами CI, такими як Jenkins [18].

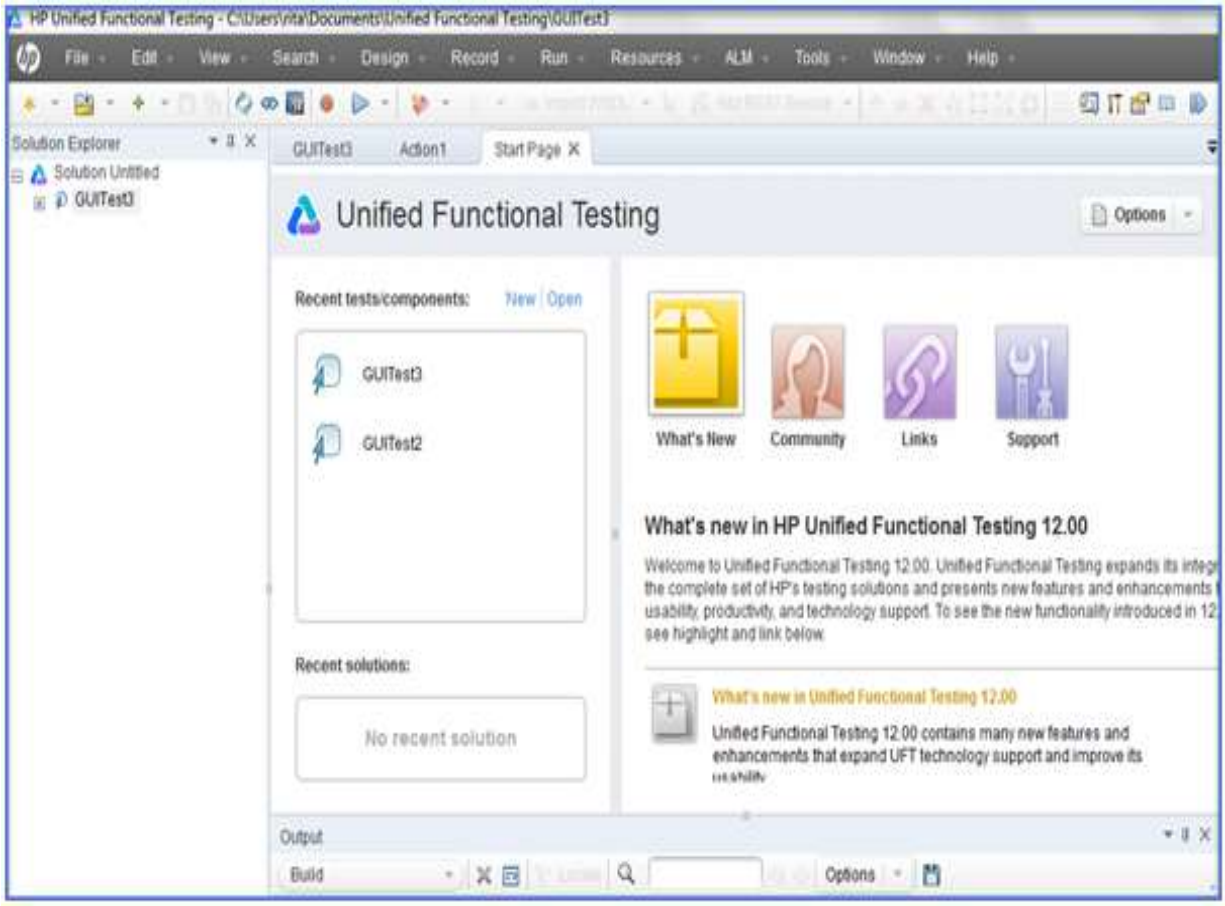


Рисунок 1.4 – Головне вікно програми UFT

TestComplete (рисунок 1.5), створений SmartBear, є ефективним інструментом для тестування десктопних, мобільних і веб-додатків. TestComplete підтримує різні мови сценаріїв, такі як: JavaScript, VBScript, Python і C ++ Script. Також як і у випадку з розглянутим раніше Katalon Studio, за допомогою TestComplete тестувальники можуть виконувати тестування з використанням ключових слів і кероване даними тестування. В інструменті також передбачена зручна функція запису і відтворення процесу тестування. TestComplete має схожу з UTF функцією розпізнавання об'єктів GUI, завдяки якій відбувається автоматичне виявлення і оновлення об'єктів користувальницького інтерфейсу, що допомагає уникнути зайвого клопоту з підтримки тестових скриптів при зміні AUT. Даний інструмент також інтегрується з Jenkins протягом CI-процесу [19].

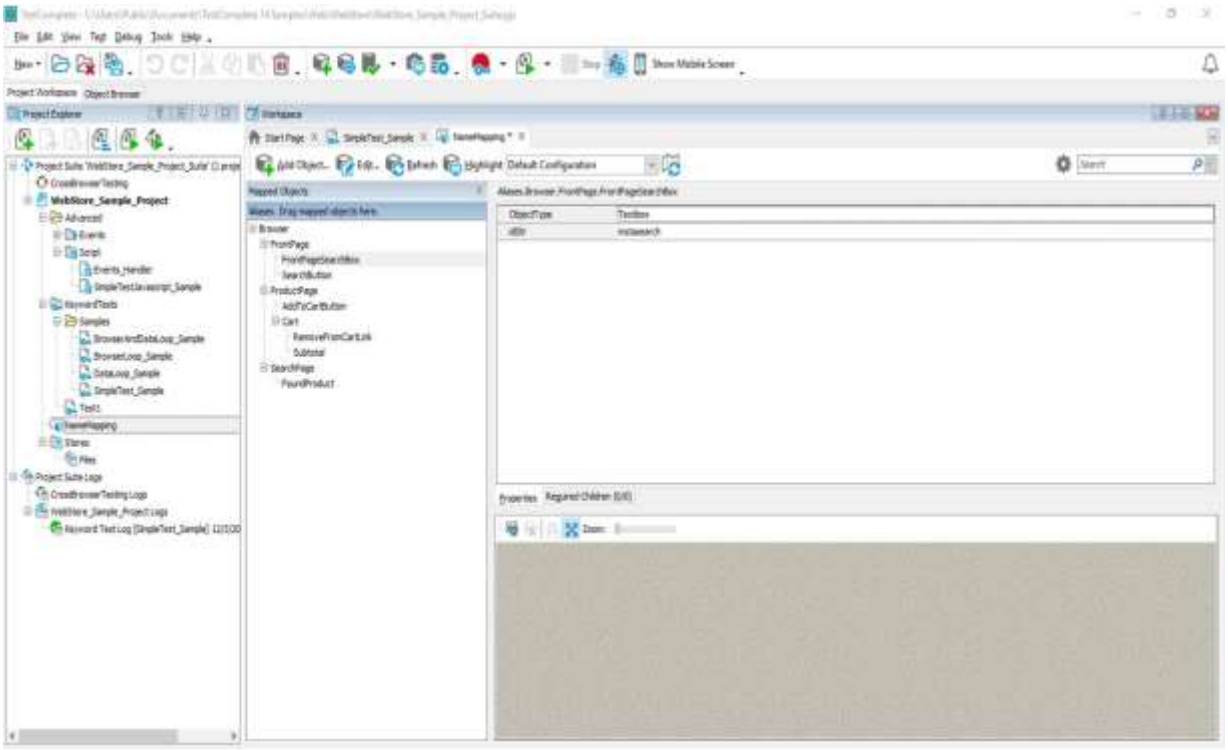


Рисунок 1.5 – Головне вікно програми TestComplete

TestPlant eggPlant (рисунок 1.6) – це інструмент автоматичного функціонального тестування, заснований на аналізі зображень, який дозволяє тестувальникам ефективно виконувати AUT. Що стосується методів тестування, то TestPlant eggPlant повністю відрізняється від традиційних інструментів тестування: в ньому моделювання процесу відбувається таким чином, як якщо б користувач займався тестуванням додатків, а не тестувальник, для якого такий процес полягає в написанні тест-скриптів. Така особливість дозволяє тестувальникам, які не володіють великими знаннями в програмуванні, застосовувати цей інструмент автоматизації тестування інтуїтивно. TestPlant eggPlant підтримується різними платформами, для нього також передбачена можливість управління лабораторією і CI-інтеграції [20].

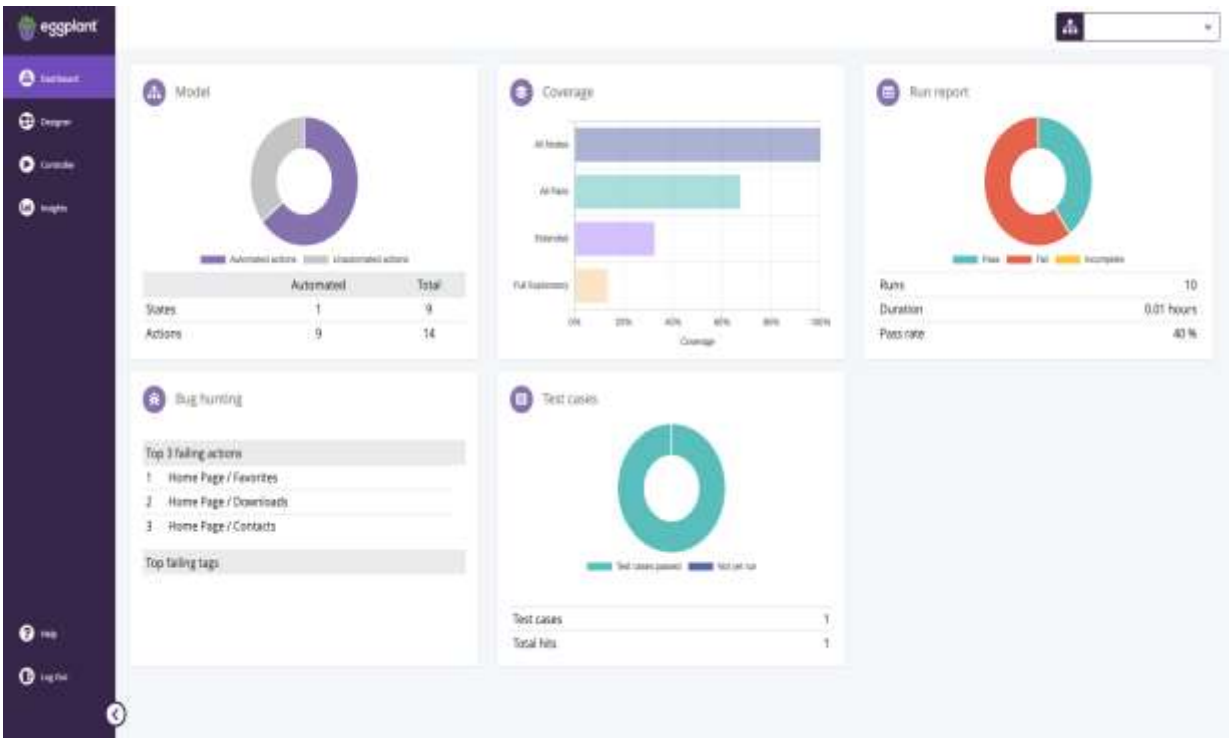


Рисунок 1.6 – Головне вікно програми TestPlant eggPlant

Ranorex (рисунок 1.7) – це платний універсальний інструмент для автоматизації тестування веб-, мобільних та десктопних додатків. Інструмент характеризується розширеними можливостями для розпізнавання GUI, застосуванням багаторазових тестових сценаріїв і можливістю запису / відтворення етапів тестування ПО. Ще однією корисною рисою даного інструменту є можливість створення тестових сценаріїв без необхідності писати код. Для тих тестувальників, хто знаходиться на самому початку шляху, така особливість виявиться прекрасною підмогою: тестувальникам не потрібно буде мати поглибленими знаннями в програмуванні, для того щоб проводити автоматичне тестування своїх проектів [21].

Результати порівняльного аналізу інструментів-застосунків, що надають можливість здійснювати процес тестування веб-застосувань зобразимо в таблиці 1.1. Варто звернути увагу на переваги та недоліки описаних систем, щоб врахувати їх у розроблюваній системі.

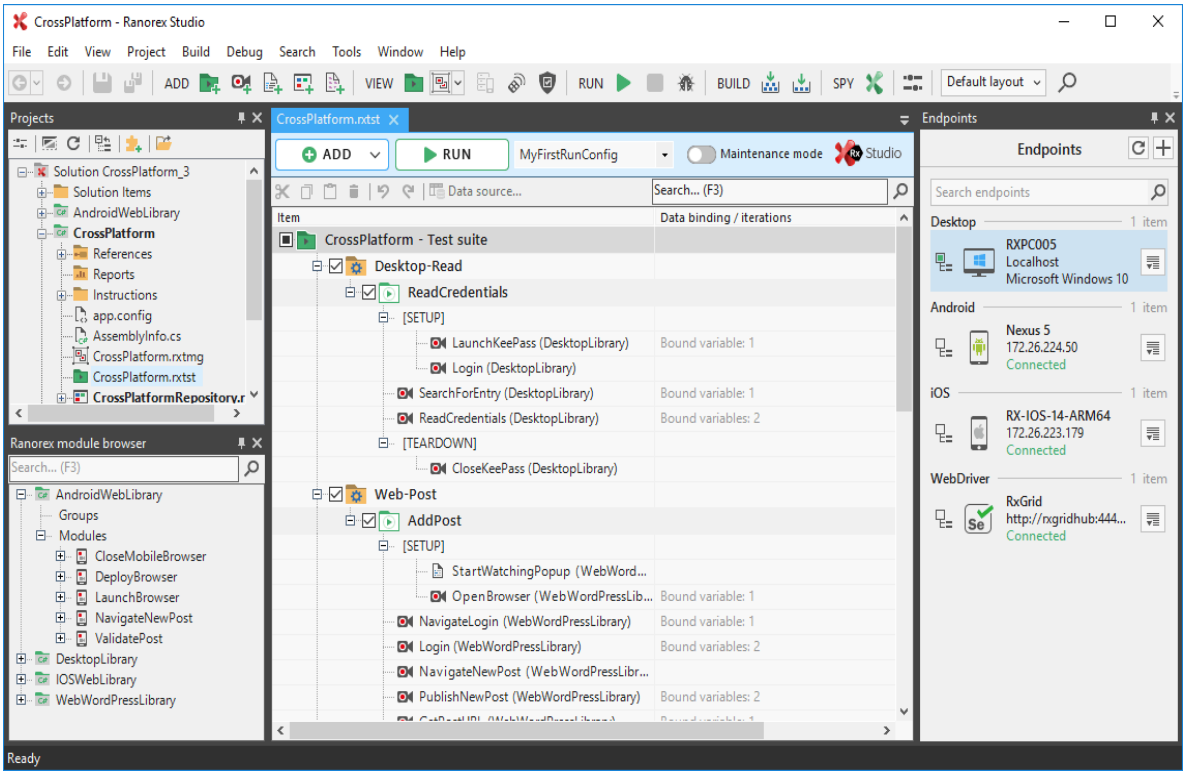


Рисунок 1.7 – Головне вікно програми Ranorex

Таблиця 1.1 – Аналіз програм-аналогів тестування веб-застосувань

Особливості / **Засторсунок**	**Доступність**	**Призначення**	**Переваги і недоліки у використанні**
1	2	3	4
Katalon Studio	платний	тестувальник	*Переваги*: можливість автоматичного створення звітів, не потребує великих знань програмування *Недоліки*: складний інтерфейс, потребує написання сценарію-скриптів тестування, відсутність україномовного інтерфейсу

Таблиця 1.1 – Продовження

1	2	3	4
Unified Functional Testing	платний	тестувальник	*Переваги*: можливість тестування графічного інтерфейсу *Недоліки*: потрібно знати програмування, потребує написання сценарію-скриптів тестування, відсутність україномовного інтерфейсу
TestComplete	платний	тестувальник	*Переваги*: наявність функції запису і відтворення процесу тестування, підтримка багатьох мов програмування *Недоліки*: потрібно знати програмування, потребує написання сценарію-скриптів тестування, відсутність україномовного інтерфейсу
TestPlant eggPlant	платний	Користувач, тестувальник	*Переваги*: не складний у використанні, моделює роботу користувача *Недоліки*: потребує написання автоматизованих сценаріїв-скриптів тестування під кожен окремий проект, відсутність україномовного інтерфейсу, платна ліцензія для використання
Ranorex	платний	тестувальник	*Переваги*: доступний для тестувальників початківців, які не мають знань програмування *Недоліки*: потребує купівлі платної ліцензії для використання відсутність україномовного інтерфейсу

Результат аналізу показав, що більшість застосунків платні, не містять україномовного інтерфейсу потребують повне написання сценарію під кожен новий об’єкт тестування, а також потребують знань мов програмування, а зазвичай тестувальники не володіють великими знаннями в програмуванні, що є частим явищем для інженера з забезпечення якості та не містять повного набору

тестів під будь-який стандартний застосунок, а це є досить суттєвими недоліками для потенційних користувачів, тобто тестувальників

Отже, процес тестування варто автоматизувати, щоб зменшити фінансові та часові витрати на здійснення тестів, підвищити ефективність та виключити фактор допущення помилки. Основною задачею програмного модуля є покрокове виконання сценарію тестів функціоналу веб-застосунків та формування відповідного звіту. Використання бібліотеки Selenium дозволяє досягти автоматизації процесу тестування.

Проблема "людського фактору" при ручному є однією із найголовніших проблем сфери тестування програмного забезпечення, саме тому автоматизація процесу тестування та створення програмного модуля для тестування дасть змогу дозволить прибрати дану проблему, скоротити час на виконання тестів та забезпечить високу ефективність їх здійснення.

1.4 Висновок до розділу 1

Описаний вище розділ присвячено аналізу методів та існуючих застосунків для автоматизованого тестування програмного забезпечення. У ході роботи над даним розділом було проаналізовано предметну область тестування веб-застосувань, визначено предмет, об'єкт та мету, описано та проаналізовано програми аналоги, проаналізовано практичне значення та сферу застосування предметної області.

2 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ТЕСТУВАННЯ WEB ЗАСТОСУВАНЬ

2.1 Аналіз та обгрунтування вибору складових компонентів інформаційної технології WEB застосувань

Інформаційна технологія – сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів [22].

Інформаційна технологія тестування веб застосувань – це сукупність програмних компонентів, що мають на меті забезпечити потребу користувачів у організації та проведенні процесу автоматизованого функціонального тестування, шляхом надання відповідних технічних засобів.

Оскільки мова йде про тестування саме функціоналу веб-застосувань, то варто зазначити, що потрібно брати до уваги саме ті існуючі інформаційні технології, які призначені для вирішення цієї проблеми. Тобто йдеться про тестування саме додатків, що написані на HTML, і відповідно для них не підійдуть інструменти, фреймворки та бібліотеки, які призначені для тестування додатків, написаних наприклад на Java чи .NET, а оскільки в роботі розглядається тестування саме веб-застосувань, то важливо враховувати кроссбраузерність обраної технології, для її використання.

Як було зазначено в попередньому розділі, Selenium та WebDriver дозволяють досягнути процесу автоматизації та можуть бути використані для створення універсальних сцернаріїв-шаблонів, що будуть використовуватись при створенні інформаційної технології тестування WEB застосувань. Selenium як програмна бібліотека, котра містить повний набір функцій, атрибутів та інших елементів, представлених у вигляді вихідного коду, що доступні користувачеві та надають вичерпні можливості користування технологією та інструмент, дасть смогу створити тестові сценарії. Кожен сценарій повинен відповідати техніці тест-дизайну, наприклад як класи еквівалентності, чи застосування методів

граничних значень. Зокрема повинні бути доступні не лише позитивні, але і негативні сценарії, оскільки система повинна проходити перевірку на стабільність при невалідних даних.

Також важливим є перелік функціоналу, що буде перевірятись. На основі порівняння багатьох різних але стандартних та подібних WEB застосувань було виявлено що вони містять подібний функціонал, наприклад конпки, радіокнопки, чекбокси, текстові поля, форми заповнення, форми реєстрації, форми авторизації, активні посилання, випадачючі списки. Також до цього переліку варто віднести різноманітні лейбли, банери, написи, які також потрібно перевірити на правильність написання.

Для того, аби автоматизоване тестування, яке буквально повторює дії користувача, було можливим, у програмному модулі повинна бути передбачена процедура пошуку елементів веб-застосування, оскільки в даному випадку функціональне тестування можливе лише при їх наявності. Цю проблему можна вирішити таким чином. Коли тестувальник виконує роботу з перевірки програмного забезпечення та створює автоматизовані тести під кожен окремий застосунок, в сценаріях тестування, окрім дій повинні бути вказані ще саме елементи сторінки, що перевіряються. Інформація про веб елемент прописується вручну для кожної команди перевірки роботи веб-застосування. Якщо перевіряється працездатність, наприклад, кнопки веб-додатку, то для того , щоб написаний автотест зміг її знайти, потрібно вказати її локатор. Цей процес можна зробити більш універсальним та спростити, створивши процедуру пошуку потрібних елементів будь-якого стандартного веб-сайта за локаторами цих елементів. У такому випадку написання кожного окремого сценарію вже не буде обов’язковою необхідністю та суттєво зменшить час, що витрачається на написання сценарію тестування та підвищить його ефективність.

Також важливим компонентом є користувацькі інтерфейси. В більшості випадків вони поділяються на консольні та графічні. В даному випадку інтерфейс розроблюваної технології повинна мати зрозумілий та зручний

програмний інтерфейс з високим різноманіттям функцій перевірки та атрибутів тестування.

Отже, проаналізовано елементи, що має містити інформаційна технологія тестування WEB застоувань та визначено її компоненти, для підвищення ефективності процесу автоматизованого функціонального тестування WEB застосувань та задоволення потреб користувачів.

2.2 Розробка структурної схеми інформаційної технології тестування WEB застосувань

Програмне забезпечення для тестування WEB застосувань складається з таких компонентів:

- модуль інтерфейсу користувача
- модуль перевірки валідності веб-застосування
- модуль об’єктів веб-застосування
- модуль управління браузерами
- модуль тестових сценаріїв
- модуль виведення результатів

Розглянемо кожну компоненту інформаційної технології та опишемо функції та задачі покладені на них.

Спочатку запускається модуль інтерфейсу користувача, який зв'язаний з модулями результатів тестування та модулем перевірки валідності веб-застосування. Він відповідає за введення та виведення інформації, чітке та правильне відображення для користувача-тестувальника необхідної інформації та результатів тестування .

Модуль перевірки валідності веб-застосування пов’язаний з модулем запуску сценаріїв тестування. Він містить алгоритм перевірки валідності веб-додатку, тобто надає можливість виконати smoke-тестування, а саме перевірити чи є тестований застосунок працездатним, оскільки тільки у такому випадку є сенс проводити подальше тестування.

Модуль об’єктів веб-застосування, що пов’язаний з модулем запуску сценаріїв тестування. У ньому здійснюється пошук елементів тестування за локаторами та селекторами, та інтегрування відповідних веб-елементів у сценарій тестування.

Модуль роботи з драйвером браузера пов’язаний з запуском сценаріїв тестування та з модулем об’єктів веб-застосування. Він виконує роль керування браузера, тобто надає доступ до веб-застосунку і дає можливість симуляції дій користувача.

Модуль тестових сценаріїв, який пов’язаний з модулем роботи з драйвером браузера та модулем об’єктів веб-застосування є головним ядром програми. У ньому, після того як модуль об’єктів поверне усі знайдені веб-елементи сторінки, для кожного з них почергово запускатиметься свій окремий сценарій перевірки.

Модуль виведення результатів пов’язаний з модулем запуску сценаріїв тестування та модулем інтерфейсу користувача і відповідає за роль виведення інформації про завершене поточне тестування у вигляді відповіді запита зі сторінки пр прецездатність певного елементу, а також скріншоту результату.

Розроблена загальна структурна схема функціонування інформаціної технології тестування WEB застосувань зображена на рисунку 2.1.

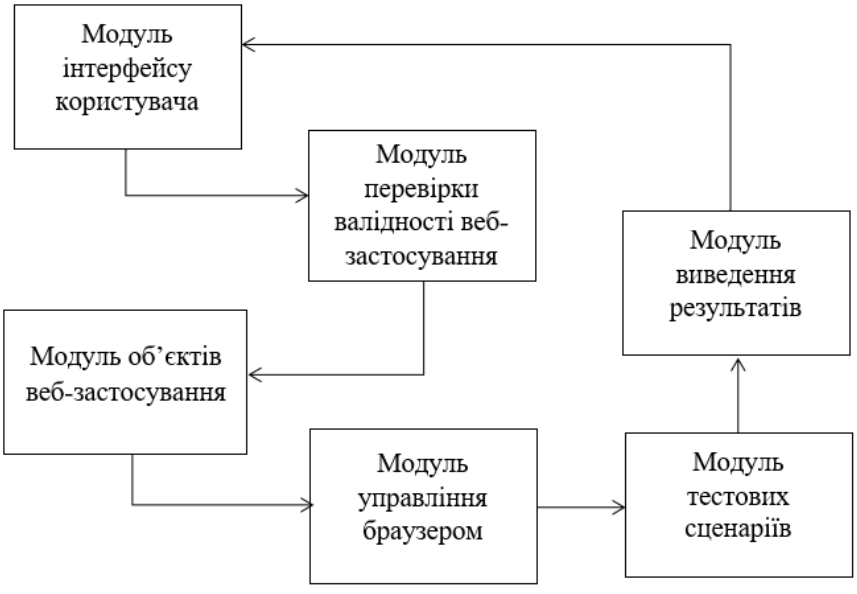


Рисунок 2.1. - Загальна структурна схема функціонування інформаціної технології тестування WEB застосувань

2.3 Удосконалення математичної моделі процесу тестування WEB застосувань

Оптимізація тестування WEB застосувань – це особлива проблема розподілу ресурсів. Тому, призначимо кожному робочому шляху програми його ефктивність виконання тесту (наприклад, пов'язану з часом виконання) та його якість виконання тесту, пов'язану з можливими знайденими помилками (наприклад, пов'язану з кількістю інструкцій та/або блоків програми, що виконуються).

Можливі два підходи щодо визначення моделей оптимізації тестування:

M_1 - модель з обмеженими ресурсами.

Метою цієї моделі є максимізація загальної ефективності при обмеженні ресурсів на виконання тестів з огляду на максимальну загальну якість виконання.

M_2 - модель підтримки якості. Метою цієї моделі є мінімізація витрат ресурсів на виконання тестів з огляду на мінімальну загальну ефективність виконання

Залежність ефективності тестів від ресурсів (час, обчислення, обсяг роботи) є асимптотичною і наведена на рисунку 2.2. Це означає, що серед усіх можливих тестів лише частина з них технічно та економічно корисна

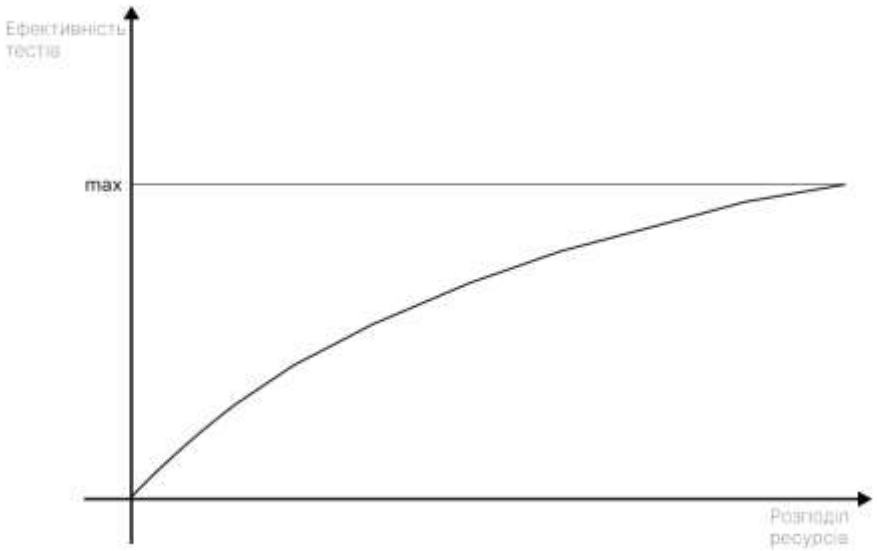


Рисунок 2.2 – Залежність ефективності тестів від розподілу ресурсів

Визначимо змінні математичної моделі.

1. Програмний блок застосунку. Складається з набору послідовних інструкцій, що виконуються послідовно.

2. Шлях програми. Послідовність повторюваних блоків (не обов'язково послідовних)

3. Довжина шляху. Кількість повторюваних блоків у шляху програми.

4. Простий шлях. Шлях без повтору модулів.

5. Виконуваний шлях. Шлях, для якого існують деякі вхідні дані, що змушують його виконувати. Підмножини виконуваних шляхів можна вибирати за допомогою типових методів функціонального тестування, не вимагаючи побудови тестів, що включають усі можливі комбінації значень вхідних даних, але лише ті, що мають значення для функціональності програми.

Припустимо, що визначаємо в середині застосунку M модулів b_i ($i = 1 \ldots M$)

Нехай N — це кількість об’єктів тестування у застосунку для запущених сценаріїв через шлях до об’єкту p_j ($i = 1 \ldots N$), визначених через функціональні тести.

6. Композиційна матриця. Матриця $A = \{ a_{i,j} \}$ визначається як $a_{i,j} = 1$ якщо блок b_i належить до виконуваного шляху $p_{j,0}$, інакше (для $i = 1 \ldots M$ та $j = 1 \ldots N$) . Його стовпці характеризують блоки b_i кожного p_j шляху, а його рядки характеризують шлях p_j для кожного блоку b_i.

7. Ступінь покриття. Ступінь покриття виконуваного шляху — це співвідношення між кількістю його окремих блоків і кількістю усіх блоків веб застосунку, тобто:

$$T_j = \frac{1}{M} \cdot \sum_{i=1}^{M} a_{i,j} \qquad (j = 1 \ldots N) \tag{2.1}$$

8. Матриця виконання. Матриця $E = \{ e_{i,j} \}$ де $e_{i,j}$ представляє кількість разів коли для блоку b_i виконується тест на тестовому шляху до об'єкта тестування $p_j (i = 1...M$ та $j = 1...N)$.

9. Довжина блоку, що виражається кількістю операторів. Пов'яжемо кожен виконуваний шлях p_j зі змінною $x_j = 1$, якщо p_j вибрано до тестування, інакше 0 ($j = 1...N$).

Нехай c_j — ресурсний показник виконання тесту з тестовим шляхом до об'єкта тестування p_j і v_j — ефективність виконання. Наприклад, спочатку можна вважати ефективність пропорційною лише часу виконання, а не залежністю від часу витраченого на їх розробку, запуск та виявлення помилок. Тоді значення ефективність виконання тесту можна вважати рівним кількості операторів (рядків коду), залучених тестовим сценарієм під час виконання p_j:

$$v_j = \sum_{i=1}^{M} e_{i,j}\, l_i \qquad (j = 1 \dots N) \tag{2.2}$$

Щоб урахувати кількість блоків, охоплених кожним тестовим шляхом p_j, ми коригуємо його значення через ступінь покриття, T_j, таким чином ефективність тестового виконання буде:

$$v_j = T_j \cdot \sum_{i=1}^{M} e_{i,j}\, l_i \qquad (j = 1 \dots N) \tag{2.3}$$

Цільова функція z_1 моделі оптимізації тестування M_1 (представляє загальну оптимальну ефективність виконання тестів):

$$z_1(x) = \sum_{j=1}^{N} v_j\, x_j \rightarrow max \tag{2.4}$$

Для моделі M_2 цільова функція, що представляє загальну якість виконання тестів, є:

$$z_2(x) = \sum_{j=1}^{N} c_j \; x_j \rightarrow max \tag{2.5}$$

У моделі M_1 максимізація загального значення ефективності повинна здійснюватися для заданої максимальної загальної якості виконання тесту, c_{max}, тобто:

$$\sum_{j=1}^{N} c_j \, x_j \leq c_{max}, x_j = \{0,1\} \; (j = 1 \ldots N) \tag{2.6}$$

Модель M_2 вимагає, щоб була досягнена якість для заданого мінімального загального значення ефективності виконання тесту, v_{min}, тобто:

$$\sum_{j=1}^{N} v_j \, x_j \leq v_{max}, x_j = \{0,1\} \; (j = 1 \ldots N) \tag{2.7}$$

Обидві моделі M_1 та M_2 - цілочисельні двовалентні моделі лінійного програмування.

Попередні базові моделі не враховують коефіцієнт покриття програми, який можна визначити як відношення між кількістю окремих модулів для яких визначено окремі тестові сценарії шляхах і загальною кількістю модулів у програмі.

Оскільки цільова функція $M_1 \; ' \; s$ і обмеження $M_2 \; ' \; s$ враховують загальну кількість виконаних, але не обов’язково окремих блоків, може статися так, що тестові сценарії, вибрані в оптимальному рішенні M_1 або M_2, охоплюють кілька окремих модулів, менше за, які можна охопити іншим неоптимальним (але,

очевидно, кращим) рішенням: тобто ефект надмірності коду написаного для тестів у суттєво різних сценаріях тестового випадку не враховується.

Однак ефект надлишковості можна визначити для кожного модулю b_i як кількість шляхів до об'єкту тестування , у яких присутній однаковий функціонал, відносно загальної кількості таких шляхів

$$RE_i = \frac{1}{N} \cdot \sum_{j=1}^{N} a_{i,j} \qquad (2.8)$$

Глобальний ефект надмірності визначається середнім значенням усіх RE_i ' *s*, тобто:

$$GREB = \frac{1}{M \cdot N} \cdot \sum_{i=1}^{M} \sum_{j=1}^{N} a_{i,j} \qquad (2.9)$$

Щоб урахувати ефект надмірності, значення ефективності тесту може бути скориговано відповідно до значення кількості повторюваного функціоналу кожного модуля. Це відповідає пов'язанню з кожним сценарієм N часткових значень (по одному для кожного шляху), що представляє розподіл ресурсів, тобто рядків коду залучених для різних тестових сценаріїв, але для одного й того ж функціоналу для усіх модулів застосування.

Введемо показник w_{ji}, часткове значення, пов'язане з модулем b_i у шляху p_j. Тоді w_{ji}, буде додатнім лише для тих шляхів, яким він належить, тобто:

$$\omega_{i,j} \begin{cases} > 0, \text{ якщо блок } b_i \text{ належить до шляху } p_j \\ \text{інакше} = 0 \end{cases}$$

А сума часткових значень ефективності тестів для модулів буде загальним значенням ефективності v_j, тобто:

$$v_j = \sum_{i=1}^{M} \omega_{i,j} \qquad (j = 1 \dots N) \tag{2.10}$$

Тоді, з формул (2.3) і (2.10) отримаємо:

$$\omega_{i,j} = e_{i,j}\, l_i T_j \qquad (i = 1 \dots M; j = 1 \dots N)$$

Щоб отримати найкраще загальне значення, тобто максимальну залученість ресурсів для різних модулів припустимо, що до включення у тестовое рішення мають залучатись ресурси у тих модулях, у яких вони мають максимальне часткове значення w_{ji}. З цього загальне значення ефективності виконання тесту з урахуванням ефекту надмірності блоків визначається загальною сумою максимальних часткових значень кожного блоку.

З вищесказаного, сформуємо нову формулу моделі M_1 :

$$max\ z_1(x) = \sum_{i=1}^{M} \max_{j=1\dots N} \{\omega_{i,j} x_j\},\ \text{якщо}\ \sum_{j=1}^{N} c_j\, x_j \le c_{max}, x_j = \{0,1\}\ \ (j = 1 \dots N).$$

Модель M_2 :

$$min\ z_2(x) = \sum_{i=1}^{N} c_j\, x_j,\ \text{якщо} \sum_{i=1}^{M} \max_{j=1\dots N} \{\omega_{i,j} x_j\} \ge v_{min},\ x_j = \{0,1\}\ (j = 1 \dots N).$$

Отже, використовуючи два підходи, можемо обчислити індекс повноти тесту без надлишковості для обох моделей як співвідношенням між отриманим загальним значенням тесту (результат тестування) та бажаним загальним значенням тесту (початковий тестовий набір), тобто:

$$R = \frac{\sum_{i=1}^{M} \max\limits_{j=1\dots N} \{\omega_{i,j} x_j\}}{\sum_{i=1}^{M} \max\limits_{j=1\dots N} \{\omega_{i,j}\}}$$

Оскільки запропоновані моделі оптимізації схожі, тепер маючи показник *R*, можемо зробити висоновок, що перша модель більше підходить для реальних ситуацій, тому використаємо модель M_1.

Оскільки M_1 є моделлю лінійного програмування, найефективнішими (від простоти застосування та швидкості збіжності до оптимального рішення) є алгоритми, які базуються на техніці «розгалуження та межі». Зокрема, підходящим є алгоритм Ленда та Дойга. Цей алгоритм має деякі цінні переваги, такі як:

- проста у застосуванні;
- максимальна кількість кроків для зближення становить приблизно 2^N (де N — кількість обраних p_j шляхів, які можна виконати);
- це комбінаторний тип, швидший, ніж перелічувальний алгоритм.

Просуваючись вперед у застосуванні алгоритму, початкова задача трансформується у все більш прості, виражені аналогічно моделі, але зі все меншим числом змінних[23]. Формулювання алгоритму для даного рішення моделі таке:

$$max(x) = \sum_{i=1}^{M} \max_{j=1\dots N} \{\omega_{i,j} x_j\}, \text{ якщо}$$

$$\sum_{j=1}^{N} c_j\, x_j \le c_{max}, x_j = \{0,1\} \quad (j = 1 \dots N).$$

Тепер на основі удосконаленої математичної моделі ми можемо створити алгоритм, який буде застосовуватись при створенні кожного тестового сценарію. Розглянемо основні кроки алгоритму, що базується на даній удосконаленій моделі тестування WEB застосвань:

1. Вибір веб застосунку для проведення тестування
2. Визначається детальний перелік функціоналу WEB застосування (вручну);
3. Визначається шлях до конкретного веб-елементу (за локатором конкретного веб-елементу);
4. Визначається конкретний модуль WEB застосування, в якому міститься даний функціонал;
5. Виявляється елемент, на який поширюється дія функціоналу;
6. Шлях інтегрується в сценарій тестування
7. Запускається процес тестування .
8. Формування результату.

Отже дана математична модель дає змогу повністю покрити функціонал тестовими сценаріями, позбавлена надлишковості, тобто зайвих кроків під час проходження тест-кейсу.

2.4 Комп’ютерне моделювання інформаційної технології тестування WEB застосувань

UML - є графічною мовою для візуалізації, опису параметрів, конструювання та документування різних систем (програм зокрема). Діаграми створюються за допомогою спеціальних CASE засобів, наприклад Rational Rose і Enterprise Architect. На основі технології UML будується єдина інформаційна модель. Наведені вище CASE засоби здатні генерувати код на різних об'єктно-

орієнтованих мовах, а так само мають дуже корисною функцією реверсивного інжинірингу. Реверсивний інжиніринг дозволяє створити графічну модель з наявного програмного коду і коментарів до нього [24].

Існують такі типи діаграм для візуалізації моделі:

- діаграма варіантів використання (use-case diagram);
- діаграма класів (class diagram);
- діаграма станів (statechart diagram);
- діаграма послідовності (sequence diagram);
- діаграма кооперації (collaboration diagram);
- діаграма компонентів (component diagram);
- діаграма розгортання (deployment diagram).

Розглянемо use-case діаграму та діаграму послідовності. Use-case діаграма відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів , що дозволяє описати систему на концептуальному рівні. Прецедент - можливість модельованої системи (частина її функціональності), завдяки якій користувач може отримати конкретний, вимірний і потрібний йому результат. Прецедент відповідає окремому сервісу системи, визначає один з варіантів її використання і описує типовий спосіб взаємодії користувача з системою. Варіанти використання зазвичай застосовуються для специфікації зовнішніх вимог до системи [25].

Use-case діаграма містить кілька типів зв’язків, таких як узагальнення, асоціація включення - <<include>>, що означає відношення між двома елементами моделі, де зміна незалежного змінює залежний елемент та розширення - <<extand>>, що означає дію, яка може викконуватись за певних умов.

Use-case діаграма користувача інформаційної технології тестування WEB застосувань відображає доступні дії відображає такі дії, як ознайомлення з інформацією про проект, що включає перегляд інформації про проект, ознайомлення з інструкцією користувача, що включає перегляд інструкції користувача, проведення тестування WEB застосувань, що включає такі дії, як

створення тест кейсу, вибір типу функціоналу та сценарію, введення даних об’єкта, що тестується, створення нового тест кейсу та запуск процесу тестування, перегляд результату тестування, що включає усі попередні кроки з проведення тестування, а також завантаження скріншоту-результату тестування.

На рисунку 2.3 зображено use-case діаграму, що відображає доступні дії користувача інформаційної технології тестування WEB застосувань.

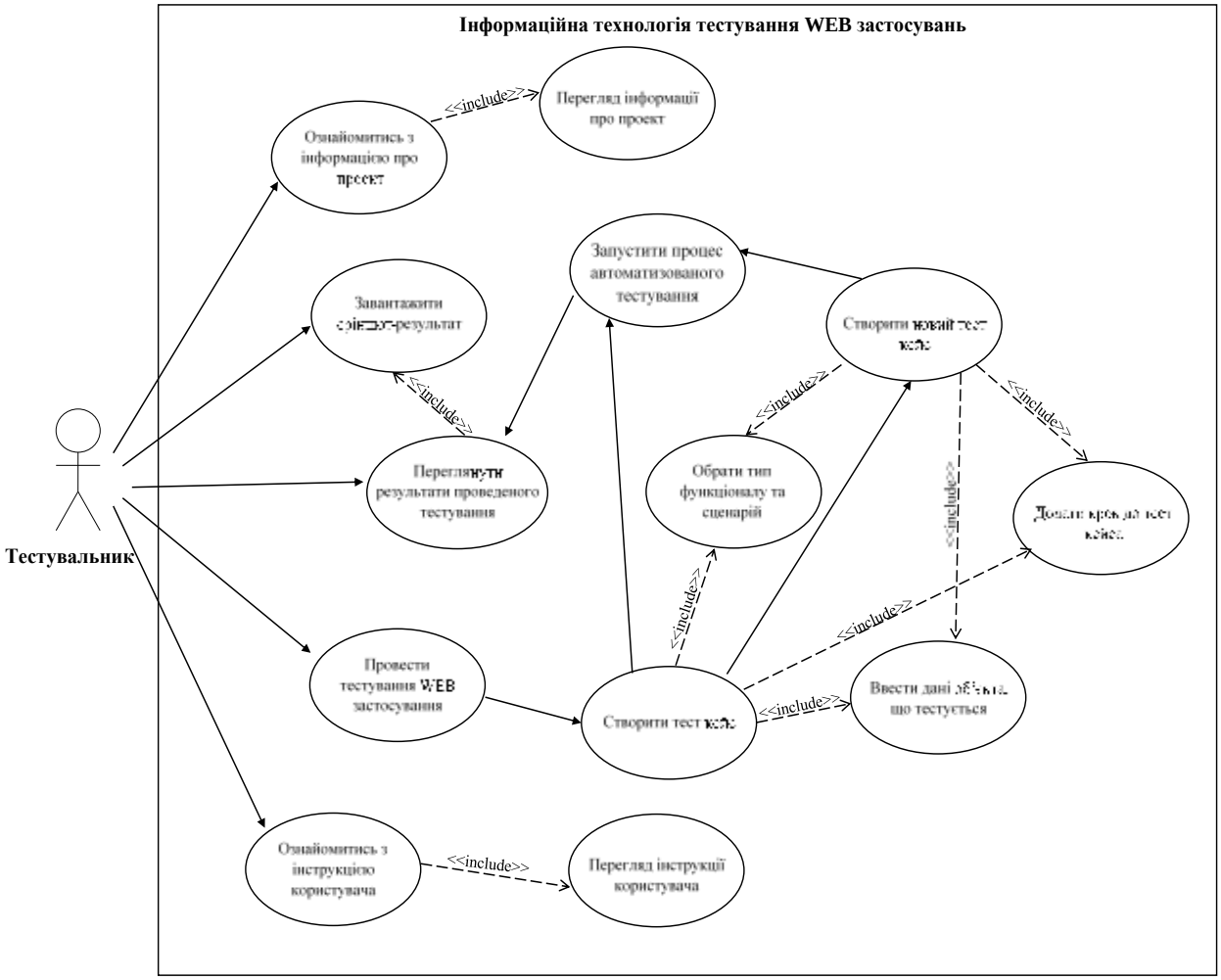


Рисунок 2.2 – Use case діаграма доступних дій користувача

Діаграма послідовності - UML-діаграма , на якій для деякого набору об'єктів на єдиній тимчасової осі показаний життєвий цикл об'єкта (створення-діяльність-знищення якоїсь сутності) і взаємодія акторів (дійових осіб) інформаційної системи в рамках прецеденту. Основними елементами діаграми послідовності є позначення об'єктів (прямокутники з назвами об'єктів),

вертикальні «лінії життя» , що відображають плин часу, прямокутники, що відображають діяльність об'єкта або виконання ним певної функції (прямокутники на пунктирній «лінії життя»), і стрілки, що показують обмін сигналами або повідомленнями між об'єктами [26].

Процес відбувається наступним чином, спочатку користувачь вводить через інфтерфейс усі необхідні дані про WEB застосуванн що тестується. Далі інформація через інтерфейс передається до самої програми, , де виконується пошук об’єетів тестуваня і відповідно перевірка правильності введенних даних, оскільки якщо не буде знайдено хоча б одної відповідності, то сценарії не запустяться. Після того як всі дані було знайдено, через WebDriver надається доступ до браузеру і на рівні серверу виконується тестування після чого дані повертаються до програми і через інтерфейс користувач може ознайомитись з результатами тестування.

На рисунку 2.4 зображено загальну UML-діаграму послідовності інформаційної технології тестування WEB застосувань.

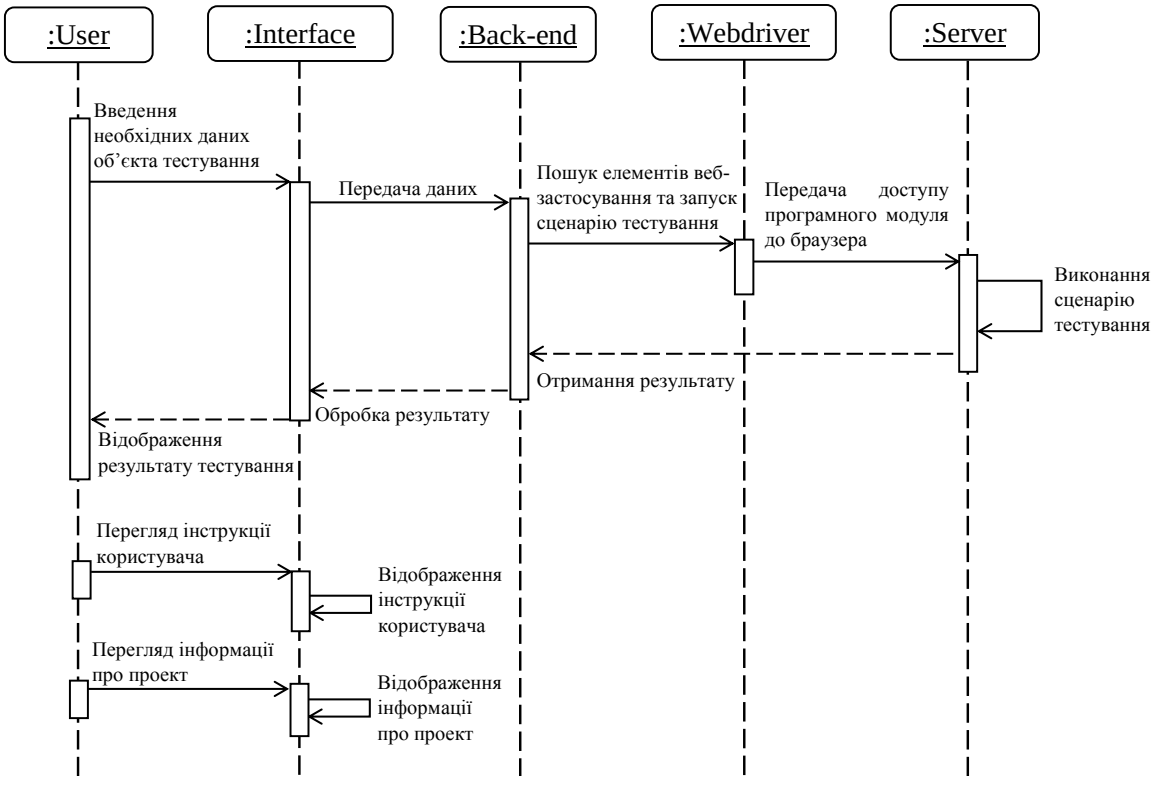


Рисунок 2.4 – UML-діаграма послідовності інформаційної технології тестування WEB застосувань

2.5 Розробка алгоритму функціонування інформаційної технології тестування WEB застосувань

Розглянемо основні етапи роботи програмного забезпечення тестування WEB застосувань.

Коли користувач запускає програмний продукт, запускається початкове вікно програми, з якого можна перейти до наступного головного вікна програми. Після цього потрібно обрати опцію для подальшої роботи з програмним модулем. Якщо користувач одразу обирає перейти до тестування, потрібно натиснути відповідну і кнопку перейти до вікна тестування та ввести посилання на WEB застосування, перевірка якого буде здійснюватись, а також необхідні дані об’єктів тестування, працездатність яких буде перевірятись. Після того як дані введені, відбувається перевірка посилання на валідність, а також відбувається процес пошуку елементів веб-застосунку. Якщо посилання не дійсне, і/або веб-елементи за вказаними локаторами не вдалось знайти то з’являється відповідне повідомлення, після чого повертаємось до попереднього кроку і вводимо дані ще раз, якщо посилання пройшло перевірку, та веб елементи знайдено то далі відбувається запуск сценаріїв тестування. Після того як тестування розпочато далі запускається Webdriver, який надає доступ до керування браузером та сценарій тестування, після чого виконується безпосередній процес тестування, після чого він завершується. Після того, як тестування звершено, відбувається перехід до вікна очікування результатів і поки іде обробка даних виводиться відповідне повідомлення. Після того як відбулась обробка даних здійснюється виведення результатів на екран, після чого можна їх переглянути і повернутись до головного меню. Якщо користувач хоче ознайомитись з інструкцією користувача здійснюється перехід до відповідного вікна, з якого можна повернутись до головного вікна програми. Якщо користувач хоче ознайомитись з інформацією про проект, здійснюється перехід до відповідного вікна, з якого можна повернутись до головного вікна програми. Якщо користувач не бажає продовжувати роботу з програмним

модулем, потрібно повернутись до головного вікна на натиснути відповідну кнопку. На рисунку 2.5 наведено загальну схему алгоритму функціонування інформаціної технології тестування WEB застосувань.

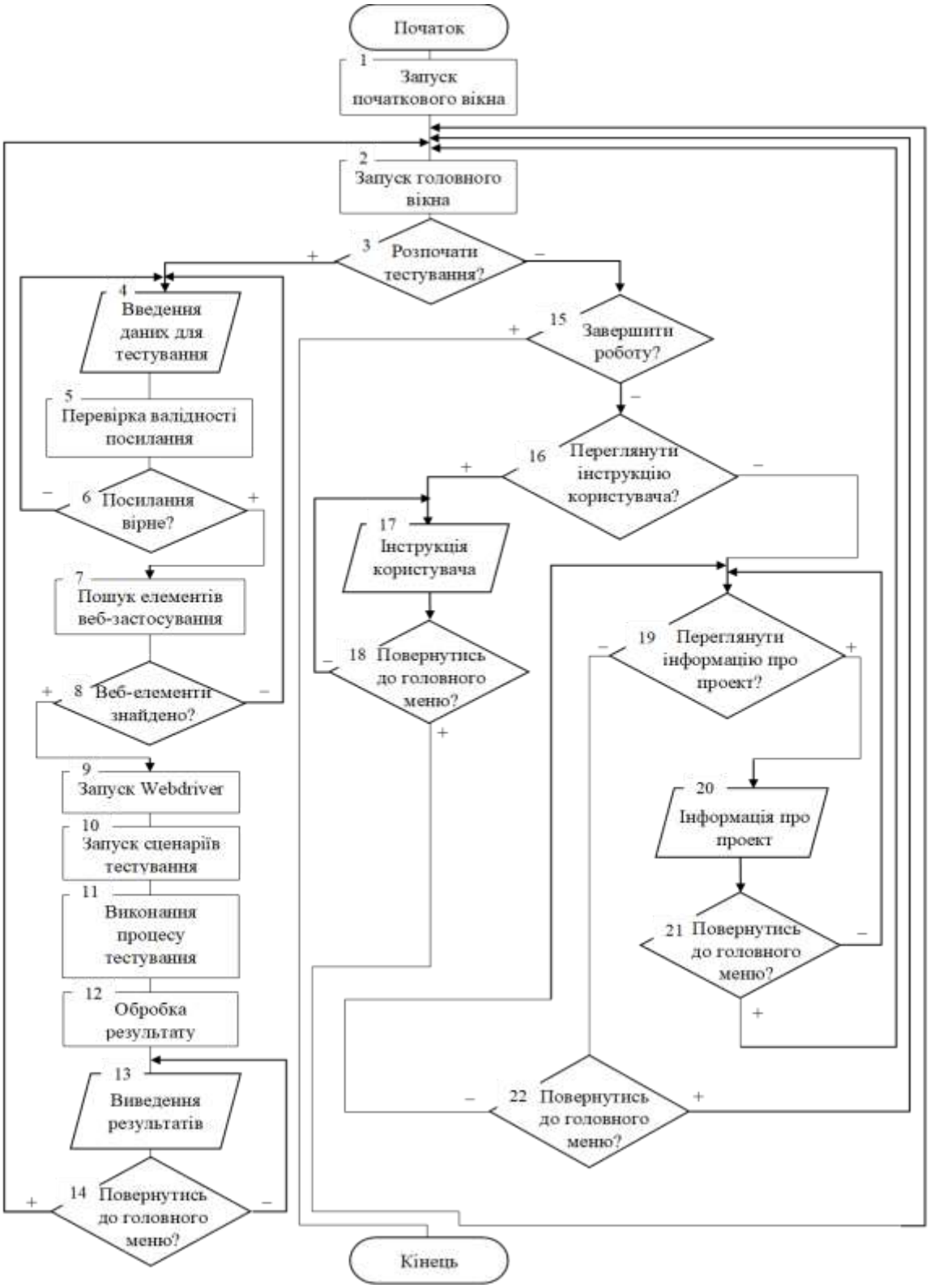


Рисунок 2.5 – Алгоритм функціонування інформаціної технології тестування WEB застосувань

2.6 Висновок до розділу 2

У даному розділі було проведено аналіз та обгрунтування вибору складових компонентів інформаційної технології тестування WEB застосувань, в результаті чого програмне забезпечення буде містити такі складові як графічний інтерфейс користувача, модуль керування браузерами, головне ядро програми, що міститиме тестові сценарії, модуль перевірки застосувань на валідність, і модуль результатів.

Розроблено загальну структурну схему інформаційної технології тестування WEB застосувань.

Удосконаленно математичну модель оптимізації процесу тестування WEB застосувань, дає можливість оцінити повноту тестування, позбавлена надлишковості, тобто зайвих кроків під час проходження тест-кейсу.

Розроблено комп'ютерне моделювання роботи інформаційної технології тестування WEB застосувань. Розроблено UML діаграму діяльності інформаційної технології , також представлена UML діаграма доступних дій користувача.

Розроблено алгоритм функціонування інформаціної технології тестування WEB застосувань.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ТЕСТУВАННЯ WEB ЗАСТОСУВАНЬ

3.1 Обґрунтування вибору мови та середовища програмування

На сьогоднішній день існує велика кількість різноманітних мов програмування. Вони відрізняються своєю архітектурою та призначенням, надають різний функціонал для роботи.

C# – об'єктно-орієнтована, структурна, імперативна мова програмування зі статично , строгою, керованою та безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft. Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі.Є дуже зрічною у відлагодженні програм.Більшість своїх особливостей успадкувала від C++, Java Delphi [26].

Існує кілька реалізацій C#:

- Компілятор Roslyn з відкритим вихідним кодом
- Реалізація C# у вигляді компілятора csc.exe включена до складу .NET Framework (включаючи .NET Micro Framework , .NET Compact Framework та його реалізації під Silverlight та Windows Phone 7).
- У складі проекту Rotor (Shared Source Common Language Infrastructure) компанії Microsoft .
- Проект Mono включає реалізацію C# з відкритим вихідним кодом .
- Проект DotGNU також включає компілятор C# із відкритим кодом.
- DotNetAnywhere - орієнтована на вбудовані системи реалізація CLR, підтримує практично всю специфікацію C # 2.0.

C++ – мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної . Розроблена Б'ярном Страуструпом у 1979 році та названа «Сі з класами». Страуструп перейменував мову у C++ у 1983 р. Стандартна бібліотека C++ включає стандартну велика частина бібліотеки C++ заснована на Стандартній Бібліотеці Шаблонів (STL). Вона надає такі важливі інструменти, як контейнери (наприклад, вектори і списки) і ітератори (узагальнені вказівники), що надають доступ до цих контейнерів як до масивів. Крім того, STL дозволяє схожим чином працювати і з іншими типами контейнерів, наприклад, асоціативними списками, стеками, чергами, можна писати узагальнені алгоритми, здатні працювати з будь-якими контейнерами або послідовностями, доступ до членів яких забезпечують ітератори [27].

Перевагами C++ є:

- Стандартизація. C++ визначається міжнародним стандартом а отже не контролюється якоюсь одною фірмою чи людиною.
- Швидкодія.
- Ефективність. Рішення розроблені на C++ можуть використовувати мінімальну необхідну кількість ресурсів таких як пам'ять, ЦПУ, енергія та інші.
- Масштабованість. На мові C++ розробляють програми для найрізноманітніших платформ і систем, які варіюються за розміром від кількох до мільйонів рядочків коду.
- Можливість роботи на низькому рівні з пам'яттю, адресами, портами.
- Можливість створення узагальнених алгоритмів для різних типів даних, їхня спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів.
- Підтримуються різні стилі та технології програмування, включаючи традиційне процедурне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси)[ ].

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована[28].

Серед основних її переваг мови Python можна наступні:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);
- відкритий код (можливість редагувати його іншими користувачами)[].

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ[29].

Наведемо порівняльну таблицю 3.1, на якій зможемо побачити плюси та мінуси кожної з обраних нами мов програмування.

Таблиця 3.1 – Порівняння мов програмування

	Python	C++	C#
1	2	3	4
Динамічна типізація	+	+	+
Статична типізація	+	-	+
Об’єктна орієнтованість	+	+	+
Багатомірні масиви	+	+	+-
Перегрузка операторів	+	+	-
Створення об’єктів на стеку	+	+	-
Умовна компіляція	+	+	+-
Ручне управління пам’яттю	+	+	+
Динамічні змінні	+	+	+
Інтерпретатор командного рядка	+	+	+
Зручність відлагодження	+	+-	+-
Підтримка Selenium	+	-	+
Швидкодія	+	+	-

Отже, враховуючі усі переваги, зокрема підтримку Selenium, для розробки інформаційної технології тестування WEB застосувань було обрано мову

програмування Python. Оскільки для розробки інформаційної технології було обрано Python, відповідно середовищем розробки обираємо PyCharm.

PyCharm є IDE для Python. IDE - Integrated development environment - інтегроване середовище розробки, комплекс програмних засобів, які дозволяють вести зручнішу розробку певною мовою програмування. Зазвичай IDE має текстовий редактор, компілятор чи інтерпретатор, налагоджувач та інше програмне забезпечення.

Початкове вікно серидовища розробки PyCharm наведено на рисунку 3.1

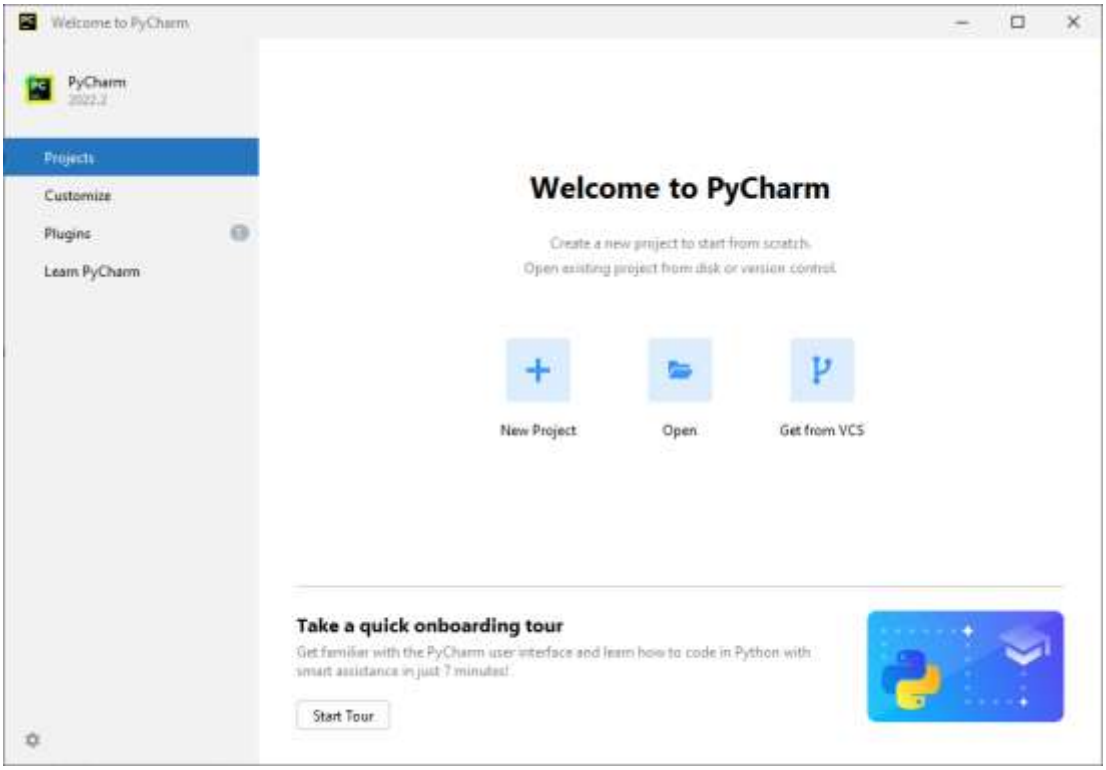


Рисунок 3.1 – Приклади роботи із середовищем розробки PyCharm

PyCharm — інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веброзробку на Django. PyCharm розроблена чеською компанією JetBrains на основі IntelliJ IDEA[30].

PyCharm працює під такими операційними системами як Windows, Mac OS X і Linux.

Створення проекту в серидовищі розробки PyCharm наведено на рисунку 3.2.

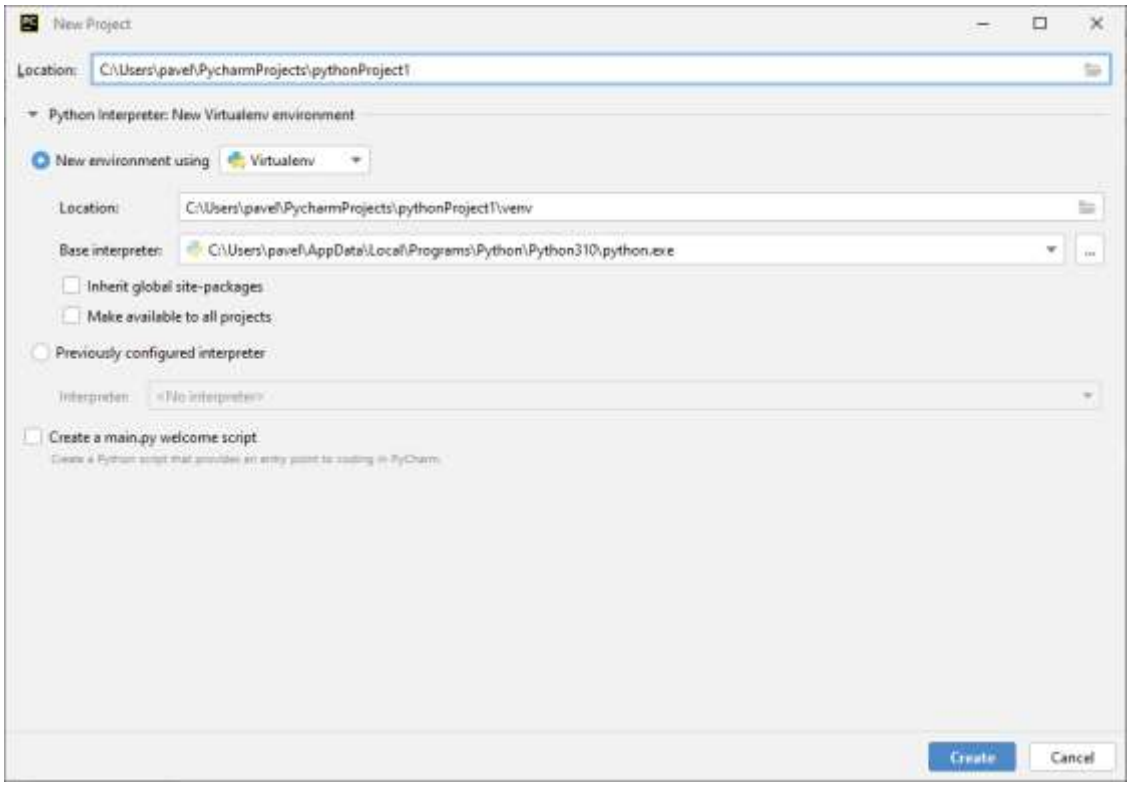


Рисунок 3.2 – Створення проекту в серидовищі розробки PyCharm

PyCharm Professional Edition має кілька варіантів ліцензій, які відрізняються функціональністю, вартістю та умовами використання. PyCharm Professional Edition безкоштовна для освітніх установ і проектів з відкритим кодом.

Існує також вільна версія Community Edition з усіченим набором можливостей, яка поширюється під ліцензією Apache 2.

PyCharm має зручний редактор коду з усіма корисними функціями: підсвічуванням синтаксису, автоматичним форматуванням, доповненням та відступами. PyCharm дозволяє перевіряти версії інтерпретатора мови на сумісність та використовувати шаблони коду. Тим, хто часто використовує документацію, буде зручно дивитися її у вікні редактора (для елементів) або в браузері (для зовнішньої документації).

PyCharm дозволяє швидко проводити рефакторинг коду, а також використовувати зручний графічний відладчик. Утиліта підтримує всі нові версії

Django, а також IronPython, Jython, Cython, PyPy wxPython, PyQt, PyGTK та багато інших інструментів[31].

Головне вікно середовища розробки Python наведено на рисунку 3.3

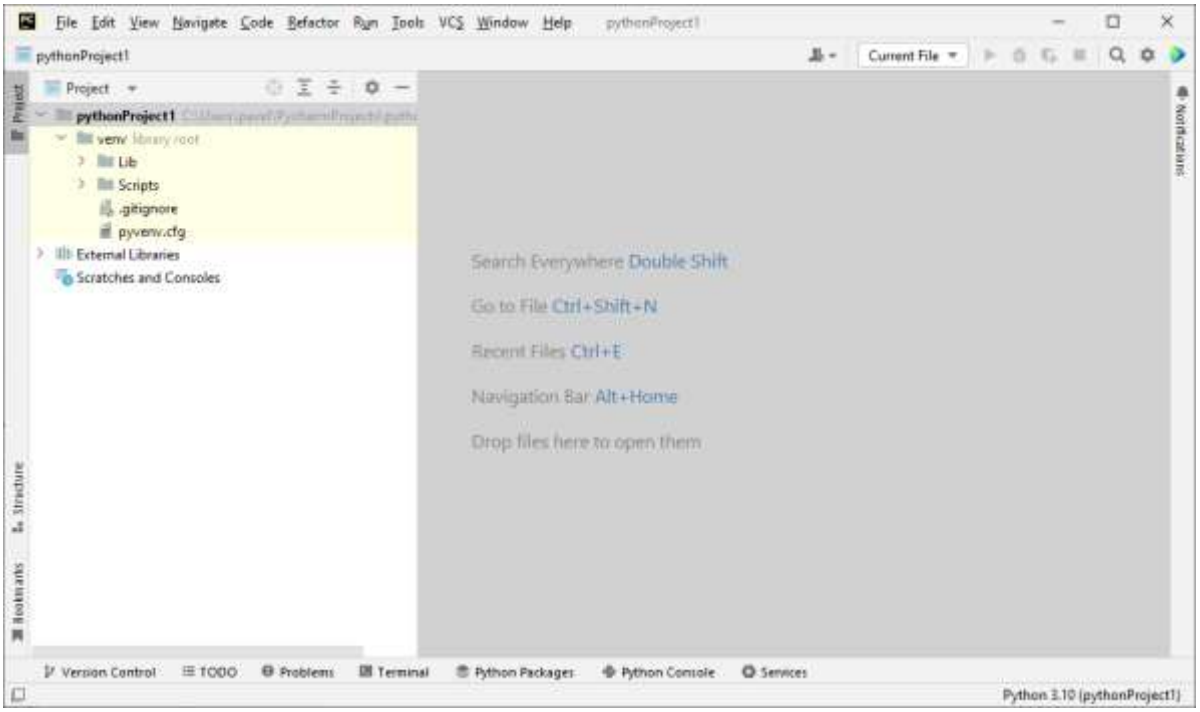


Рисунок 3.3 – Головне вікно середовища розробки Python

У PyCharm можна проводити інтегроване Unit тестування, використовувати інтерактивні консолі для Python, Django, SSH, відладчика та баз даних.

PyCharm має велику колекцію плагінів, і його можна використовувати у зв'язці з різними трекерами на зразок JIRA, Youtrack, Lighthouse, Redmine, Trac і таке інше. PyCharm робить розробку максимально продуктивною завдяки функціям автодоповнення та аналізу коду, миттєвому підсвічуванню помилок та швидким виправленням. Автоматичні рефакторинги допомагають ефективно редагувати код, а зручна навігація дозволяє миттєво переміщатися за проектом.

Підсумовуючи, можна зазначити що дане середовище розробки має наступні можливості:

- Функціональний редактор Python
- Інструмент запуску тестів та графічний відладчик
- Навігація за кодом та рефакторингом

- Інспекції коду
- Підтримка систем контролю версій
- Інструменти для наукових обчислень
- Веб розробка
- Веб-фреймворки Python
- Python-профілювальник
- Можливості віддаленої розробки
- Підтримка баз даних та SQL

Приклад роботи в середовищі розробки PyCharm наведено на рисунку 3.4

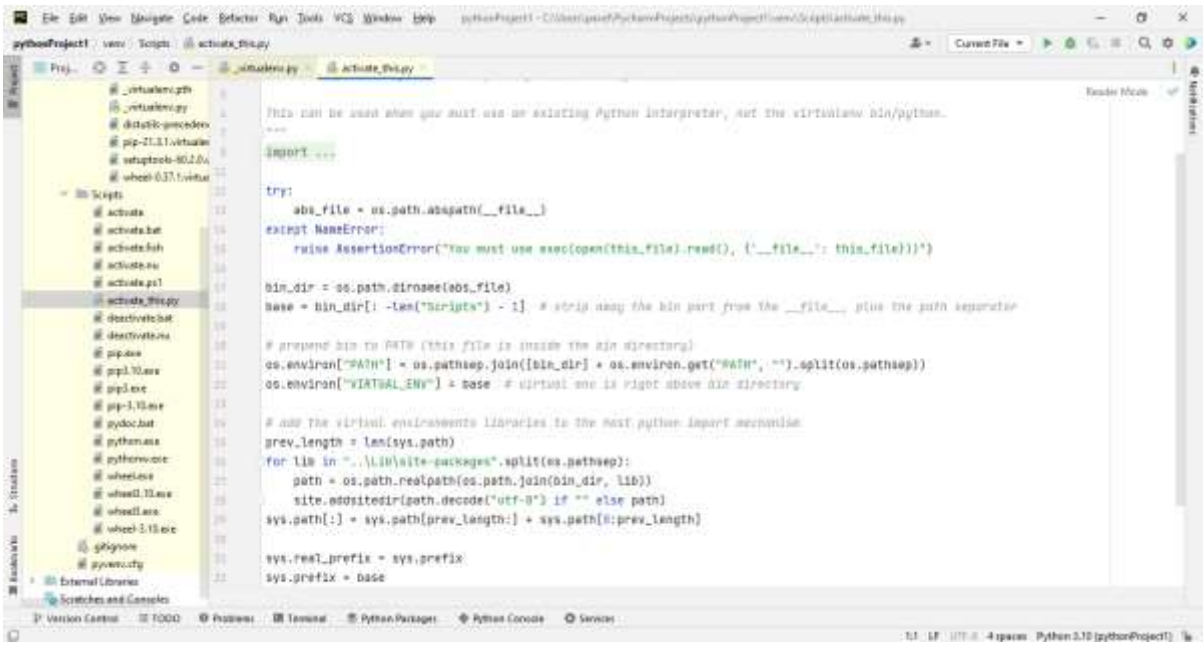


Рисунок 3.4 – Приклад роботи в середовищі розробки PyCharm

3.2 Програмна реалізація інтерфейсу програмного забезпечення тестування WEB застосувань

Інтерфейс користувача — засіб зручної взаємодії користувача з інформаційною системою, сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача[ ]. Якщо мова йде про програмні інтерфейси, то їх можна поділити на графічні та консольні.

Протилежність графічному інтерфейсу - командний рядок, дозволяє керувати програмою за допомогою текстових команд. Такий інтерфейс реалізований у терміналі macOS та командному рядку Windows.

GUI – це графічний інтерфейс користувача, оболонка програми, з якою ми взаємодіємо за допомогою клавіатури та миші. На сучасних операційних системах майже всі програми працюють із графічним інтерфейсом, і ми щодня стикаємося з GUI: читаємо статті у браузері, набираємо текст у редакторі або граємо в ігри.

Для реалізації графічного інтерфейсу мовою програмування Python у середовищі програмування PyCharm можна використовувати декілька підходів.

Для роботи з GUI в Python є чотири бібліотеки:

- Tkinter;
- Kivy;
- Python QT;
- wxPython.

Найбільш відповідним для розробки інформаційної технології тестування WEB застосувань є бібліотека Tkinter, оскільки вона не вимагає додаткової установки і дозволяє швидко створювати програми з простим графічним інтерфейсом.

Спочатку створимо головне вікно програми. Для цього підключимос бібліотеку за допомогою директиви import. Після цього створимо клас Window, у якому міститимуться усі головні та базові атрибути та параметри – висота, ширина, заголовок, і тд.

Створення головного вікна зображено на рисунку 3.5. та результат роботи коду на рисунку 3.6

Рисунок 3.5 - Створення головного вікна програми

Рисунок 3.6 – Продовження

Наступним кроком буде створення інших (дочірніх) вікон, між якими буде відбуватись перехід від головного і назад. Для цього створимо новий клас ChildWindow, у який міститиме базові налаштування програмного вікна. Для

того щоб встановити зв’язок між батьківським та поточним вікном додаємо аргумент та об’єкт parent. Також створюємо метод для нашого дочірнього вікна.

Аналогічним способом створюємо інші дочірні вікна, які міститмуть компоненти програмного забезпечення – вікно тестування, вікно результатів, звіт та інструкцію користувача.

Створення дочірніх вікон зображено на рисунку 3.7, на рисунку 3.8 та на рисунку 3.9.

```
from child_window import ChildWindow


class Window:
    def __init__(self, width, height, title="MyWindow", resizable=(False, False), icon=None):
        self.root = Tk()
        self.root.title(title)
        self.root.geometry(f"{width}x{height}+200+200")
        self.root.resizable(resizable[0], resizable[1])
        if icon:
            self.root.iconbitmap(icon)

    def run(self):
        self.root.mainloop()

    def create_child(self, width, height, title="Child", resizable=(False, False), icon=None):
        ChildWindow(self.root, width, height, title, resizable, icon)


if __name__ == "__main__":
    window = Window(500, 500, "TKINTER")
    window.create_child(200, 100)
    window.run()
```

Рисунок 3.7 – Створення дочірніх вікон

```
    def run(self):
        self.draw_widgets()
        self.root.mainloop()

    def draw_widgets(self):
        top_frame = LabelFrame(self.root, text="Top frame")
        bottom_frame = LabelFrame(self.root, text="Bottom frame")
        top_frame.pack(padx=10, pady=10, expand=1, anchor=NW)
        bottom_frame.pack(ipadx=10, ipady=10, expand=1, fill=BOTH)

        self.label.pack(anchor=NW, padx=100, pady=30)
        Label(top_frame, width=30, height=2, bg='red', text="First").pack(side=LEFT, padx=10)
        Label(top_frame, width=30, height=2, bg='orange', text="Second").pack(side=LEFT)
        Label(bottom_frame, width=30, height=2, bg='yellow', text="Third").pack(side=LEFT)
        Label(bottom_frame, width=30, height=2, bg='green', text="Fourth").pack(side=LEFT)
        Label(bottom_frame, width=30, height=2, bg='cyan', text="Fifth").pack(side=LEFT)

    def create_child(self, width, height, title="Child", resizable=(False, False), icon=None):
        ChildWindow(self.root, width, height, title, resizable, icon)


if __name__ == "__main__":
    window = Window(500, 500, "TKINTER")
    window.create_child(200, 100)
    window.run()
```

Рисунок 3.8 – Продовження 1

```
def run(self):
        self.draw_widgets()
        self.root.mainloop()

    def draw_widgets(self):
        l1 = Label(self.root, width=30, height=2, bg='orange', text="Second")
        l2 = Label(self.root, width=30, height=2, bg='yellow', text="Third")
        l3 = Label(self.root, width=30, height=2, bg='green', text="Fourth")

        l1.grid(row=0, column=0)
        l2.grid(row=0, column=1)
        l3.grid(row=1, column=0, columnspan=2)
        Label(self.root, width=30, height=2, bg='red', text="First").grid(row=2, column=0)

        l1.grid_remove()
        l1.grid_forget()
        l1.grid()

        print(l1.grid_info())
        print(self.root.grid_size())
        print(self.root.grid_location(x=20, y=50))


    def create_child(self, width, height, title="Child", resizable=(False, False), icon=None):
        ChildWindow(self.root, width, height, title, resizable, icon)
```

Рисунок 3.9 – Продовження 2

Далі перейдемо до створення елементів самого програмного забезпечення. За домогою методів entry та insert створюємо поле введення для пошуку веб елементів, кнопки, заповнюємо інформацією вікно інструкції користувача, сворюємо вікно виведення звіту, де інформація буде подаватись у вигляді або інформації що тест завершено успішно, або скріншот та вказане конкретне місце WEB застосунку – веб елемент, над яким буде здійснюватись тестування. Для цього використаємо бібліотеку messagebox, що відповідно дозволить виконуватись даним процесам. Данії дії - наповнення програмного інтерфейсу елементами наведено на рисунках 3.10 та 3.11

```python
from tkinter import messagebox as mb

class Window:
    def __init__(self, width, height, title="MyWindow", resizable=(False, False), icon=r"resources/feather.ico"):
        self.root = Tk()
        self.root.title(title)
        self.root.geometry(f"{width}x{height}+200+200")
        self.root.resizable(resizable[0], resizable[1])
        if icon:
            self.root.iconbitmap(icon)

        self.login_entry = Entry(self.root)
        self.age_entry = Entry(self.root)
        self.password_entry = Entry(self.root, show="*")

        self.entry = Entry(self.root)
        self.entry.insert(0, "Hey...")

    def run(self):
        self.draw_widgets()
        self.root.mainloop()

    def draw_widgets(self):
        Label(self.root, text="Login:", justify=LEFT).grid(row=0, column=0, sticky=W)
```

Рисунок 3.10 – Наповнення програмного інтерфейсу елементами

```python
        self.root.mainloop()

def draw_widgets(self):
    Label(self.root, text="Login:", justify=LEFT).grid(row=0, column=0, sticky=W)
    self.login_entry.grid(row=0, column=1, sticky=W+E, padx=5, pady=5)
    Label(self.root, text="Age:", justify=LEFT).grid(row=1, column=0, sticky=W)
    self.age_entry.grid(row=1, column=1, sticky=W+E, padx=5, pady=5)
    Label(self.root, text="Password:", justify=LEFT).grid(row=2, column=0, sticky=W)
    self.password_entry.grid(row=2, column=1, sticky=W+E, padx=5, pady=5)

    text_var = StringVar(value="Text")
    Entry(self.root, width=30, fg="blue", font=("Verdana", 9), justify=CENTER, relief=SUNKEN, bd=3,
          selectbackground="green", selectforeground="yellow", textvariable=text_var).pack()

    self.entry.pack()
    Button(self.root, text="Show text", width=10, command=self.get_text).pack()
    Button(self.root, text="Repeat", width=10, command=self.repeat_text).pack()
    Button(self.root, text="Insert", width=10, command=self.insert).pack()
    Button(self.root, text="Clear", width=10, command=self.clear).pack()

    Button(self.root, text="Save", width=10, command=self.save_data).grid(row=3, column=0, padx=5, sticky=E)
    Button(self.root, text="Quit", width=10, command=self.exit).grid(row=3, column=1, sticky=E)

def exit(self):
    choice = mb.askyesno("Quit")
```

Рисунок 3.11 – Продовження

І переходимо до заверщального етапу тобто до створення діалогових та контекстних вікон програмного забезпечення за допомогою методу messegebox. Фрагменти коду наведено на рисунках 3.12 та 3.13

```
        Button(self.root, text="Save", width=10, command=self.save_data).grid(row=3, column=0, padx=5, sticky=E)
        Button(self.root, text="Quit", width=10, command=self.exit).grid(row=3, column=1, sticky=E)

    def exit(self):
        choice = mb.askyesno("Quit")
        if choice:
            self.root.destroy()

    def save_data(self):
        login = self.login_entry.get()
        age = int(self.age_entry.get())
        password = self.password_entry.get()

        mb.showinfo("User Data", f"Hello, {login}! You're {age} y.o.")

    def get_text(self):
        text = self.entry.get()
        mb.showinfo("Entry text", text)

    def repeat_text(self):
        text = self.entry.get()
        self.entry.insert(0, text)
        self.entry.insert(END, text)
```

Рисунок 3.12 – Фрагмент коду, що відповідає за створення діалогових вікон

```
    def repeat_text(self):
        text = self.entry.get()
        self.entry.insert(0, text)
        self.entry.insert(END, text)

    def insert(self):
        self.entry.insert(INSERT, "_in_")
        self.entry.insert(ANCHOR, "_sel_")

    def clear(self):
        self.entry.delete(0, END)

    def button_action(self):
        Label(self.root, text="New label").pack()

    def create_child(self, width, height, title="Child", resizable=(False, False), icon=None):
        ChildWindow(self.root, width, height, title, resizable, icon)


if __name__ == "__main__":
    window = Window(500, 500, "TkINTER")
    window.create_child(200, 100)
    window.run()
```

Рисунок 3.13 – Продовження

3.3 Програмна реалізація основних компонентів інформаційної технології тестування WEB застосувань

Згідно з розробленими загальною структурною схемою функціонування, а також UML діаграмами, головним компонентом програмного забезпечення є модуль тестових сценаріїв. Він виконує найбільшу частину роботи, та запускається після того, як була виконана перевірка посилання веб-застосунку на валідність та після запуску модуля об’єктів веб-додатку. Він містить такі основні методи:

- Open() – відкриває сторінку в браузері за певною адресою;
- Click() - робить натискання по елементу сторінки;
- Type() - вводить значення в текстове поле сторінки;
- Select() – вибирає значення зі списку;
- SelectWindow() - переключає фокус на інше вікно;
- GetTitle() - повертає Title для поточної сторінки;
- GetValue() - повертає значення елемента сторінки;
- GoBack() - повертає на попередню сторінку;
- Close() - закриває поточне вікно.

Фрагмент коду, що реалізує даний модуль наведено на рисунку 3.14

Рисунок 3.14 – Фрагмент коду, що реалізує роботу модуля запуску та виконання сценаріїв тестування

Перед модулем запуску та виконання сценаріїв тестування, запускаються модулі перевірки валідності веб-застосування та пошуку елементів веб-застосування. Ці модулі дуже важливі оскільки повинна бути передбачена процедура початкового посилання, оскільки якщо воно не працює, то фактично немає об’єкту тестування, а без наявності веб-елементів тестувати буде просто нічого.

Для перевірки валідності посилання використаємо наступні методи:

- void GoToUrl(string url)- перейти за вказаною адресою;
- Request.GetResponse() – виконує запит на сервер та повертає статус веб-застосунку.

Фрагмент коду, що відповідає за роботу даного модуля наведено на рисунку 3.15.

```
import requests
import urllib.request

def is_valid(url, min_attributes, qualifying_attr, qualifying):
    qualifying = min_attributes
if qualifying_attr is None:
else: qualifying_attr token: object = urllib.parse.urlparse(url)
all([getattr(token, qualifying_attr) for qualifying_attr in qualifying_attr])

def is_valid2(url, qualifying=None):
    url = raw_input(" ")

check = requests.get(url)
if check.status_code == 404:
    print("Посилання не вірне! Перевірте введені дані", url)

    domens = ['.com', '.net', '.org', '.pro', '.ua', ]
def check_site(site_name):
    for domen in domens:
        site = f'https://www.{site_name}{domen}'
        try:
            response = requests.get(site)
            print("Посилання не вірне! Перевірте введені дані")
    check_site()
```

Рисунок 3.15 – Фрагмент коду, що відображає реалізацію модуля перевірки валідності веб-застосування

Модуль, що відповідає за пошук елементів веб-додатку, що тестується містяти у собі методи, які знаходять елементи веб-застосування, тобто різні кнопки, поля, посилання і т.д., за назвами атрибутів працездатність яких вже безпосередньо перевіряється головним модулем програми. Розглянемо основні з них. Пошук може здійснюватись з використанням локаторів. Основними локаторами, які використовуються в даній розробці є:

- find_element_by_id
- find_element_by_name
- find_element_by_xpath
- find_element_by_link_text
- find_element_by_partial_link_text
- find_element_by_tag_name
- find_element_by_class_name
- find_element_by_css_selector

де, id - як локатора використовується атрибут id (унікальний ідентифікатор) елемента сторінки;

name - як локатора використовується атрибут name елемента сторінки;

identifier - використовується атрибут id елемента, якщо по id - у елемент не знайдений, то пошук буде вестися по атрибуту name;

dom - пошуку елемента відбувається по DOM висловом;

xpath - використовується для пошуку елемента по XPath виразу;

link - пошук посилань із зазначеним текстом;

css - даний тип локаторів заснований на описах таблиць стилів.

Відповідно, методи які використовуються для пошуку дані локатори є:

- OpenQA.Selenium.IWebElement FindElement(OpenQA.Selenium.By by) - пошук елемента на сторінці. Параметр By вказує на механізм пошуку елементу, тобто локатори, які описані вище. Повертає знайдений елемент, що задовольняє, умовам пошуку.

- System.Collections.ObjectModel.ReadOnlyCollection FindElements(Open QA.Selenium.By by) – те ж саме, що і FindElement, тільки повертає всі елементи, що задовольняють умові пошуку.

У випадку написання автотестів під кожен окремий веб-застосунок, потрібно прописувати новий сценарій, оскільки різні додатки містять різний функціонал, який у свою чергу містить свій унікальний локатор, по якому здійснюється пошук і саме це потребує написання нового сценарію. Для того, щоб уникнути цього недоліку, використаємо пошук по локаторах . Тобто користувач програмного продукту, завчасно знає всі вимоги до веб-застосування, що тестується, та вводить в програму лише адресу веб-застосунка та локатор, за якими буде здійснюватись пошук веб-елементів і відповідно подальше тестування застосунку.

Фрегменти коду, що відображають реалізацію модуля пошуку елементів веб-додатку зображено на рисунку 3.16 та рисунку 3.17

```
def __init__(self, session, id_) -> None:
    self.session = session
    self._id = id_

def __eq__(self, other_shadowroot) -> bool:
    return self._id == other_shadowroot._id

def __hash__(self) -> int:
    return int(md5_hash(self._id.encode("utf-8")).hexdigest(), 16)

def __repr__(self) -> str:
    return '<{0.__module__}.{0.__name__} (session="{1}", element="{2}")>'.format(
        type(self), self.session.session_id, self._id
    )
def find_element(self, by: str = By.ID, value: str = None):
    if by == By.ID:
        by = By.CSS_SELECTOR
        value = f'[id="{value}"]'
    elif by == By.CLASS_NAME:
        by = By.CSS_SELECTOR
        value = f".{value}"
    elif by == By.NAME:
        by = By.CSS_SELECTOR
        value = f'[name="{value}"]'

    return self._execute(Command.FIND_ELEMENT_FROM_SHADOW_ROOT, {"using": by, "value": value})["value"]
```

Рисунок 3.16 – Фрагмент коду, що відображає реалізацію модуля пошуку елементів веб-додатку

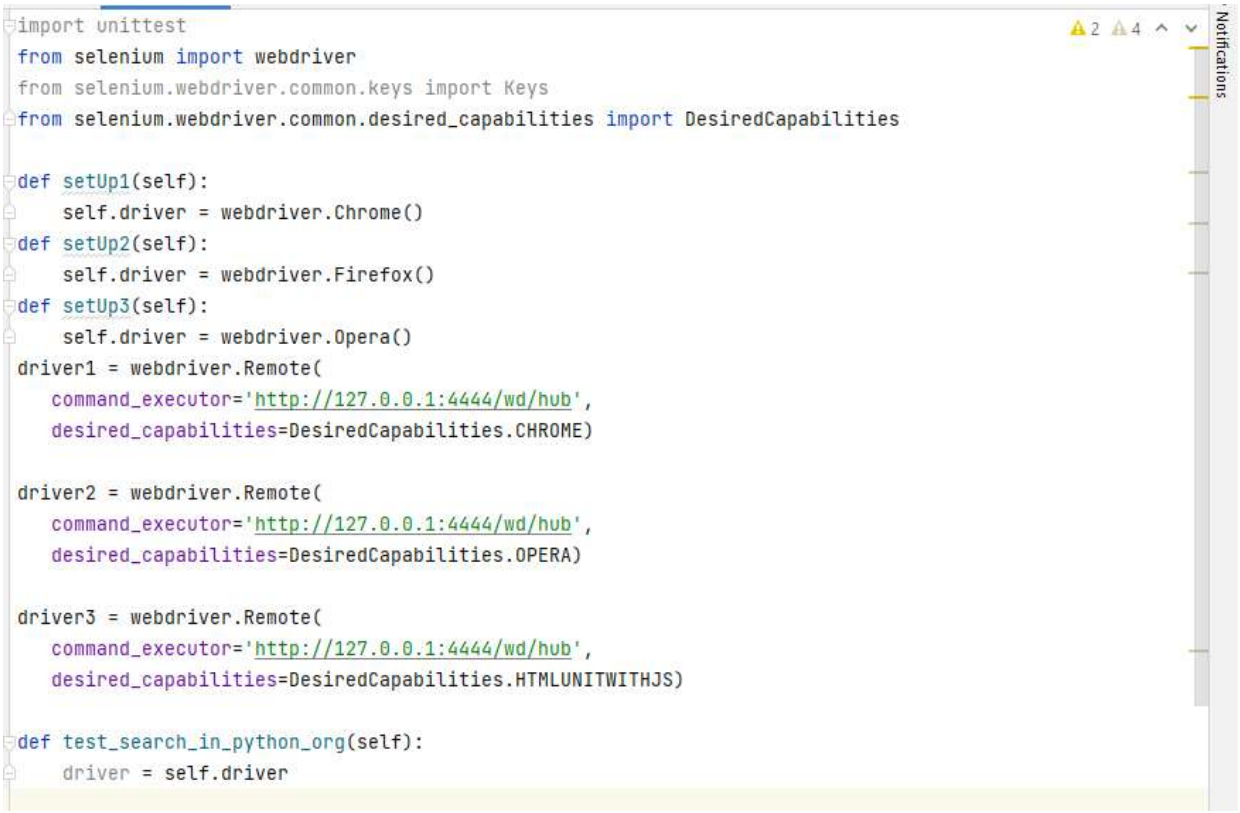
```
    elif by == By.NAME:
        by = By.CSS_SELECTOR
        value = f'[name="{value}"]'

    return self._execute(Command.FIND_ELEMENT_FROM_SHADOW_ROOT, {"using": by, "value": value})["value"]

def find_elements(self, by: str = By.ID, value: str = None):
    if by == By.ID:
        by = By.CSS_SELECTOR
        value = f'[id="{value}"]'
    elif by == By.CLASS_NAME:
        by = By.CSS_SELECTOR
        value = f".{value}"
    elif by == By.NAME:
        by = By.CSS_SELECTOR
        value = f'[name="{value}"]'

    return self._execute(Command.FIND_ELEMENTS_FROM_SHADOW_ROOT, {"using": by, "value": value})["value"]

def _execute(self, command, params=None):

    if not params:
        params = {}
    params["shadowId"] = self._id
    return self.session.execute(command, params)
```

Рисунок 3.17 – Продовження

Для того, аби здійснення сценаріїв тестування було можливим, потрібен доступ до браузера. Саме драйвер браузера Webdriver надає доступ до його керування. Тому, щоб надати доступ для здійснення взаємодії роботи із браузером, потрібно налаштувати роботу програмного модуля. Спочатку підключимо потрібні бібліотеки:

- import unittest
- from selenium import webdriver
- from selenium.webdriver.common.keys import Keys
- from selenium webdriver common desired_capabilities import Desired Capabilities

Вони надають можливість використовувати описані вищі методи, за допомогою яких, можна запускати та здійснювати перевірку працездатності веб-застосування. На рисунку 3.18 наведено фрагмент коду, де використовується підключення до драйверу браузера, в якому здійснюється перевіка роботи веб-додатку.



```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

def setUp1(self):
    self.driver = webdriver.Chrome()
def setUp2(self):
    self.driver = webdriver.Firefox()
def setUp3(self):
    self.driver = webdriver.Opera()
driver1 = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.CHROME)

driver2 = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.OPERA)

driver3 = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.HTMLUNITWITHJS)

def test_search_in_python_org(self):
    driver = self.driver
```

Рисунок 3.18 – Фрагмент коду, що відповідає за підключення до драйверу браузера, для надання доступ до його керування

3.4 Тестування та аналіз результатів роботи програмного забезпечення

Перед тим як проаналізувати ефективність розробленого програмного забезпечення, виконаємо так зване smoke-тестування, тобто виконаємо базову перевірку працездатності програмного забезпечення, виконання покладених на неї задач.

Після запуску відображається початкове вікно програми з привітанням, з можливістю переходу до основного вікна програми. Результати роботи початкового вікна програмного модуля наведено на рисунку 3.19

Рисунок 3.19 – Результати роботи початкового вікна програмного забезпечення

Після того, як запустилось головне вікно програмного модуля, що наведено на рисунку 3.20, можна переглянути інструкцію користувача, натиснувши відповідну кнопку «Інструкція користувача», де міститься покроковий опис роботи із програмним модулем і повернутись назад до головного вікна додатка. Результати роботи вікна «Інструкція користувача» наведено на рисунку 3.21

Рисунок 3.20 – Результати роботи головного вікна програмного модуля

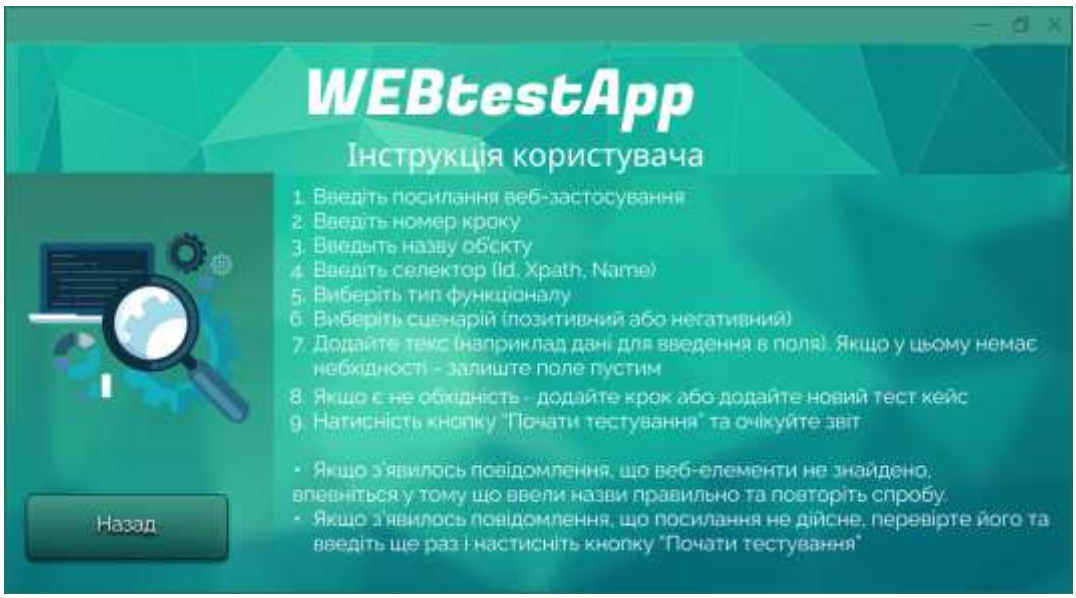


Рисунок 3.21 – Результати роботи вікна «Інструкція користувача»

Також у головному вікні міститься перехід на вікно, де знаходиться інформація про проект. Для того щоб її переглянути, потрібно натиснути на кнопку «Про проект». Ознайомившись з інформацією з цього вікна можна повернутись назад до головного вікна додатка. Результати роботи вікна «Про проект» наведено на рисунку 3.22

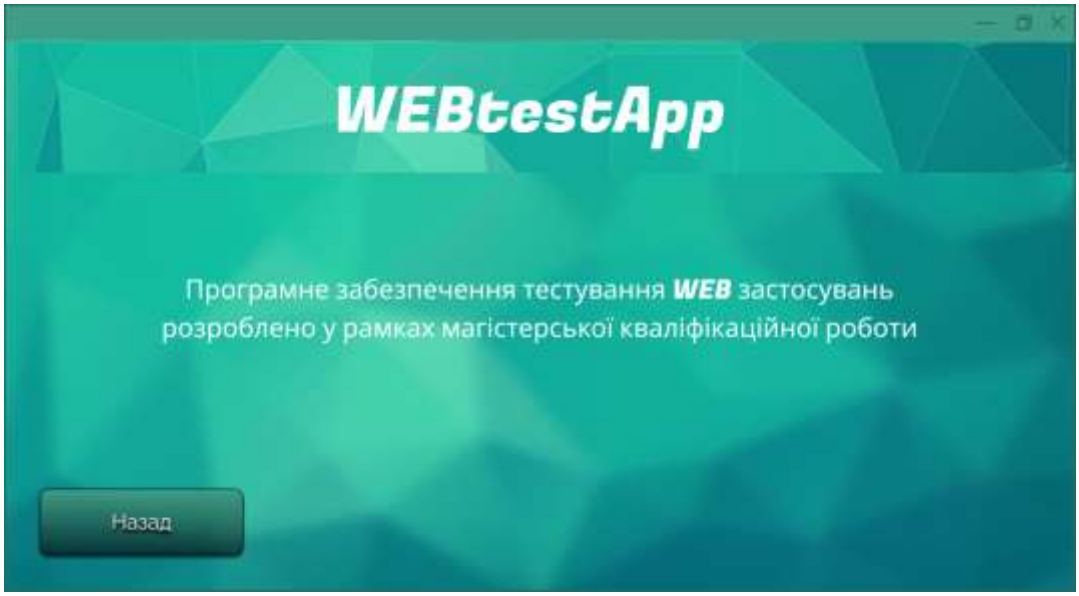


Рисунок 3.22 – Результати роботи вікна «Про проект»

Перейдемо до безпосередньої перевірки програмного модуля. Для того, аби перейти до тестування, натискаємо кнопку «+ Нове тестування». Після натиснення кнопки, відбувається перехід до наступного вікна програми, яке містить два поля для введення та обробки вхідних даних. Спочатку перевіримо чи працює перевірка посилання на валідність та загалом перевірка введених даних. Для цього спочатку введемо в поле будь який текст, що не є посиланням і перевіримо результат роботи програми. Після введення некоректних даних, на екрані з’явилось повідомлення про те, що посилання не вірне.

Отже, можна зробити висновок, що дана функція перевірки посилання на валідність при введені некоректних даних працює. Результати перевірки посилання на валідність наведені на рисунку 3.23

Рисунок 3.23 – Результати перевірки посилання на валідність

Необхідно перевірити пошук елементів веб-застосування. Для цього спочатку вводимо заздалегідь обране посилання у відповідне поле. Далі, у поле, де потрібно ввести назви атрибутів, за якими буде здійснюватися пошук безпосередніх об’єктів, працездатність яких буде перевірятись, введемо такі дані, тобто назви атрибутів, яких не містить код сторінки веб-застосування. Натискаємо кнопку «Старт» і бачимо повідомлення, що вказані елементи не

знайдені. Отже дана функція пошуку об’єктів тестування при введені недійсних даних працює. Результати перевірки пошуку об’єктів тестування за назвами атрибутів наведено на рисунку 3.24.

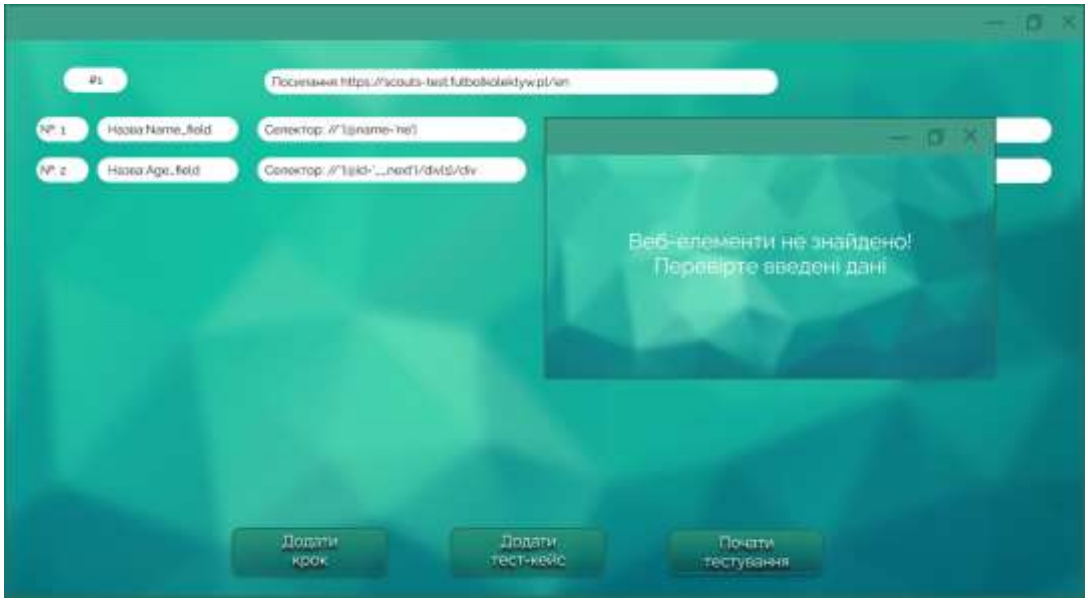


Рисунок 3.24 – Результати перевірки об’єктів тестування за локаторами

Перейдемо до безпосередньої перевірки виконання тестування. Для цього сформуємо певний набір даних – посилання на веб-застосунок, перевірка якого буде здійснюватись, назви об’єктів тестування, локатори веб-елементів, що містяться в коді цієї сторінки, працездатність яких буде перевірятись, а також для поля введення тестові дані. Також обираємо браузер, тип функціоналу та сценарій. Дана дія наведена на рисунку 3.25

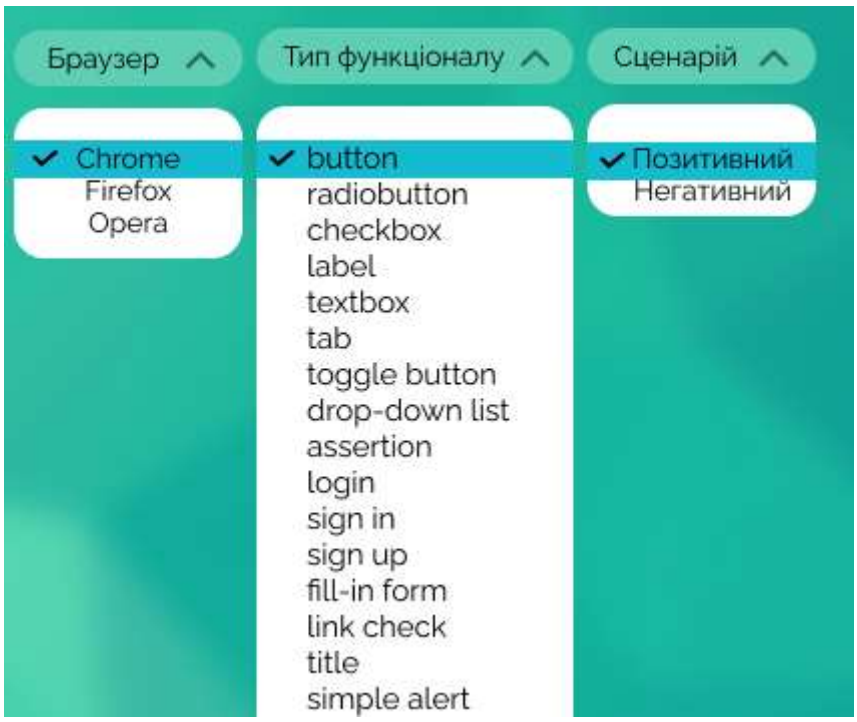


Рисунок 3.25 – Вибір браузера, функціоналу та сценарію

Вводимо такий набір даних:

1. Назви:

- Register
- Language
- Login
- Password
- Create

2. Локатори:

- //*[@id='__next']/form/div
- //*[@id='__next']/form/div/div[1]/h5
- //*[@id='login']
- //*[@id='password']
- //*[@id='__next']/div[1]/main/div[2]

3. Браузер - Chrome
4. Тип функціоналу – button та input field
5. Сценарій – для button позитивний а для input field - негативний

6. Для об’єктів Login та Password вводимо невалідні дані «#$%^&*», оскільки сценарій тестування обрано негативний.

Після того як ввели всі дані у відповідні поля, після чого натискаємо кнопку «Почати тестування». Дана дія наведена на рисунку 3.26

Рисунок 3.26 – Результати роботи програмного модуля

Після того, як було введено коректні дані, здійснюється перехід на наступне вікно, де відбувається очікування результату, про що інформує відповідне повідомлення у даному вікні. Результат роботи вікна очікування наведено на рисунку 3.27

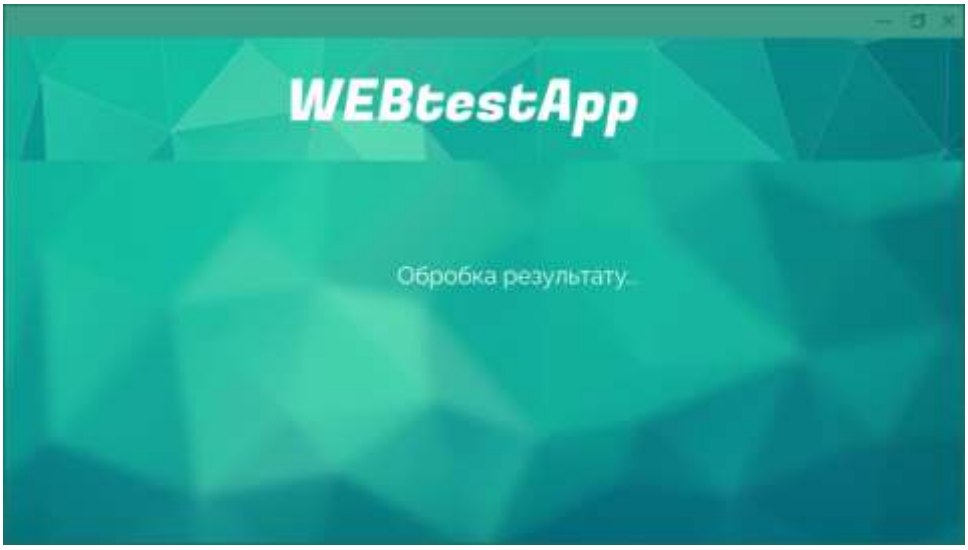


Рисунок 3.27 – Результат роботи вікна очікування

Після того, як виконання сценарію тестування було завершено відбувається перехід до вікна з повідомленням про результати, в якому вони відображаються у вигляді пунктів для кожного кроку тест кейса. Про результат повідомляють passed та failed позначені відповідно зеленим і червоним кольором. Також доступна опція завантажити скрін із результатом тестування для кожного пункту. В даному випадку було здійснено перевірку п’яти елементів веб-застосування і відображено результат «pass:3, fail:2», а оскільки для поля введення логіну та пароля було обрано саме негативний сценаріть, тобто перевірка роботи при невалідних даних, то це свідчить про те, що всі п’ять об’єктів перевірки функціонують відповідно до вимог і некоректної роботи веб-застосування не було виявлено.

Результат роботи програмного модуля наведено на рисунку 3.28.

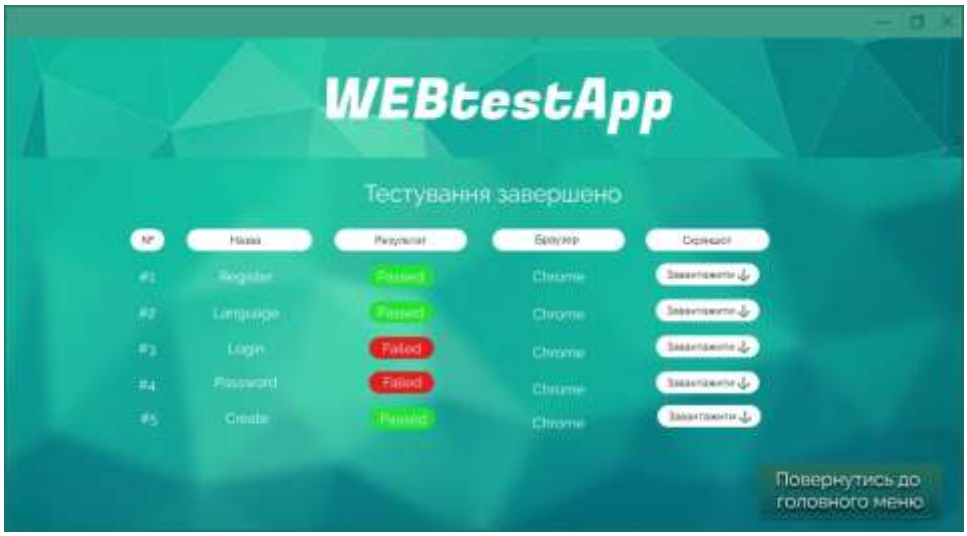


Рисунок 3.28 – Продовження 2

Загалом було проведено більше 300 запусків програми, із різними тестовими наборами даних, для кожного типу функціоналу та сценарію передбаченого програмним забезпеченням під час яких некоректної роботи програмного забезпечення виявлено не було.

Автоматизація тестування сильно підвищує проектні ризики, а тому існують спеціальні підходи щодо оцінки застосовності та ефективності

автоматизованого тестування. Якщо висловити всю їхню суть дуже коротко, то в першу чергу слід врахувати:

• Витрати часу на ручне виконання тест-кейсів та виконання цих же тест-кейсів, але вже автоматизованих. Чим відчутніша різниця, тим вигіднішою є автоматизація.

• Кількість повторень виконання тих самих тест-кейсів. Чим вона більша, тим більше часу ми зможемо заощадити за рахунок автоматизації.

• Витрати часу на налагодження, оновлення та підтримку автоматизованих тест-кейсів. Цей параметр найскладніше оцінити, і саме він становить найбільшу загрозу успіху автоматизації, тому тут для проведення оцінки слід залучати найдосвідченіших фахівців.

• Наявність у команді відповідних фахівців та їхнє робоче завантаження.

Ефективність автоматизованого тестування можна розрахувати за формулою 3.1[32]:

$$A_{effect} = \frac{N \cdot T^{overall}}{N \cdot T^{development}_{automated} + T^{support}_{automated}} \cdot 100\% \qquad (3.1)$$

Де, A_{effect} - коефіцієнт вигоди від використання автоматизації,

N - Запланована кількість білдів програми;

$T^{overall}$- розрахунковий час розробки підготовки, виконання та аналізу тестових даних, розробки тест-кейсів;

$T^{development}_{automated}$ — розрахунковий час виконання та аналізу результатів автоматизованого тестування;

$T^{support}_{automated}$ – розрахунковий часу виділеного на проєкт (загальний аналіз результатів)

Обрахуємо ефективність для створеної інформаційної технології. Для цього візьмемо дані про стандартний веб-застосунок . Приймемо що N = 1,

розрахунковий час розробки, виконання та аналізу результатів ручного тестування становитиме 30 годин на кожен білд, розрахунковий час виконання та аналізу результатів автоматизованого тестування становитиме 5 години, а час на увесь проект відведено 150 годин

$$A_{effect} = \frac{30 \text{ год}}{2 \text{ год} + 150 \text{ год}} \cdot 100\%$$

$$A_{effect} = 19\%$$

Порівняємо ефективність інформаційної технології, з програмами-аналогами, що містять подібні рішення та наведемо порівняльну таблицю 3.2. Оцінку проведемо так само для стандартного веб-застосунку, а тому деякі параметри будуть незмінні для різних програмних засобів, оскільки об’єкт тестування не змінюється.

Таблиця 3.2 – Порівняння ефективності

Програмний засіб	N	$T^{overall}$	$T^{development}_{automated}$	$T^{support}_{automated}$	A_{effect}
Розроблене програмне забезпечення	1	25	2	130	19%
Ranorex	1	35	6	200	16%
Katalon Studio	1	40	5	250	15%

З порівняльної таблиці видно, що для розробленого програмного забезпечення ефективність вища в середньому на 3,5% у порівнянні з найкращим

показником програми аналога. Отже, ефективність тестування підвищено, а мету магістерської кваліфікаційної роботи досягнуто

3.5 Висновок до розділу 3

У третьому розділі розглянуто розглянуто переваги та недоліки трьох об’єктно-орієнтованих мов програмування та обгрунтовано вибір мови Python для розробки інформаційної технології, а також середовища програмування PyCharm. Розроблено інтерфейс користувача та програмне забезпечення з використанням фреймворку Selenium та технології WebDriver. Проведено тестування розробленого програмного забезпечення, що здійснювалось на основі smoke-тестування. Усього здійснювалось 300 запусків програми, із різними тестовими наборами даних, для кожного створеного сценарію. Проведене тестування підтверджує правильність роботи усіх покладених на неї задач, і здійснено оцінку ефективності тестування, за відповідною методикою обчислення, яка показує, що у порівнянні із двома подібними програмними рішеннями (Ranorex та Katalon Studio, що мають відповідну ефективність 16 та 15 %) ефективність розробленого програмного забезпечення становить 19% і в середньому підвищилась на 3,5%.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту розробки інформаційної технології тестуваня WEB застосувань, як результату науково-технічної діяльності. Дана розробка розширює функціональні можливості програмного забезпечення тестуваня WEB застосувань за рахунок використанням інформаційної технології.

Аналогами можуть бути програмне забезпечення Ranorex, приблизна ціна 145168 грн. або Katalon studio, приблизна ціна 74973 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів[33]. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за п’ятибальною шкалою)					
	0	1	2	3	4
1	*2*	*3*	*4*	*5*	*6*
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах

Таблиця 4.1 – Продовження

1	2	3	4	5	6
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам анало-гів	Ціна продукту дещо нижча за ціни аналогів	Ціна продукту значно нижча за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Таблиця 4.1 – Продовження

1	2	3	4	5	6
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5ти років. Термін окупності інвестицій більший 5ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій менший 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що потребує значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту потребує незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 4.2

Середньоарифметичну суму балів $СБ_c$ розраховують за формулою:

$$\text{СБ}_\text{c} = \frac{\sum_{i=1}^{3} \text{СБ}_i}{3} \qquad (4.1)$$

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	4	4
Цінова політика	3	4	4
Технічні та споживчі властивості виробу	3	3	3
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	4
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	3	4	3
Матеріально-технічна база	3	3	4
Термін реалізації ідеї	3	3	3
Супровідна документація	2	3	4
Сума	37	42	44
Середньоарифметична сума балів $СБ_с$	$СБ_с = \frac{37+42+44}{3} = 41$		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високою, що досягається за рахунок того, що, дана розробка розширює функціональні можливості програмного забезпечення для тестування WEB застосувань за рахунок використанням інформаційної технології.

Оцінимо рівень конкурентоспроможності розробки.

Процедура визначення якісних одиничних параметричних індексів за технічними показниками (показниками якості) здійснюється за відповідними формулами.

Якщо збільшення величини параметра свідчить про підвищення якості нової розробки , то одиничний параметричний індекс розраховується за формулою:

$$q_i = \frac{P_i}{P_{\text{баз } i}} \tag{4.2}$$

Якщо зменшення величини параметра свідчить про підвищення якості нової розробки , то одиничний параметричний індекс розраховується за формулою:

$$q_i = \frac{P_{\text{баз } i}}{P_i} \tag{4.3}$$

де q_i – одиничний параметричний індекс, розрахований за і-м параметром;

P_i – значення і-го параметра розробки;

$P_{баз\ i}$ – аналогічний параметр базової розробки-аналога, з якою проводиться порівняння.

Технічні, економічні а також «м’які» параметри аналога та нової науково-технічної розробки подано в таблиці 4.4

Таблиця 4.4 - Технічні та економічні параметри аналога та нової науково-технічної розробки

Параметр	Одиниця виміру	Аналог	Нова розробка	Індекс зміни значення параметра	Коефіцієнт вагомості
Технічні					
Кількість тестових методів	шт.	80	100	1,25	0,5
Ефективність покриття коду тест кейсами	%	70	80	1,14	0,3
Швидкодія	-	0,06	0,1	1,6	0,2
Економічні					
Вартість	грн.	74973	15000	0,2	0,6
Витрати на купівлю та переклад національною мовою технічної інформації та інструкцій	грн.	2000	800	0,4	0,3
Витрати на навчання персоналу	грн.	20000	5000	0,25	0,1

Груповий показник конкурентоспроможності за нормативними параметрами розраховується як добуток частинних показників за кожним параметром за формулою:

$$I_{\text{нп}} = \prod_{i=1}^{n} q_i \qquad (4.4)$$

де $I_{нп}$ – загальний показник конкурентоспроможності за нормативними параметрами;

q_i – одиничний (частинний) показник за і-м нормативним параметром;

n – кількість нормативних параметрів, які підлягають оцінюванню.

$$I_{\text{нп}} = 1$$

Значення групового параметричного індексу за технічними параметрами визначається з урахуванням вагомості (частки) кожного параметра:

$$I_{\text{тп}} = \sum_{i=1}^{n} q_i \cdot \alpha_i \qquad (4.5)$$

де I_{mn} – груповий параметричний індекс за технічними показниками (порівняно з аналогом);

q_i – одиничний параметричний показник і-го параметра;

α_i – вагомість і-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$

n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

$$I_{\text{тп}} = 1{,}25 \cdot 0{,}5 + 1{,}14 \cdot 0{,}3 + 1{,}6 \cdot 0{,}2 = 1{,}287$$

Груповий параметричний індекс за економічними параметрами (за ціною споживання) розраховується за формулою:

$$I_{\text{еп}} = \sum_{i=1}^{m} q_i \cdot \beta_i \qquad (4.6)$$

де I_{en} – груповий параметричний індекс за економічними показниками;

q_i – економічний параметр і-го виду;

β_i – частка і-го економічного параметра, $\sum_{i=1}^{n} \beta_i = 1$

m – кількість економічних параметрів, за якими здійснюється оцінювання.

$$I_{\text{еп}} = 0{,}85$$

Бажане значення $I_{en} \leq 1$, оскільки чим нижча ціна споживання, тим вищий рівень конкурентоспроможності розробки.

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розраховують інтегральний показник конкурентоспроможності за формулою:

$$К_{ІНТ} = I_{нп} \cdot \frac{I_{тп}}{I_{еп}} \quad (4.7)$$

$$К_{ІНТ} = 1 \cdot \frac{1,287}{0,85} = 1,5$$

Оскільки $К_{інт} > 1$, це означає, що наково-технічна розробка перевищує зразок і буде конкурентноспроможною на ринку, а також враховуючи сучасний розвиток інформаційних технологій відбувається дуже стрімко, що у свою чергу сприяє великому зросту кількості веб розробок, то створене програмне забезпечення для тестування WEB застосувань буде мати попит при виведенні на ринок.

4.2 Прогнозування витрат на виконання науково-технічної інформаційної технології тестування WEB застосувань

Витрати, пов’язані з проведенням науково-технічної, дослідноконструкторської, конструкторсько-технологічної роботи, створенням дослідного зразка і здійсненням виробничих випробувань, під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуються за такими статтями:

- витрати на оплату праці;
- відрахування на соціальні заходи;
- матеріали;

- паливо та енергія для науково-виробничих цілей;
- витрати на службові відрядження;
- спецустаткування для наукових (експериментальних) робіт;
- програмне забезпечення для наукових (експериментальних) робіт;
- витрати на роботи, які виконують сторонні підприємства, установи і організації;
- інші витрати;
- накладні (загальновиробничі) витрати.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_i \frac{M_{ni} \cdot t_i}{T_p} \qquad (4.8)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

Мпі – місячний посадовий оклад конкретного дослідника, грн;

ti – кількість днів роботи конкретного дослідника, дн.;

Тр – середня кількість робочих днів в місяці, Тр=21…23 дні. Проведені розрахунки наведено в таблиці 4.5.

Таблиця 4.5 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	25000	1136,6	35	39772,72
Інженер	20000	909,09	35	31818,18
Всього				71590,9

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плата розробників, які приймали участь в розробці інформаційної технології тестування WEB застосувань

Додаткова заробітна плата прийнято розраховувати як 11 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot \frac{11\%}{100\%} \quad (4.9)$$

$$З_д = 71590{,}9 \cdot \frac{11\%}{100\%} = 7874{,}9 \text{ (грн.)}$$

4.2.2 Відрахування на соціальні заходи

До статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов'язкове державне соціальне страхування та для здійснення заходів щодо соціального захисту населення (*ЄСВ – єдиний соціальний внесок*).

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н=\left(З_о+З_р+З_{дод}\right)\cdot\frac{Н_{зп}}{100\%} \quad (4.10)$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

$$З_н=(71590,9+7874,9)\cdot\frac{22\ \%}{100\%}=17482,47\ (грн.)$$

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації.

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$M=\sum_{j=1}^{n}Н_j\cdot Ц_j\cdot К_j-\sum_{j=1}^{n}В_j\cdot Ц_{вj} \quad (4.11)$$

де Hj – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Цj$ – вартість матеріалу j-го найменування, грн/кг;

Кj – коефіцієнт транспортних витрат, (*Кj* = 1,1 … 1,15);

Bj – маса відходів j-го найменування, кг;

Цвj – вартість відходів j-го найменування, грн/кг.

Приймемо, що коефіцієнт транспортних витрат *Кj* = 1,1. Проведені розрахунки наведено у таблиці 4.6.

Таблиця 4.6 - Витрати за сировину та матеріали

Найменування матеріалу, марка, тип, сорт	**Ціна, грн за 1 кг, грн**	**Норма витрат, кг**	**Величина відходів, кг**	**Ціна відходів, грн/кг**	**Вартість витраченого матеріалу, грн**
Офісний папір А4	240	1	0,2	2,5	263,5
Тонер для принтера	328	1	0,1	2,5	360,55
Всього:					624,05

4.2.4 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування, верстатів, пристроїв, інструментів, приладів, стендів, апаратів, механізмів, іншого спецобладнання, необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Вартість спецустаткування визначається за прейскурантом гуртових цін або за даними базових підприємств за відпускними і договірними цінами. До балансової вартості устаткування окрім прейскурантної вартості входять витрати на його транспортування і монтаж, тому ці витрати беруться додатково в розмірі 10…12% від вартості устаткування. Балансову вартість спецустаткування розраховують за формулою:

$$В_{спец} = \sum_{j=1}^{n} Ц_і \cdot С_{пр.і} \cdot К_і \quad (4.12)$$

де *Ці* – ціна придбання одиниці спецустаткування даного виду, марки, грн;

Спр.і – кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

Кі – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (*Кі* = 1,10…1,12);

k – кількість найменувань устаткування.

Приймемо, що *Кі* = 1,10. Проведені розрахунки наведено у таблиці 4.7.

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

Найменування комплектуючих	**Кількість, шт**	**Ціна за одиницю, грн**	**Вартість, грн**
Проектор	1	7148	7862,8
Екран для проектора	1	539	592,9
Стіл	1	1765	1941,5
Крісло	1	2280	2508
Всього:			12905,2

4.2.5 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення. До балансової вартості програмного забезпечення входять

витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{прг.і} \cdot К_{і} \quad (4.13)$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{прг.і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($Кі$ = 1,10…1,12);

k – кількість найменувань програмних засобів

Проведені розрахунки наведено у таблиці 4.8.

Таблиця 4.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	**Кількість, шт**	**Ціна за одиницю, грн**	**Вартість, грн**
PyCharm	1	-	9491,15
Miccrosoft Office	1	-	7873,52
Всього:			19101,137

4.2.6 Амортизація обладнання, програмних засобів та приміщень

До статті «Амортизація обладнання, програмних засобів та приміщень» відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпечення

для проведення науково-дослідної роботи, *за його наявності* в дослідній організації або на підприємстві.

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$\text{А} = \frac{\text{Ц}}{\text{Т}_\text{в}} \cdot \frac{t_{\text{вик}}}{12} \qquad (4.14)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 15500 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2 міс.

$$A_{обл} = \frac{15500}{5} \cdot \frac{2}{12} = 516{,}6 \text{ грн.}$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.9. Для розрахунку амортизації нематеріальних ресурсів використовується формула:

$$\text{А}_\text{н.р.} = \text{Ц}_\text{н.р.} \cdot \text{Н}_\text{а} \cdot \frac{t_{\text{вик}}}{12} \qquad (4.15)$$

Норму амортизації $\text{Н}_\text{а}$ приймемо за 11 %.

Таблиця 4.9 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Ноутбук (DELL Vostro 15 3000)	15500	5	2	516,66
Приміщення	1100000	20	2	9166.66
Ліцензійна ОС, та спеціалізовані ліцензійні нематеріальні ресурси	7899	_	2	144,815
Всього				9828,135

4.2.7 Паливо та енергія для науково-виробничих цілей

До статті «Паливо та енергія для науково-виробничих цілей» належать витрати на придбання у сторонніх підприємств, установ і організацій будь-якого палива, що витрачається з технологічною метою на проведення досліджень. Стаття формується у разі виконання енергоємних наукових досліджень за методом прямого внесення витрат і досягає значної питомої ваги у собівартості досліджень.

$$В_е = \sum_{i=1}^{k} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (4.16)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії);

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для η_i приймемо значення 0,99, а $K_{впі}$ - 0,9.

$$В_е = \frac{0{,}045 \cdot 250 \cdot 6{,}02 \cdot 0{,}9}{0{,}99} + \frac{0{,}015 \cdot 250 \cdot 6{,}02 \cdot 0{,}9}{0{,}99}$$

$$В_е = 61{,}56 + 20{,}52 = 82{,}08$$

Проведені розрахунки наведено у таблиці 4.10.

$$Ц_е = (Ц_{опт} + Ц_{розп} + Ц_{пост}) \cdot \left(1 + \frac{ПДВ}{100\%}\right) \quad (4.17)$$

де $Ц_{опт}$ - середня оптова ціна електроенергії, яка визначається оператором ринку (без ПДВ), грн за 1 кВт·год;

$Ц_{розп}$ - вартість розподілу електроенергії окремою енергорозподільною компанією (без ПДВ), грн за 1 кВт·год;

$Ц_{пост}$ - вартість постачання електроенергії від енергорозподільної компанії до конкретного споживача (без ПДВ), грн за 1 кВт·год.

ПДВ - величина податку на додану вартість, %, у 2022 році ПДВ=20%.

$$Ц_е = (3{,}411 + 1{,}257 + 0{,}346) \cdot \left(1 + \frac{20\%}{100\%}\right) = 6{,}02 \text{ грн.}$$

Таблиця 4.10 – Витрати на електроенергію

Найменування обладнання	**Встановлена потужність, кВт**	**Тривалість роботи, год**	**Сума, грн**
Ноутбук (DELL Vostro 15 3000)	0,045	250	61,56
Wi-fi роутер	0,015	250	20,52
Всього			82,08

4.2.8 Службові відрядження

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%} \quad (4.18)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження».

$$В_{св} = (71590{,}9\ ) \cdot \frac{20\%}{100\%} = 14318{,}18$$

4.2.9 Витрати на роботи, які виконують сторонні підприємства, установи і організації.

«Витрати на роботи, які виконують сторонні підприємства, установи і організації» належать витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100\%} \quad (4.19)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = (71590{,}9\ ) \cdot \frac{40\%}{100\%} = 28636{,}36$$

4.2.10 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100\%} \quad (4.20)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = (71590{,}9\ ) \cdot \frac{50\%}{100\%} = 35795{,}45$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%} \quad (4.21)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = (71590{,}9\ ) \cdot \frac{120\%}{100\%} = 85909{,}08$$

4.2.11 Витрати на проведення науково-дослідної інформаційної технології тестування WEB застосувань.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.22)$$

$В_{заг}$=71590,9+7874,9+17482,47+624,05+19101,137+12905,2+9828,135+82,08 +14318,8+28636,36+35795,45+85909,08 = 304148,562 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науковотехнічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \quad (4.23)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи. Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η =0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9.

$$ЗВ = \frac{304148{,}562}{0{,}5} = 608297{,}124 \text{ грн.}$$

4.4 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

Припустимо, що результати науково-технічної розробки можуть бути впроваджені у 2023, а протягом 3-х років після її впровадження очікуються основні позитивні результати для потенційного інвестора. Як уже було зазначено вище, аналогами можуть бути програмне забезпечення Ranorex, приблизна ціна 145168 грн. або Katalon studio, приблизна ціна 74973 грн. Попит даної розробки спямований на ІТ-компанії, розробників програмного забезпечення та тестувальників які працюють в ІТ-компаніях.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

У даному випадку науково-технічною розробкою є інформаційна технологія тестування веб застосувань, що являє собою програмне забезпечення. Тому основу майбутнього економічного ефекту буде формувати:

ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу;

N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки;

Цб – вартість послуги у році до впровадження інформаційної системи;

±ΔЦо – зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розрахуємо за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right) \qquad (4.24)$$

де $\pm\Delta Ц_o$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки);

$\pm\Delta Ц_o$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість становить 20%, а коефіцієнт λ=0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ=0,2...0,5;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році ϑ=18%.

Припустимо, що при прогнозованій ціні 15000 грн. Як уже зазначалося вище, за одиницю виробу термін збільшення прибутку складе 3 роки. Після завершення її розробки і вдосколналення, можна буде підняти її ціну на 1000 грн. Кількість одиниць реалізованої продукції також можк збільшитись. Протягом першого року 700 шт., протягом другого – на 950 шт., і протягом третього на 1000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було, тому $\pm\Delta Ц_о = 0$.

$$\Delta П_1 = (0 \cdot 800 + 16000 \cdot 700) \cdot 0{,}8333 \cdot 0{,}4 \cdot \left(1 - \frac{18}{100}\right) = 3061210{,}88$$

$$\Delta П_2 = (0 \cdot 800 + 16000 \cdot 1650) \cdot 0{,}8333 \cdot 0{,}4 \cdot \left(1 - \frac{18}{100}\right) = 7215711{,}36$$

$$\Delta П_3 = (0 \cdot 800 + 16000 \cdot 2650) \cdot 0{,}8333 \cdot 0{,}4 \cdot \left(1 - \frac{18}{100}\right) = 11588869{,}76$$

Отже очікуваний комерційний ефект за три роки складе 21865792грн.

Розрахуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$\text{ПП} = \sum_{i=1}^{T} \frac{\Delta \text{П}_i}{(1+\tau)^t} \qquad (4.25)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$\text{ПП} = \frac{3061210{,}88}{(1+0{,}1)^1} + \frac{7215711{,}36}{(1+0{,}1)^2} + \frac{11588869{,}76}{(1+0{,}1)^3} = 17453206{,}1764$$

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науковотехнічної розробки. Для цього можна використати формулу:

$$\text{PV} = \text{k}_{\text{інв}} \cdot \text{ЗВ} \qquad (4.26)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на

підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$ =2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$\text{PV} = 2 \cdot 608297{,}124 = 1216594{,}248 \text{ грн.}$$

Тоді абсолютний економічний ефект Еабс або чистий приведений дохід (*NPV, Net Present Value*) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \qquad (4.27)$$

де *ПП* – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн; *PV* – теперішня вартість початкових інвестицій, грн.

$$E_{абс} = 17453206{,}1764 - 1216594{,}248 = 16236611{,}9284$$

Оскільки величина $E_{абс}$ має велике додатне значення, то це може свідчити про потенційну зацікавленість інвесторів у впровадженні та комерціалізації цієї науково-технічної розробки. Але для остаточного прийняття рішення про впровадження науково-технічної розробки та виведення її на ринок (тобто її комерціалізації) цього недостатньо.

Для остаточного прийняття рішення з цього питання розрахуємо внутрішню економічну дохідність $E_в$ або показник внутрішньої норми дохідності (*IRR, Internal Rate of Return*) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науковотехнічної розробки, розраховується за формулою:

$$E_{\text{в}} = \sqrt[T_{\text{ж}}]{1 + \frac{\text{E}_{\text{абс}}}{PV}} - 1 \qquad (4.28)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_{\text{в}} = \sqrt[3]{1 + \frac{16236611{,}9284}{1216594{,}248}} - 1 = 1{,}42$$

Далі визначимо бар'єрну ставку дисконтування $\tau_{мін}$, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть, за формулою:

$$\tau_{\text{мін}} = d + f \qquad (4.29)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,9...0,12;

f – показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина f =0,05...0,5, але може бути і значно вищою.

$$\tau_{\text{мін}} = 0{,}12 + 0{,}05 = 0{,}17$$

Так як величина $E_в > \tau_{мін}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки та виведенні її на ринок, тобто в її комерціалізації.

Далі розраховуємо період окупності інвестицій $T_{ок}$ (*DPP, Discounted Payback Period*), які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$\mathrm{T_{ок}} = \frac{1}{E_{\mathrm{в}}} \qquad (4.30)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$\mathrm{T_{ок}} = \frac{1}{1{,}42} = 0{,}69$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,69 роки, то фінансування даної наукової розробки є доцільним.

4.5 Висновки до економічної частини

Економічна частина даної роботи містить розрахунок витрат на розробку інформаційної технології тестування WEB застосувань, сума яких складає 608297,124 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки.

В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналоги і є високо конкурентоспроможним. Період окупності складе близько 0,69 роки.

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи здійснено розробку інформаційної технології тестування WEB застосувань та програмного забезпечення з використанням технології Selenium Webdriver

У першому розділі магістерської роботи було проведено аналіз преметної області, а саме сформульовано постановку задачі, здійснено аналіз технологій автоматизованного тестування програмних засобів, проведено огляд методів та засобів вирішення задачі автоматизованного тестування WEB застосувань.

Найефективнішим та найбільш підходящим для даної розробки було обгрунтовано та обрано метод тестування чорної скриньки функціоналу графічного інтерфейсу користувача WEB застосувань, продемострировано вигідність інформаційної технології у порівнянні з ручним тестуванням. Також було проведено огляд програм-аналогів, недоліками яких є надмірна складність використання, що полягає у необхідності володіння мовами програмування і написання сценаріїв-скриптів тесуваня для кожного об’єкту і відсутність україномовного інтерфейсу.

У другому розділі магістерської кваліфікаційної роботи було проведено аналіз та обгрунтовано вибір складових компонентів інформаційної технології тестування WEB застосувань, розроблено структурну схему інформаційної технології, що містить 6 головних модулів, які виконують покладені функції на програмне забезпечення, а саме, модуль інтерфейсу користувача, модуль перевірки валідності веб-застосування, модуль об’єктів веб-застосування, модуль управління браузерами, модуль тестових сценаріїв та модуль виведення результатів. Удосконаленно математичну модель оптимізації процесу тестування WEB застосувань, дає можливість оцінити повноту тестування, позбавлена надлишковості, тобто зайвих кроків під час проходження тест-кейсу. здійснено комп’ютерне моделювання інформаційної технології тестування WEB застосувань і на основі математичної моделі розроблено і удосконалено

алгоритм функціонування інформаційної технології тестування WEB застосувань .

У третьому розділі розглянуто розглянуто переваги та недоліки трьох об’єктно-орієнтованих мов програмування та обгрунтовано вибір мови Python для розробки інформаційної технології, а також середовища програмування PyCharm. Розроблено інтерфейс користувача та програмне забезпечення з використанням фреймворку Selenium та технології WebDriver. Проведено тестування розробленого програмного забезпечення, що здійснювалось на основі smoke-тестування. Усього здійснювалось 100 заспусків програми із тестовою вибіркою у розмірі 50 об’єктів тестування для кожного створеного сценарію. Проведене підтверджує правильність роботи усіх покладених на неї задач, і здійснено оцінку ефективності тестування, за відповідною методикою обчислення, яка показує, що у порівнянні із двома подібними програмними рішеннями (Ranorex та Katalon Studio, що мають відповідну ефективність 16 та 15 %) ефективність розробленого програмного забезпечення становить 19% і в середньому підвищилась на 3,5%.

У четвертому розділі було виконано розрахунок на розробку інформаційної технології тестування WEB застосувань, сума яких складає 608297,124 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналоги і є високо конкурентоспроможним. Період окупності складе близько 0,69 роки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вовчарук П. Ю., Озеранський В. С., Перевозніков С. І. – Особливості тестування функціоналу веб-застосунків – Репозиторій ВНТУ [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/11656

2. Вовчарук П. Ю., Озеранський В. С. – особливості автоматизованого тестування веб-застосувань [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15401/12972

3. Software testing [Електронний ресурс] – Режим доступу: https://economictimes.indiatimes.com/definition/software-testing

4. Тестування ПЗ [Електронний ресурс] – Режим доступу: https://qalearning.com.ua/theory/about_qa/shpargalka-z-testuvannya/

5. Верифікація та валідація [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/verifikatsiya-ta-validatsiya/

6. Баг репорт [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/bug-report/

7. Методи тестування. Статичне та динамічне тестування [Електронний ресурс] – Режим доступу: https://inlnk.ru/qO404

8. Тестування білої скриньки [Електронний ресурс] – Режим доступу: https://uk.itpedia.nl/2018/02/05/white-box-testing-onder-de-loep/

9. Тестування чорної скриньки [Електронний ресурс] – Режим доступу: https://uk.jejakjabar.com/wiki/Black-box_testing

10. Функціональне тестування [Електронний ресурс] – Режим доступу: https://cutt.ly/PbECq98

11. Атоматизовне тестування [Електронний ресурс] – Режим доступу: https://gist.github.com/codedokode/a455bde7d0748c0a351

12. Selenium [Електронний ресурс] – Режим доступу: https://cutt.ly/HbECU0E

13. Тестування веб-застосунків копонентами Selenium [Електронний ресурс] – Режим доступу: https://www.uplab.ru/blog/automated-testing-of-website/

14. SeleniumWebdriver [Електронний ресурс] – Режим доступу: https://unetway.com/tutorial/selenium-webdriver

15. Основні поняття та методи Webdriver [Електронний ресурс] – Режим доступу: https://artstroy.net/selenium-webdriver-vvedenie/

16. Вигідність автоматизації [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/otus/blog/461257/

17. Програми для тестування веб-застосунків [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/342234/

18. Unified Functional Testing [Електронний ресурс] – Режим доступу: https://www.guru99.com/uft-qtp-automation-testing.html

19. TestComplete [Електронний ресурс] – Режим доступу: https://smartbear.com/product/testcomplete

20. TestPlant eggPlant [Електронний ресурс] – Режим доступу: https://www.eggplantsoftware.com/

21. Ranorex [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/2gis/blog/188302/

22. Інформаційні технології [Електронний ресурс] – Режим доступу:http://apeps.kpi.ua/shcho-take-informatsiini-technologii/en

23. Математична модель тестування веб застосувань [Електронний ресурс] – Режим доступу:https://www.researchgate.net/publication/230635418_A_Mathematical_Modelling_Approach_for_Software_Testing_Optimization

24. Основи UML [Електронний ресурс] – Режим доступу: https://docs.kde.org/stable5/uk/umbrello/umbrello/uml-basics.html#about-uml

25.Use-case діаграма [Електронний ресурс] – Режим доступу: https://nationalteam.worldskills.ru/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/

26. UML діаграма послідовності (sequence diagram) [Електронний ресурс] – Режим доступу: http://flash.retejo.info/cxefpagxo/uml/diagrama-poslidovnosti

26. Коноваленко І.В. Програмування мовою C# / Коноваленко І.В.. – Тернопіль: ТНТУ, 2016. – 227 с.

27. Трофименко О.Г., Прокоп Ю.В., Швайко І.Г., Буката Л.М., Косирева Л.А., Леонов Ю. Г., Ясинський В. В.. «С++. Основи програмування. Теорія та практика: підручник.». - 2010.

28. Python. History and License [Електронний ресурс] – Режим доступу: https://docs.python.org/3/license.html

29. Python. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

30. PyCharm - The Python IDE for Professional Developers [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/pycharm/

31. PyCharm [Електронний ресурс] – Режим доступу:https://uk.wikipedia.org/wiki/PyCharm

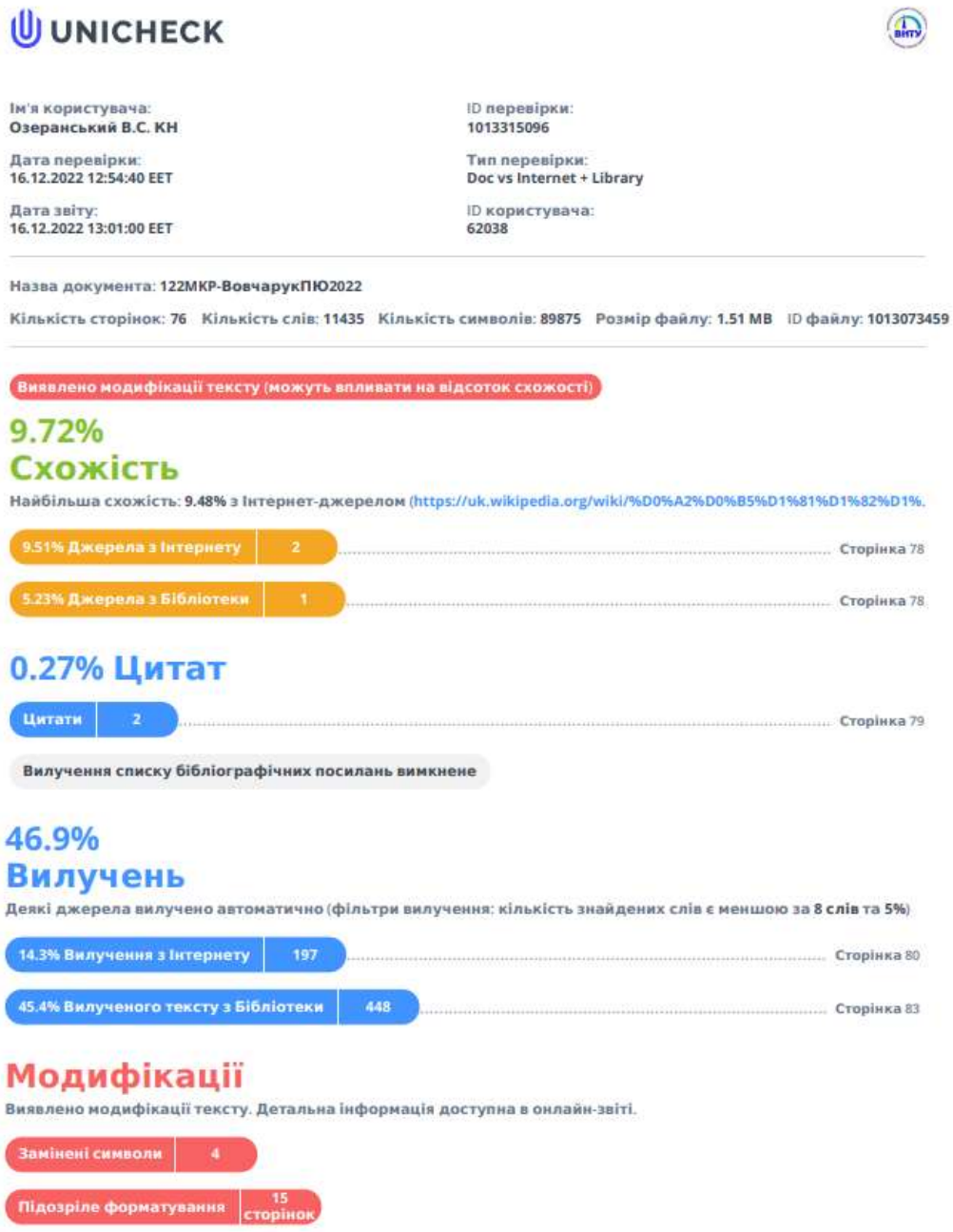
32. Ефективність автоматизованого тестування [Електронний ресурс] – Режим доступу: https://svyatoslav.biz/software_testing_book_download_en/

33. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с

ДОДАТКИ

Додаток А (обов'язковий)

Результат перевірки на плагіат в онлайн-системі UNICHECK



UNICHECK

Ім'я користувача:
Озеранський В.С. КН

ID перевірки:
1013315096

Дата перевірки:
16.12.2022 12:54:40 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.12.2022 13:01:00 EET

ID користувача:
62038

Назва документа: 122МКР-ВовчарукПІО2022

Кількість сторінок: 76 Кількість слів: 11435 Кількість символів: 89875 Розмір файлу: 1.51 MB ID файлу: 1013073459

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

9.72%
Схожість

Найбільша схожість: 9.48% з Інтернет-джерелом (https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%.

9.51% Джерела з Інтернету 2 Сторінка 78

5.23% Джерела з Бібліотеки 1 Сторінка 78

0.27% Цитат

Цитати 2 Сторінка 79

Вилучення списку бібліографічних посилань вимкнене

46.9%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 8 слів та 5%)

14.3% Вилучення з Інтернету 197 Сторінка 80

45.4% Вилученого тексту з Бібліотеки 448 Сторінка 83

Модифікації

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

Замінені символи 4

Підозріле форматування 15 сторінок

Додаток Б (обов'язковий)

Лістинг програми

```
from selenium.common.exceptions import ElementClickInterceptedException
from selenium.common.exceptions import ElementNotInteractableException
from selenium.common.exceptions import ElementNotSelectableException
from selenium.common.exceptions import ElementNotVisibleException
from selenium.common.exceptions import ImeActivationFailedException
from selenium.common.exceptions import ImeNotAvailableException
from selenium.common.exceptions import InsecureCertificateException
from selenium.common.exceptions import InvalidArgumentException
from selenium.common.exceptions import InvalidCookieDomainException
from selenium.common.exceptions import InvalidCoordinatesException
from selenium.common.exceptions import InvalidElementStateException
from selenium.common.exceptions import InvalidSelectorException
from selenium.common.exceptions import InvalidSessionIdException
from selenium.common.exceptions import JavascriptException
from selenium.common.exceptions import MoveTargetOutOfBoundsException
from selenium.common.exceptions import NoAlertPresentException
from selenium.common.exceptions import NoSuchCookieException
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoSuchFrameException
from selenium.common.exceptions import NoSuchShadowRootException
from selenium.common.exceptions import NoSuchWindowException
from selenium.common.exceptions import ScreenshotException
from selenium.common.exceptions import SessionNotCreatedException
from selenium.common.exceptions import StaleElementReferenceException
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import UnableToSetCookieException
from selenium.common.exceptions import UnexpectedAlertPresentException
from selenium.common.exceptions import UnknownMethodException
from selenium.common.exceptions import WebDriverException


class ErrorCode:
    """Error codes defined in the WebDriver wire protocol."""

    # Keep in sync with org.openqa.selenium.remote.ErrorCodes and errorcodes.h
    SUCCESS = 0
    NO_SUCH_ELEMENT = [7, "no such element"]
    NO_SUCH_FRAME = [8, "no such frame"]
    NO_SUCH_SHADOW_ROOT = ["no such shadow root"]
    UNKNOWN_COMMAND = [9, "unknown command"]
    STALE_ELEMENT_REFERENCE = [10, "stale element reference"]
    ELEMENT_NOT_VISIBLE = [11, "element not visible"]
    INVALID_ELEMENT_STATE = [12, "invalid element state"]
    UNKNOWN_ERROR = [13, "unknown error"]
    ELEMENT_IS_NOT_SELECTABLE = [15, "element not selectable"]
    JAVASCRIPT_ERROR = [17, "javascript error"]
    XPATH_LOOKUP_ERROR = [19, "invalid selector"]
    TIMEOUT = [21, "timeout"]
    NO_SUCH_WINDOW = [23, "no such window"]
    INVALID_COOKIE_DOMAIN = [24, "invalid cookie domain"]
    UNABLE_TO_SET_COOKIE = [25, "unable to set cookie"]
```

```
            UNEXPECTED_ALERT_OPEN = [26, "unexpected alert open"]
            NO_ALERT_OPEN = [27, "no such alert"]
            SCRIPT_TIMEOUT = [28, "script timeout"]
            INVALID_ELEMENT_COORDINATES = [29, "invalid element coordinates"]
            IME_NOT_AVAILABLE = [30, "ime not available"]
            IME_ENGINE_ACTIVATION_FAILED = [31, "ime engine activation failed"]
            INVALID_SELECTOR = [32, "invalid selector"]
            SESSION_NOT_CREATED = [33, "session not created"]
            MOVE_TARGET_OUT_OF_BOUNDS = [34, "move target out of bounds"]
            INVALID_XPATH_SELECTOR = [51, "invalid selector"]
            INVALID_XPATH_SELECTOR_RETURN_TYPER = [52, "invalid selector"]

            ELEMENT_NOT_INTERACTABLE = [60, "element not interactable"]
            INSECURE_CERTIFICATE = ["insecure certificate"]
            INVALID_ARGUMENT = [61, "invalid argument"]
            INVALID_COORDINATES = ["invalid coordinates"]
            INVALID_SESSION_ID = ["invalid session id"]
            NO_SUCH_COOKIE = [62, "no such cookie"]
            UNABLE_TO_CAPTURE_SCREEN = [63, "unable to capture screen"]
            ELEMENT_CLICK_INTERCEPTED = [64, "element click intercepted"]
            UNKNOWN_METHOD = ["unknown method exception"]

            METHOD_NOT_ALLOWED = [405, "unsupported operation"]


        class ErrorHandler:
            """Handles errors returned by the WebDriver server."""

            def check_response(self, response: Dict[str, Any]) -> None:
                """Checks that a JSON response from the WebDriver does not have an
                error.
                :Args:
                 - response - The JSON response from the WebDriver server as a
    dictionary
                   object.
                :Raises: If the response contains an error message.
                """
                status = response.get("status", None)
                if not status or status == ErrorCode.SUCCESS:
                    return
                value = None
                message = response.get("message", "")
                screen: str = response.get("screen", "")
                stacktrace = None
                if isinstance(status, int):
                    value_json = response.get("value", None)
                    if value_json and isinstance(value_json, str):
                        import json

                        try:
                            value = json.loads(value_json)
                            if len(value.keys()) == 1:
                                value = value["value"]
                            status = value.get("error", None)
                            if not status:
                                status = value.get("status", ErrorCode.UNKNOWN_ERROR)
                                message = value.get("value") or value.get("message")
                                if not isinstance(message, str):
                                    value = message
                                    message = message.get("message")
                            else:
                                message = value.get("message", None)
```

```
        except ValueError:
            pass

    exception_class: Type[WebDriverException]
    if status in ErrorCode.NO_SUCH_ELEMENT:
        exception_class = NoSuchElementException
    elif status in ErrorCode.NO_SUCH_FRAME:
        exception_class = NoSuchFrameException
    elif status in ErrorCode.NO_SUCH_SHADOW_ROOT:
        exception_class = NoSuchShadowRootException
    elif status in ErrorCode.NO_SUCH_WINDOW:
        exception_class = NoSuchWindowException
    elif status in ErrorCode.STALE_ELEMENT_REFERENCE:
        exception_class = StaleElementReferenceException
    elif status in ErrorCode.ELEMENT_NOT_VISIBLE:
        exception_class = ElementNotVisibleException
    elif status in ErrorCode.INVALID_ELEMENT_STATE:
        exception_class = InvalidElementStateException
    elif (
        status in ErrorCode.INVALID_SELECTOR
        or status in ErrorCode.INVALID_XPATH_SELECTOR
        or status in ErrorCode.INVALID_XPATH_SELECTOR_RETURN_TYPER
    ):
        exception_class = InvalidSelectorException
    elif status in ErrorCode.ELEMENT_IS_NOT_SELECTABLE:
        exception_class = ElementNotSelectableException
    elif status in ErrorCode.ELEMENT_NOT_INTERACTABLE:
        exception_class = ElementNotInteractableException
    elif status in ErrorCode.INVALID_COOKIE_DOMAIN:
        exception_class = InvalidCookieDomainException
    elif status in ErrorCode.UNABLE_TO_SET_COOKIE:
        exception_class = UnableToSetCookieException
    elif status in ErrorCode.TIMEOUT:
        exception_class = TimeoutException
    elif status in ErrorCode.SCRIPT_TIMEOUT:
        exception_class = TimeoutException
    elif status in ErrorCode.UNKNOWN_ERROR:
        exception_class = WebDriverException
    elif status in ErrorCode.UNEXPECTED_ALERT_OPEN:
        exception_class = UnexpectedAlertPresentException
    elif status in ErrorCode.NO_ALERT_OPEN:
        exception_class = NoAlertPresentException
    elif status in ErrorCode.IME_NOT_AVAILABLE:
        exception_class = ImeNotAvailableException
    elif status in ErrorCode.IME_ENGINE_ACTIVATION_FAILED:
        exception_class = ImeActivationFailedException
    elif status in ErrorCode.MOVE_TARGET_OUT_OF_BOUNDS:
        exception_class = MoveTargetOutOfBoundsException
    elif status in ErrorCode.JAVASCRIPT_ERROR:
        exception_class = JavascriptException
    elif status in ErrorCode.SESSION_NOT_CREATED:
        exception_class = SessionNotCreatedException
    elif status in ErrorCode.INVALID_ARGUMENT:
        exception_class = InvalidArgumentException
    elif status in ErrorCode.NO_SUCH_COOKIE:
        exception_class = NoSuchCookieException
    elif status in ErrorCode.UNABLE_TO_CAPTURE_SCREEN:
        exception_class = ScreenshotException
    elif status in ErrorCode.ELEMENT_CLICK_INTERCEPTED:
        exception_class = ElementClickInterceptedException
    elif status in ErrorCode.INSECURE_CERTIFICATE:
        exception_class = InsecureCertificateException
```

```
                elif status in ErrorCode.INVALID_COORDINATES:
                    exception_class = InvalidCoordinatesException
                elif status in ErrorCode.INVALID_SESSION_ID:
                    exception_class = InvalidSessionIdException
                elif status in ErrorCode.UNKNOWN_METHOD:
                    exception_class = UnknownMethodException
                else:
                    exception_class = WebDriverException
                if not value:
                    value = response["value"]
                if isinstance(value, str):
                    raise exception_class(value)
                if message == "" and "message" in value:
                    message = value["message"]

                screen = None  # type: ignore[assignment]
                if "screen" in value:
                    screen = value["screen"]

                stacktrace = None
                st_value = value.get("stackTrace") or value.get("stacktrace")
                if st_value:
                    if isinstance(st_value, str):
                        stacktrace = st_value.split("\n")
                    else:
                        stacktrace = []
                        try:
                            for frame in st_value:
                                line = frame.get("lineNumber", "")
                                file = frame.get("fileName", "<anonymous>")
                                if line:
                                    file = f"{file}:{line}"
                                meth = frame.get("methodName", "<anonymous>")
                                if "className" in frame:
                                    meth = f"{frame['className']}.{meth}"
                                msg = "    at %s (%s)"
                                msg = msg % (meth, file)
                                stacktrace.append(msg)
                        except TypeError:
                            pass
                if exception_class == UnexpectedAlertPresentException:
                    alert_text = None
                    if "data" in value:
                        alert_text = value["data"].get("text")
                    elif "alert" in value:
                        alert_text = value["alert"].get("text")
                    raise exception_class(message, screen, stacktrace, alert_text)  #
type: ignore[call-arg]  # mypy is not smart enough here
                raise exception_class(message, screen, stacktrace)


        import contextlib
        import copy
        import pkgutil
        import types
        import typing
        import warnings
        from abc import ABCMeta
        from base64 import b64decode
        from base64 import urlsafe_b64encode
        from contextlib import asynccontextmanager
        from contextlib import contextmanager
```

```
        from importlib import import_module
        from typing import Dict
        from typing import List
        from typing import Optional
        from typing import Union

        from selenium.common.exceptions import InvalidArgumentException
        from selenium.common.exceptions import JavascriptException
        from selenium.common.exceptions import NoSuchCookieException
        from selenium.common.exceptions import NoSuchElementException
        from selenium.common.exceptions import WebDriverException
        from selenium.webdriver.common.by import By
        from selenium.webdriver.common.html5.application_cache import ApplicationCache
        from selenium.webdriver.common.options import BaseOptions
        from selenium.webdriver.common.print_page_options import PrintOptions
        from selenium.webdriver.common.timeouts import Timeouts
        from selenium.webdriver.common.virtual_authenticator import Credential
        from selenium.webdriver.common.virtual_authenticator import
    VirtualAuthenticatorOptions
        from selenium.webdriver.common.virtual_authenticator import (
            required_virtual_authenticator,
        )
        from selenium.webdriver.support.relative_locator import RelativeBy

        from .bidi_connection import BidiConnection
        from .command import Command
        from .errorhandler import ErrorHandler
        from .file_detector import FileDetector
        from .file_detector import LocalFileDetector
        from .mobile import Mobile
        from .remote_connection import RemoteConnection
        from .script_key import ScriptKey
        from .shadowroot import ShadowRoot
        from .switch_to import SwitchTo
        from .webelement import WebElement

        _W3C_CAPABILITY_NAMES = frozenset(
            [
                "acceptInsecureCerts",
                "browserName",
                "browserVersion",
                "pageLoadStrategy",
                "platformName",
                "proxy",
                "setWindowRect",
                "strictFileInteractability",
                "timeouts",
                "unhandledPromptBehavior",
                "webSocketUrl",
            ]
        )

        _OSS_W3C_CONVERSION = {"acceptSslCerts": "acceptInsecureCerts", "version":
    "browserVersion", "platform": "platformName"}

        cdp = None


        def import_cdp():
            global cdp
            if not cdp:
                cdp = import_module("selenium.webdriver.common.bidi.cdp")
```

```
      def _make_w3c_caps(caps):
          """Makes a W3C alwaysMatch capabilities object.
          Filters out capability names that are not in the W3C spec. Spec-compliant
          drivers will reject requests containing unknown capability names.
          Moves the Firefox profile, if present, from the old location to the new
Firefox
          options object.
          :Args:
           - caps - A dictionary of capabilities requested by the caller.
          """
          caps = copy.deepcopy(caps)
          profile = caps.get("firefox_profile")
          always_match = {}
          if caps.get("proxy") and caps["proxy"].get("proxyType"):
              caps["proxy"]["proxyType"] = caps["proxy"]["proxyType"].lower()
          for k, v in caps.items():
              if v and k in _OSS_W3C_CONVERSION:
                  # Todo: Remove in 4.7.0 (Deprecated in 4.5.0)
                  w3c_equivalent = _OSS_W3C_CONVERSION[k]
                  warnings.warn(
                      f"{k} is not a w3c capability.  use `{w3c_equivalent}`
instead.  This will no longer be"
                      f" converted in 4.7.0",
                      DeprecationWarning,
                      stacklevel=2,
                  )
                  always_match[w3c_equivalent] = v.lower() if k == "platform" else v
              if k in _W3C_CAPABILITY_NAMES or ":" in k:
                  always_match[k] = v
          if profile:
              moz_opts = always_match.get("moz:firefoxOptions", {})
              # If it's already present, assume the caller did that intentionally.
              if "profile" not in moz_opts:
                  # Don't mutate the original capabilities.
                  new_opts = copy.deepcopy(moz_opts)
                  new_opts["profile"] = profile
                  always_match["moz:firefoxOptions"] = new_opts
          return {"firstMatch": [{}], "alwaysMatch": always_match}


      def get_remote_connection(capabilities, command_executor, keep_alive,
ignore_local_proxy=False):
          from selenium.webdriver.chromium.remote_connection import
ChromiumRemoteConnection
          from selenium.webdriver.firefox.remote_connection import
FirefoxRemoteConnection
          from selenium.webdriver.safari.remote_connection import
SafariRemoteConnection

          candidates = [RemoteConnection, ChromiumRemoteConnection,
SafariRemoteConnection, FirefoxRemoteConnection]
          handler = next((c for c in candidates if c.browser_name ==
capabilities.get("browserName")), RemoteConnection)

          return handler(command_executor, keep_alive=keep_alive,
ignore_proxy=ignore_local_proxy)


      def create_matches(options: List[BaseOptions]) -> Dict:
          capabilities = {"capabilities": {}}
```

```
            opts = []
            for opt in options:
                opts.append(opt.to_capabilities())
            opts_size = len(opts)
            samesies = {}

            # Can not use bitwise operations on the dicts or lists due to
            # https://bugs.python.org/issue38210
            for i in range(opts_size):
                min_index = i
                if i + 1 < opts_size:
                    first_keys = opts[min_index].keys()

                    for kys in first_keys:
                        if kys in opts[i + 1].keys():
                            if opts[min_index][kys] == opts[i + 1][kys]:
                                samesies.update({kys: opts[min_index][kys]})

            always = {}
            for k, v in samesies.items():
                always[k] = v

            for i in opts:
                for k in always:
                    del i[k]

            capabilities["capabilities"]["alwaysMatch"] = always
            capabilities["capabilities"]["firstMatch"] = opts

            return capabilities


        class BaseWebDriver(metaclass=ABCMeta):
            """Abstract Base Class for all Webdriver subtypes.
            ABC's allow custom implementations of Webdriver to be registered so
            that isinstance type checks will succeed.
            """


        class WebDriver(BaseWebDriver):
            """Controls a browser by sending commands to a remote server. This server
            is expected to be running the WebDriver wire protocol as defined at
            https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol.
            :Attributes:
             - session_id - String ID of the browser session started and controlled by
this WebDriver.
             - capabilities - Dictionary of effective capabilities of this browser
session as returned
                 by the remote server. See
https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities
             - command_executor - remote_connection.RemoteConnection object used to
execute commands.
             - error_handler - errorhandler.ErrorHandler object used to handle errors.
            """

            _web_element_cls = WebElement
            _shadowroot_cls = ShadowRoot

            def __init__(
                self,
                command_executor="http://127.0.0.1:4444",
                desired_capabilities=None,
```

```
                browser_profile=None,
                proxy=None,
                keep_alive=True,
                file_detector=None,
                options: Union[BaseOptions, List[BaseOptions]] = None,
            ) -> None:
                """Create a new driver that will issue commands using the wire
                protocol.
                :Args:
                 - command_executor - Either a string representing URL of the remote
server or a custom
                     remote_connection.RemoteConnection object. Defaults to
'http://127.0.0.1:4444/wd/hub'.
                 - desired_capabilities - A dictionary of capabilities to request when
                     starting the browser session. Required parameter.
                 - browser_profile - A
selenium.webdriver.firefox.firefox_profile.FirefoxProfile object.
                     Only used if Firefox is requested. Optional.
                 - proxy - A selenium.webdriver.common.proxy.Proxy object. The browser
session will
                     be started with given proxy settings, if possible. Optional.
                 - keep_alive - Whether to configure
remote_connection.RemoteConnection to use
                     HTTP keep-alive. Defaults to True.
                 - file_detector - Pass custom file detector object during
instantiation. If None,
                     then default LocalFileDetector() will be used.
                 - options - instance of a driver options.Options class
                """
                if desired_capabilities:
                    warnings.warn(
                        "desired_capabilities has been deprecated, please pass in an
Options object with options kwarg",
                        DeprecationWarning,
                        stacklevel=2,
                    )
                if browser_profile:
                    warnings.warn(
                        "browser_profile has been deprecated, please pass in an
Firefox Options object with options kwarg",
                        DeprecationWarning,
                        stacklevel=2,
                    )
                if proxy:
                    warnings.warn(
                        "proxy has been deprecated, please pass in an Options object
with options kwarg",
                        DeprecationWarning,
                        stacklevel=2,
                    )
                if not keep_alive:
                    warnings.warn(
                        "keep_alive has been deprecated. We will be using True as the
default value as we start removing it.",
                        DeprecationWarning,
                        stacklevel=2,
                    )
                capabilities = {}
                # If we get a list we can assume that no capabilities
                # have been passed in
                if isinstance(options, list):
                    capabilities = create_matches(options)
```

```
                else:
                    _ignore_local_proxy = False
                    if options:
                        capabilities = options.to_capabilities()
                        _ignore_local_proxy = options._ignore_local_proxy
                    if desired_capabilities:
                        if not isinstance(desired_capabilities, dict):
                            raise WebDriverException("Desired Capabilities must be a
dictionary")
                        capabilities.update(desired_capabilities)
                self.command_executor = command_executor
                if isinstance(self.command_executor, (str, bytes)):
                    self.command_executor = get_remote_connection(
                        capabilities,
                        command_executor=command_executor,
                        keep_alive=keep_alive,
                        ignore_local_proxy=_ignore_local_proxy,
                    )
                self._is_remote = True
                self.session_id = None
                self.caps = {}
                self.pinned_scripts = {}
                self.error_handler = ErrorHandler()
                self._switch_to = SwitchTo(self)
                self._mobile = Mobile(self)
                self.file_detector = file_detector or LocalFileDetector()
                self._authenticator_id = None
                self.start_client()
                self.start_session(capabilities, browser_profile)

            def __repr__(self):
                return f'<{type(self).__module__}.{type(self).__name__}
(session="{self.session_id}")>'

            def __enter__(self):
                return self

            def __exit__(
                self,
                exc_type: typing.Optional[typing.Type[BaseException]],
                exc: typing.Optional[BaseException],
                traceback: typing.Optional[types.TracebackType],
            ):
                self.quit()

            @contextmanager
            def file_detector_context(self, file_detector_class, *args, **kwargs):
                """Overrides the current file detector (if necessary) in limited
                context. Ensures the original file detector is set afterwards.
                Example:
                with webdriver.file_detector_context(UselessFileDetector):
                    someinput.send_keys('/etc/hosts')
                :Args:
                 - file_detector_class - Class of the desired file detector. If the
class is different
                      from the current file_detector, then the class is instantiated
with args and kwargs
                      and used as a file detector during the duration of the context
manager.
                 - args - Optional arguments that get passed to the file detector
class during
                      instantiation.
```

```
             - kwargs - Keyword arguments, passed the same way as args.
            """
            last_detector = None
            if not isinstance(self.file_detector, file_detector_class):
                last_detector = self.file_detector
                self.file_detector = file_detector_class(*args, **kwargs)
            try:
                yield
            finally:
                if last_detector:
                    self.file_detector = last_detector

        @property
        def mobile(self) -> Mobile:
            return self._mobile

        @property
        def name(self) -> str:
            """Returns the name of the underlying browser for this instance.
            :Usage:
                ::
                    name = driver.name
            """
            if "browserName" in self.caps:
                return self.caps["browserName"]
            raise KeyError("browserName not specified in session capabilities")

        def start_client(self):
            """Called before starting a new session.
            This method may be overridden to define custom startup behavior.
            """
            pass

        def stop_client(self):
            """Called after executing a quit command.
            This method may be overridden to define custom shutdown
            behavior.
            """
            pass

        def start_session(self, capabilities: dict, browser_profile=None) -> None:
            """Creates a new session with the desired capabilities.
            :Args:
             - capabilities - a capabilities dict to start the session with.
             - browser_profile - A
selenium.webdriver.firefox.firefox_profile.FirefoxProfile object. Only used if
Firefox is requested.
            """
            if not isinstance(capabilities, dict):
                raise InvalidArgumentException("Capabilities must be a
dictionary")
            if browser_profile:
                if "moz:firefoxOptions" in capabilities:
                    capabilities["moz:firefoxOptions"]["profile"] =
browser_profile.encoded
                else:
                    capabilities.update({"firefox_profile":
browser_profile.encoded})
            w3c_caps = _make_w3c_caps(capabilities)
            parameters = {"capabilities": w3c_caps}
            response = self.execute(Command.NEW_SESSION, parameters)
            if "sessionId" not in response:
```

```
                response = response["value"]
            self.session_id = response["sessionId"]
            self.caps = response.get("value")

            # if capabilities is none we are probably speaking to
            # a W3C endpoint
            if not self.caps:
                self.caps = response.get("capabilities")

        def _wrap_value(self, value):
            if isinstance(value, dict):
                converted = {}
                for key, val in value.items():
                    converted[key] = self._wrap_value(val)
                return converted
            if isinstance(value, self._web_element_cls):
                return {"element-6066-11e4-a52e-4f735466cecf": value.id}
            if isinstance(value, self._shadowroot_cls):
                return {"shadow-6066-11e4-a52e-4f735466cecf": value.id}
            if isinstance(value, list):
                return list(self._wrap_value(item) for item in value)
            return value

        def create_web_element(self, element_id: str) -> WebElement:
            """Creates a web element with the specified `element_id`."""
            return self._web_element_cls(self, element_id)

        def _unwrap_value(self, value):
            if isinstance(value, dict):
                if "element-6066-11e4-a52e-4f735466cecf" in value:
                    return self.create_web_element(value["element-6066-11e4-a52e-
4f735466cecf"])
                if "shadow-6066-11e4-a52e-4f735466cecf" in value:
                    return self._shadowroot_cls(self, value["shadow-6066-11e4-
a52e-4f735466cecf"])
                for key, val in value.items():
                    value[key] = self._unwrap_value(val)
                return value
            if isinstance(value, list):
                return list(self._unwrap_value(item) for item in value)
            return value

        def execute(self, driver_command: str, params: dict = None) -> dict:
            """Sends a command to be executed by a command.CommandExecutor.
            :Args:
             - driver_command: The name of the command to execute as a string.
             - params: A dictionary of named parameters to send with the command.
            :Returns:
              The command's JSON response loaded into a dictionary object.
            """
            params = self._wrap_value(params)

            if self.session_id:
                if not params:
                    params = {"sessionId": self.session_id}
                elif "sessionId" not in params:
                    params["sessionId"] = self.session_id

            response = self.command_executor.execute(driver_command, params)
            if response:
                self.error_handler.check_response(response)
```

```
                    response["value"] = self._unwrap_value(response.get("value",
None))
                    return response
                # If the server doesn't send a response, assume the command was
                # a success
                return {"success": 0, "value": None, "sessionId": self.session_id}

            def get(self, url: str) -> None:
                """Loads a web page in the current browser session."""
                self.execute(Command.GET, {"url": url})

            @property
            def title(self) -> str:
                """Returns the title of the current page.
                :Usage:
                    ::
                        title = driver.title
                """
                return self.execute(Command.GET_TITLE).get("value", "")

            def pin_script(self, script: str, script_key=None) -> ScriptKey:
                """Store common javascript scripts to be executed later by a unique
                hashable ID."""
                script_key_instance = ScriptKey(script_key)
                self.pinned_scripts[script_key_instance.id] = script
                return script_key_instance

            def unpin(self, script_key: ScriptKey) -> None:
                """Remove a pinned script from storage."""
                try:
                    self.pinned_scripts.pop(script_key.id)
                except KeyError:
                    raise KeyError(f"No script with key: {script_key} existed in
{self.pinned_scripts}") from None

            def get_pinned_scripts(self) -> List[str]:
                return list(self.pinned_scripts)

            def execute_script(self, script, *args):
                """Synchronously Executes JavaScript in the current window/frame.
                :Args:
                 - script: The JavaScript to execute.
                 - \\*args: Any applicable arguments for your JavaScript.
                :Usage:
                    ::
                        driver.execute_script('return document.title;')
                """
                if isinstance(script, ScriptKey):
                    try:
                        script = self.pinned_scripts[script.id]
                    except KeyError:
                        raise JavascriptException("Pinned script could not be found")

                converted_args = list(args)
                command = Command.W3C_EXECUTE_SCRIPT

                return self.execute(command, {"script": script, "args":
converted_args})["value"]

            def execute_async_script(self, script: str, *args):
                """Asynchronously Executes JavaScript in the current window/frame.
                :Args:
```

```
                - script: The JavaScript to execute.
                - \\*args: Any applicable arguments for your JavaScript.
            :Usage:
                ::
                    script = "var callback = arguments[arguments.length - 1]; " \\
                             "window.setTimeout(function(){ callback('timeout') },
3000);"
                    driver.execute_async_script(script)
            """
            converted_args = list(args)
            command = Command.W3C_EXECUTE_SCRIPT_ASYNC

            return self.execute(command, {"script": script, "args":
converted_args})["value"]

        @property
        def current_url(self) -> str:
            """Gets the URL of the current page.
            :Usage:
                ::
                    driver.current_url
            """
            return self.execute(Command.GET_CURRENT_URL)["value"]

        @property
        def page_source(self) -> str:
            """Gets the source of the current page.
            :Usage:
                ::
                    driver.page_source
            """
            return self.execute(Command.GET_PAGE_SOURCE)["value"]

        def close(self) -> None:
            """Closes the current window.
            :Usage:
                ::
                    driver.close()
            """
            self.execute(Command.CLOSE)

        def quit(self) -> None:
            """Quits the driver and closes every associated window.
            :Usage:
                ::
                    driver.quit()
            """
            try:
                self.execute(Command.QUIT)
            finally:
                self.stop_client()
                self.command_executor.close()

        @property
        def current_window_handle(self) -> str:
            """Returns the handle of the current window.
            :Usage:
                ::
                    driver.current_window_handle
            """
            return self.execute(Command.W3C_GET_CURRENT_WINDOW_HANDLE)["value"]
```

```
            @property
            def window_handles(self) -> List[str]:
                """Returns the handles of all windows within the current session.
                :Usage:
                    ::
                        driver.window_handles
                """
                return self.execute(Command.W3C_GET_WINDOW_HANDLES)["value"]

            def maximize_window(self) -> None:
                """Maximizes the current window that webdriver is using."""
                command = Command.W3C_MAXIMIZE_WINDOW
                self.execute(command, None)

            def fullscreen_window(self) -> None:
                """Invokes the window manager-specific 'full screen' operation."""
                self.execute(Command.FULLSCREEN_WINDOW)

            def minimize_window(self) -> None:
                """Invokes the window manager-specific 'minimize' operation."""
                self.execute(Command.MINIMIZE_WINDOW)

            def print_page(self, print_options: Optional[PrintOptions] = None) -> str:
                """Takes PDF of the current page.
                The driver makes a best effort to return a PDF based on the
                provided parameters.
                """
                options = {}
                if print_options:
                    options = print_options.to_dict()

                return self.execute(Command.PRINT_PAGE, options)["value"]

            @property
            def switch_to(self) -> SwitchTo:
                """
                :Returns:
                    - SwitchTo: an object containing all options to switch focus into
                :Usage:
                    ::
                        element = driver.switch_to.active_element
                        alert = driver.switch_to.alert
                        driver.switch_to.default_content()
                        driver.switch_to.frame('frame_name')
                        driver.switch_to.frame(1)
                        driver.switch_to.frame(driver.find_elements(By.TAG_NAME,
"iframe")[0])
                        driver.switch_to.parent_frame()
                        driver.switch_to.window('main')
                """
                return self._switch_to

            # Navigation
            def back(self) -> None:
                """Goes one step backward in the browser history.
                :Usage:
                    ::
                        driver.back()
                """
                self.execute(Command.GO_BACK)

            def forward(self) -> None:
```

```
            """Goes one step forward in the browser history.
            :Usage:
                ::
                    driver.forward()
            """
            self.execute(Command.GO_FORWARD)

        def refresh(self) -> None:
            """Refreshes the current page.
            :Usage:
                ::
                    driver.refresh()
            """
            self.execute(Command.REFRESH)

        # Options
        def get_cookies(self) -> List[dict]:
            """Returns a set of dictionaries, corresponding to cookies visible in
            the current session.
            :Usage:
                ::
                    driver.get_cookies()
            """
            return self.execute(Command.GET_ALL_COOKIES)["value"]

        def get_cookie(self, name) -> typing.Optional[typing.Dict]:
            """Get a single cookie by name. Returns the cookie if found, None if
            not.
            :Usage:
                ::
                    driver.get_cookie('my_cookie')
            """
            with contextlib.suppress(NoSuchCookieException):
                return self.execute(Command.GET_COOKIE, {"name": name})["value"]

        def delete_cookie(self, name) -> None:
            """Deletes a single cookie with the given name.
            :Usage:
                ::
                    driver.delete_cookie('my_cookie')
            """
            self.execute(Command.DELETE_COOKIE, {"name": name})

        def delete_all_cookies(self) -> None:
            """Delete all cookies in the scope of the session.
            :Usage:
                ::
                    driver.delete_all_cookies()
            """
            self.execute(Command.DELETE_ALL_COOKIES)

        def add_cookie(self, cookie_dict) -> None:
            """Adds a cookie to your current session.
            :Args:
             - cookie_dict: A dictionary object, with required keys - "name" and
"value";
                optional keys - "path", "domain", "secure", "httpOnly", "expiry",
"sameSite"
            Usage:
                driver.add_cookie({'name' : 'foo', 'value' : 'bar'})
                driver.add_cookie({'name' : 'foo', 'value' : 'bar', 'path' : '/'})
```

```
                    driver.add_cookie({'name' : 'foo', 'value' : 'bar', 'path' : '/',
'secure':True})
                    driver.add_cookie({'name': 'foo', 'value': 'bar', 'sameSite':
'Strict'})
                """
                if "sameSite" in cookie_dict:
                    assert cookie_dict["sameSite"] in ["Strict", "Lax", "None"]
                    self.execute(Command.ADD_COOKIE, {"cookie": cookie_dict})
                else:
                    self.execute(Command.ADD_COOKIE, {"cookie": cookie_dict})

            # Timeouts
            def implicitly_wait(self, time_to_wait: float) -> None:
                """Sets a sticky timeout to implicitly wait for an element to be
found,
                or a command to complete. This method only needs to be called one time
                per session. To set the timeout for calls to execute_async_script, see
                set_script_timeout.
                :Args:
                 - time_to_wait: Amount of time to wait (in seconds)
                :Usage:
                    ::
                        driver.implicitly_wait(30)
                """
                self.execute(Command.SET_TIMEOUTS, {"implicit":
int(float(time_to_wait) * 1000)})

            def set_script_timeout(self, time_to_wait: float) -> None:
                """Set the amount of time that the script should wait during an
                execute_async_script call before throwing an error.
                :Args:
                 - time_to_wait: The amount of time to wait (in seconds)
                :Usage:
                    ::
                        driver.set_script_timeout(30)
                """
                self.execute(Command.SET_TIMEOUTS, {"script": int(float(time_to_wait)
* 1000)})

            def set_page_load_timeout(self, time_to_wait: float) -> None:
                """Set the amount of time to wait for a page load to complete before
                throwing an error.
                :Args:
                 - time_to_wait: The amount of time to wait
                :Usage:
                    ::
                        driver.set_page_load_timeout(30)
                """
                try:
                    self.execute(Command.SET_TIMEOUTS, {"pageLoad":
int(float(time_to_wait) * 1000)})
                except WebDriverException:
                    self.execute(Command.SET_TIMEOUTS, {"ms": float(time_to_wait) *
1000, "type": "page load"})

            @property
            def timeouts(self) -> Timeouts:
                """Get all the timeouts that have been set on the current session.
                :Usage:
                    ::
                        driver.timeouts
                :rtype: Timeout
```

Додаток В (обов'язковий)

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ТЕСТУВАННЯ WEB ЗАСТОСУВАНЬ

Виконав: студент 2-го курсу,
групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Вовчарук П. Ю.
(прізвище та ініціали)

Керівник: к.т.н., ст. викладач каф. КН
Озеранський В. С.
(прізвище та ініціали)
«15» 12 2022 р.

Вінниця ВНТУ - 2022 рік

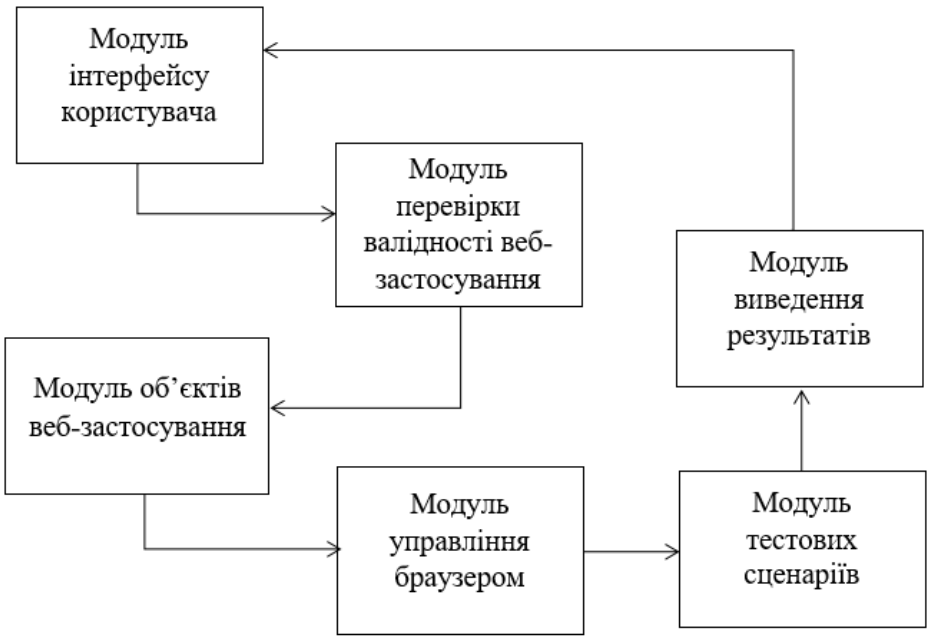


Рисунок В.1 - Структурна схема інформаційної технології тестування WEB застосувань

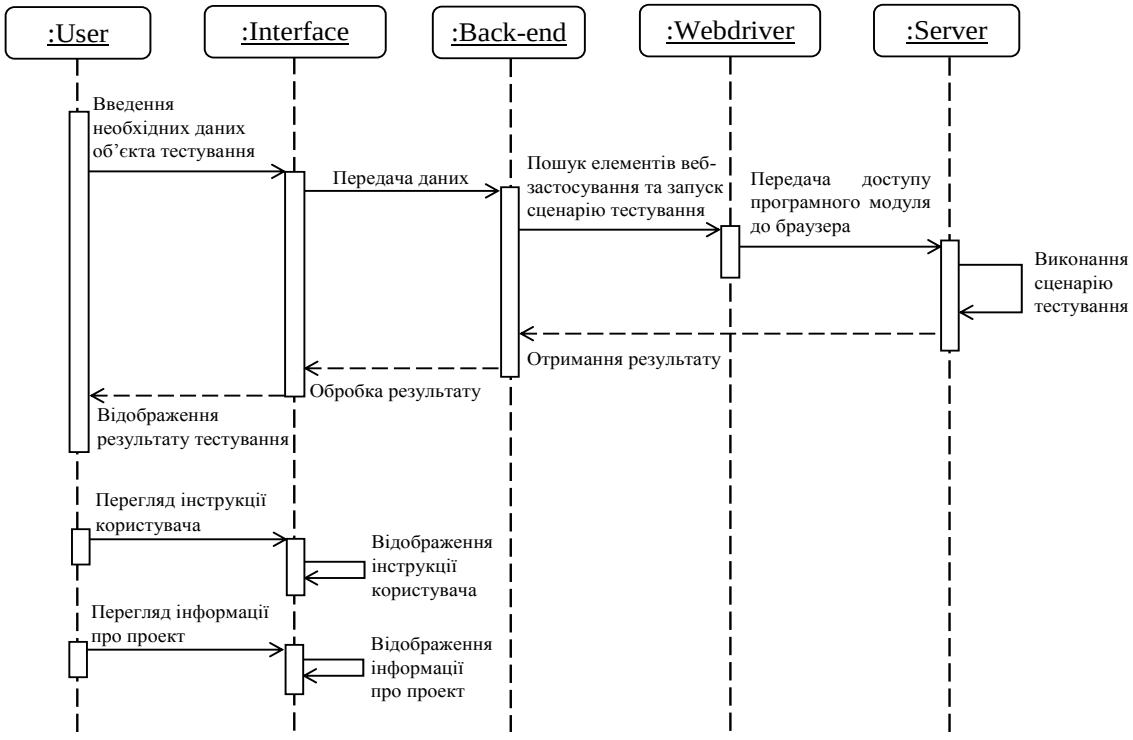


Рисунок В.2 - UML діаграма послідовностей

Рисунок Г.3 – Принцип роботи Selenium Webdriver

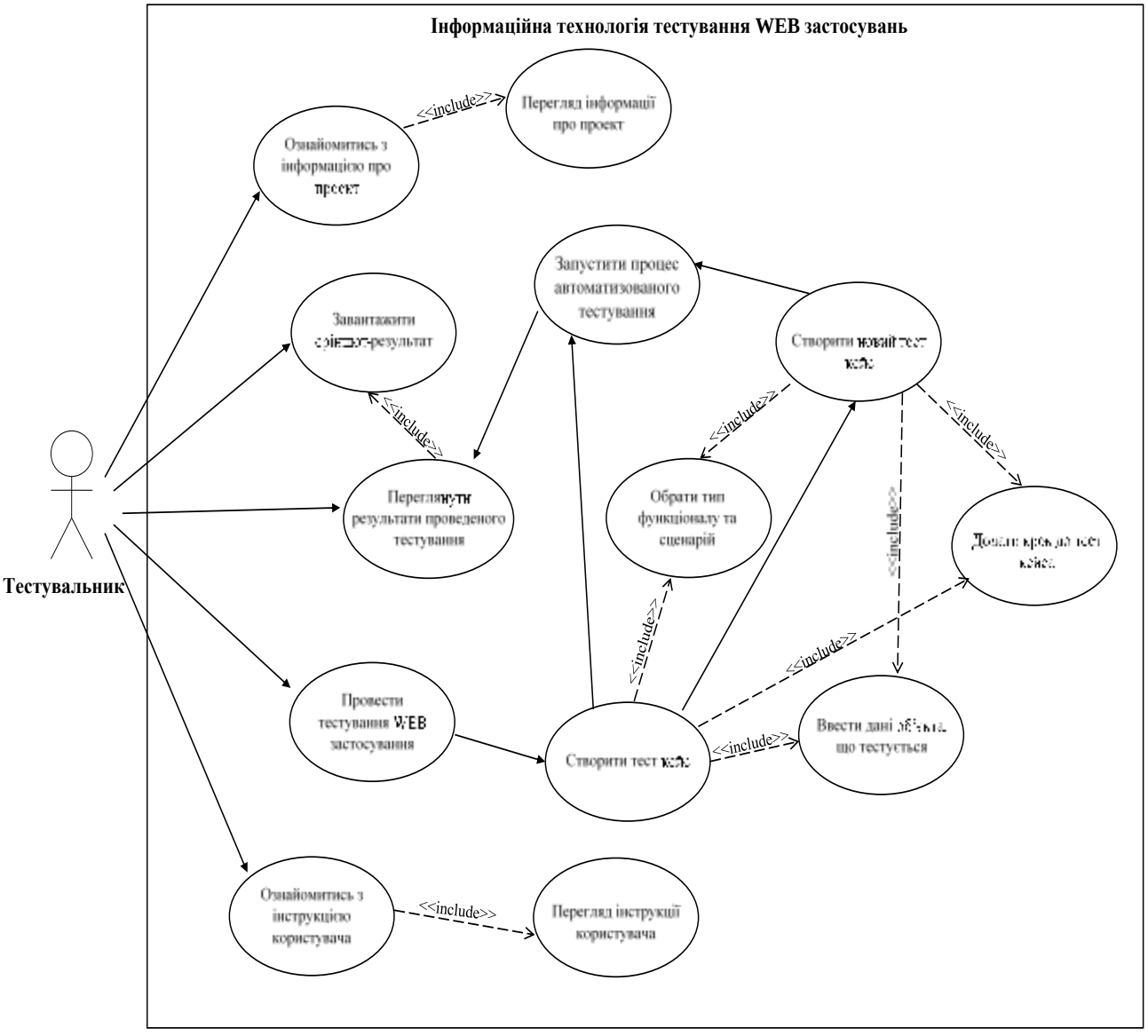


Рисунок В.4 - Usecase-діаграма доступних дій користувача

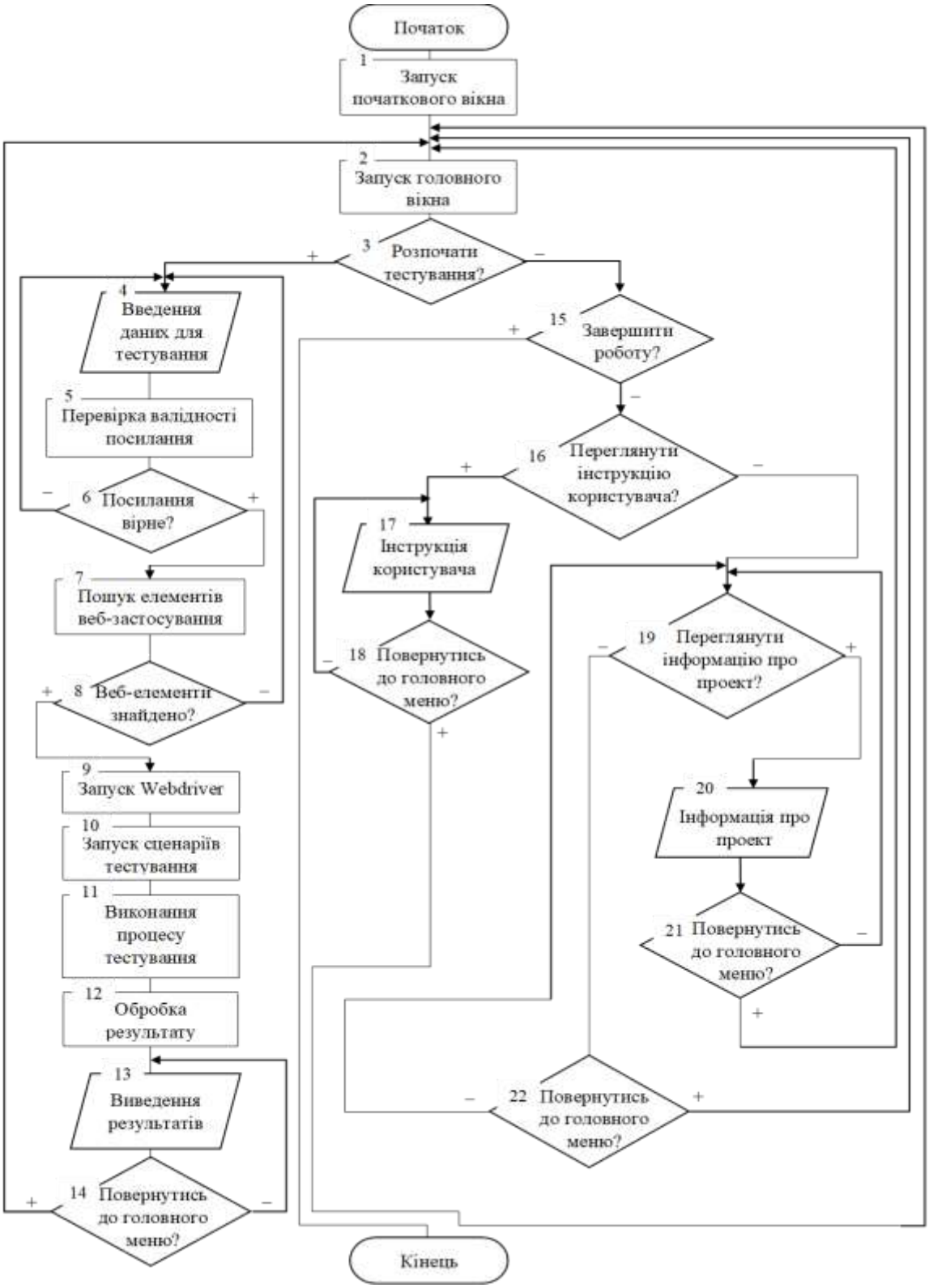


Рисунок В.5 - Алгоритм функціонування інформаційної технології тестування WEB застосувань

Рисунок В.6 – Робочі вікна програмного забезпечення

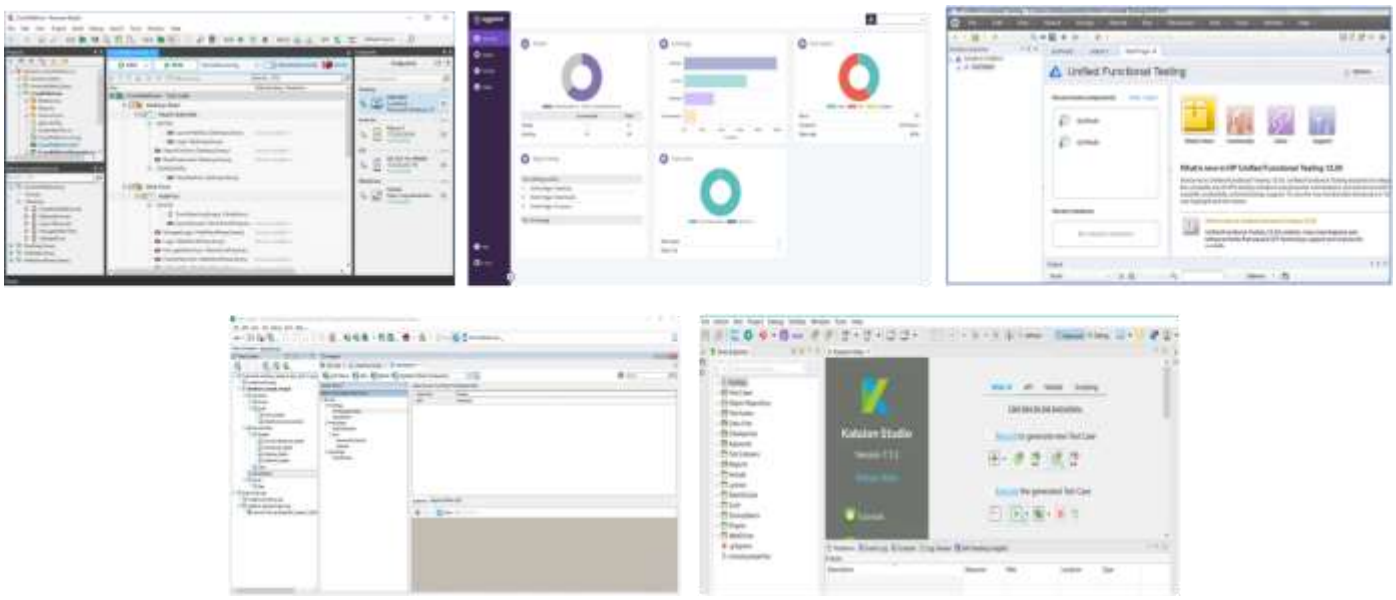


Рисунок В.7 – Робочі вікна програмного забезпечення

Програмний засіб	N	$T^{overall}$	$T^{development}_{automated}$	$T^{support}_{automated}$	A_{effect}
Розроблене програмне забезпечення	1	25	2	130	19%
Ranorex	1	35	6	200	16%
Katalon Studio	1	40	5	250	15%

Рисунок В.8 – Розрахунок ефективності

Додаток Г (довідниковий)

Інструкція користувача

Для початку роботи потрібно запустити середовище розробки PyCharm перейти до проекту. Після цього, у вікні проекту, яке відкриється після попереднього кроку натиснути кнопку запуску. Виконання цих дій наведено на рисунку Г.1. та рисунку Г.2.

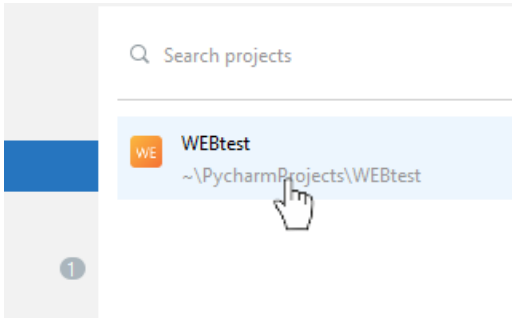


Рисунок Г.1 – Запуск програми

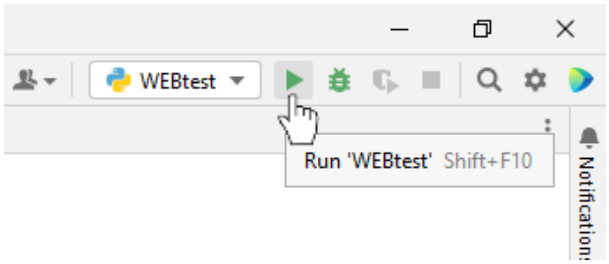


Рисунок Г.2 – Продовження

Після запуску програми модуля відображається початкове вікно програми з привітанням, що наведено на рисунку Г.3, з можливістю переходу до основного вікна програми. Для цього потрібно натиснути кнопку «Розпочати роботу»

Рисунок Г.3 – Початкове вікно програми

Після того, як запустилось головне вікно програмного модуля, що наведено на рисунку Г.4, можна переглянути інструкцію користувача, натиснувши відповідну кнопку «Інструкція користувача», де міститься покроковий опис роботи із програмним модулем і повернутись назад до головного вікна додатка. Результати роботи вікна «Інструкція користувача» наведено на рисунку Г.5

Рисунок Г.4 Головне вікно програмного модуля

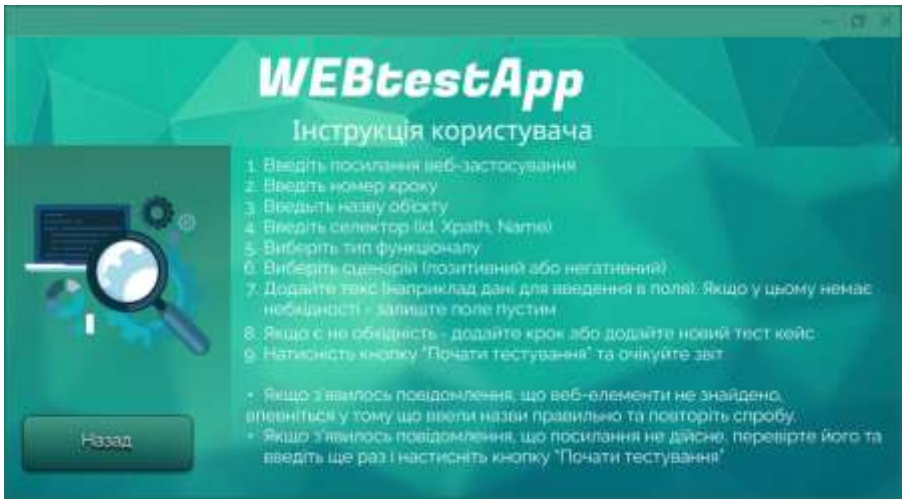


Рисунок Г.5 – Вікно «Інструкція користувача»

У головному вікні міститься перехід на вікно, де знаходиться інформація про проект, що наведено на рисунку А.6. Для того щоб її переглянути, потрібно натиснути на кнопку «Про проект». Ознайомившись з інформацією з цього вікна можна повернутись назад до головного вікна додатка.

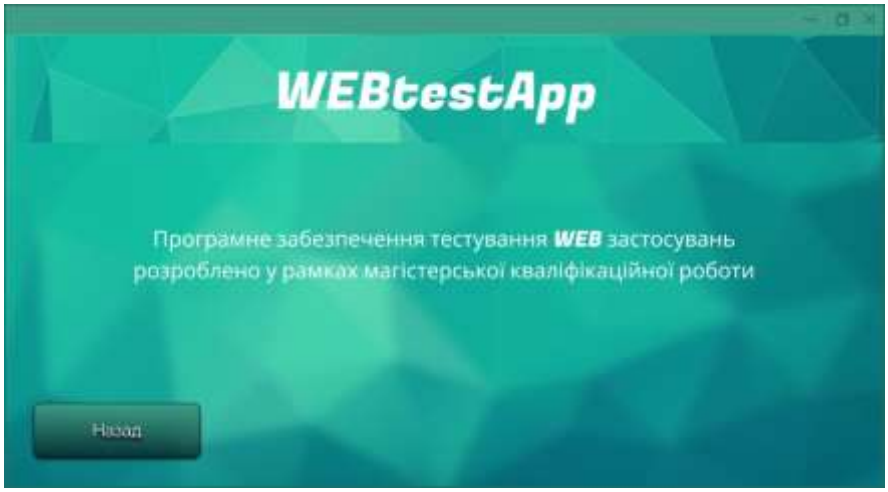


Рисунок Г.6 – Вікно «Про проект»

Для того, щоб розпочати тестування, необхідно у головному вікні програми натиснути на кнопку «Нове тестування». Після цього запускається вікно, що наведено на рисунку Г.7.

Рисунок Г.7– Вікно тестування

У верхнє поле введення потрібно ввести посилання на веб-застосунок, перевірка якого буде здійснюватись, а внижні - назви об’єктів тестування, локатори веб-елементів, що містяться в коді цієї сторінки, працездатність яких буде перевірятись, а також для поля введення - тестові дані, в іншому випадку, коли не потрібно вводити додаткові тестові дані поле потрібно залишити пустим. Також потрібно обрати браузер, тип функціоналу та сценарій з випадаючого меню, що зявляється при наведенні курсору на нього. Дана дія наведена на рисунку Г.8

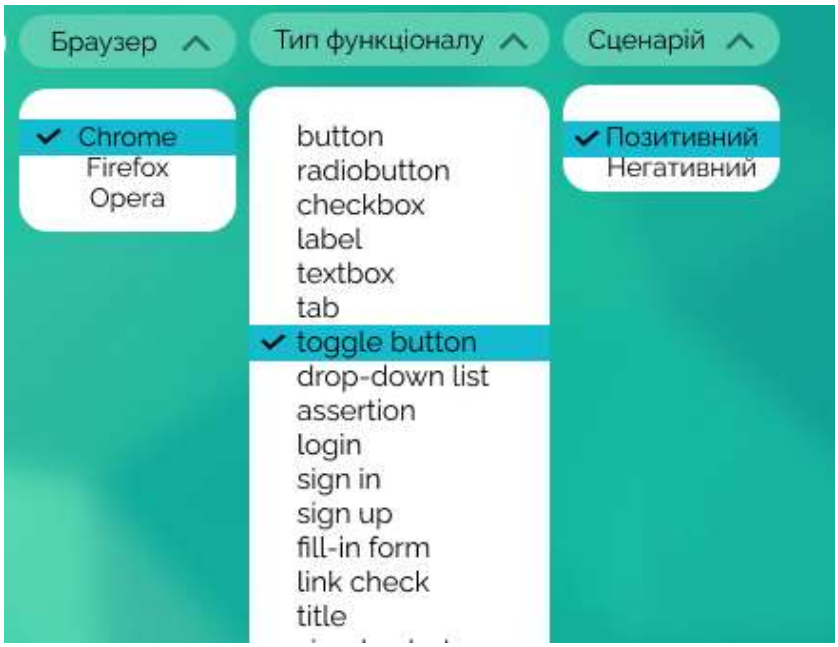


Рисунок Г.8 – Вибір браузера, функціоналу та сценарію

Після того, як користувач натиснув кнопку «Старт», виводиться вікно з повідомленням про очікування результату, що наведено на рисунку Г.9. Коли тестування завершується, відображається вікно результатів, що наведено на рисунку Г.10, в якому міститься інформація про загальну кількість успішних тестів та помилок. Після цього потрібно натиснути кнопку «повернутись до головного меню», а далі можна повторити усі вище перераховані дії або завершити роботу програми, натиснувши відповідну кнопку.

Рисунок Г.9 – Вікно очікування результату

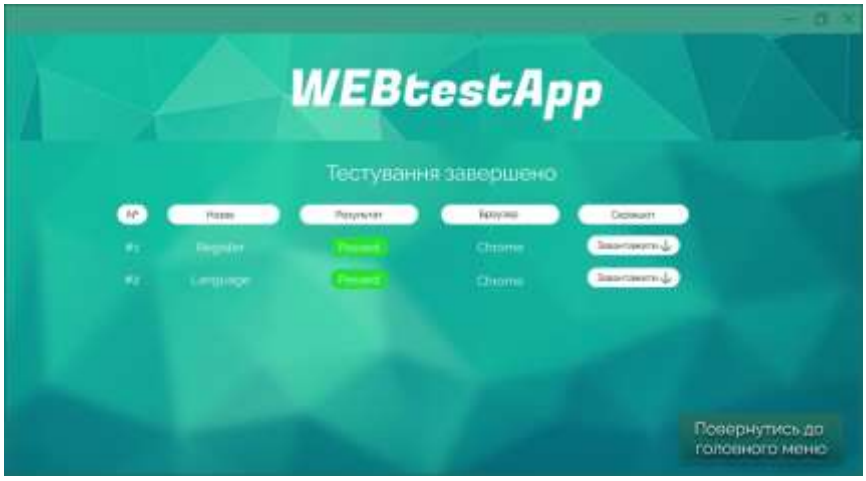
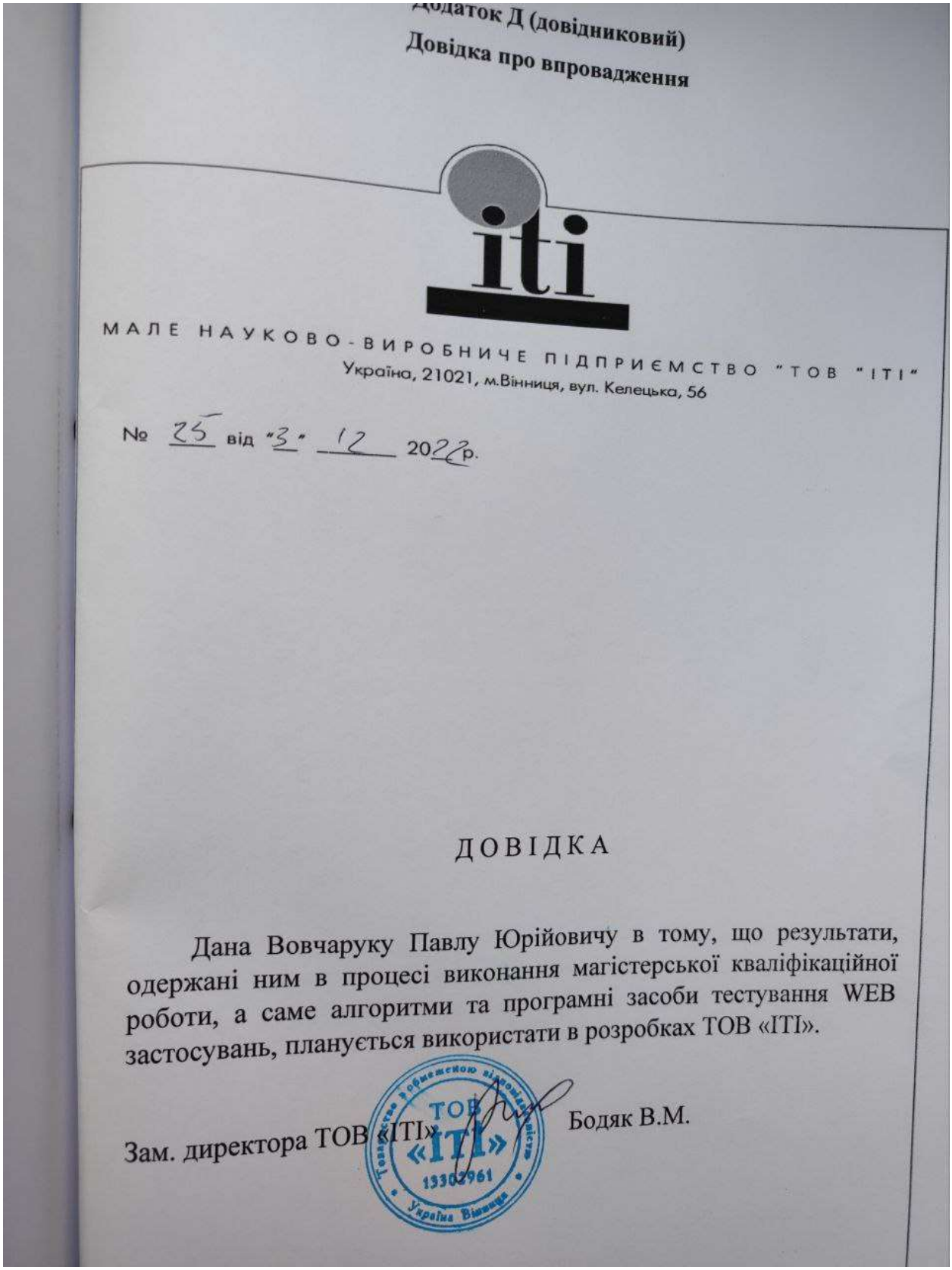


Рисунок Г.10 – Вікно результату

Додаток Д (довідниковий)



Додаток Д (довідниковий)
Довідка про впровадження

МАЛЕ НАУКОВО-ВИРОБНИЧЕ ПІДПРИЄМСТВО "ТОВ "ІТІ"
Україна, 21021, м.Вінниця, вул. Келецька, 56

№ 25 від "3" 12 2022р.

ДОВІДКА

Дана Вовчаруку Павлу Юрійовичу в тому, що результати, одержані ним в процесі виконання магістерської кваліфікаційної роботи, а саме алгоритми та програмні засоби тестування WEB застосувань, планується використати в розробках ТОВ «ІТІ».

Зам. директора ТОВ «ІТІ» Бодяк В.М.

ТОВ «ІТІ» 13302961 Україна Вінниця