Міністерство освіти і науки України
Сумський державний університет
Навчально-науковий інститут бізнес-технологій «УАБС»
Кафедра економічної кібернетики

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему «Автоматизоване тестування веб-додатків»

Виконав студент 2 курсу, групи ЕК.м-81а
(номер курсу) (шифр групи)
спеціальності 051 «Економіка»
(«Економічна кібернетика»)
Ляшко С.Ю.
(прізвище, ініціали студента)

Керівник
доцент, к.т.н. Гриценко К.Г.
(посада, науковий ступінь, прізвище, ініціали)

Суми – 2019 рік

РЕФЕРАТ

кваліфікаційної магістерської роботи на тему

«АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ»

студента Ляшка Сергія Юрійовича
(прізвище, ім’я, по батькові)

Актуальність даної роботи випливає з необхідності автоматизації процесу тестування веб-додатків для підвищення якості тестування, а також зменшення трудових і фінансових ресурсів, які на це витрачаються.

Метою даної роботи є скорочення ресурсів, що витрачаються на тестування веб-додатків, і підвищення рівня ефективності такого тестування.

Задачі, поставлені для досягнення мети роботи:

- проведення аналізу існуючих методів тестування веб-додатків;
- аналіз існуючих засобів автоматизації тестування веб-додатків і вибір найкращого із них;
- розробка прототипу тест кейсів для автоматизації тестування веб-додатків на обраних засобах автоматизації;
- визначення швидкості автоматичного виконання розроблених прототипів тест кейсів засобами автоматизації та порівняння її з ручним тестуванням.

Об’єктом дослідження виступає процес тестування веб-додатків.

Предметом дослідження є засоби та методи, що використовуються для автоматизації тестування веб-додатків.

В дослідженні використовувалися аналітичні методи, а також алгебраїчні методи для оцінки ефективності отриманих результатів.

Наукова новизна отриманих результатів полягає у тому, що було проведено порівняння найбільш розповсюджених до цього часу засобів автоматизації тестування веб-додатків та нових засобів, які ще не є широко розповсюдженими, але мають переваги та дозволять полегшити та покращити процес автоматизації тестування.

Було розроблено прототипи тест кейсів для автоматизації тестування веб-додатків, зокрема сайтів електронної комерції. За необхідності тест кейси можна правити в залежності від веб-додатку, тестування якого проводиться.

Ключові слова: автоматизація, веб-додаток, тест кейс, тестування, прототип, регрессія, програмне забезпечення.

Зміст кваліфікаційної магістерської роботи викладено на 47 сторінках. Список використаних джерел із 69 найменувань розміщений на 6 сторінках. Робота містить 7 таблиць, 6 рисунків, а також 2 додатки, розміщених на 7 сторінках.

Рік виконання кваліфікаційної роботи – 2019 рік.

Рік захисту роботи – 2019 рік.

Міністерство освіти і науки України

Сумський державний університет

Навчально-науковий інститут бізнес-технологій «УАБС»

Кафедра економічної кібернетики

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.е.н., професор
_____ О.В. Кузьменко
“__” _____ 2019 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
(спеціальність 051 «Економіка («Економічна кібернетика»)»)
студенту 2 курсу, групи ЕК.м-81а

Ляшку Сергію Юрійовичу
(прізвище, ім’я, по батькові студента)

1. Тема роботи Автоматизоване тестування веб-додатків
затверджена наказом по університету від «29» жовтня 2019 року № 2163-III
2. Термін подання студентом закінченої роботи «17» грудня 2019 року
3. Мета кваліфікаційної роботи скорочення ресурсів, що витрачаються на тестування веб-додатків, і підвищення рівня ефективності такого тестування
4. Об’єкт дослідження процес тестування веб-додатків
5. Предмет дослідження засоби та методи, що використовуються для автоматизації тестування веб-додатків
6. Кваліфікаційна робота виконується на матеріалах ТОВ «НЕТКРЕКЕР»
7. Орієнтовний план кваліфікаційної роботи, терміни подання розділів керівникові та зміст завдань для виконання поставленої мети

Розділ 1 Аналіз підходів і методів тестування програмного забезпечення 08 листопада 2019 року
(назва – термін подання)

У розділі 1 розглянути основні типи, методи та рівні тестування програмного забезпечення
(зміст конкретних завдань до розділу, які повинен виконати студент)

Розділ 2 Інструменти для тестування веб-додатків 28 листопада 2019 року
(назва – термін подання)

У розділі 2 охарактеризувати основні інструменти, що використовуються для тестування веб-додатків, провести їх порівняльний аналіз і обгрунтувати вибір інструменту для тестування веб-додатків
(зміст конкретних завдань до розділу, які має виконати студент)

Розділ 3 Розробка прототипів тестових сценаріїв _____06 грудня 2019 року_____
(назва – термін подання)

У розділі 3 обрати веб-додатки, на основі яких будуть розроблятись прототипи тестових сценаріїв, створити тест кейси для обраних веб-додатків і їх програмну реалізацію, оцінити очікувані ефекти від впровадження автоматизованого тестування веб-додатків_____
(зміст конкретних завдань до розділу, які повинен виконати студент)

8. Консультації з роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			

9. Дата видачі завдання: «02» жовтня 2019 року

Керівник кваліфікаційної роботи _____ _____
(підпис) (ініціали, прізвище)

Завдання до виконання одержав _____ _____
(підпис) (ініціали, прізвище)

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1 АНАЛІЗ ПІДХОДІВ І МЕТОДІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6

1.1 Основні типи тестування програмного забезпечення 6

1.1.1 Поняття про ручне та автоматизоване тестування. 6

1.1.2 Що необхідно автоматизувати. 7

1.1.3 Коли необхідно автоматизувати процес тестування. 7

1.1.4 Якими засобами автоматизується процес тестування. 8

1.2 Методи тестування програмного забезпечення 9

1.2.1 Тестування «чорної скриньки» (Black Box Testing) 9

1.2.2 Тестування «білої скриньки» (White Box Testing) 10

1.2.3 Тестування «сірої скриньки» (Grey Box Testing) 11

1.3 Рівні тестування програмного забезпечення 13

1.3.1 Функціональне тестування. 13

1.3.2 Нефункціональне тестування 18

1.4 Висновки до першого розділу 23

РОЗДІЛ 2 ІНСТРУМЕНТИ ДЛЯ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ 24

2.1 Selenium WebDriver 24

2.1.1 Коротка характеристика Selenium IDE 24

2.1.2 Коротка характеристика Selenium Remote Control (Selenium RC) 25

2.1.3 Коротка характеристика WebDriver 26

2.1.4 Selenium Grid 27

2.1.5 Короткі висновки по Selenium 27

2.2 TestCafe 28
2.2.1 Переваги TestCafe Studio 29
2.2.2 Порівняння TestCafe і TestCafe Studio 32
2.2.3 Порівняння TestCafe і Selenium 33
2.3 Висновки до другого розділу 34
РОЗДІЛ 3 РОЗРОБКА ПРОТОТИПІВ ТЕСТОВИХ СЦЕНАРІЇВ В СИСТЕМІ TestCafe Studio 36
3.1 Вибір веб-додатків, на основі яких будуть розроблятись прототипи тестових сценаріїв 36
3.2 Розробка тест кейсів для тестування обраних веб-додатків 36
3.2.1 Розробка тест кейсів для тестування першого веб-додатку 37
3.2.2 Розробка тест кейсів для тестування другого веб-додатку 38
3.3 Програмна реалізація розроблених тест кейсів 39
3.3.1 Опис основних дій, умов, які можна виконувати та перевіряти за допомогою TestCafe Studio 39
3.4 Виконання підготовлених тест кейсів в TestCafe Studio 40
3.5 Виконання регресійного тестування для другого веб-додатку 44
3.6 Оцінка проведення ручного та автоматизованого тестування 45
3.7 Висновки до третього розділу 46
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 48
ДОДАТКИ 54

ВСТУП

Як знає більшість людей, задіяних у галузі розробки програмного забезпечення, існують суттєві відмінності між ручним і автоматизованим тестуванням. Тестування вручну вимагає більше часу та зусиль, щоб впевнитись у тому, що тестований додаток виконує всі функції, які на нього покладено.

Актуальність даної роботи випливає з необхідності автоматизації процесу тестування веб-додатків для підвищення якості тестування, а також зменшення трудових і фінансових ресурсів, які на це витрачаються. Стрімке зростання кількості та складності розроблюваних веб-додатків стало поштовхом для появи ресурсів та додатків, які дозволяють автоматизувати та полегшити процес тестування.

Метою даної роботи є скорочення ресурсів, що витрачаються на тестування веб-додатків, і підвищення рівня ефективності такого тестування.

Задачі, поставлені для досягнення мети роботи:

- проведення аналізу існуючих методів тестування веб-додатків;
- аналіз існуючих засобів автоматизації тестування веб-додатків і вибір найкращого із них;
- розробка прототипу тест кейсів для автоматизації тестування веб-додатків на обраних засобах автоматизації;
- визначення швидкості автоматичного виконання розроблених прототипів тест кейсів засобами автоматизації та порівняння її з ручним тестуванням.

Об'єктом дослідження виступає процес тестування веб-додатків.

Предметом дослідження є засоби та методи, що використовуються для автоматизації тестування веб-додатків.

Методи дослідження. В дослідженні використовувалися аналітичні методи, а також алгебраїчні методи для оцінки ефективності отриманих результатів.

Наукова новизна отриманих результатів полягає у тому, що було проведено порівняння найбільш розповсюджених до цього часу засобів автоматизації тестування веб-додатків та нових засобів, які ще не є широко розповсюдженими, але мають переваги та дозволять полегшити та покращити процес автоматизації тестування.

РОЗДІЛ 1 АНАЛІЗ ПІДХОДІВ І МЕТОДІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Основні типи тестування програмного забезпечення

1.1.1 Поняття про ручне та автоматизоване тестування.

Ручне тестування.

Цей тип включає тестування програмного забезпечення вручну, тобто без використання автоматизованого інструменту. У цьому типі тестер бере на себе роль кінцевого користувача та перевіряє програмне забезпечення, щоб виявити будь-яку несподівану поведінку чи помилку. Існують різні етапи ручного тестування, такі як тестування блоків, інтеграційне тестування, системне тестування тощо.

Для перевірки програмного забезпечення тестери використовують тестовий план, тестові кейси або тестові сценарії для забезпечення повноти тестування. Ручне тестування також включає дослідницьке тестування, оскільки тестери досліджують програмне забезпечення для виявлення помилок у ньому.

Автоматизоване тестування.

Автоматизоване тестування – це коли тестувальник пише скрипти та використовує інше програмне забезпечення для тестування програмного забезпечення. Цей процес передбачає автоматизацію ручного процесу. Автоматичне тестування використовується для повторного запуску тестових сценаріїв, які виконувались вручну, швидко та багаторазово [6, 25, 34, 49, 50].

Крім регресійного тестування, автоматизоване тестування також використовується для тестування програми з точки зору навантаження,

продуктивності та стресостійкості. Це збільшує покриття тесту, підвищує точність, економить час та гроші порівняно з ручним тестуванням [24, 29, 31].

1.1.2 Що необхідно автоматизувати.

Неможливо автоматизувати все в тестуванні програмного забезпечення. Однак тестування частин програмного забезпечення, в яких користувач може здійснювати транзакції, такі як форма входу або реєстраційні форми тощо, будь-яка область, де значна кількість користувачів може одночасно отримати доступ до програмного забезпечення, повинно бути автоматизованим.

Крім того, всі елементи графічного інтерфейсу, з’єднання з базами даних, перевірка вимог до введення тексту тощо можуть бути ефективно перевірені шляхом автоматизації процесу тестування [33, 43].

1.1.3 Коли необхідно автоматизувати процес тестування.

Автоматизація тестування програмного забезпечення буде виправданою, якщо наступні твердження будуть справедливі для продукту, який тестується:

- великий та критичний проект;
- проекти, які потребують частого тестування одних і тих же областей;
- вимоги не змінюються часто;
- доступ до програми для навантаження та тестування перфомансу з багатьма віртуальними користувачами;
- стабільне програмне забезпечення в аспекті ручного тестування;
- наявність часу.

1.1.4 Якими засобами автоматизується процес тестування.

Автоматизація здійснюється за допомогою підтримуючої комп’ютерної мови, наприклад Java Script, та автоматизованого програмного забезпечення. Існує маса інструментів, які можна використовувати для написання сценаріїв автоматизації [17, 23, 30]. Перш ніж говорити про інструменти, слід зазначити процеси, які можна використовувати для автоматизації тестування:

- визначення областей програмного забезпечення для автоматизації;
- підбір відповідного інструменту для автоматизації тестування;
- написання тестових сценаріїв;
- розробка тестових скриптів;
- виконання сценаріїв;
- створення звіти про результати;
- визначення потенційних помилок чи проблем з продуктивністю програмного забезпечення.

Нижче наведено інструменти, які можна використовувати для автоматизації тестування:

- TestCafe
- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LaodRunner
- Visual Studio Test Professional
- WATIR

1.2 Методи тестування програмного забезпечення

1.2.1 Тестування «чорної скриньки» (Black Box Testing)

Техніка тестування, коли невідомо як система працює всередині, – це тестування Black Box. Тестувальник не звертає уваги на системну архітектуру та не має доступу до програмного коду. Як правило, під час такого підходу до тестування тестер взаємодіє з користувальницьким інтерфейсом системи, надаючи вхідні дані та вивчаючи вихідні, не знаючи, як і де дані були оброблені.

Переваги методу:

- добре підходить та ефективний для програмних додатків з великим обсягом коду;
- не потрібен доступ до коду;
- чітко відокремлює точку зору користувача від точки зору розробника через чітко визначені ролі для роботи з додатком;
- велика кількість добре кваліфікованих тестувальників може перевірити додаток, не маючи знань щодо розробки, мови програмування чи операційних систем, які використовувалися.

Недоліки методу:

- обмежене покриття тестових випадків, оскільки фактично виконується лише вибрана кількість тестових сценаріїв;
- неефективне тестування пов’язане з тим, що тестер має обмежені знання про програму;
- покриття тестування функціоналу наосліп, оскільки тестер не може орієнтуватися на конкретні сегменти коду або області, схильні до помилок;
- тест-кейси складно спроектувати.

1.2.2 Тестування «білої скриньки» (White Box Testing)

Тестування білої скриньки – це детальне дослідження внутрішньої логіки та структури коду. Тестування білої скриньки також називається тестуванням скла або тестуванням відкритої коробки. Для того щоб виконати таке тестування поля в додатку, тестеру необхідно володіти знаннями про внутрішню роботу коду. Тестувальник повинен мати доступ до програмного коду та з'ясувати, який сектор коду може призводити до помилки.

Оскільки тестер володіє знаннями програмного коду, стає дуже легко дізнатися, який тип даних може допомогти для ефективного тестування програми.

Переваги методу:

- допомагає в оптимізації коду;
- зайві рядки коду можуть бути видалені, що може призвести до прихованих дефектів;
- завдяки знанням тестера програмного коду, можна досягнути максимального покриття під час написання тест-кейсів.

Недоліки методу:

- через те, що для тестування білої скриньки потрібен висококваліфікований тестер, витрати збільшуються;
- іноді неможливо зазирнути в кожен закуток коду, щоб виявити приховані помилки, які можуть створити проблеми, оскільки багато випадків буде не перевірено;
- складно підтримувати тестування білої скриньки, оскільки необхідне використання спеціалізованих інструментів, таких як аналізатори коду та інструменти для налагодження коду (debugging tools).

1.2.3 Тестування «сірої скриньки» (Grey Box Testing)

Тестування «сірої скриньки» – це методика тестування програми з обмеженими знаннями про внутрішню роботу та будову програми. В тестуванні програмного забезпечення термін "чим більше ви знаєте, тим краще" набуває великого значення в процесі виконання тесту.

Управління основою системи завжди дає тестувальнику перевагу над будь-ким із обмеженими знаннями середовища. На відміну від тестування "чорної скриньки", коли тестер тестує лише користувальницький інтерфейс програми, у тесті "сірої скриньки" тестер має доступ до проектних документів та бази даних. Маючи ці знання, тестувальник може підготувати тест-дату та тестові сценарії при складанні плану тестування.

Переваги методу:

- там, де це можливо, об’єднуються переваги тестування "чорної" та "білої" скриньок;
- тестери "сірої скриньки" не покладаються на вихідний код; натомість вони покладаються на визначення інтерфейсу та функціональні характеристики, зафіксовані в документації;
- на основі обмеженої інформації, яка доступна, тестер може створити чудові тестові сценарії, особливо які стосуються протоколів зв’язку та обробки даних;
- тест робиться з точки погляду на систему користувача, а не дизайнера.

Недоліки методу:

- оскільки доступ до вихідного коду відсутній, то можливість перегляду коду та тестового покриття обмежена;
- тести можуть бути зайвими, якщо розробник програмного забезпечення вже запустив виконання тестового випадку;

- тестування всіх можливих вхідних даних неможливе, оскільки це займе необґрунтовану кількість часу, тому деякий програмний функціонал залишаться неперевіреним.

Таблиця 1.1 – Порівняння методів тестування

	Black Box Testing	Grey Box Testing	White Box Testing
1	Внутрішня робота програми не повинна бути відомою	Дещо відомо про внутрішню будову	Тестер повністю володіє знаннями про внутрішню будову додатку
2	Також відомий як тестування закритого типу, тестування на основі даних та функціональне тестування	Іншим терміном є напівпрозорі випробування, оскільки тестер має обмежені знання про внутрішню будову програми	Також відомий як прозоре тестування, структурне тестування або тестування на основі коду
3	Виконується кінцевими користувачами, а також тестерами та розробниками	Виконується кінцевими користувачами, а також тестерами та розробниками	Зазвичай виконується тестерами та розробниками
4	Тестування базується на зовнішніх очікуваннях, внутрішня поведінка програми невідома	Тестування проводиться на основі діаграм баз даних високого рівня та діаграм потоку даних	Внутрішня будова системи повністю відома, і тестер може відповідно розробляти сценарії випробувань
5	Це найменш трудомісткий і вичерпний	Частково трудомісткий та вичерпний	Найбільш вичерпний і трудомісткий вид тестування
6	Не підходить для тестування по алгоритму	Не підходить для тестування по алгоритму	Підходить для тестування алгоритму

Продовження таблиці 1.1

7	Можна зробити лише методом проб і помилок	Будову системи та внутрішні дані можна перевірити, якщо вони відомі	Будову системи та внутрішні дані можна краще перевірити

1.3 Рівні тестування програмного забезпечення

Можна виділити два основні рівні тестування програмного забезпечення: функціональне та нефункціональне.

1.3.1 Функціональне тестування.

Це тип тестування, який базується на специфікаціях програмного забезпечення, яке підлягає тестуванню. Додаток тестується шляхом введення вхідних даних, а потім вивчаються результати, які повинні відповідати функціоналу, для якого він призначений. Функціональне тестування програмного забезпечення проводиться на повністю інтегрованій системі для оцінки відповідності системи її визначеним вимогам. Існує п'ять кроків, які виконуються при тестуванні програми на функціональність:

- крок I - визначення функцій, які повинна виконувати система;
- крок II - створення тестових даних на основі специфікацій програми;
- крок III - висновок заснований на тестових даних та специфікаціях програми;
- крок IV - написання тестових сценаріїв та виконання тест кейсів;

- крок V - порівняння фактичних та очікуваних результатів на основі виконаних тест кейсів.

Практика ефективного тестування дозволяє побачити вищезазначені кроки, застосовані до політики тестування в різних організаціях, а отже, це дозволяє дотримуватись найсуворіших стандартів щодо якості програмного забезпечення.

1.3.1.1 Модульне тестування (Unit Testing)

Цей тип тестування проводиться розробниками до того, як білд буде переданий команді тестувальників для офіційного виконання тест кейсів. Модульне тестування проводиться відповідними розробникам. Розробники використовують тестові дані, окремі від тестових даних групи з забезпечення якості (quality assurance team).

Мета модульного тестування - виділити кожну частину програми та показати, що окремі частини працюють правильно з точки зору вимог та функціональності.

Обмеження модульного тестування.

Модульне тестування не може знайти кожну помилку програми. Неможливо оцінити кожен шлях виконання у кожному програмному застосуванні. Те саме стосується одиничного тестування. Існує обмеження на кількість сценаріїв і тестових даних, які розробник може використовувати для перевірки програмного коду. Тож після того, як він вичерпав усі варіанти, не залишається іншого вибору, як зупинити тестування модуля та об'єднати сегмент коду з іншими.

1.3.1.2 Інтеграційне тестування (Integration Testing)

Тестування комбінованих частин програми, щоб визначити, чи правильно вони працюють разом, називається інтеграційним тестуванням. Також перевіряється взаємодії додатку з зовнішніми системами. Існує два методи проведення тестування інтеграції: тестування знизу вгору та тестування інтеграції зверху вниз.

Тестування інтеграції знизу вгору починається з тестування окремих одиниць, після чого поступово випробовуються поєднання вищих рівнів одиниць, що називаються модулями або білдами.

В тестуванні інтеграції зверху вниз, модулі вищого рівня перевіряються спочатку, а модулі нижчого рівня тестуються після цього. У всеосяжному середовищі розробки програмного забезпечення зазвичай спочатку проводять тестування знизу вгору, після чого - тестування зверху вниз.

1.3.1.3 Системне тестування (System Testing)

Це наступний рівень тестування - тестування системи в цілому. Після інтеграції всіх компонентів додаток у цілому ретельно перевіряється на предмет того, що він відповідає стандартам якості. Цей тип тестування проводиться спеціалізованою групою тестування.

Чому системне тестування є важливим:

- тестування системи - це перший крок у життєвому циклі розробки програмного забезпечення, де програма тестується в цілому;
- додаток тестується ретельно, щоб переконатися, що він відповідає функціональним та технічним характеристикам;

- додаток тестується в середовищі, що дуже близьке до виробничого середовища, де програма буде розміщена;
- тестування системи дозволяє нам перевірити, перевірити та підтвердити як бізнес-вимоги, так і архітектуру програм.

1.3.1.4 Регресійне тестування (Regression Testing)

Щоразу, коли відбувається зміна в певній області коду програмного додатка, це може вплинути й на інші області коду в програмі. Щоб перевірити, що виправлена помилка не призвела до порушення інших функціональних можливостей або правил, проводиться регресійне тестування. Його метою є забезпечити впевненість, що зміни, такі як виправлення помилок, не призвели до появи іншої помилки в додатку.

Чому регресійне тестування є важливим:

- мінімізує прогалини в тестуванні, коли додаток із внесеними змінами має бути протестовано;
- тестування змін в додатку, щоб переконатися, що внесені зміни не вплинули на будь-яку іншу область програми;
- пом'якшує ризики, коли регрес-тестування проводиться в додатку;
- покриття тесту збільшується без шкоди для термінів;
- збільшення швидкості видачі продукту.

1.3.1.5 Приймальне тестування (Acceptance Testing)

Це, мабуть, найважливіший вид тестування, оскільки він проводиться командою із забезпечення якості, яка перевіряє, чи відповідає програма

заявленим специфікаціям та чи відповідає вимогам клієнта. Команда QA матиме набір попередньо написаних сценаріїв та тестових випадків, які будуть використані для тестування програми.

Приймальне тестування призначене не лише для того, щоб вказати на прості орфографічні помилки, косметичні помилки або прогалини в інтерфейсі, але також вказати на будь-які помилки в додатку, які призведуть до збоїв у роботі системи або великих помилок у програмі.

Виконуючи приймальні тести на системі, тестувальна група визначить, як буде працювати програма під час постійного використання. Існують також юридичні та договірні вимоги щодо прийняття системи в роботу [59].

Альфа-тестування.

Цей тест є першим етапом тестування і проводитиметься серед команд (розробників та команд із забезпечення якості). Тестування модулів, тестування інтеграції та тестування системи у поєднанні відомі як альфа-тестування. Під час цієї фази в додатку буде перевірено наступне:

- орфографічні помилки;
- несправні посилання;
- додаток буде протестовано на машинах з найнижчою специфікацією для перевірки часу завантаження та будь-яких проблем із затримкою.

Бета-тестування.

Цей тест виконується після успішного альфа-тестування. При бета-тестуванні частина ймовірних користувачів тестує додаток. Бета-тестування також відоме як тестування перед випуском. Бета-тестові версії програмного забезпечення інколи розподіляються серед широкої аудиторії в Інтернеті, частково для того, щоб виконати тест в реальних умовах. Якщо розроблюване програмне забезпечення призначене для внутрішнього користування в певній компанії, то для тестування залучають співробітників замовника. На цьому етапі аудиторія буде тестувати наступне:

- користувачі встановлюють, запускають додаток та надсилають свої відгуки команді проекту;

- типографічні помилки, заплутаність потоку додатків і навіть збої;
- отримуючи зворотний зв'язок, команда проекту може вирішити проблеми перед тим, як випустити програмне забезпечення фактичним користувачам;
- чим більше виправлених проблем, які вирішують реальні проблеми користувачів, тим вище буде якість вашої програми;
- наявність більш якісної програми під час випуску для широкої громадськості підвищить задоволеність клієнтів.

1.3.2 Нефункціональне тестування

Цей вид заснований на тестуванні програми на основі її нефункціональних атрибутів. Нефункціональне тестування програмного забезпечення передбачає тестування програмного додатку на основі вимог, які не є функціональними за своєю суттю, але також важливі, такі як продуктивність, безпека, користувальницький інтерфейс тощо. Деякі з важливих і часто використовуваних нефункціональних типів тестування описуються далі [66].

1.3.2.1 Тестування продуктивності

Тестування продуктивності здебільшого використовується для виявлення будь-яких вузьких місць або проблем з продуктивністю, а не для пошуку помилок у програмному забезпеченні. Існують різні причини, які сприяють зниженню продуктивності програмного забезпечення:

- затримки мережі;

- обробка на стороні клієнта;
- обробка транзакцій в базі даних;
- балансування навантаження між серверами;
- відображення даних.

Тестування ефективності розглядається як один із важливих та обов'язкових типів тестування з точки зору наступних аспектів:

- швидкість (тобто час відгуку, надання та доступ до даних);
- ємність;
- стабільність;
- масштабованість.

Це може бути або якісна, або кількісна перевірка, і її можна розділити на різні під види, такі як навантажувальне тестування (Load testing) та стрес-тестування (Stress testing) [65, 68].

1.3.2.1.1 Навантажувальне тестування

Процес тестування поведінки програмного забезпечення шляхом застосування максимального навантаження з точки зору доступу до програмного забезпечення та маніпулювання великими вхідними даними. Це можна зробити як в нормальних, так і в пікових умовах навантаження. Цей тип тестування визначає максимальну потужність програмного забезпечення та його поведінку у піковий час.

Більшу частину часу тестування навантаження проводиться за допомогою автоматизованих інструментів, таких як Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test тощо [48].

Віртуальні користувачі (VUsers) визначені в автоматизованому інструменті тестування і сценарій виконується для перевірки тестування завантаження для програмного забезпечення. Кількість користувачів може збільшуватися або зменшуватися одночасно або поступово, виходячи з вимог.

1.3.2.1.2 Стрес-тестування

Цей тип тестування включає тестування поведінки програмного забезпечення в ненормальних умовах.

Стрес-тестування відбирає ресурси, застосовує навантаження понад фактичну межу навантаження.

Основна мета - протестувати програмне забезпечення, застосовуючи навантаження на систему та переймаючи ресурси, що використовуються програмним додатком для визначення точки зламу. Це тестування може бути проведено шляхом тестування різних сценаріїв, таких як:

- вимкнення або перезапуск мережевих портів випадковим чином;
- увімкнення та вимкнення бази даних;
- запуск різних процесів, що споживають такі ресурси, як процесор, пам'ять, сервер тощо.

1.3.2.2 Тестування зручності використання (Usability Testing)

Цей вид тестування включає різні поняття та визначення тестування зручності використання з точки зору програмного забезпечення. Це чорна скринька і використовується для виявлення будь-яких помилок та

удосконалень у Програмному забезпеченні шляхом спостереження за користувачами за їх використанням та експлуатацією.

Зручна для користувачів система повинна виконувати наступні п’ять цілей:

- проста для вивчення;
- проста для запам’ятовування;
- ефективна у використанні;
- задовільна у використанні;
- просту для розуміння.

Окрім різних визначень зручності користування, існують деякі стандарти та моделі якості та методи, які визначають зручність використання у вигляді атрибутів та допоміжних атрибутів, таких як ISO-9126, ISO-9241-11, ISO-13407 та IEEE std.610.12 тощо.

Тестування інтерфейсу користувача включає тестування графічного інтерфейсу користувача програмного забезпечення. Це тестування гарантує, що графічний інтерфейс повинен відповідати вимогам щодо кольору, вирівнювання, розміру та інших властивостей.

З іншого боку, тестування зручності використання забезпечує те, що хороший та зручний користувальницький графічний інтерфейс розроблений та простий у використанні для кінцевого користувача [37]. Тестування користувальницького інтерфейсу можна розглядати як підрозділ тестування зручності використання.

1.3.2.3 Тестування безпеки (Security Testing)

Тестування безпеки включає тестування програмного забезпечення з метою виявлення будь-яких недоліків та вразливих місць з точки зору безпеки

та вразливості [18, 23, 32]. Нижче наведено основні аспекти, які перевіряються під час тестування безпеки:

- конфіденційність;
- цілісність;
- аутентифікація;
- доступність;
- авторизація;
- програмне забезпечення захищене від відомих і невідомих загроз;
- дані програмного забезпечення захищені;
- програмне забезпечення відповідає всім нормам безпеки;
- перевірка та валідація вхідних даних.

1.3.2.4 Тестування на портативність (Portability Testing)

Тестування портативності включає тестування програмного забезпечення з наміром, що воно повинно бути повторно використане, а також може бути переміщене з іншого пристрою. Далі наведені стратегії, які можна використовувати для тестування на портативність:

- перенесено встановлене програмне забезпечення з одного комп'ютера на інший;
- створення виконавчого файлу (.exe) для запуску програмного забезпечення на різних платформах.

Тестування на портативність може розглядатися як один з підрозділів системного тестування, оскільки цей тип тестування включає загальне тестування програмного забезпечення щодо його використання в різних середовищах [14, 17]. Основні напрямки тестування на портативність - це комп'ютерне обладнання, операційні системи та браузери. Нижче наведено деякі передумови для тестування на портативність:

- програмне забезпечення повинно бути розроблено та закодовано, враховуючи вимоги до портативності;
- модульне тестування було проведено на пов'язаних компонентах;
- проведено інтеграційне тестування;
- тестове середовище створено.

1.4 Висновки до першого розділу

В першому розділі магістерської роботи було проаналізовано основні методи та підходи з тестування програмного забезпечення.

Були розглянуті поняття ручного та автоматизованого тестування. Були вказані ситуації, коли необхідно вводити автоматизоване тестування та засоби, за допомогою яких це проводиться.

Проаналізовано переваги та недоліки різних підходів до тестування програмного забезпечення.

Можна сказати, що автоматизація тестування програмного забезпечення буде виправданою, якщо хоча б частково будуть виконуватись зазначені умови. В такому випадку автоматизація тестування дозволить значно знизити витрати на тестування, а відповідно й розробку програмного продукту. Також це дозволить покращити якість програмного забезпечення, що розробляється або підтримується. Під час регресійного тестування це може значно зменшити час на його виконання й аналіз виявлених помилок, якщо такі будуть.

РОЗДІЛ 2 ІНСТРУМЕНТИ ДЛЯ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

2.1 Selenium WebDriver

Selenium - це інструмент з відкритим кодом, який використовується для автоматизації тестів, проведених у веб-браузерах.

Selenium складається з чотирьох компонент:

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid

На даний момент Selenium RC і WebDriver об'єднані в єдину систему, щоб утворити Selenium 2.

2.1.1 Коротка характеристика Selenium IDE

Selenium Integrated Development Environment (IDE) - це найпростіший фреймворк в наборі Selenium і є найлегшим для вивчення. Це плагін Firefox, який можна встановити так само просто, як і інші плагіни. Однак через свою простоту, Selenium IDE слід використовувати лише як інструмент прототипування. Якщо треба створити більш досконалі автоматизовані тести, потрібно буде використовувати або Selenium RC, або WebDriver.

Переваги Selenium IDE:

- дуже простий та зрозумілий
- не потрібно мати навичок з програмування, лиже мінімальні знання з HTML

- можна завантажені підготовлені тести і потім використати в Selenium RC та WebDriver
- підтримує розширення

Недоліки Selenium IDE:

- доступний лише на браузері Firefox
- розроблений лише для розробки прототипів тестів
- не підтримує ітерації та кроки, для виконання яких необхідні певні умови
- виконання тесту повільне в порівнянні з Selenium RC та WebDriver

2.1.2 Коротка характеристика Selenium Remote Control (Selenium RC)

Selenium RC тривалий час був основною тестування всього проекту Selenium. Це перший автоматизований інструмент веб-тестування, який дозволив користувачам використовувати мову програмування, яку вони віддають перевагу. З версії 2.25.0 RC може підтримувати наступні мови програмування:

- Java
- C#
- PHP
- Python
- Perl
- Ruby

Переваги:

- кросбраузерність і кросплатформеність
- може виконувати зациклені операції
- підтримує тестування на основі даних
- має завершену API

- швидший у виконанні тестів ніж IDE

Недоліки:

- встановити складніше ніж IDE
- необхідні навички з програмування
- необхідний запущений Selenium RC сервер
- API має надлишкові та іноді спантеличуючі команди
- взаємодія з браузерами менш реалістична
- час виконання тестів нижче ніж у WebDriver

2.1.3 Коротка характеристика WebDriver

WebDriver є кращим, ніж Selenium IDE і Selenium RC у багатьох аспектах. Він реалізує більш сучасний і стабільний підхід в автоматизації тестів в браузері. WebDriver, на відміну від Selenium RC, не покладається на JavaScript для автоматизації. Він контролює браузер, безпосередньо спілкуючись з ним.

Мови, що підтримуються, такі ж, як і в Selenium RC:

- Java
- C#
- PHP
- Python
- Perl
- Ruby

Переваги WebDriver:

- простіший в установці ніж Selenium RC
- посилає команди безпосередньо браузеру
- взаємодія з браузерами більш реалістична
- не потребує окремих компонент, таких як RC Server

- швидше виконання тестів ніж в IDE і RC

Недоліки WebDriver:

- встановити складніше ніж Selenium IDE
- потребує знань в програмуванні
- не може відразу підтримувати нові браузери
- не має вбудованого механізму для генерації результатів тестування

2.1.4 Selenium Grid

Selenium Grid - це інструмент, який використовується разом із Selenium RC для одночасного запуску паралельних тестів на різних машинах та різних браузерах. Паралельне виконання означає виконання декількох тестів одночасно.

Особливості:

- дозволяє одночасне виконання тестів у кількох браузерах та середовищах;
- економить час.

Використовує концепцію концентраторів і вузлів. Концентратор виступає центральним джерелом команд Selenium до кожного підключеного до нього вузла.

2.1.5 Короткі висновки по Selenium

Весь набір інструментів Selenium складається з чотирьох компонентів:

- Selenium IDE - додаток для Firefox, який ви можете використовувати лише для створення порівняно простих тестових випадків та тестових наборів.
- Selenium Remote Control, також відомий як Selenium 1, це перший інструмент Selenium, який дозволив користувачам використовувати мови програмування при створенні складних тестів.
- WebDriver - новіший прорив, який дозволяє тестовим сценаріям спілкуватися безпосередньо з браузером, тим самим контролюючи його на рівні ОС.
- Selenium Grid - це також інструмент, який використовується із Selenium RC для виконання паралельних тестів у різних браузерах та операційних системах.

Selenium RC і WebDriver були об'єднані, щоб утворити Selenium 2.

На відміну від попередників (наприклад HP Quick Test Pro), Selenium більш вигідний. Він одним з перших дозволив запускати декілька тестів одночасно.

2.2 TestCafe

TestCafe - це чистий інструмент node.js для автоматизації тестування веб-додатків. За допомогою TestCafe можна писати тести на JavaScript та TypeScript, що цілком логічно, враховуючи, що інструмент, який використовується для тестування в Інтернеті, повинен дозволяти писати тести за допомогою однієї з основних технологій Інтернету (JavaScript).

TestCafe - це безкоштовний інструмент із відкритим кодом. TestCafe не потребує плагінів браузера, він використовує проксі-сервер для перезапису URL-адреси під назвою Hammerhead, який емулює команди за допомогою API

DOM та вводить JavaScript у браузер. Цей проксі переписує всі властивості відповідних об’єктів JavaScript, які містять значення URL для імітації дій користувача в тестованій програмі. Це дозволяє йому підтримувати кілька браузерів без будь-яких драйверів браузера.

TestCafe Studio розроблений з нуля для сучасного веб-тестування та браузерів і не покладається на Selenium або інші застарілі тестові платформи.

Для початківців та тих, хто не має досвіду кодування, TestCafe Studio усуває необхідність вручну генерувати тестові сценарії. Для досвідчених інженерів із забезпечення якості та розробників програмного забезпечення TestCafe Studio включає інструменти, призначені для підвищення продуктивності та скорочення часу, необхідного для тестування складних веб-додатків.

Як відомо, ринок веб-тестувань включає в себе безліч фреймворків, розроблених для імітації активності користувачів та автоматизації end-to-end тестування. На жаль, більшість цих інструментів вимагають від користувачів запису великих об’ємів коду та розшифрування складних конфігурацій.

Щоб допомогти вирішити притаманні недоліки існуючих фреймворків та дозволити витрачати менше часу на написання/управління тестовими сценаріями, TestCafe Studio має різні корисні функції.

2.2.1 Переваги TestCafe Studio

1. Повністю інтегрований візуальний рекордер веб-тестів.

Візуальний тестовий рекордер TestCafe Studio спостерігає за діями користувача під час навігації/взаємодії з активними елементами веб-сторінки та автоматично генерує етапи тестування. Без додаткової обробки ці записані

кроки можна використовувати для швидкого тестування веб-додатку у різних браузерах настільних, мобільних та хмарних платформ.

На відміну від "псевдовізуальних тестових рекордерів", які генерують неправильні селектори, які не можуть надійно відтворити візуально записані тести, візуальний тестовий рекордер TestCafe Studio був створений для отримання послідовних результатів із надійним тестовим відтворенням.

2. Автоматично створені селектори елементів.

Побудова селекторів елементів сторінки, які використовуються для взаємодії з сторінкою веб-додатку, трудомістка. Також є велика ймовірність допустити помилки.

На відміну від інших інструментів, TestCafe Studio автоматично генерує селектори для кожного елемента сторінки в рамках створюваного тесту. Він підтримує всі основні елементи HTML.

Для покриття найширшого можливого сценарію тестування, TestCafe Studio генерує оптимальний селектор елементів і набір альтернатив. Якщо потрібно натиснути кнопку, можна вибрати її в тесті, використовуючи її підписи, як це робив би користувач. Якщо, однак, мета перевірити, чи відображається правильний підпис, можна використовувати ідентифікатор або ім’я класу для ідентифікації кнопки або іншого елементу сторінки.

Звичайно, можна вручну редагувати автоматично створені селектори TestCafe в залежності від потреб. Можна навіть створити селектори з нуля - конструктор TestCafe Studio’s Selector Constructor був розроблений, щоб передати контроль у ваші руки. Це є дуже корисною функцією, адже розширює можливості для створення ефективних тестових сценаріїв, а також використання написаних тестів для одного додатку в тестуванні іншого зробивши мінімальні правки в коді тесту.

3. Крос-платформенні та крос-браузерні тести.

TestCafe Studio - це кросплатформенний додаток, який працює на macOS, Windows та Linux.

Тестові файли/сценарії є платформно-агностичними: незалежно від того, де вони були записані, тести можуть виконуватися на різних платформах ОС, пристроях (настільних/мобільних), хмарних сервісах тестування (наприклад, BrowserStack або SauceLabs) та системах постійної інтеграції (в т.ч. CircleCI, Bitbucket, Azure, TeamCity, Jenkins, Travis, GitLab тощо).

Платформа і агностичний підхід до веб-тестування дозволяють швидко знаходити помилки, пов’язані з платформою та браузером, і гарантують, що тестований веб-додаток забезпечує однакові функції користувачам різних платформ/операційних систем.

4. Не потрібні додаткові інструменти для обслуговування.

TestCafe Studio не потребує Selenium/WebDriver, або інших плагінів веб-переглядача чи сторонніх додатків. Однією з безпосередніх переваг цієї автономної архітектури є ефективність.

За допомогою TestCafe Studio можна розпочати тестування, як тільки продукт буде встановлений. Немає потреби шукати драйвери та плагіни.

Можливо, важливішою є незалежність системи TestCafe. Коли вийде нова основна версія браузера, який використовувався для тестування, можна бути впевненим, що існуючі веб-тести не вийдуть з ладу через відсутність плагінів або розширень.

5. Вбудований механізм очікування.

Тести веб-програмного забезпечення мають асинхронний характер. Дії користувача рідко дають миттєві результати, особливо якщо ці дії супроводжуються зворотними переходами на сервер або інтегрованими ефектами анімації [4].

В майже у всіх існуючих системах для тестування веб-додатків доводиться вручну прописувати в коді спеціальні елементи, які будуть змушувати виконання тесту призупинитись на певний час. Або ж доводиться зупиняти тест вручну, що спричиняє затримки та нераціональне використання ресурсів.

TestCafe Studio по іншому обробляє асинхронні операції. Перед кожною дією чи твердженням система автоматично чекає, коли цільовий елемент стане доступним (завантажений, видимий, не затемнений іншим елементом тощо).

2.2.2 Порівняння TestCafe і TestCafe Studio

Як було вказано раніше, TestCafe Studio має графічний інтерфейс з більш широким спектром доступних функцій, ніж просто TestCafe. Надалі буде проведено їх порівняння.

Таблиця 2.1 – Порівняння TestCafe і TestCafe Studio

	TestCafe	TestCafe Studio
Немає необхідності у WebDriver, плагінах браузера та інших інструментах	+	+
Крос-платформеність та крос-браузерність наявні за замовчуванням	+	+
Можливість написання тестів на JavaScript або TypeScript	+	+
Стабільні тести завдяки механізму інтелектуального запиту	+	+
Тести проходять швидко завдяки механізму автоматичного очікування та одночасному виконанню тестів	+	+
Користувацькі плагіни для тестувальника	+	+
Можливість використання сторонніх модулів Node.js у тестових сценаріях	+	+
Інтеграція з популярними CI системами	+	+
Безкоштовна та з відкритим кодом	+	
Механізм візуального запису тестів		+
Інтерактивний редактор тестів		+

Продовження таблиці 2.1

Генерація автоматичного селектора		+
Запуск менеджера конфігурацій		+
Графічний інтерфейс, схожий на IDE		+

З таблиці порівнянь очевидно, що TestCafe Studio включає майже всі особливості та переваги TestCafe за вийнятком того, що вона є платною. Але є можливість використання 30ти-денної безкоштовної версії. Для тестування можливостей додатку, а також дослідження можливостей автоматизації тестування веб-додатків цього буде достатньо. Для постійного використання, звісно, буде необхідно придбати платну підписку.

2.2.3 Порівняння TestCafe і Selenium

Selenium простий, а також легкий для вивчення. Він існує вже досить давно і має прихильників у галузі QA.

TestCafe є новим інструментом автоматизації, але має розширені функції.

TestCafe здатний тестувати мобільні додатки, тоді як Selenium потребує підтримки від Selendroid або Appium для тестування мобільних додатків.

Таблиця 2.2 – Порівняння TestCafe Studio і Selenium

Особливість	Selenium	TestCafe
Гнучке ліцензування	З відкритим кодом та доступний у вільному доступі.	Доступний як платна, так і безкоштовна версії

Продовження таблиці 2.2

Плагіни	У ньому є кілька плагінів, які можуть бути, а можуть і не бути доступними у вільному доступі. Плагіни потрібні в Selenium з усіма пов'язаними проблемами.	Виключає використання плагінів, і користувач може легко протестувати додатки в будь-якому HTML-браузері, настільному ПК або мобільному пристрої.
Зрозумілість і простота вивчення	Користувачам, які вже володіють мовою програмування, це може бути легко. Більшість користувачів, які використовують Selenium, є професіоналами.	Його API одночасно зрозумілий і простий у використанні. Користувач може витрачати менше часу на запам'ятовування та більше часу на тестування .
100% веб-тестове середовище	Його основна мета - протестувати веб-додаток, і він базується на середовищі веб-тесту.	Також базується на середовищі веб-тестування, все - від запису до виконання та аналізу - на веб-основі.
Спільнота користувачів	Має велику спільноту користувачів, яка може забезпечити будь-яке рішення будь-якої проблеми.	Оскільки це новий інструмент, у нього немає великої спільноти.

Можна побачити, що і Selenium має переваги в деяких аспектах. Але все ж лідером є TestCafe.

2.3 Висновки до другого розділу

В другому розділі магістерської роботи було проведено огляд існуючих засобів для автоматизації тестування веб-додатків. Було розглянуто Selenium, як один з засобів, що був розроблений та використовується достатньо давно, а також TestCafe – більш інноваційне рішення. Кожне з рішень має свої сильні

та слабкі сторони, але так як TestCafe є більш новим, в ньому були частково вирішені проблеми, які притаманні для Selenium. Також TestCafe є простішим, що дозволяє зберегти час тестувальника не втрачаючи ефективності тесту. Що, відповідно, дозволяє зберегти ресурси компанії.

Одним з найважливіших аспектів, які можна виділити це те, що в TestCafe була вирішена проблема механізму очікування. Обробка асинхронних операцій іншим чином дозволила підвищити ефективність тестування.

Ще одним важливим моментом є те, що використовуючи TestCafe можна значно скоротити витрати часу на написання тестів. Завдяки механізму візуального запису тестів можна створити бажаний тест та відредагувати його за допомогою редактора. Потім створений та відредагований тест можна буде використовувати для інших тестових потреб зробивши мінімальні правки.

РОЗДІЛ 3 РОЗРОБКА ПРОТОТИПІВ ТЕСТОВИХ СЦЕНАРІЇВ В СИСТЕМІ TestCafe Studio

3.1 Вибір веб-додатків, на основі яких будуть розроблятись прототипи тестових сценаріїв

Для репрезентативності результатів оцінки ефективності використання програмного забезпечення для автоматизації тестування було обрано 2 веб-додатки електронної комерції:

1) інтернет-магазин Rozetka https://rozetka.com.ua/
2) проект, створений на платформі Tilda: http://upgradepc.tilda.ws/

Перший веб-додаток був обраний з наступних причин:

- стабільність роботи;
- наповненість веб-додатку різними елементами;
- доступ до об’ємного каталогу продукції.

Другий веб-ресурс був обраний з наступних причин:

- відкритий доступ до ресурсу;
- можливість вносити зміни до коду власного проекту.

Те, що другий веб-ресурс дозволяє вносити зміни в код тестованої сторінки дозволить перевірити, як на ці зміни відреагує запуск тесту.

3.2 Розробка тест кейсів для тестування обраних веб-додатків

Тест кейс – це набір дій, що виконуються для перевірки певної функції або функціональності тестованої програми. Тест кейс містить етапи тестування, дані тестування, попередні умови, післяумови, розроблені для

конкретного сценарію тестування для перевірки будь-яких вимог. Тестовий випадок включає конкретні змінні чи умови, за допомогою яких тестувальник може порівняти очікувані та фактичні результати, щоб визначити, чи функціонує програмний продукт правильно й у відповідності до вимог замовника.

3.2.1 Розробка тест кейсів для тестування першого веб-додатку

Для першого веб-додатку було описано п’ять тест кейсів які будуть виконані для перевірки базових функції сайту.

Таблиця 3.1 – Тест кейси для тестування першого веб-додатку

№ кейсу	Що перевіряється	Опис необхідних кроків
1	Можливість змінити мову інтерфейсу	Змінити мову сайту на українську
2	Каталог товарів	Змінити мову сайту на українську, відкрити каталог товарів та знавігуватися до категорії
3	Сортування товарів	Перейти до категорії та відсортувати товари за популярністю
4	Фільтрація товарів	Знавігуватися до категорії товарів та з допомогою бокового меню виконати фільрацію товарів
5	Навігація на сторінці	Знайти товар та перевірити навігацію на сторінці опису

3.2.2 Розробка тест кейсів для тестування другого веб-додатку

Для першого веб-додатку було описано п’ять тест кейсів які будуть виконані для перевірки базових функцій сайту.

Таблиця 3.2 – Тест кейси для тестування другого веб-додатку

№ кейсу	Що перевіряється	Опис необхідних кроків
1	Можливість змінити мову інтерфейсу	Змінити мову сайту на англійську
2	Оформити замовлення	Оформити замовлення на сайті
3	Оформити замовлення використовуючи навігацію сайту	Оформити замовлення на сайті використовуючи навігацію
4	Оформити замовлення на англомовній версії сайту	Змінити мову сайту на англійську та оформити замовлення
5	Перевірити контент на сайті	Перевірити текст над слоганом сайту

Тестові дані, які будуть використані для оформлення замовлень:

ім’я – Сергій;

електронна пошта - hollow3809@gmail.com;

номер телефону: +380997256452.

3.3 Програмна реалізація розроблених тест кейсів

3.3.1 Опис основних дій, умов, які можна виконувати та перевіряти за допомогою TestCafe Studio

Основні дії, які можуть виконуватися при виконанні тесту:

1) *Click* – натиснення на елемент сторінки.

Приклад:

.click(Selector('.menu-language__item').nth(1).find('a').withText('UA')).

2) *Double Click* – подвійний натиск на елемент сторінки.

Приклад:

.doubleClick('#dyplom').

3) *Hover* – розташувати курсор над елементом веб-сторінки.

Приклад:

.hover(Selector('li').withText('Смартфоны, ТВ и электроника').find('.menu-categories__link.js-menu-categories__link'))

4) *Type Text* – введення тексту у вхідний елемент.

Приклад:

.typeText(Selector('#form72296172').find('[name="Email"]'), 'hollow3809@gmail.com');

.typeText(Selector('#form72296172').find('[name="Phone"]'), '+380997256452').

5) *Press Key* – натиснення вказаних клавіш клавіатури.

Приклад:

.pressKey('tab');

.pressKey('shift+alt').

6) *Navigate* – перехід за вказаним посиланням.

Приклад:

.navigateTo(' https://hard.rozetka.com.ua/ua/').

7) *Upload* – завантаження файлів до веб-додатку.

Приклад:

```
.setFilesToUpload('#upload-input', [
        './examples/S1.jpg',
        './ examples/S2.jpg',
        './ examples/S3.jpg'
    ]).
```

8) *Resize Window* – зміна розміру вікна браузера.

Є три способи змінити розмір вікна:

- встановити розміри:
  ```
  .resizeWindow(200, 100)
  ```
- підігнати розміри під властивості поточного обладнання:
  ```
  .resizeWindowToFitDevice('Mi 9t pro', {
      portraitOrientation: true
    })
  ```
- використати максимальний розмір вікна:
  ```
  .maximizeWindow() [4].
  ```

Можна використовувати умови (Assertions) для перевірки того, чи відповідає стан тестованої системи певним твердженням, чи ні. TestCafe надає широкий вибір операторів умов, які можуть бути використані для покращення якості тестування.

3.4 Виконання підготовлених тест кейсів в TestCafe Studio

TestCafe Studio надає широкі можливості під час програмної реалізації підготовлених тест кейсів: можна відкрити браузер через додаток, підготувати необхідні дії, а потім в TestCafe Studio редагувати тест кейси за допомогою графічних елементів або безпосередньо в коді. Після написання тест кейсів

можна експортувати їх у вигляді JavaScript коду для використання у інших додатках або імпорту на іншу машину.

Програмну реалізацію (лістинги) розроблених прототипів тест кейсів можна знайти в додатку Б.

TestCafe Studio дозволяє запускати виконання тест кейсів в різних браузерах/платформах та з різними параметрами. Приклад конфігурації наведено на рисунку 3.1.

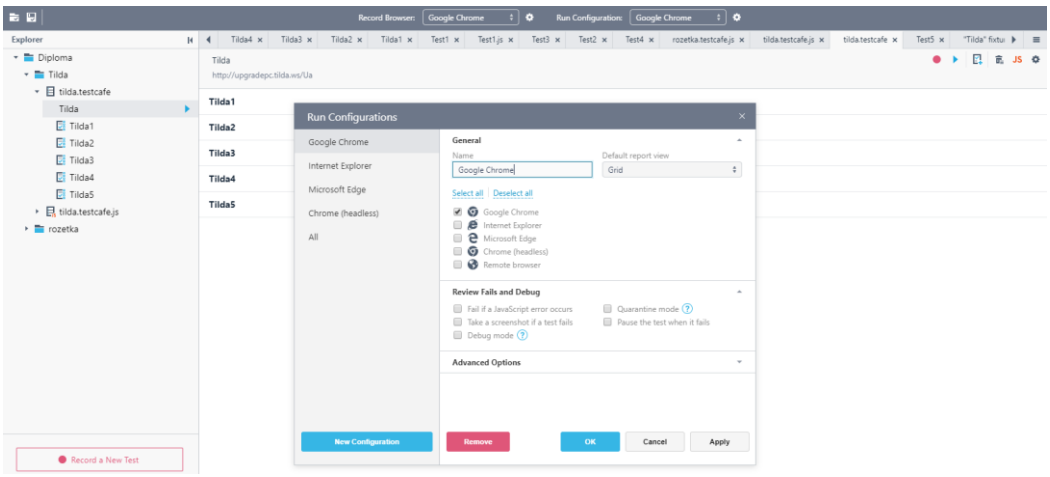

Рисунок 3.1 – Конфігурація запуску виконання тест кейсів

Для виконання підготовлених тест кейсів необхідно завантажити додаток TestCafe Studio. Для завантаження тест кейсів необхідно натиснути кнопку ‘Open Test Directory’ і обрати директорію на комп’ютері, де зберігаються тест кейси. Після завантаження кейсів, або якщо вони розроблялися на комп’ютерів додатку де буде проходити запуск, можна знавігуватися безпосередньо до тест кейсу і натиснути на кнопку ‘Run Test’ як зображено на рисунку 3.2.

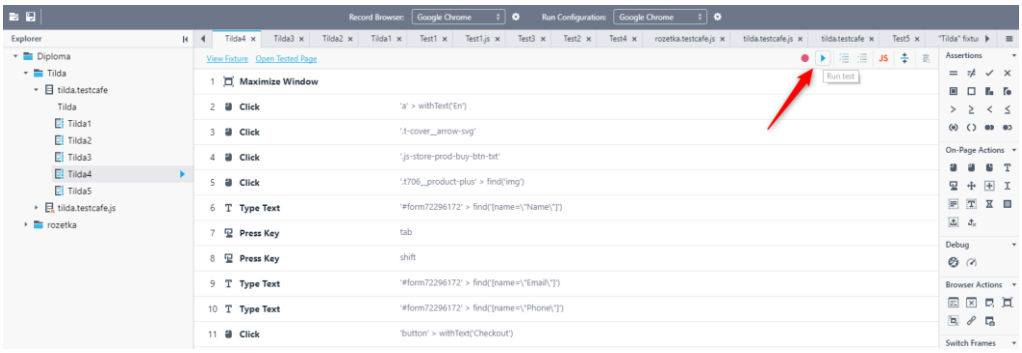

Рисунок 3.2 – Запуск виконання одного тест кейсу

Якщо було підготовлено декілька тест кейсів, об’єднаних в групу (тест сценарій), то можна запустити їх всі відразу як показано на рисунку 3.3.

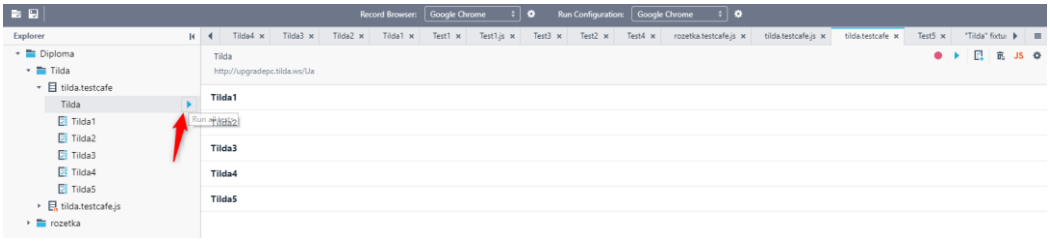

Рисунок 3.3 – Запуск всіх тест кейсів

Після виконання запуску тест кейсів користувачу надається звіт, де можна переглянути статус (був кейс успішно виконаний чи були помилки, які спричинили падіння виконання кейсу), час виконання та інші деталі. Відповідні приклади наведені на рисунках 3.4 та 3.5.

Результати запуску виконання тест кейсів для першого веб-додатку наведено на рисунку 3.4.

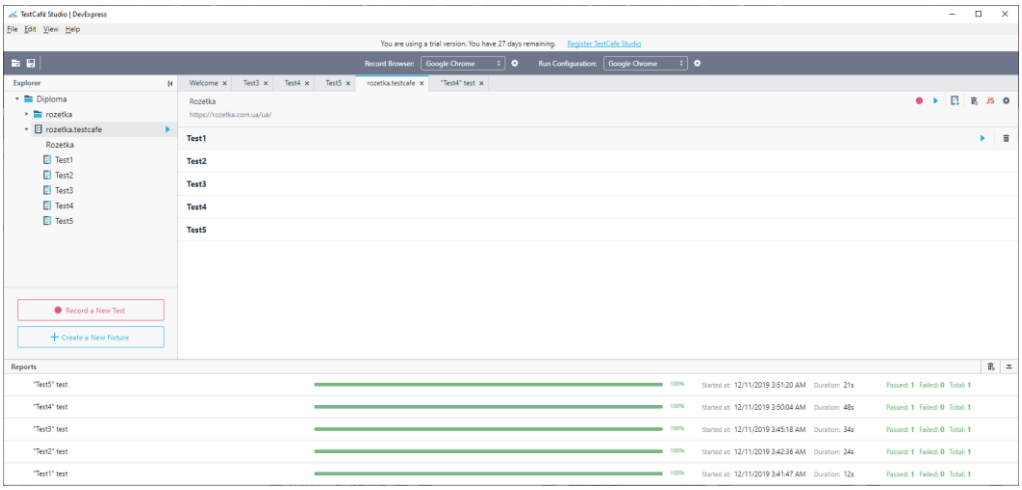

Рисунок 3.4 – Результат виконання тест кейсів для першого веб-додатку

Результати запуску виконання тест кейсів для другого веб-додатку наведено на рисунку 3.5.

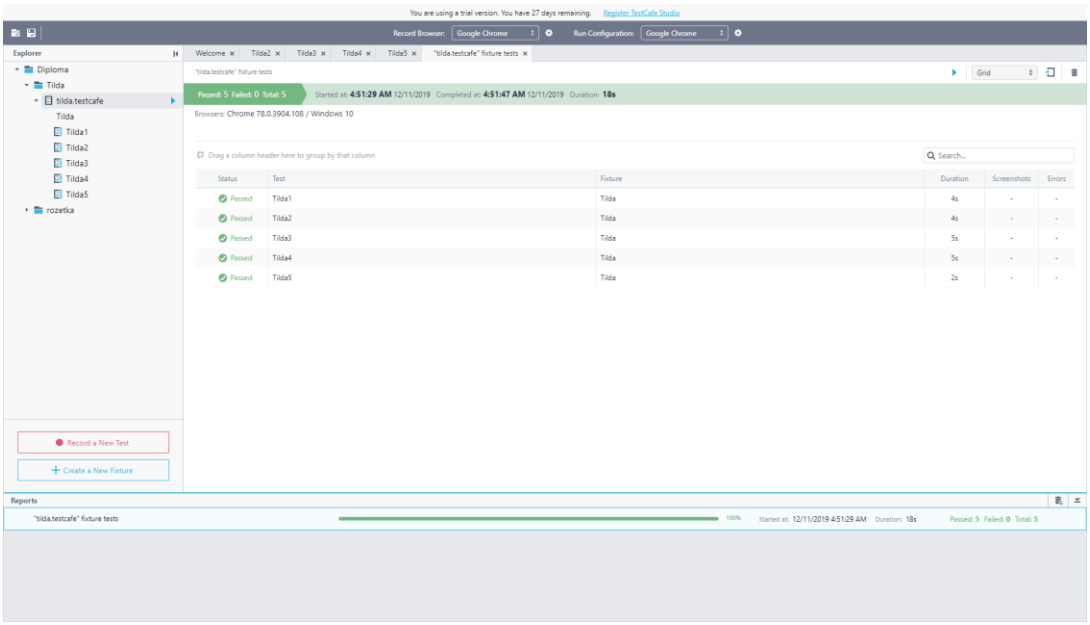

Рисунок 3.5 – Результат виконання тест кейсів для другого веб-додатку

3.5 Виконання регресійного тестування для другого веб-додатку

Регресійне тестування – це виконання списку тест кейсів для тестування певного програмного продукту після того, як до нього були внесені зміни [29, 33].

Регресійне тестування допомагає зрозуміти, чи були після змін в коді продукту допущені помилки, які можуть призвести до порушення роботи додатку та втрати основних функцій.

Під час виконання кваліфікаційної роботи, після першого виконання тест кейсів для другого веб-додатку до його контенту було внесено зміни. Щоб зрозуміти, як це відобразиться у звіті, було виконано повторний запуск тест кейсів.

Як можна побачити на рисунку 3.6, під час виконання п’ятого тест кейсу було знайдену помилку і статус виконання проставлено ‘Failed’. Додаток дозволяє переглянути, яка саме умова не була виконана, і що призвело до неуспішного виконання сценарію.

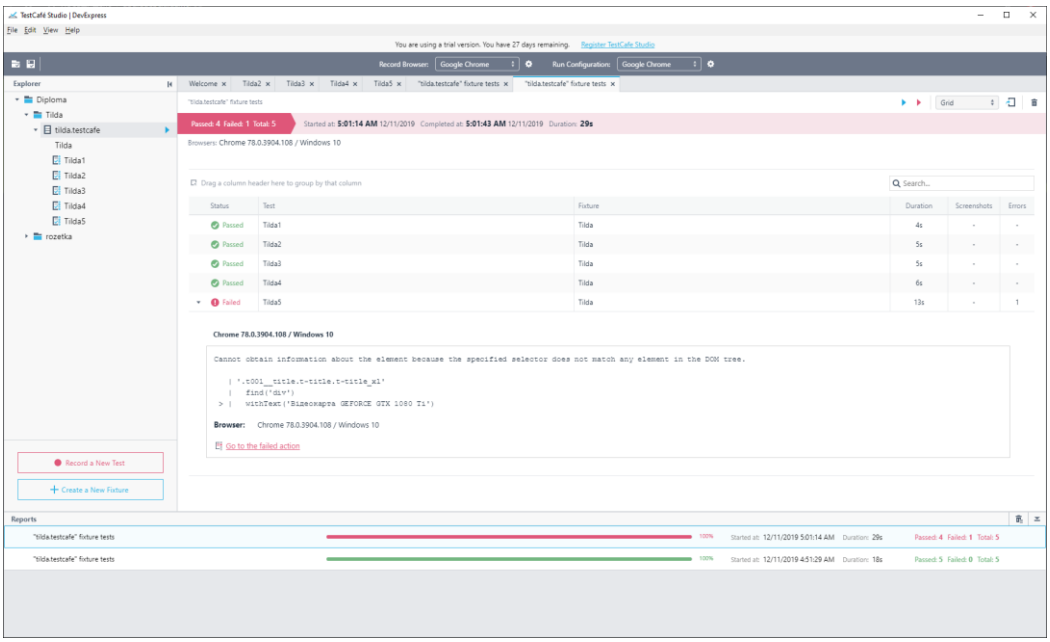

Рисунок 3.6 – Результати виконання тест кейсів для регресійного тестування

Таким чином, можна зробити висновок про те, що додаток TestCafe Studio чудово підходить не лише для звичайного виконання тест кейсів, а й для регресійного тестування. Це дозволить мінімізувати помилки, які можуть виникнути при зміні функціоналу/контенту додатку, що, відповідно, дозволить покращити його якість і задоволеність кінцевого користувача.

3.6 Оцінка проведення ручного та автоматизованого тестування

Для репрезентативності результати, отримані після виконання тест кейсів, були занесені до таблиць 3.3 та 3.4.

Таблиця 3.3 – Витрати часу на тестування першого веб-додатку

Веб-додаток №1		
№ кейсу	Ручне тестування, с	Автоматизоване тестування, с
1	40	21
2	80	48
3	70	34
4	130	24
5	60	12
Разом	380	139

Таблиця 3.4 – Витрати часу на тестування другого веб-додатку

Веб-додаток №2		
№ кейсу	Ручне тестування, с	Автоматизоване тестування, с
1	20	4
2	120	4
3	120	5
4	120	5
5	20	2
Разом	400	20

З представлених результатів добре видно, що автоматизоване тестування першого веб-додатку займає майже в три рази менше часу (380/139≈2,73). Для тестування другого веб-додатку цей показник є ще більшим – автоматизоване тестування займає в 20 разів менше часу, ніж ручне.

3.7 Висновки до третього розділу

У третьому розділі магістерської роботи було розроблено прототипи тест кейсів для автоматизації тестування веб-додатків. Було проведено їх програмну реалізацію та виконання за допомогою додатку TestCafe Studio.

З результатів тестування очевидна вигода від впровадження автоматизації веб-додатків. Автоматизація дозволяє значно скоротити витрати трудових та грошових ресурсів на тестування програмного забезпечення в процесі його розробки та подальших змін (регресійне тестування).

ВИСНОВКИ

В даній магістерській роботі було проведено аналіз основних понять, методів та засобів тестування програмного забезпечення, зокрема веб-додатків, визначено задачі, які повинні бути вирішені в результаті виконання роботи.

Проведено аналіз існуючих методів тестування веб-додатків і програмного забезпечення в цілому. Розглянуто основні питання щодо забезпечення якості створюваного продукту в розрізі загальної теорії тестування програмного забезпечення.

Проаналізовано існуючі засоби автоматизації тестування веб-додатків. В результаті аналізу було виділено декілька додатків, серед яких досить розповсюджений Selenium та новий відносно нього TestCafe. Для дослідження був обраний TestCafe Studio, який є більш новим, функціональним та зручним для користування для всіх тестувальників, а не лише для тих, хто має знання у сфері програмування.

Було розроблено прототипи тест кейсів для автоматизації тестування веб-додатків, зокрема сайтів електронної комерції. За необхідності тест кейси можна правити в залежності від веб-додатку, тестування якого проводиться.

Після використання розроблених тест кейсів для автоматичного тестування було визначено швидкість їх виконання та порівняно з відповідними показниками при ручному виконанні аналогічних дій. Результати виконання тестування наглядно показали, на скільки автоматизоване тестування може бути ефективнішим за ручне.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. [Електронний ресурс]. – Режим доступу: http://developer.android.com/tools/testing/testing_ui.html - UI Testingfr Android.

2. [Електронний ресурс]. – Режим доступу: http://upgradepc.tilda.ws/

3. [Електронний ресурс]. – Режим доступу: http://www.vogella.com/articles/AndroidTesting/article.html#roboelectric_overview – Android application testing with the Android test framework.

4. [Електронний ресурс]. – Режим доступу: https://devexpress.github.io/testcafe/documentation/test-api/

5. [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/24973/

6. [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Test_automation

7. [Електронний ресурс]. – Режим доступу: https://rozetka.com.ua/ua/

8. [Електронний ресурс]. – Режим доступу: https://sqa.lviv.ua/yaki-ye-typy-testuvannya

9. A Bertolino, P. Inverardi, and H. Muccini. An Explorative Journey from Architectural Tests Definition down to Code Tests Execution. In IEEE Proc. Int. Conf. on Software Engineering, ICSE2001, pp. 211-220, May 2001.

10. An Andreas Johnson. Architecture –Based Verification Of Software-Intensive Systems. Mälardalen University School of Innovation, Design and Engineering 2010 March 8,Västerås Sweden.

11. Anderson G., Asare S.D., Ayalew Y., Garg D.,Gopolang B., Masizana-Katongo A., Mogotlhwane O., Mpoeleng D., and Nyongesa H.O. (2007) Towards a Bilingual SMS Parser for HIV/AIDS Information Retrieval in Botswana, In Proceedings of the secondIEEE/ACM International Conference of Information and Communication Technologies and Development (ICTD),pp 329-333, Bangalore, India.16.

12. Aw, A., Zhang, M., Xiao, J., and Su, J. (2006) A phrase-based statistical model for SMS text normalization, In Proceedings of COLING-ACL

13. B. Beizer. Software Testing Techniques. Van Nos-trand Reinhold, New York, NY, 1990.

14. Badri H.S. Systematic Software Architecture Based Testing Approach – A Case Study. In International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 9, September 2012.

15. Baier, C., & Katoen J. P. (2008). Principles of Model Checking. Cambridge, MA: The MIT Press

16. Bernardo, M., Inverardi, P., 2003. Formal Methods for Software Architectures, Tutorial Book on Software Architectures and Formal Methods. SFM-03: SA Lectures, LNCS, vol. 2804.

17. Bertolino and P. Inverardi. Architecture based software testing. In Proc. of Int'l Software Architecture. Workshop, pages 62-64, October 1996.

18. Binder R.V. Testing Object-Oriented Software: Models, Patterns, and tools / Binder. R.V. – Addison Wesley, 1999. – 1203 p.

19. Bosch, J., 2000. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. ACM Press/Addison-Wesley Publishing Co.

20. Cédrick, F., and Sébastien, P. (2010) A Translated Corpus of 30,000 French SMS, In Proceedings of the48th Annual Meeting of the Association for Computational Linguistics, pp 770-779 Uppsala,Sweden13.

21. Copestake, A., Corbett, P., Murray-Rust, P., Rupp, C. J.,Siddharthan, A., Teufe, S., and Waldron, B. (2006) An Architecture for Language Processing for Scientic Texts, In Proceedings of the UK e-Science Programme All Hands Meeting (AHM2006), Nottingham, UK.17.

22. D. Richardson and A. Wolf. Software testing at the architectural level. In Proc. of Int'l Software. Arch. Workshop, pages 6-70, October 1996.

23. D. Richardson, J. Stafford, and A. Wolf. A formal approach to architecture based software testing. Technical report, University of California, Irvine, 1998.

24. D. S. Rosenblum and E. J. Weyuker. Predicting the cost-effectiveness of regression testing strategies. In Proceedings of the ACMSIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering, Oct. 1996.

25. Fewster M. Software Test Automation / M. Fewster, D. Graham. – ACM Press, 1999. – 600 p.

26. Genereux, M. (2002) An example-based Semantic Parserfor National language, In Proceedings of EMCSR 2002, Vienna, Austria.19.

27. Gruner, S. (2011). Problems for a Philosophy of Software Engineering. Minds and Machines, 21(2), 275-299.

28. GUI Testing Using Computer Vision / [Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller]. – ACM, CHI, 2010.

29. H. Leung and L. White. A cost model to compare regression test strategies. In Proc. of Conf. on Software. Maint. Pages 201-208, Oct.1991.

30. H. Muccini, M. Dias, and D. Richardson. Systematic Testing of Software Architectures in the C2 style. Extended version of the ETAPS 2004 publication.

31. Harsh Bhasin, Manoj. Regression Testing Using Coupling and GeneticAlgorithms.IN (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (1), 2012, 3255 – 3259.

32. Hoare., C. A. R. (1969). An Axiomatic Basis for Computer Programming. Comm. ACM, 12(10),576-580.

33. Henry Muccini, Marcio Dias, Debra J. Richardson. Software Architecture-Based Regression Testing, 10 February 2006. In: TheJournal of Systems and Software 79 (2006) 1379–1396.

34. http://automated-testing.info/tools - Web site about Tests Automation.

35. http://standards.ieee.org/findstds/standard/829-1983.html - 829-1983 - IEEE Standard for Software Test Documentation.

36. Kern, S. E., and Jaron, D. (2003) HealthcareTechnology, economics and policy: an evolving balance,IEEE Eng Med Biol Mag 22, 16-19.2.

37. Kobel, M. (2005) Parsing by Example, Institut furInformatik und angewandte Mathematik.18.

38. Kohana, YII, Symfony, CodeIgniter. – [Электронный ресурс]. – Режим доступу: http://www.whydrupal.ru/sravnenie-php-freimvorkov-kohana-yiisymfony-codeigniter

39. Kohn, L. T., Corrigan, J. M., and Donaldson, M. S.,(Eds.) (1999) To err is human: building a safer healthsystem.8.

40. Komatinemi S. Pro Android 3 / S. Komatinemi, MacLean D., S.Y. Hashimi. – Apress, 2011. – 1200 p.

41. Lee, R. G., Shen, H. S., Lin, C. C., Chang, K. C., andChen, J. H. (2000) Home telecare system using cabletelevision plants- an experimental field trial., IEEE TransInf Technol Biomed 4, 37-44.

42. Lin, J. C. (1999) Applying telecommunicationstechnology to health-care delivery, IEEE Eng Med BiolMag, 28-31.4.

43. M. J. Harrold. Architecture-Based Regression Testing of Evolving Systems. In Proc. Int. Workshop on the Role of SoftwareArchitecture in Testing and Analysis (ROSATEA), CNR-NSF, pp. 73-77, July 1998.

44. Masizana-Katongo, A., and Ama-Njoku, T. (2011)Example-Based Parsing Solution for a HIV and AIDSFAQ System, International Journal of Research andReviews in Wireless Communications (IJRRWC) 1, 59-65.15.

45. Muccini, H., Bertolino, A., Inverardi, P., 2003. Using software architecture for code testing. IEEE Transactions on SoftwareEngineering 30 (3), 160–171.

46. Murphy M.L. The Busy Coder's Guide to Android Development / Murphy M.L. – Commonsware LLC, 2008. – 400 p.

47. Parsloe, C. (2003) World apart? Healthcare technologiesfor longlife disease management, IEEE Eng Med BiolMag, 53-56.11.

48. Perry, D.E., Wolf, A.L., 1992. Foundations for the Study of Software Architecture 17 (4), 40–52.

49. Samanta, S. K., Achilleos, A., Moiron, S. R. F., Woods,J., and Ghanbari, M. (2010) Automatic languagetranslation for mobile SMS, International Journal of Information Communication Technologies and HumanDevelopment (IJICTHD) 2, 43-58.12.

50. Sikuli: using GUI screenshots for search and automation / [Tom Yeh, Tsung-Hsiang Chang, Robert C. Miller] // In Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09). – ACM, NY, USA. – P. 183-192.

51. Singh, M. P. (2002) Treating healthcare, IEEE InternetComputing, 4-5.10.

52. Streiter, O. (2000) Reliability in Example-Based Parsing,In Workshop TAG+5, Paris France.20.

53. Tagg, C. (2009) A Corpus Linguistics Study of SMSText Messaging, In Department of English, p 402,School of English, Drama and American and CanadianStudies, Birmingham.14.

54. Wells, P. N. T. (2003) Can technology truly reducehealthcare costs, IEEE Eng Med Biol Mag, 20-25.3.

55. YII-framework. Часть 1. Введение: – [Електронний ресурс] – Режим доступу: http://www. Linkexchanger .su /2010 / 418.html

56. Zhang, J., Stahl, J. N., Huang, H. K., Zhou, X., Lou, S.L., and Song, K. S. (2000) Real-time teleconsultationwith high resolution and large-volume medical forcollaborative healthcare, IEEE Trans Inf TechnolBiomed 4, 178-185.5.

57. Абдикеев Н. М. Проектирование информационных систем (современные технологии): Учебное пособие. / Н. М. Абдикеев. – М. : КОС-ИНФ, Рос. экон. акад., 2003. – 140 с. 77

58. Ахтырченко К.В. Методы и технологии реинжиниринга ИС / К.В. Ахтырченко, Т.П. Сорокваша / Труды Института Системного

Программирования РАН. – [Электронный ресурс]. – Режим доступу: http://citforum.ru/SE/project/isr.

59. Башмаков А.И. Интеллектуальные информа-ционные технологии: Учеб. пособие. / А.И. Башмаков, И.А. Башмаков. – М. : Изд-во: МГТУ им. Н.Э. Баумана, 2005. – 304 с.

60. Васвани В. Разработка веб-приложений на PHP / В. Васвани– Питер, 2012. – 432 с.

61. Винниченко И. В. Автоматизация процессов тестирования / Винниченко И. В. – СПб.: Питер, 2011. – 203 с. – ISBN 5-469-00798-7.

62. Дастин Э. Автоматизированное тестирование программного обеспечения. Внедрение. Управление и эксплуатация пер з англ. / Э. Дастин, Д. Рэшка, Д. Пол – М: Лорі, 2013. – 589 с. – ISBN 5-85582-186-2.

63. Калбертсон Р. Быстрое тестирование пер. з англ. / Калбертсон Р., Браун К., Кобб Г. – М.: Вільямс, 2011. – 374 с. – ISBN 5-8459-0336-X.

64. Книга веб-программиста: секреты профессиональной разработки веб-сайтов: пер з англ. / [Хоган Б., Уоррен К., Уэбер М. та ін.] – СПб.:Питер, 2013. – 208 с. – ISBN 978-5-459-01510-2.

65. Плаксин М. А. Тестирование и отладка программ для профессионалов будущих и настоящих / Плаксин М. А. – М.:Бином. Лаборатория знаний, 2013. – 168 с – ISBN 978-5-94774-458-3.

66. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Савин Р.– М.:Дело, 2007. – 316 с. – ISBN 978-5-7749-0460-0.

67. Тамре Л. Введення в тестування програмного забезпечення: пер. з англ. / Тамре Л. – М.: Вільямс, 2013. – 368 с. – ISBN 5-8459-0394-7

68. Тестирование производительности Web-приложений Microsoft .NET пер. з англ.: Microsoft Corporation. – М.: Русская редакция, 2013. – 352 с. – ISBN 5-7502-0224-0.

69. Уиттакер Д. Как естируют в Google пер з англ. / Д. Уиттакер, Д. Арбон, Д. Каролло – СПб.: Питер, 2014. – 320 с. – 978-5-496-00893-8.

ДОДАТКИ

Додаток А

(Обов’язковий)

SUMMARY

Liashko S.Yu. Automated testing of web applications. – Masters level Qualification Thesis. Sumy State University, Sumy, 2019.

The necessity of automation of web application testing process was investigated. The main purpose of the research is to select a software product that will help improve the efficiency and quality of web application testing, as well as to develop prototypes of test cases performed by the application to analyze the benefits of automation.

Keywords: automation, web application, test case, testing, prototype, regression, software.

АНОТАЦІЯ

Ляшко С.Ю. Автоматизоване тестування веб-додатків. – Кваліфікаційна магістерська робота. Сумський державний університет, Суми, 2019 р.

У роботі досліджено необхідність проведення автоматизації процесу тестування веб-додатків. Основною метою дослідження є вибір програмного продукту, яких допоможе підвищити ефективність та якість тестування веб-додатків, а також розробка прототипів тест кейсів виконуваних додатком для аналізу отриманої користі від впровадження автоматизації.

Ключові слова: автоматизація, веб-додаток, тест кейс, тестування, прототип, регрессія, програмне забезпечення.

Додаток Б

(Інформаційний)

ЛІСТИНГ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТЕСТ КЕЙСІВ

Програмна реалізація тест кейсів для першого веб-додатку:

Відкриття веб-додатку:

```
fixture `Rozetka`
   .page `https://rozetka.com.ua/ua/`;
```

Код першого тест кейсу:

```
test('Test1', async t => {
   await t
      .click(Selector('.side-menu__toggler').find('svg'))
      .click(Selector('.menu-
language__item').nth(1).find('a').withText('UA'));
});
```

Код другого тест кейсу:

```
test('Test2', async t => {
   await t
      .click(Selector('.side-menu__toggler').find('svg'))
      .click(Selector('.menu-language__item').nth(1).find('a').withText('UA'))
      .click(Selector('.side-menu__toggler').find('svg'))
      .click(Selector('.menu-main__link').find('span').withText('Каталог
товарів'))
      .click(Selector('[alt="Смартфони, ТВ і електроніка"]'))
      .click(Selector('.fat-
popular__item').nth(6).find('a').withText('Відеокамери'));
});
```

Код третього тест кейсу:

```
test('Test3', async t => {
    await t
        .click(Selector('.side-menu__toggler').find('svg'))
        .click(Selector('.menu-language__item').nth(1).find('a').withText('UA'))
        .click(Selector('.side-menu__toggler').find('svg'))
        .click(Selector('.menu-main__link').find('span').withText('Каталог
товарів'))
        .click(Selector('[alt="Смартфони, ТВ і електроніка"]'))
        .click(Selector('.fat-popular__item').find('a').withText('Мобільні
телефони'))
        .click(Selector('.select-css.ng-untouched.ng-pristine.ng-valid'))
        .click(Selector('option').withText('Популярні'));
});
```

Код четвертого тест кейсу:

```
test('Test4', async t => {
    await t
        .maximizeWindow()
        .click(Selector('.menu-toggler'))
        .hover(Selector('li').withText('Смартфоны, ТВ и
электроника').find('.menu-categories__link.js-menu-categories__link'))
        .click(Selector('.menu__hidden-
list').nth(15).find('a').withText('Смартфоны'))
        .drag(Selector('a').withText('Диагональ экрана'), 1, 1)
        .click(Selector('i').withText('6\" - 6.49\"'))
        .click(Selector('span').withText('4000 мА*ч + (117)'))
        .click(Selector('span').withText('6 ГБ (19)'));
});
```

Код п’ятого тест кейсу:

```
test('Test5', async t => {
    await t
        .maximizeWindow()
        .typeText(Selector('.search-form.ng-untouched.ng-pristine.ng-valid').find('[name="search"]'), 'Mi 9t pro 6/128')
        .pressKey('enter')
        .click(Selector('a').withText('Характеристики'))
        .click(Selector('a').withText('Фото'))
        .click(Selector('a').withText('Отзывы 134'))
        .click(Selector('a').withText('Покупают вместе'))
        .click(Selector('a').withText('Все о товаре'));
});
```

Програмна реалізація тест кейсів для другого веб-додатку:

Відкриття веб-додатку:

```
fixture `Tilda`
    .page `http://upgradepc.tilda.ws/Ua`;
```

Код першого тест кейсу:

```
test('Tilda1', async t => {
    await t
        .maximizeWindow()
        .click(Selector('a').withText('En'))
        .click(Selector('a').withText('Ua'));
});
```

Код другого тест кейсу:

```
test('Tilda2', async t => {
    await t
        .maximizeWindow()
        .click(Selector('.js-store-prod-buy-btn-txt'))
        .click(Selector('#form72293672').find('[name="Name"]'))
        .pressKey('shift+alt')
        .typeText(Selector('#form72293672').find('[name="Name"]'), 'Сергій')
        .pressKey('tab')
        .pressKey('shift+alt')
        .pressKey('shift+alt')
        .typeText(Selector('#form72293672').find('[name="Email"]'),
'hollow3809@gmail.com')
        .pressKey('tab')
        .typeText(Selector('#form72293672').find('[name="Phone"]'),
'+380997256452')
        .click(Selector('button').withText('Замовити'));
});
```

Код третього тест кейсу:

```
test('Tilda3', async t => {
    await t
        .maximizeWindow()
        .click(Selector('td').withText('Придбати відеокарту'))
        .click(Selector('.js-store-prod-buy-btn-txt'))
        .click(Selector('#form72293672').find('[name="Name"]'))
        .pressKey('shift+alt')
        .typeText(Selector('#form72293672').find('[name="Name"]'), 'Сергій')
        .pressKey('tab')
        .pressKey('shift+alt')
        .pressKey('shift+alt')
        .typeText(Selector('#form72293672').find('[name="Email"]'),
'hollow38092')
        .pressKey('backspace')
        .typeText(Selector('#form72293672').find('[name="Email"]'),
'@gmail.com')
        .pressKey('tab')
        .typeText(Selector('#form72293672').find('[name="Phone"]'),
'+380997256452')
        .click(Selector('button').withText('Замовити'));
});
```

Код четвертого тест кейсу:

```
test('Tilda4', async t => {
    await t
        .maximizeWindow()
        .click(Selector('a').withText('En'))
        .click(Selector('.t-cover__arrow-svg'))
        .click(Selector('.js-store-prod-buy-btn-txt'))
        .click(Selector('.t706__product-plus').find('img'))
        .typeText(Selector('#form72296172').find('[name="Name"]'), 'Serhii')
        .pressKey('tab')
        .pressKey('shift')
        .typeText(Selector('#form72296172').find('[name="Email"]'),
'hollow3809@gmail.com')
        .typeText(Selector('#form72296172').find('[name="Phone"]'),
'+380997256452')
        .click(Selector('button').withText('Checkout'));
});
```

Код п’ятого тест кейсу:

```
test('Tilda5', async t => {
    await t
        .maximizeWindow()
        .expect(Selector('.t001__title.t-title.t-
title_xl').find('div').withText('Відеокарта GEFORCE GTX 1080
Ti').textContent).eql("Відеокарта GEFORCE GTX 1080 Ti  ");
});
```