МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів автоматизації тестування програмних e-commerce systems
(тема)

Виконав:
Студент 2 курсу, групи ІПЗздм-19-2
Ільченко А.В.
(прізвище, ініціали)

Спеціальність 121, «Інженерія програмного забезпечення»
(код і повна назва спеціальності)

Тип програми Освітньо – наукова
(освітньо-професійна або освітньо-наукова)

Керівник доцент Вечур О.В.
(посада, прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121, «Інженерія програмного забезпечення»
(код і повна назва спеціальності)

Тип програми Освітньо – наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Ільченко Аліни Вячеславівни
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів автоматизації тестування програмних e-commerce systems

затверджена наказом університету від 26.03.2021 № 34

2. Термін подання роботи до екзаменаційної комісії 21 05 2021р.
3. Вихідні дані до роботи Методи автоматизованого тестування ПЗ, ОС Windows, мова програмування Java, середовище розробки IntelliJIDEA
4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області і постановка задачі, методи та інстументи для автоматизації тестування програмного забезпечення e-commerce systems

5. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування Розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доцент Вечур О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.02.2021	виконано
2	Постановка задачі	18.03.2021	виконано
3	Дослідження методів та інстументів	02.04.2021	виконано
4	Проектування та розробка фремворку	07.04.2021	виконано
5	Підготовка пояснювальної записки	04.05.2021	виконано
6	Підготовка презентації та доповіді	05.05.2021	виконано
7	Перевірка на академічний плагіат	11.05.2021	виконано
8	Нормоконтроль	11.05.2021	виконано
9	Рецензування	13.05.2021	виконано
10	Знесення диплома в електронний архів	15.05.2021	виконано
11	Попередній захист	18.05.2021	виконано
12	Допуск до захисту у зав. Кафедри	21.05.2021	виконано

Дата видачі завдання 25 . Січня 2021р.

Студент _____
(підпис)

Керівник роботи _____ доцент Вечур О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 102 с., 27 рис., 1 табл., 23 джер.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ, СИСТЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ, ТЕСТУВАННЯ ВЕБ ЗАСТОСУНКУ ЕЛЕКТРОННОЇ КОМЕРЦІЇ, ТЕСТОВІ СЦЕНАРІЇ, ТЕСТОВИЙ ФРЕЙМВОРК, SELENIUM, BDD.

Об’єктом дослідження є процес забезпечення якості та надійності тестування e-commerce веб-застосування шляхом створення автотестів.

Метою роботи є підвищення ефективності тестування ecommerce веб-застосування шляхом дослідження методів автоматизації тестування програмних ecommerce system, вибору найоптимальнішого фреймворку та інструментів для створення автотестів, їх подальшої підтримки, перевикористання та розширення.

Методи розробки базуються на таких технологіях, як Selenium, Java, Spring, Cucumber, BDD, Allure.

В результаті роботи було досліджено технології, фреймворки, іструменти та методи для автоматизації, які на сьогоднішній день зарекомендували себе в процесі конфігурації системи для створення автотестів. Удосконалено спосіб розробки фреймворку для автоматизації та написання тестів за рахунок спрощення процесу їх імплементації, перевикористання і підтримки.

AUTOMATION TESTING, AUTOMATION, E-COMMERCE WEB SYSTEMS, E-COMMERCE WEB APPLICATION TESTING, TEST SCENARIOS, TEST FRAMEWORK, SELENIUM, BDD.

The object of the study is the process of ensuring the quality and reliability of testing e-commerce web application by creating automated tests.

The aim of the work is to increase the efficiency of e-commerce testing of web applications by researching methods of automation testing ecommerce system, selection of the the most optimal framework and tools for creating autotests, their further support, reuse and expansion.

Development methods are based on such technologies as Selenium, Java, Spring, Cucumber, BDD, Allure.

As a result of the work, technologies, frameworks, tools and methods for automation were studied, which for today have proven themselves in the process of configuring the system for creating autotests. Improved the way to develop a framework for automation and writing tests by simplifying the process of their implementation, reuse and support.

Я, Ільченко Аліна Вячеславівна, студентка групи ІПЗздм-19-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів автоматизації тестування програмних e-commerce systems», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 9
ВСТУП 10
1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 14
1.1 Аналіз проблемної області дослідження 14
1.1.1 Аналіз вимог 15
1.1.2 Стадія проектування 15
1.1.3 Розробка 16
1.1.4 Системне тестування 16
1.1.5 Тестування та налагодження 17
1.1.6 Експлуатація та підтримка 17
1.2 Види тестування 18
1.2.1 Функціональне тестування 18
1.2.2 Нефункціональне тестування 19
1.3 Виявлення проблем автоматизації та актуалізація рішень 21
1.4 Моделі створення автоматизованих тестів 24
1.5 Постановка задачі 25
2 АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК В ОБЛАСТІ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ 27
2.1 Визначення критеріїв аналізу 27
2.2 Порівняльна характеристика існуючих розробок 28
2.2.1 Selenium 28
2.2.2 Katalon Studio 29
2.2.3 Unified Functional Testing (UFT) 30
2.2.4 Watir 31

2.2.5 Test Complete 31
2.2.6 Robot Framework 32
2.3 Порівняльна характеристика інструментів 34
2.3.1 JBehave 34
2.3.2 JUnit 36
2.3.3 Serenity 37
2.3.4 TestNG 38
3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ 40
3.1 Формування вимог до програмної системи 40
3.2 Вибір основної мови програмування 40
3.3 Вибір технологій і специфікації 42
3.3.1 SAP Hybris 42
3.3.2 Cucumber 42
3.3.3 Selenium WebDriver 43
3.3.4 Бібліотека тестування jUnit 43
3.3.5 Система автоматичного складання проекту Maven 44
3.3.6 Spring 44
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 45
4.1 Реалізація системи автоматизованого тестування 45
4.1.1 Архітектура тестового фреймворку 45
4.1.2 Впровадження BDD підходу 50
4.1.3 Опис об'єктів веб-сторінки 52
4.1.4 Структура проектування класів 53
4.1.5 Інтеграція Cucumber зі Spring 54

4.1.6 Додаткові класи для Selenium WebDriver 56

4.1.7 Конфігурація запуску тестів 57

4.1.8 Характеристика результатної інформації..... 58

4.1.9 Система звітів по виконаним тестам 58

ВИСНОВКИ 61

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 64

ДОДАТОК А..... 67

ДОДАТОК Б 68

ДОДАТОК В..... 69

ДОДАТОК Г 79

ДОДАТОК Д..... 82

ДОДАТОК Е 84

ДОДАТОК Ж..... 85

ДОДАТОК И..... 87

ДОДАТОК К..... 89

ДОДАТОК Л..... 95

ДОДАТОК М..... ..101

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

API – Application Programming Interface (Прикладний програмний інтерфейс)

BDD – Behaviour Driven Development (Розробка керована поведінкою)

QA – Quality Assurance (Забезпечення якості)

ВСТУП

На сьогоднішній день сучасні реалії світу та інтернету неможливо уявити без різноманітних веб-застосувань. Вони вже достатньо давно увійшли в наше життя, стали популярними, інтуїтивно зрозумілими та простими серед користувачів. Сфера їх розробки постійно розвиється і з кожним днем вони стають все більш новітніми, гнучкими, кросплатформенними та набирають популярність. Як результат такі веб-застосування мають складнішу та більшу архітектуру, що у свою чергу все важче підтримувати та розробляти. На сьогодні, Agile – як підхід до розробки програмного забезпечення, в ході якого надається якась частина задокументованих описаних вимог, не завжди виявляється доцільним. Саме тому впровадження процесу автоматизації тестування програмного продукту за допомогою автоматичних тестових скриптів є однією з невід'ємних частин на проектах і представляє собою підхід, який стає все більше актуальним серед багатьох сучасних компаній.

Нині розробкою більшості складних та сучасних веб-додатків займаються крупні комерційні компанії. Важливими характеристиками при розробці функціоналу становлять показники надійності, якості та відповідності до наданих вимог щодо його фактичного результату, адже від справної та злагодженої роботи продукту напряму залежать прибутки підприємств-розробників. Таким чином відбувається активний розвиток та впровадження нових технологій, методів та інструментів за допомогою яких можна прискорити та спростити процеси реалізації та випуску продукту. Також слід враховувати те, щоб швидко можна було підлаштовуватися у разі зміни вимог замовника, підхід до процесу розробки повинен бути достатньо гнучким. Продукт, що розробляється має відповідати одночасно як вимогам замовника, так і загальним вимогам та стандартам якості. Гарною практикою вважається, щоб команда розробників на будь якій стадії реалізації мала доступ до проміжних версій продукту, які вже працюють коректно та відповідають стандартам якості.

Така активність як тестування ПЗ необхідна для того, щоб отримати загальну оцінку стану якості розробленого функціоналу, виявити відповідність до встановлених вимог, ідентифікувати дефекти з метою в кінцевому результаті реалізувати якісний продукт [1].

Автоматизація тестування є невід'ємною складовою протягом всіх етапів контролю якості розробки програмного продукту. Для реалізації автотестів, конфігурації для їх запуску, отримання звітності існують різноманітні сучасні інструменти та технології, які можуть значно зменшити час тестування і спростити та покращити процес.

Організація процесу автоматичного тестування дозволяє виявити проблеми як з вимогами так і з функціоналом на всіх етапах життєвого циклу розробки [2]:

– в ході активної фази розробки, кожного разу при будь-якій внесеній розробником зміні в програмний код, інженер з автоматизованого тестування може швидко запустити один тестовий сценарій або за потребою тестовий набір, щоб перевірити правильність роботи як нової імплементованої функціональності так і запевнитись, що новий функціонал на вплинув і не зламав коректну поведінку старого. Таким чином, звіт про походження тестів дозволить точно ідентифікувати помилку, причину та місце збою. В свою чергу виконуючи такі перевірки розробник буде завжди впевнений і тому, що його частина коду не зашкодила роботі програми;

– великою перевагою автоматизації є те, що вона майже повністю заміщує процес ручного тестування, як наслідок терміни на розробку продукту займають значно менше часу. Крім того, автотести можуть покривати сценарії проходження для різноманітних додатків, форм та будь якої іншої частини функціоналу. Як результат, тестовий сценарій, який був написаний один раз може слугувати інструментом для багаторазового пошуку помилок;

– механізм рандомної генерації даних, який може використовуються в тестових скриптах, дозволяє виявити непередбачені людським фактором помилки. Алгоритми автоматичної рандомної генерації даних здатні створити тестовий набір, який в ході ручного тестування інженер міг навіть і не передбачити [3];

– великі проекти з високим навантаженням вимагають більшої уваги до деталей та більш ретельного контролю якості. У довгостроковій перспективі лише тестування автоматизації буде ефективним для фінансових проектів, сайтів електронної комерції, значно покращить якість та зменшить витрати на розробку [4];

– проходження тестів без стороннього втручання. В той час як відбувається процес виконання тестів тестувальник може переключити увагу на якісь інші справи. Також тестові сценарії можуть виконуватися в неробочий час (цей метод краще, тому що навантаження на локальну мережу знижене);

– результати проходження тестів автоматично генеруються в звіти, зберігаються та надсилаються на аналіз;

– базуючись на результатах та історії проходження автотестів на певних стадіях розробки ПЗ, інженер з тестування отримує дуже цінну статистичну інформацію, на підставі якої можна проаналізувати як загальну кількість помилок, так і помилки, отримані при імплементації певного функціоналу. Допомагає простежити тенденцію багів в ході одного спринту або за весь час створення продукту загалом, тим проектам чи компаніям, які дотримуються гнучких методологій розробки. Навіть час, який був витрачений на аналіз, виправлення та повторне тестування виправлених помилок, їх загальне співвідношення. Ця інформація являється корисним і невід'ємним ресурсом для звітування перед замовниками, для використання при подальшій розробці, щоб у свою чергу на основі неї зробити якісь висновки та покращити процес, враховуючи недоліки попередніх релізів.

Метою даної роботи є підвищення ефективності тестування e-commerce веб-застосування шляхом дослідження методів автоматизації тестування програмних e-commerce system, вибору найоптимальнішого фреймворку та інструментів для створення автотестів, їх подальшої підтримки, перевикористання та розширення.

Для досягнення поставленої мети виконані наступні завдання:

– вивчення теоретичного матеріалу з тестування та автоматизованого тестування;

– проведений аналіз основних фреймворків для автоматизованого тестування і обраний найоптимальніший, що вирішуватиме поставлене завдання;

– змодельований фреймворк для автоматизованого тестування;

– представлена програмна реалізація створення автоматизованих перевірок для тестування веб-застосувань;

– здійснений аналіз розробленої системи для автоматизації та її ефективність у використанні в ході розробки ПЗ.

Об’єкт дослідження – процес забезпечення якості та надійності тестування e-commerce веб-застосування шляхом створення автотестів.

Предмет дослідження – технології та фреймворк як інструмент для тестування, який буде слугувати для забезпечення зручності при розробці автоматизованих тестів для тестування e-commerce веб-застосувань, дослідження методів автоматизації тестування програмних e-commerce system.

Наукова новизна досліджених результатів – проведено дослідження технологій, фреймворків та іструментів для автоматизації, які на сьогоднішній день зарекомендували себе в процесі конфігурації системи для створення автотестів. Удосконалено спосіб розробки фреймворку для автоматизації та написання тестів за рахунок спрощення процесу їх імплементації, перевикористання і підтримки.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області дослідження

Тестування програмного забезпечення являється процесом дослідження того, наскільки реальна поведінка системи відповідає вимогам і очікуванням. Тести, які використовуються для цих перевірок, ґрунтуються на задокументованих вимогах замовника. Необхідно максимально підвищити ймовірність того, що тестований продукт буде працювати як належить за будь-яких обставин і відповідати всім описаним вимогам. Також важливо надавати актуальну інформацію про стан продукту на даний момент.

Час від початкового моменту створення якого-небудь програмного продукту, його розробки, впровадження та вилучення з експлуєтації і є життєвий цикл програмного забезпечення.

Етапи тестування ПЗ – це процес, який включає в себе певну кількість окремих частин, має кінцеву мету і виконується протягом усього життєвого циклу розробки ПЗ (додаток В, рисунок В.1).

Загальний процес тестування зазвичай збігається з циклом реалізації програмного продукту або відповідає його певному етапу. Перевірка стійкості ПЗ схожа на вивчення системи від кінцевого користувача, тому тестувальника слід підключати до роботи ще на етапі збору та аналізу вимог. Найкраще, коли обговорення елементів продукту проходить за участю розробника, QA-аналітика і користувача.

Далі розглянемо більш детально кожен з етапів, які проходить програмний продукт від ідеї до експлуатації.

1.1.1 Аналіз вимог

Дана стадія є однією з найважливіших, оскільки визначає успіх всього проекту. На даному етапі формуються цілі і завдання проекту, встановлюється галузь застосування програмного забезпечення і визначаються граничні умови, базові сутності та взаємозв'язок між ними. Узгоджуються терміни і кошти на розробку ПЗ, формується і підписується технічне завдання. Умови і критерії роботи системи, як правило, визначаються замовником або менеджером проекту в процесі спілкування з клієнтом, або аналізом стандартів і нормативної документації. Ці вимоги можуть бути як функціональними, так і нефункціональними. Тестувальник працює над статичним тестуванням вимог: досліджує їх повноту, логічність, однозначність і визначеність, також знаходить слабкі місця в тестовому покритті і виявляє можливі ризики. Виникають обставини, коли наявні вимоги не тестуються, але використовуються на етапі дизайну і розробки. І вже готовий продукт направляється в відділ QA. Тоді кількість знайдених дефектів, вартість і масштаб їх виправлення можуть значно зрости.

1.1.2 Стадія проектування

Один з найважливіших складових етапів, тому що користувачеві доведеться постійно взаємодіяти з системою. Продукт повинен бути інтуїтивно зрозумілий у використанні, мати приємний інтерфейс, а також бути оптимізованим (користувач робить мінімальні дії для досягнення результату). На цій стадії відбувається розробка функціональної специфікації ПЗ, проектується архітектура системи, визначається найбільш підходяща СУБД (Система управління базами даних), проектуються структури зберігання даних, йде збір та обговорення вимог до апаратного та програмного забезпечення, складається перелік необхідних

організаційних заходів для впровадження ПЗ, а також перелік документів що регламентують його використання. На даному етапі тестувальник перевіряє існуючі прототипи ПЗ на відповідність вимогам замовника, коректності відображення візуальних елементів і зручності використання.

1.1.3 Розробка

На даній стадії будуються прототипи як цілої програмної системи, так і її частин, здійснюється фізична реалізація структур даних, розробляється програмний код, виконується налагоджувальне тестування, створюється технічна документація. В результаті етапу кодування з'являється робоча версія продукту. Під час процесу розробки системи слід провести модульне, інтеграційне і системне тестування. Для початку тестуються окремі компоненти програми. Це дозволяє виявити, в якій саме частині коду є помилка і досить швидко усунути її. Ще модульне тестування допомагає відстежити, чи не призвела зміна коду до появи нових помилок в уже перевірених місцях продукту. Також це допомагає краще зрозуміти роль кожного модуля системи. Інтеграційне тестування дозволяє перевірити, як компоненти коду об'єднуються і взаємодіють один з одним. Також воно визначає зв'язок між модулями, обладнанням або різними системами.

1.1.4 Системне тестування

І, нарешті, приступаємо до системного тестування. На цій стадії мається на увазі перевірка всіх компонентів і модулів в цілому. Цей етап життєвого циклу в ході розробки ПЗ має на увазі під собою загальний тест системи на предмет інтеграції її компонентів. Це означає, що у випадку, якщо система складається з

різних модулів, нам потрібно перевірити, наскільки добре чи наскільки погано кожен із них працює всередині системи. Більше того, на цьому етапі важливо виконувати тестування користувацького інтерфейсу.

1.1.5 Тестування та налагодження

Тестування ПЗ тісно пов'язане з етапами проектування та реалізації. На даному етапі тестувальники перевіряють систему на наявність дефектів незалежно від того, чи проводилося це раніше. Проводиться повне тестування інтерфейсу та функціоналу продукту. Всі виявлені помилки повинні бути задокументовані в баг-трекері. Також слід провести регресійне тестування, щоб дослідити систему на предмет дефектів, які могли з'явитися після усунення інших багів. По завершенню процесу налагодження потрібно надати оцінку якості продукту, наскільки відповідають вимоги замовника реальній роботі системи.

1.1.6 Експлуатація та підтримка

Після того, як продукт надходить в реліз, залишається необхідність в тестуванні, так як буде відбуватися оновлення ПЗ, будуть з'являтися нові баги, які були упущені або баги, пов'язані з експлуатацією кінцевого користувача. В такому випадку потрібне втручання відділу QA. У випадку знаходження кінцевими користувачами якихось пост-релізних багів, звітність про їх виявлення надається у формі звітів про дефекти команді розробки, яка, в залежності від пріорітету і серйозності проблеми приймає рішення про те, здійснювати швидке виправлення (так званий hot-fix), або проблему можна відкласти та полагодити уже в наступній релізній версії програмного продукту.

Всі зміни, що вносяться до програмного забезпечення необхідно ретельно протестувати. ПЗ повинне продовжувати виконувати спочатку закладені в ньому бізнес-функції і не порушувати працездатність інших функцій і всієї системи в цілому.

Процес тестування ПЗ охоплює всі етапи життєвого циклу розробки. Він безперервний, тривалий і вимагає наявності досить досвідченої команди тестувальників, щоб охопити всі етапи тестування. А моніторинг і аналіз всього процесу допомагає спланувати і, в разі необхідності, внести зміни для підвищення ефективності подальших завдань. Дана частина сучасного процесу розробки ПЗ допомагає замовнику, команді розробників, і головне, кінцевому користувачеві отримати продукт високої якості.

1.2 Види тестування

Всі види робіт, що проводяться тестувальником можна умовно розділити на два: функціональне і нефункціональне тестування.

Щоб було трохи простіше зрозуміти всі градації тестування, представимо їх перелік у вигляді рисунку В.2, додатку В.

1.2.1 Функціональне тестування

Функціональне тестування – тестування ПЗ, головна мета якого це перевірка можливості реалізації функціональних вимог додатка, тобто здатність додатка в заданих критеріях вирішувати покладені на нього (на додаток) завдання.

Вимоги включають в себе:

– захищеність;

– відповідність стандартам;

– здатність до взаємодії;

– функціональна придатність;

– точність.

Різні типи функціонального тестування містять наступне:

– Unit Testing – це перший рівень функціонального тестування для тестування будь-якого програмного забезпечення. При цьому тестувальник самостійно протестує модуль програми або перевірить всю функціональність модуля, що називається модульним тестуванням;

– Integration Testing – це другий рівень функціонального тестування, де ми перевіряємо потік даних між залежними модулями або інтерфейс між двома функціями;

– System Testing – тестувальник проходиться по кожному атрибуту програмного забезпечення та перевіряє, чи кінцева функція працює відповідно до бізнес-вимог і аналізує програмний продукт як цілісну систему.

1.2.2 Нефункціональне тестування ПЗ

В першу чергу перевірка на відповідності не функціональним вимогам:

– зручність – в основному проводиться оцінка зручності для користувачів;

– масштабованість – перевіряється як вертикальна так і горизонтальна масштабованість тестованої програми;

– продуктивність – здатність роботи програми при різних навантаженнях;

– безпека – захист призначених для користувача даних, захист даних додатка, стійкість на злам;

– портативність – сумісність і переносимість додатка для і під різні оточення, платформи тощо;

– надійність – поведінка системи при різних непередбачених ситуаціях, здатність обробки нестандартних дій користувача.

Воно надає детальну інформацію про продуктивність програмного продукту та використовувані технології.

Нефункціональне тестування класифікується за різними частинами тестування, про які буде йти мова далі.

При тестуванні програмного забезпечення за доступом до коду програмного продукту тестування може бути класифіковано на три різні типи тестування, а саме:

- White Box Testing – розробник перевірить кожен рядок коду, перш ніж передати його групі тестування або відповідним інженерам з тестування;
- Black Box Testing – тестувальник проаналізує програмне забезпечення на відповідність вимогам, виявить дефекти чи помилку та відправить його назад команді розробників;
- Grey Box Testing – включає доступ до внутрішнього коду для розробки тестових кейсів. Тестування сірого ящика проводить людина, яка знає програмування, а також тестування;
- Performance Testing – тестувальник перевірить роботу програми, симулювавши деяке навантаження. У цьому типі нефункціонального тестування тестувальник зосередиться лише на кількох аспектах, таких як час відгуку, навантаження, масштабованість та стабільність програмного забезпечення чи програми [5].

Performance Testing включає різні типи тестування, які є такими: тестування навантаження, стрес-тестування, тестування масштабованості, випробування на стійкість. Usability Testing аналізується зручність використання програми та виявляються помилки в інтерфейсі кінцевого користувача програмного забезпечення. Compatibility Testing перевіряється функціональність програми в конкретному апаратному та програмному середовищі.

За класифікацією об'єктів, які можуть бути протестовані: конфігураційне, стрес-тестування, юзабіліті, безпеки, документації, стабільності, локалізації та інтернаціоналізації та інтерфейсу користувача.

В залежності від типу перевірки даних та відтворення сценаріїв: позитивні та негативні тестові сценарії.

Існує кілька фаз тестування ПЗ: Alpha, Smoke, New Feature Testing, Regression, Acceptance та Beta. Вони виконуються в певних проміжках часу, іноді один за іншим і разом забезпечують випуск високоякісного програмного забезпечення.

Основуючись на рівні підготовки та освіченості тестувальника, може виконуватися ad hoc testing – інтуїтивне тестування та тестування документації.

У свою чергу, ці види тестування ПЗ мають безліч різноманітних розгалужень по особливостям тестування. Так, самотестування може ще поділятися на рівні тестування, які в тій чи іншій мірі можуть перетинатися між собою. Ще існує більш детальна розбивка по цілям, хронології, знанням системи, сценаріями тощо.

1.3 Виявлення проблем автоматизації тестування та актуалізація рішень

Автоматизація тестування – це застосування інструментів та технологій для тестування програмного забезпечення з метою зменшення зусиль на тестування, надання можливостей швидше та доступніше. Це допомагає створювати більш якісне програмне забезпечення з меншими зусиллями. Це не тільки виконання автоматичних тестів, це написання і використання різноманітних скриптів для аналізу результатів, підготовки тестових даних, парсинга документів тощо, тобто автоматизація всіх рутинних і повторюваних завдань для полегшення процесу тестування [6].

Загалом тестування можна розділити на 3 види: ручне, автоматизоване і частково автоматизоване:

– ручне тестування. Даний вид означає, що все тестування проводиться руками, без використання скриптів і написання автотестів. Тобто всі тестові дані

складаються і забиваються руками, прогін тестів також здійснюється руками, ну і звичайно відбувається ручний аналіз виконання тестів і ручна фіксація багів;

– автоматизоване тестування. Тут вже присутня максимальна автоматизація процесу тестування. Тобто генеруються автоматично тестові дані, автоматично виконується прогін тестів, і відбувається автоматичний аналіз прогону тестів і фіксація багів;

– частково автоматизоване тестування. В основному використовується в тих випадках, коли або немає необхідності все автоматизувати, або нема можливості, грошей або часу на автоматизацію. В даному виді тестування використовується напівавтоматична генерація тестових даних, тести виконуються в автоматичному чи напівавтоматичному режимі, відбувається ручний аналіз прогону тестів і ручна фіксація багів [7].

Як вже було сказано, автоматизація служить для полегшення роботи тестувальника і поліпшення якості тестування, а значить і продукту. Основні цілі, для яких служить автоматизація, зображені на рисунку В.3, додатку В.

Тобто автоматизація покриває весь спектр робіт з тестування програмного продукту.

Відповідно до інформації порталу «Про Тестінг» [8], автоматизоване тестування ПЗ (Software Automation Testing) – це дії, спрямовані на перевірку тестової системи, в ході якого основні функції та кроки тесту виконуються автоматично за допомогою інструментів для автоматизованого тестування.

Software Automation Test Engineer – це технічний фахівець (тестувальник або розробник програмного забезпечення), що забезпечує створення, налагодження та підтримку працездатного стану тестових скриптів, тестових наборів та інструментів для автоматизованого тестування.

Інструмент для автоматизованого тестування (Automation Test Tool) – це програмне забезпечення, за допомогою якого фахівець з автоматизованого тестування здійснює створення, налагодження, виконання і аналіз результатів прогону тестових скриптів.

Тестовий Скрипт (Test Script) – це набір певних інструкцій, який автоматично виконується в тестовій системі, щоб перевірити, чи працює система належним чином.

Тестовий набір (Test Suite) – це набір тестових сценаріїв або тестових процедур, що виконуються під час конкретного тестового запуску, для перевірки певної частини ПЗ, об'єднаної загальною функціональністю або цілями.

Тести для запуску (Test Run) – це комбінація тестових скриптів або тестових наборів для подальшого спільного запуску (послідовного або паралельного, в залежності від переслідуваних цілей і можливостей інструменту для автоматизованого тестування)».

Для написання і роботи даних артефактів найзручніше застосовувати загальний фреймворк автоматизованого тестування.

Основною проблемою при автоматизації є підготовка та контроль перерахованих артефактів, а також налаштування необхідного середовища для запуску і виконання автоматичних тестів. При цьому є проблема залучення в процес підтримки автоматичних тестів якомога більшої кількості фахівців, що займаються тестуванням, пов'язана з можливим недоліком кваліфікації або спеціалізованих знань мов і середовищ розробки. Для написання автоматичних тестів, більшість інструментів автоматизації вимагають від тестувальника розуміння скриптової мови (Java, Java Script, C# тощо). Зазвичай інструменти дозволяють створювати тести за допомогою запису і відтворення, але, як правило, такі скрипти не надто ефективні, не можуть бути перевикористані і важкі в підтримці.

Фреймворк автоматичного тестування – це набір умов, концепцій і практик, спрямований на перевикористання, зменшення витрат на підтримку і підвищення надійності використання тестів. Використання фреймворку вирішує більшість описаних проблем, що передбачає використання інструменту широким колом фахівців, включаючи розробників і фахівців з ручного тестування.

1.4 Моделі створення автоматизованих тестів

Виділяють переважно дві моделі створення автоматизованих тестів: Record and play, Page Object. Розглянемо кожну з них детальніше:

– модель Record and play. Ця модель передбачає запис дій тестувальника здійснених у вікні браузера. Записуються положення курсора, кліки мишки і клавіш на клавіатурі. Це дозволяє швидко створити робочий автоматизований тест, але такі тести є вкрай крихкими і підтримка таких тестів практично неможлива або дуже трудомістка. При автоматизації тестування розроблюваного проекту такі тести принесуть більше збитків ніж прибутку, адже скрипти записані по моделі Record and play не дозволяють відхилень;

– модель Page Object. У свою чергу модель Page Object передбачає кодування сценарію тесту вручну на одній з мов програмування. На початку відбувається опис об'єктів сторінки, а потім дій, які потрібно зробити з цими об'єктами. Це дозволяє створювати гнучкі і стійкі до мінорних змін тести, наприклад якщо на веб-сторінці відбувся зсув поля реєстрації, але ключові атрибути полів і кнопок залишилися незмінними, то автотест створений за моделлю record and play доведеться перезаписувати буквально з нуля, а тест створений за моделлю Page Object продовжить нормально функціонувати.

Підбиваючи підсумки, в першу чергу, варто зауважити, що ручне і автоматизоване тестування – це взаємодоповнюючі технології. Тому, як у сучасному світі складно випустити якісний продукт без автоматизації, також неможливо обійтися і без ручного тестування. По-друге, хотілося б відзначити, що автоматизовані тести бувають різні і слід вибирати такі тести, які найбільше підійдуть конкретно до даного проекту. Таким чином:

– автоматизація бере вагому участь у проектних процесах і застосовується на всіх рівнях тестування;

– автоматизоване та ручне тестування є взаємодоповнюючими технологіями;

– процес автоматизації все одно потребує людської участі;

– автоматизація потребує певні фінансові додаткові інвестиції, але, в ту ж чергу, дозволяє покращити якість продукту, який розробляється;

– автоматизоване тестування гарантує детерміновану перевірку функціональності;

– автоматизація є розробкою (програмуванням);

– підтримка автотестів потребує додаткових витрат в ході активної зміни системи, що тестується.

1.5 Постановка задачі

На підставі попередньо зроблених висновків дамо характеристику основної мети роботи.

Метою даної роботи є підвищення ефективності тестування e-commerce веб-застосування шляхом дослідження методів автоматизації тестування програмних e-commerce system, вибору найоптимальнішого фреймворку та інструментів для створення автотестів, їх подальшої підтримки, перевикористання та розширення.

Фреймворк повинен бути спроектований таким чином, щоб у майбутньому інженери з автоматизації мали можливість легко підтримувати, перевикористовувати та розширювати тестові набори та дані, якщо в ході тестування продукту будуть раптово змінюватися вимоги або функціонал, і потрібно буде швидко переписати тестові сценарії і програмний код.

Таким чином, необхідно виконати наступні задачі:

– сформулювати та описати вимоги до програмної системи, яка буде розроблятися;

– обрати програмне забезпечення та мову для реалізації фреймворку для автоматизації;

– визначити технології та підхід до реалізації;

– підключити всі необхідні бібліотеки, залежності та плагіни, які будуть вхідною точкою для початку розробки фреймворку;

– реалізувати Page Object архітектуру для представлення класів, методів та локаторів;

– створити тестові сценарії для тестування продукту;

– виконати запуск тестів та представити результати запуску тестів;

– описати архітектуру розробленого фреймворку з висновками;

– описати переваги використання розробленого фреймворку в ході тестування веб-додатків на прикладі e-commerce системи.

2. АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК В ОБЛАСТІ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

2.1 Визначення критеріїв аналізу

В якості критеріїв для аналізу ринку існуючого ПЗ для автоматизації тестування будуть використовуватися:

а) поширеність інструменту – мається на увазі наявність бази користувачів, за рахунок яких збільшується ймовірність знаходження відповідей на ці запитання з використання інструменту, а також ймовірність актуальності знань для розробників. Важливість цього критерію в тому, щоб розробники автотестів мали можливість скористатися накопиченим досвідом інших розробників, а не вирішували всі проблеми «з нуля»;

б) підтримка – чи підтримується фреймворк, тобто як часто виходять оновлення/виправлення помилок. Підтримка важлива для можливості виправлення виявлених помилок, що впливають на стабільність роботи системи;

в) ціна інструмента – варіантами будуть платне ПЗ, безкоштовне, умовно безкоштовне. Ціна інструменту є економічним критерієм, визначаючим розрахунок прямої ефективності від впровадження програмного засобу для автоматизації;

г) відкритість вихідного коду – відкритий вихідний код, або пропрієтарний формат програмного засобу. Цей критерій дозволяє оцінити можливість доопрацювання інструменту для власних потреб;

д) підтримка браузерів – так як розглядається фреймворк саме для тестування веб-додатків, важливу роль відіграє підтримка нових версій популярних браузерів;

е) підтримка операційних систем. Підтримка якомога більшої кількості операційних систем впливає на вибір того чи іншого інструменту, в тому випадку, якщо тестоване ПЗ передбачає мультиплатформеність;

ж) мова програмування, на якому написаний фреймворк впливає для бізнесу на простоту/складність пошуку фахівців для розробки і підтримки тестів і самого фреймворку;

з) можливості взаємодії зі сторонніми бібліотеками в рамках Continuous Integration / Continuous Delivery. Від цього критерію залежить, можливо вбудувати тести в процес безперервної розробки і постачання ПЗ, що є важливим критерієм для оцінки застосовності інструменту;

и) можливості роботи з моделлю BDD (Behaviour Driven Development). Цей критерій відповідає за рішення необхідності наявності спеціалізованих знань з розробки ПЗ у користувачів фреймворку;

к) простота установки, налаштування і зручність користування. Аналогічно пункту и) цього списку, простота використання впливатиме на розширення списку користувачів фреймворку.

2.2 Порівняльна характеристика існуючих розробок

В даному підрозділі в таблиці И.1 (додаток И) представлена порівняльна характеристика існуючих фреймворків для автоматизації тестування, які є поширеними рішеннями в даній області за описаними в попередньому пункті критеріями.

2.2.1 Selenium

На сьогодні, Selenium інструмент є одним з найбільш популярних фреймворків з відкритим вихідним кодом, призначеним для використання в автоматизації тестування. Розроблений десь в далеких двохтисячних і розвивається

протягом наступного десятиліття, за цей час він встиг завоювати серця багатьох тестувальників, особливо тих, у кого в розпорядженні є просунуті навички програмування і досвід написання скриптів. Selenium можна розглядати в якості родоначальника деяких сучасних інструментів автоматизації тестування, наприклад: Katalon Studio, Watir, Protractor і Robot Framework.

Операційних систем, в яких можна працювати з Selenium, є декілька, це Linux, Windows та OS X. Інструмент взаємодіє з багатьма браузерами – Headless браузери, Firefox, Chrome та IE. Вибір мови програмування для написання скриптів також є широким, їх можна реалізувати на C#, Java, Python, Groovy, Ruby, PHP і Perl.

Однак варто зазначити, що даний фреймворк має як плюси, так і мінуси. До переваг можна віднести гнучкість, а також можливість написання складних і ефективних скриптів для тестування розробляються. З іншого боку, для того щоб почати працювати з Selenium, тестувальник повинен володіти неабиякими знаннями в програмуванні і бути готовим приділяти деяку кількість свого часу і енергії для написання спеціальних фреймів і бібліотек, які забезпечують виконання певних функцій в процесі тестування, ліцензія безкоштовна [9].

2.2.2 Katalon Studio

Це всеосяжний набір інструментів для тестування веб, мобільних додатків, API та десктопу, який допомагає подолати загальні проблеми автоматизації тестування веб-інтерфейсу, наприклад, спливаючі вікна, iFrame та час очікування. Робота з даним інструментом не залежить від попередніх навичок тестувальника, можна починати роботу навіть з базовими знаннями в сфері тестування. Адже спеціалісти навіть без технічної підготовки та досвіду програмування здатні самостійно реалізувати проект з автоматизації, а для розробників та тестувальників з більшим досвідом Katalon Studio значно зекономить час для написання нових та

підтримки існуючих скриптів. Він може легко інтегрувати автоматизовані тести у конвеєр CI / CD, чудово працює в зв'язці з такими відомими інструментами під час тестування ПЗ: JIRA, Jenkins і Git. Бонусом інструменту є функція Katalon Analytics, яка надає повноцінну інформацію про стан та процес тестування за допомогою звітів, які можуть бути у вигляді графічних схем, діаграм, метрик та статистичних збірок. Ліцензія безкоштовна [10].

2.2.3 Unified Functional Testing (UFT)

Означає уніфіковане функціональне тестування. Раніше він був відомий як QTP (QuickTest Professional). Один із провідних інструментів автоматизованого функціонального тестування, є продуктом компанії HP у своїй лінійці. Для розробки автоматизованих тестів UFT використовує мову VBScript [11]. Це єдина мова, яка повністю підтримується IDE UFT One. Браузери, які підтримуються: Internet Explorer, Edge, Firefox, Google Chrome. Підтримується операційна система Windows різних версій та CI з Jenkins. UFT One в основному використовується для функціонального, регресійного та сервісного тестування. Використовуючи UFT One, можна автоматизувати дії користувачів у веб- або клієнтському комп'ютерному додатку, а також тестувати та виявляти помилки в одних і тих же діях для різних користувачів, різному наборі даних, в різних операційних системах Windows та / або різних браузерах. Автоматизація за допомогою UFT One, якщо планується та виконується належним чином, може заощадити значний час та гроші порівняно з ручним тестуванням. Він відомий своєю простотою використання та підтримкою з боку постачальника та великої спільноти тестувальників автоматизації. З цієї причини кваліфіковані фахівці UFT One завжди користуються попитом, ліцензія платна [11].

2.2.4 Watir

Це група бібліотек Ruby для автоматизування веб-браузерів. Це дозволяє писати тести, які легко читати та підтримувати. Іншими словами, це простий та гнучкий інструмент. Він являється безкоштовним інструментом з відкритим кодом, який не вимагає витрат на використання цього інструменту. Використовує Ruby, повнофункціональну сучасну мову сценаріїв, а не власний сценарій постачальника. Watir підтримує будь-які додатки – не важливо, на якій технології він розроблений. Він підтримує лише Internet Explorer у Windows. Watir-WebDriver підтримує різні браузери, такі як Chrome, Firefox, Internet Explorer та браузер Opera. Він також працює в безголовому режимі (HTMLUnit). Ruby дає можливість підключатися до баз даних, читати файли даних та електронні таблиці, ліцензія безкоштовна [12].

2.2.5 Test Complete

Це потужний автоматизований інструмент тестування графічного інтерфейсу для мобільних, веб- та десктопних додатків. Дає можливість створювати точні та повторювані автоматизовані тести на багатьох пристроях, платформах та середовищах. Інструмент дозволяє командам писати сценарії різними мовами, включаючи JavaScript, VBScript та Python, або створювати складні тести за допомогою можливостей запису та відтворення. Він постачається із готовою інтеграцією з інструментами CI/CD, такими як Jenkins та Jira, популярними фреймворками з відкритим кодом та інструментами, такими як Selenium та SoapUI, а також власною підтримкою BDD, ліцензія платна [13].

2.2.6 Robot Framework

Представляє собою фреймворк для автоматичного тестування з відкритим вихідним кодом, в якому реалізується підхід тестування на основі ключових слів для приймального тестування і розробки через приймальне тестування (ATDD). Robot Framework надає можливість вирішення різних завдань автоматизації тестування. Однак його можливості можуть бути розширені завдяки впровадженню додаткових бібліотек за допомогою Python і Java. Наприклад, Selenium WebDriver – популярна зовнішня бібліотека, яка використовується в Robot Framework. Тестувальники можуть використовувати Robot Framework як фреймворк для автоматичного тестування не тільки веб-додатків, але для додатків під Android та iOS. Він виявиться простим у вивченні для тих тестувальників, хто вже знайомий з тестуванням на основі ключових слів, ліцензія безкоштовна [14].

Отже, проаналізувавши існуючі аналоги інструментів, які можна було б узяти за основу створення цілісного фреймворку для автоматизації веб-додатку e-commerce системи, можна сказати, що фаворитом є Selenium, так як основними достоїнствами і критеріями, в порівнянні з наявними на ринку продуктами, у нього будуть:

– прозорість: забезпечує високу спритність та максимальну прозорість серед міжфункціональних команд процесу життєвого циклу розробки програмного забезпечення (для розробників, відділу забезпечення якості, клієнтів та керівництва);

– незалежність від платформи: скорочує час тестувальника програмного забезпечення для написання тестових сценаріїв для кожної платформи, що перевіряється. Оскільки це відповідає принципу написання одного тестового сценарію і працює на будь-якій платформі;

– сприяє постійним зусиллям щодо інтеграції: безперервна інтеграція та безперервна доставка є сучасними гнучкими механізмами процесу розробки ПЗ. Селен шляхом автоматизації загального процесу може виступити каталізатором

для стимулювання зусиль організацій щодо успішного впровадження практики безперервної інтеграції (CI);

– скорочує час виконання: Framework Selenium полегшує командам тестування автоматичний паралельний запуск декількох тестових кейсів на платформах із багатьма браузерами. Це скорочує час виконання, забезпечуючи надзвичайну якість випробувань.

Мовою розробки фреймворку вибрана мова Java, як одна з найбільш поширених мов на сьогодні по ряду індексів. Наприклад, відповідно до широко відомих індексом TIOBE Programming Community Index на квітень 2021 року, мова Java займає друге місце за поширеністю з рейтингом 16,38% [15]. Обравши Selenium, можна писати свої сценарії будь-якою з мов програмування, в тому числі і на Java, і Selenium в найкоротші терміни перетворить їх у сумісні з Selenium коди. Так як на відміну від тих же Unified Functional Testing чи Watir, які є аналогом Selenium, вони підтримують тільки по одній мові, що є не дуже добре, або Test Complete, який працює з різними мовами програмування, але не підтримує Java. Крім того, кожна мова, що підтримується Selenium, має спеціальні фреймворки, які допомагають у написанні тестового сценарію для автоматизації тестування Selenium. Отже, обравши Selenium як інструмент для тестування автоматизації, нам не доведеться турбуватися про підтримку мови та фреймворку, оскільки Selenium це зробить за нас.

BDD. На додаток до попереднього пункту, ще більше поширення фреймворк може отримати внаслідок можливості розробляти автотести з використанням технології Behaviour Driven Development, що дозволяє навіть незнайомим ні з одною мовою програмування фахівцям в області тестування ПЗ, як аналізувати поточні написані тести, так і розробляти нові, використовуючи розроблені базові кроки. Наприклад, відкриття певної сторінки, натискання, заповнення веб-елементів, перевірка текстових та інших даних на сторінках, можна робити, не знаючи традиційних мов програмування.

Серед існуючих аналогів цю технологію підтримують лише Test Complete і Watir, про які я вже зазначала, але вони не працюють з Java. А BDD технологія одна

з найбільш ефективних і кращих підходів до розробки автотестів для веб-додатків, які в тому числі представляють собою великі e-commerce системи з низкою вимог і функціоналу, підхід до їх реалізації, підтримки, перевикористання та розширення до яких потрібно ще з самого початку налаштувати таким чином, щоб це все мало структурований, зрозумілий та водночас легковісний вигляд не тільки для розробників автотестів, але і для людей, які не мають достатньої технічної підготовки.

Інтеграція Selenium із Sikuli (візуальне середовище для тестування графічного інтерфейсу за допомогою зображень чи скріншотів) дозволяє перевірити веб-програми, що включають відео та зображення.

Selenium заохочує тестувальників писати скрипт для покриття сценарію однією мовою програмування та запускати (повторно використовувати) ті самі тестові сценарії на багатьох платформах браузера. WebDriver стає частиною стандарту W3C для всіх браузерів і, таким чином, браузери, які на сьогодні існують, автоматично підтримуватимуть Selenium. Цікавим аспектом цього засобу автоматизації тестувань є те, що він пропонує великий набір опцій для тестування, порівняння результатів і нарешті перевірки, чи відповідають вони очікуваній поведінці програми чи ні.

2.3 Порівняльна характеристика інструментів

2.3.1 JBehave

Це один з кращих фреймворків тестування Java, він підтримує BDD. Він є покращеною версією TDD (Test-Driven Development і ATDD (Acceptance Test-Driven Development) розробки на основі тестування та приймального тестування.

BDD покликана зробити тестування більш доступним і інформативним для новачків і фахівців. Вона ставить на перше місце поведінковий фактор, приділяючи особливу увагу взаємодії з користувачем і автоматизації.

Переваги:

– у продуктів відмінні характеристики завдяки продуманим деталям і інтуїтивності фреймворку;

– відмінно працює для BDD, пропонуючи інтеграцію між окремими командами програмістів, зайнятими різними проектами зі схожими специфікаціями;

– у JBehave використовується напівформальна мова, що дуже допомагає розробникам. Наявність словника предметної області допомагає забезпечити узгодженість команди;

– зацікавлені сторони, а також менеджери проектів отримують кращу прозорість QA-груп і високу продуктивність команди розробників, оскільки специфікації мають однаковий формат.

У JBehave, як і у будь-який інший платформи BDD тестування, є єдиний мінус: показник успішності інструменту в основному зводиться до важливості спілкування між учасниками, що працюють над проектом. Між керуючими, тестувальниками, програмістами і зацікавленими сторонами.

Проблеми в спілкуванні можуть призвести до появи сумнівів і питань без відповіді. Це може вилитися в розробку програми, повної помилок, або яка не відповідає вимогам замовника, що призведе до взаємних звинувачень всіх учасників проекту.

Принцип роботи фреймворку JBehave схожий з платформами типу Serenity. Якщо ви хочете розробляти автоматизовані приймальні тести ефективно, варто використовувати його разом з Serenity. Концепція Serenity заснована на BDD розробці, що також дозволяє користувачам складати докладний звіт тестування.

2.3.2 JUnit

Це відкритий фреймворк регресійного тестування, в основному використовується досвідченими розробниками ПЗ для виконання юніт-тестування, прискорення програмування і підвищення якості коду в Java. Основна його мета – дозволити Java-розробникам писати скрипти і реалізовувати повторювані тест-кейси. Це один з найбільш часто використовуваних фреймворків модульного тестування Java.

Найчастіше JUnit застосовується розробниками для невеликих фрагментів коду. Також можна виконувати автоматизовані тести сайтів, комбінуючи Selenium WebDriver з JUnit для автоматизованого тестування Java. При додаванні будь-якого свіжого коду вам буде запропоновано перезавантажити тест-кейси і переконатися у відсутності неполадок.

Серед інших фреймворків тестування, JUnit має ряд особливостей:

– завдяки підтримці функцій Java 8, він може використовуватися як для інтеграційних, так і для юніт-тестів;

– програмісти і розробники, що працюють в цій тестовій середовищі, знаходять її вкрай корисною. Легше читати код і досліджувати наявність вразливостей;

– виявлення помилок на ранній стадії і читабельність підвищують надійність коду;

– практично всі основні IDE, такі як Eclipse, NetBeans, Maven, Ant і IntelliJIDEA, сумісні з фреймворком. Так що ви можете легко написати і запустити модульне тестування безпосередньо з них;

– використовуючи оновлену версію JUnit, можна легко перевірити наявність винятків. Також можна виконувати тест-кейси, написані на попередній версії JUnit.

Недоліки: фреймворк JUnit не може виконувати тести залежностей. Для цього нам потрібен TestNG.

JUnit – це широко використовуваний фреймворк на основі Java. І TestNG і JUnit виконують аналогічну роботу, і їх ключові характеристики в чомусь однакові, за винятком можливостей тестування залежностей. Процедура виконання параметризованого тестування також відрізняється в обох фреймворках.

Так як JUnit використовується дуже давно, він має велику підтримку Java-спільноти. Він також визначено як стандартна платформа модульного тестування шляхом інтеграції Selenium WebDriver для додатків на основі Java. Незважаючи на те, що у TestNG ще мало користувачів і спільнота стає дедалі більше, вибір між фреймворкуми JUnit і TestNG залежить від потреб конкретних додатків.

2.3.3 Serenity

Фреймворк, раніше відомий як Thucydides, є одним з найбільш широко використовуваних відкритих BDD-фреймворків тестування Java. Він допоможе вам в написанні більш чітких, структурованих і настроюються критеріїв приймання. Це розширює спектр можливостей в Selenium WebDriver і JUnit.

З цим фреймворком можна створювати докладні звіти про тестування. Ви також можете зіставляти автоматизоване тестування до своїх потреб. Цей фреймворк автоматизації робить процес Java-розробки більш надійним і ефективним.

Особливості:

– проект з відкритим доступом;

– докладні звіти про результати тестування і реалізації;

– звіт в діловому форматі для кожного тесту.

Переваги:

– цей фреймворк автоматизації допомагає тестувальника створювати тест-кейси для сервісів REST;

– він підтримує грань між описом і розумінням тест-кейса і підходом до його реалізації;

– користувачам не треба витрачати зайвий час на підтримку або створення - власної платформи автоматизації;

– Serenity можна об'єднувати з багатьма фреймворками, наприклад з такими платформами для Java розробки як Spring, іншими фреймворками типу Selenium, JBehave, і інструментами управління проектами типу JIRA.

Недоліки: слабка сторона Serenity схожа з Java фреймворком JBehave. Регулярне спілкування між учасниками проекту необхідно для забезпечення якості підтримки BDD.

Цей фреймворк Java-тестування в основному використовується для написання і звітності критеріїв приймання. Звіти, написані на Serenity, дуже ефективні, корисні, сповнені і інформативні, на відміну від інших BDD фреймворків типу JBehave. Це допомагає розробникам в написанні тест-кейсів автоматизації високої якості. Serenity також підтримує Selenium WebDriver і Rest Assured, дозволяючи QA-інженерам складати критерії автоматизованого приймального тестування простіше і швидше.

2.3.4 TestNG

Є одним з найбільш популярних і потужних фреймворків Java-тестування, що використовуються для інтеграційного, функціонального і юніт-тестування. Був створений Седріком Беустом в 2004 році і зараз оновлений до версії 7. Він є прямим конкурентом таких фреймворків як JUnit. TestNG схожий на JUnit, але доповнений незвичайними анотаціями та класними функціями (не підтримує JUnit).

TestNG можна використовувати практично у всіх типах тестування ПЗ, включаючи end-to-end, модульне, інтеграційне і функціональне тестування. Як TestNG, так і JUnit є фреймворками на основі Java, які дозволяють писати тести і

вивчити кінцеві результати. Якщо тестування пройшло успішно, ви побачите зелену смугу, якщо немає – червону.

Крім усунення проблем старих фреймворків, у TestNG чимало переваг:

– дозволяє створювати і запускати паралельне тестування декількох фрагментів коду;

– під час імплементації тест-кейса можна створювати HTML-звіт;

– тест-кейси можна легко впорядкувати і згрупувати за пріоритетністю;

– виконання тестів набагато простіше, потрібно просто запустити тестування фронтенду і бази даних на фреймворку;

– пріоритети можна легко встановити шляхом параметризації даних і використання анотацій.

Недоліки: TestNG може не підійти, якщо в проекті не потрібна пріоритезація тест-кейсів. До того ж, на установку TestNG потрібно більше часу.

Як ми вже знаємо, платформа тестування JUnit має той же функціонал, що і TestNG, але у неї є деякі недоліки. Коли тест виконується ізольовано, JUnit відмінно себе проявляє. Але при наявності будь-якої залежності ви не зможете контролювати, який тест завершиться раніше. TestNG Java фреймворки допомагають виконувати тест-кейси саме так, як хочеться вам.

3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

3.1 Формування вимог до програмної системи

Спочатку необхідно точно сформулювати основні ідеї, які визначають функціонал системи автоматизованого функціонального тестування. Чітко складені вимоги до проекту є гарантією того, що він буде виконаний максимально якісно і матиме повністю необхідний функціонал.

Таким чином, до розроблюваної системи автоматизованого функціонального тестування ставляться такі вимоги:

– простота створення і підтримки тестів навіть для людини, що раніше не працювала в рамках автоматизації тестування, але володіє навиком програмування;

– ефективність – система повинна бути оптимізованою, швидкодіючою і здатною здійснювати тестування на різних рівнях від інтеграційного до системного;

– наявність розвиненої системи звітів – існує необхідність в автоматичному формуванні різних видів звітів про стан проекту (кількість багів, швидкодія тощо);

– модульність – система повинна складатися з окремих модулів, які будуть відповідати за окрему задачу розробки тестів;

– можливість розширення – функціонал системи повинен бути легко розширюваним.

Архітектура фреймворку для автоматизації зображена на рисунку В.4 додатку В.

3.2 Вибір основної мови програмування

Для початку необхідно обрати мову програмування, за допомогою якої можна було б реалізувати систему, що задовольняє всім вимогам та дозволяє

взаємодіяти з найпопулярнішими інструментами тестування. Вибір був зроблений на користь мови Java.

Це об’єктно-орієнтована та загальна мова програмування, яка допомагає створювати програми та додатки на будь-якій платформі. Java пропонує безліч переваг, які дозволяють вам дотримуватися її.

Будь-яку мову можна вважати простою, якщо її легко вивчити та зрозуміти. Синтаксис Java простий, його легко писати, вивчати, підтримувати та розуміти, код легко налагоджувати. Більше того, Java є менш складною, ніж такі мови, як C та C++, оскільки багато складних особливостей цих мов видаляються з Java, такі як явна концепція вказівників, класи зберігання, перевантаження оператора та багато іншого.

Обравши її для проекту можна підвищити гнучкість і багаторазовість використання коду. Використовуючи концепцію ООП (Об'єктно-орієнтоване програмування), ми можемо легко використовувати об’єкт в інших програмах. Це також допомагає нам підвищити безпеку, пов’язуючи дані та функції в єдине ціле і не дозволяючи доступу до них зовнішньому світу. Це також допомагає організувати більші модулі в менші, щоб їх було легко зрозуміти [16].

Також Java пропонує дуже ефективну користь, надаючи функцію незалежності від платформи, яка є функцією записувати один раз, де завгодно. Складений код, тобто байт-код Java, не залежить від платформи і може працювати на будь-якій машині, незалежно від операційної системи. Ми можемо запустити цей код на будь-якій машині, яка підтримує віртуальну машину Java (JVM).

До того ж, важливим критерієм, яким я керувалась при виборі мови програмування є те, що основною складовою фреймворку для автоматизації буде інтеграція зі Spring фремворком, потужним інструментом для розробки і тестування великих веб-додатків. Детальніше про ньгоо буде описано згодом.

Для розробки проекту була обране середовище (IDE) IntelliJ IDEA Comunity

Edition версії 2019.2 компанії JetBrains. Дана IDE є безкоштовною, призначена для розробки проектів на Java і має великі можливості розширення за рахунок додаткових плагінів.

3.3 Вибір технологій і специфікації

3.3.1 SAP Hybris

В якості тестованого продукту является e-commerce рішення на платформі SAP Hybris (SAP Commerce Cloud).

Ринок електронної комерції стрімко розвивається. SAP йде в ногу з розвитком хмарних технологій для електронної комерції та рішень для цифрового взаємодії з клієнтами. Включаючи модульний модуль SAP C / 4HANA Customer Experience Suite в ту саму хмару, що включає SAP Commerce Cloud, клієнти SAP тепер мають повну технологічну платформу для створення власного наскрізного багатоканального комерційного досвіду для своїх кінцевих споживачів. SAP Commerce Cloud – це комплексне комерційне рішення, яке підтримує все, починаючи від управління замовленнями у фоновому режимі і закінчуючи досвід покупки для кінцевих користувачів [17].

3.3.2 Cucumber

Однією з великих проблем автоматизації тестування є підтримка, формалізація і розуміння тест кейсів. Наприклад якщо в компанію прийшла нова людина, то вона витратить величезну кількість часу, щоб розібратися, що відбувається в момент роботи того чи іншого тесту. Мінімізувати цю проблему допоміг фреймворк Cucumber.

Cucumber використовується для написання програмного коду по методології Behavior-Driven-Development, що перекладається як «розробка через поведінку». Суть методології в тому, щоб спочатку формалізувати вимоги, а потім вже відштовхуючись від цих вимог реалізовувати функціонал. У разі системи автоматизованого тестування цей підхід зручний для формалізації тест кейсів на

природній мові, а потім реалізації кожного кроку у вигляді програмного коду. Це дозволяє внести ясність і чіткість в сам тест кейс, прискорює розуміння його роботи і того, що і як він тестує [18].

3.3.3 Selenium WebDriver

Для роботи з будь-яким сайтом або веб-додатком використовуються браузер. Браузер – прикладне програмне забезпечення для перегляду веб-сторінок. Перш ніж взаємодіяти з сайтом, потрібно вирішити задачу взаємодії з браузером як таким. На допомогу прийшов Selenium WebDriver.

Цей драйвер є ключовим елементом системи і дозволяє імітувати роботу користувача з браузером: кликати на посилання, заповнювати поля, очищати поля,натискати на кнопки, переходити за посиланням тощо. WebDriver здатний взаємодіяти з більшістю популярних браузерів: InternetExplorer, MozillaFireFox, Google Chrome тощо.

3.3.4 Бібліотека тестування jUnit

Далі були проведені дослідження популярних бібліотек для тестування, і вибір був зроблений на користь бібліотеки jUnit. Вона дозволить додати в будь-яке місце нашого тесту якусь точку порівняння фактичного і очікуваного результату. jUnit – бібліотека для модульного тестування ПЗ на мові Java. Належить до родини фреймворків xUnit для різних мов програмування. jUnit породив цілу екосистему розширень – EasyMock, HttpUnit тощо. jUnit був портований на інші мови, включаючи PHP (PHPUnit), C # (NUnit), Python (PyUnit), Fortran (fUnit),),JavaScript (JSUnit) та інші [19].

3.3.5 Система автоматичного складання проекту Maven

Для Java існує система збірки Maven, призначена для автоматичного складання проектів, розроблювана співтовариством Apache Software Foundation. Для отримання структури проекту Maven використовує POM мову, яка являється підмножиною мови XML. Самі ж проектні файли включають в себе інформацію з описом специфікацій, плагінів, порядок конфігурації залежностей з іншими проектами. На рисунку В.5 додатку В представлений звязок компонентів системи автоматизованого тестування.

Архітектура системи передбачає при першому відкритті проекту на новій машині в IDE імпорт файлу pom.xml для автоматичного збирача Maven. Встановлюються залежності між компонентами проекту і завантажуються відсутні бібліотеки.

3.3.6 Spring

В проекті використовується Spring. Це одна з найпопулярніших платформ з відкритим кодом для розробки корпоративних додатків. Він забезпечує комплексну підтримку інфраструктури для розробки додатків на основі Java.

Spring також дозволяє розробнику створювати високопродуктивні, багаторазові та вільні зв’язки корпоративних програм Java.

Інтеграція Cucumber-Spring призначена для спрощення автоматизації тестування. Після інтеграції тестів на Cucumber із Spring, ми зможемо виконувати їх разом із збіркою Maven.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Реалізація системи автоматизованого тестування

4.1.1 Архітектура тестового фреймворку.

Тестовий фреймворк складається з основного кореневого каталогу src, який в свою чергу містить два основні модулі – пакет main для вихідного коду, і пакет test для фіча файлів (додаток В, рисунок В.6).

Пакет main складається під-пакетів java і resources, його структура зображена на рисунку В.7 додатку В. Каталог java містить конфігураційні класи, класи необхідні для запуску тестів, для створення скріншотів, наприклад в пакеті elements знаходяться класи, які містять опис об'єктів з тестованих веб-сторінок, і власноруч створені методи для взаємодії з цими елементами (DropDown, Link, TextBlock і тд).

Власна програмна реалізація методу для взаємодії з елементами на веб-сторінках, що включає в собі функцію нажаття на елемент, представлена у додатку Д в пункті Д.1.

Для взаємодії з Dropdown елементом у додатку Д у пункті Д.2 наведений приклад програмної реалізації методу для вибору значень зі спадаючого списку за певними атрибутами.

Методи для взаємодії з елементами дозволяють системі тестування розуміти описані об'єкти тестованих веб-сторінок та перевірити існування деяких елементів, які тестуються на веб-сторінці на основі WebDriver API.

Пакет framework складається з класів з константами (наприклад, TimeoutConstants, WebElementAttributeConstants), що в свою чергу містять статичні незмінні значення, які використовуються в реалізації програми.

Пакет core.bean складається з допоміжних класів, створення екземплярів яких надалі відбувається у раніше описаному класі ContextConfig, де відбувається їх впровадження до ApplicationContext сховища.

В даному пакеті в папці common знаходяться DTO (Data Transfer Object) класи, які містять сутності для передачі інформації та використання в методах. Це

класи, які містять тільки поля і властивості, не містять методи для маніпуляції з даними. Також розташовані папки з реалізацією абстрактних базових класів та інтерфейсів.

В папці transactional знаходяться класи та інтерфейси, які містять опис компонентів з тестованих веб-сторінок і самі веб-сторінки, та методи необхідні для взаємодії з ними. Також реалізовані методи з певними умовами, які вебдрайвер повинен виконати в ході потрібного тесту. Є класи з тестовою датою та Enum, які використовуються в тестових сценаріях. І є папка flow, яка містить в собі класи за певним функціоналом, в якому зібрані загальні методи, які часто зустрічаються в сценаріях і на потребують реалізації окромого Step definition, про який буде йтися далі, а являють собою зібранням певної послідовної кількості дій, які можна прописати в одном методі і викотристовувати і потрібному кейсі.

Пакет util оперує утильними класами, також відомими як Helper class – це клас, який містить лише статичні методи, він не має статусу і не може бути створений. Він містить купу пов’язаних методів, тому їх можна використовувати повторно у всій програмі. Як приклад DateTimeUtils, який має допоміжні методи для роботи з датою та часом (додаток В, рисунок В.8).

Пакет resources містить в собі конфігураційні файли та класи необхідні для коректної роботи системи, chromedriver.exe версії 89.0.4.

Файл configuration.properties, призначений для того, щоб зберігати в ньому якісь статичні дані необхідні проекту, конфігурації та важливі анотації, призначені для запуску тесту на локальному середовищі (додаток В, рисунок В.9).

В даному фреймворці указуємо адресу посилання з портом для локального запуску тесту. Прописуемо page.load.timeout – 100, це визначає кількість часу, протягом якого Selenium буде чекати завантаження сторінки. І в разі того, якщо ми захочемо запустити одразу всі тести в кілька потоків, то використовуємо атрибут thread.count, в нашому випадку у нього буде значення – 6. Атрибут language.parameter відповідає за те, на якій мові буде відкриватися інтерфейс тестового сайту, цей параметр передається в посилання.

У файлі allure.properties вказується необхідна директорія, куди буде генеруватися звіт з результатами виконаного тесту. В моєму випадку це - allure.results.directory=target/allure-results.

В розробленому тестовому фреймворці будуть використовуватися основні залежності: Junit – версії 4.12, allure-cucumber4-jvm.

Пакет test так само містить в собі два під-пакета java і resources. Пакет resources містить фіча файли. Файли з формалізованими тест кейсами мають розширення .feature і зазвичай носять назву самого тест кейса, наприклад AddToBasket.feature. Зразок тестового сценарію представлений на рисунку В.10, додатку В.

Ключове слово Feature використовується для формалізації назви тест-кейсу, а ключове слово Scenario служить для швидкого опису самого алгоритму тестування або очікуваного результату.

Для зв'язку формалізованого тест-кейсу з його описом на мові програмування використовуються ключові слова: Given, When, Then, And.

Працюють дані ключові слова подібно посиланнях в інтернеті, кажучи про те що існує якийсь метод, який відноситься конкретно до даного кроку тест-кейсу. Cucumber підтримує безліч природних мов для формалізаціі тест кейсів, що робить його одним з найсильніших фреймворків для досягнення простоти і зрозумілості тестів і тест кейсів, які до них відносяться.

Ім'я фічи зазвичай відповідає тому функціоналу, який буде тестуватися, а сам тестовий сценарій реалізується на мові Gherkin. Директорія з файлами .feature зображена на рисунку В.11, додатку В.

Варто зазначити що для роботи з браузером Google Chrome потрібно попередньо завантажити драйвер для Selenium WebDriver з зовнішнього ресурсу. Після складання проект повністю готовий для створення нових і виконання вже існуючих тестів.

Щоб мати змогу використовувати Cucumber у своїй програмі, потрібно скачати необхідні плагіни, і додати бібліотеку Cucumber до свого проекту.

Для запуску фреймворку з середовища Intellij Idea, потрібні такі плагіни як Cucumber for Java та Gherkin.

Java пакет містить java класи, які називаються Step Definition та CucumberTest клас.

Step Definition – це невеликий фрагмент коду з прикріпленим до нього шаблоном або, іншими словами, це java-метод в класі з анотацією над ним. Анотація, за якою слідує шаблон, використовується для зв'язування визначення кроку з усіма відповідними кроками, а код – це те, що Cucumber виконає, коли побачить крок Gherkin. Cucumber знаходить файл визначення кроку за допомогою коду Glue в параметрах Cucumber.

При запуску тесту Cucumber проходить за сценарієм крок за кроком. Взявши крок, він відокремлює ключове слово від опису кроку і намагається знайти в Java-класах пакету зазначеного в опції glue анотацію з регулярним виразом, відповідним опису. Знайшовши збіг, фреймворк викликає метод зі знайденою анотацією. Якщо кілька регулярних виразів задовольняють опису кроку, фреймворк викидає помилку.

CucumberTest являється класом, який буде точкою входу для наших тестів (додаток В, рисунок В.12). Також слід взяти до уваги, що назва класу повинна мати закінчення Test, інакше тести не будуть запускатися.

Нижче наведено структуру класу CucumberTest:

– features – шлях до папки з .feature файлами. Фреймворк буде шукати файли в цій і в усіх дочірніх папках;

– glue – пакет, в якому знаходяться класи з реалізацією кроків і конфігурацій;

– strict – якщо true, то при зустрічі неопрацьованого кроку тест зупиниться з помилкою. False – неопрацьовані кроки пропускаються. За замовчуванням false;

– plugin – опція плагіна використовується для вказівки різних варіантів форматування вихідних звітів;

– monochrome – ми використовуємо монохромний параметр всередині анотації @CucucmberOptions. Цей параметр може бути встановлений як true або false. Якщо встановлено як true: це означає, що висновки консолі для тесту

Cucumber набагато читабельніші та видаляють будь-який нечитабельний символ. false: тоді вивід консолі не такий читабельний, як мав би бути.

При запуску класу CucumberTest відбувається ініціалізація фреймворку Cucumber, який шукає ключові слова Before , Test і After , а потім визиває методи, що відповідають ключовим словам:

– Before. На даній стадії необхідно очистити або створити директорію для звітів і запустити браузер, потім необхідно з файлу .feature отримати ім'я сценарію і вивести в консоль, що почав виконуватися наш сценарій. Якщо зараз виконується безліч сценаріїв, то відбувається перевірка на дублювання сценаріїв, якщо ми вже перевіряли даний сценарій, то повторна перевірка проведена не буде. За ці дії відповідає метод setUp () класу GlobalHooks;

– Test. На даній стадії відбувається ініціалізація об'єкта Cucumber, який буде виконувати тест і взаємодіяти з Selenium WebDriver, і виконання тестового сценарію відповідно до формалізованого тест кейсу;

– After. Після отримання результату тесту, будь він позитивний або негативний, відбувається завершення роботи WebDriver і створення звітних файлів в папці зі звітами. Так само в консоль виводиться інформація про завершення тесту.

Найшвидший спосіб запуску теста це використання значків запуску поруч із необхідною функцією або сценарієм або кліком правої кнопки миші на конкретний сценарій та нажаттям рядку «Run». Є можливість відлатки сценарія, в цьому разі аналогічним способом обирається рядок зі значенням «Debug».

Після завершення запуску тестів в IntelliJ IDEA результати відображаються у вікні інструмента «Run» на вкладці «Test Runner».

Консоль праворуч показує результати поточного тестового сеансу. Це дозволяє побачити детальну інформацію про виконання тесту та про те, чому ваші тести провалились або були проігноровані.

Панель інструментів «Test Runner», розташована над списком результатів тесту, дозволяє показувати та приховувати успішні та проігноровані тести, відображати, скільки часу потрібно для запуску кожного тесту, експорту, імпорту та сортування результатів тесту.

4.1.2 Впровадження BDD підходу

Як було уже раніше зазначено у розділі з обраною специфікацією для проекту, в ході реалізації даного фреймворку, був обраний BDD підхід, де всі тестові сценарії мають таку структуру, яка дозволяє бізнес-вимогам перетворитись на тестові кейси, які переглядаються бізнесом та користувачами за потреби.

Основними критеріями, якими я керувалася в ході вибору структури до написання тестів це були можливість інтеграції Selenium Webdriver з даною технологією і також той факт, що в якості тестового продукту буде виступати e-commerce сайт, який по своїй суті представляє собою великий та складний механізм з широким спектром функціоналу та вимогами до нього.

Тому всі ті переваги, які містить в собі BDD підхід до розробки авто тестів є найоптимальнішим для тестування даного проекту, і далі я дам цьому пояснення. E-commerce – також відома як електронна комерція або інтернет-торгівля, стосується купівлі-продажу товарів або послуг за допомогою Інтернету, а також передачі грошей та даних для здійснення цих операцій.

Електронна комерція часто використовується для позначення продажу фізичних товарів в Інтернеті, але вона також може описувати будь-яку комерційну операцію, яка здійснюється через Інтернет. Потрапляти на подібні електронні веб-магазини покупці можуть через рекомендаційні системи, які мають інформацію про переваги користувачів, їх смаки до вибору товарів чи послуг та загалом інформацію про характеристики, рейтинг та схожість об’єктів, які цікавлять покупця. Вони формують рекомендаційний список потрібних об’єктів, послуг чи товарів основуючись на цій інформаціїї та спрощують процедуру вибору покупця серед широкого різноманіття та спектру послуг чи товарів, що в свою чергу є плюсом для магазинів, так як підвищується рівень продажу товарів певної групи або і об’єктів в цілому [20].

BDD призначений для спільної роботи. Кожен, від замовника до тестувальника, повинен мати можливість легко брати участь у розробці продукту. І кожен може написати сценарії поведінки, оскільки вони написані простою мовою.

Сценарії фокусуються на очікуваній поведінці товару. Кожен сценарій фокусується на одній конкретній речі. Поведінка описується простою мовою, і будь-яку двозначність можна пояснити за допомогою простої бесіди або прикладу. Немає нечитабельного коду або незрозумілого технічного жаргону. Чіткість гарантує, що клієнт отримує те, що хоче клієнт.

Фреймворки BDD спрощують перетворення сценаріїв на автоматизовані тести. Інженеру автоматизації просто потрібно написати метод або функцію для виконання операцій кожного кроку.

Кроки «Given-When-Then» можна повторно використовувати між сценаріями. Основна реалізація кожного кроку не змінюється. Код автоматизації стає дуже модульним. Кроки сценарію можна параметризувати, щоб бути ще більш багаторазовими. Параметризація може допомогти команді прийняти загальний набір кроків, що багаторазово використовуються, і надихає на здоровіші обговорення при написанні сценаріїв.

Scenario outlines дозволяють легко запускати один і той же сценарій з різними комбінаціями значень на вхід. Це простий, але потужний спосіб розширити охоплення тестуванням без дублювання коду, що є недоліком автоматизації тестів.

Суть даної конструкції полягає в тому, що в місця, позначені символами «<>», вставляються дані з таблиці Examples. Тест буде запускатися по черзі для кожного рядка з даної таблиці. Назви колонок повинно збігатися з назвою місць вставки даних (додаток В, рисунок В.13).

BDD має ефект сніжної кулі: сценарії стають простішими та швидшими для написання та автоматизації, оскільки додається більше визначень кроків. Зазвичай сценарії поділяють загальні кроки. Іноді для нових сценаріїв потрібні не що інше, як різні параметри кроку або лише один новий рядок.

І наостанок ще потрібно виділити, те що сценарії BDD легко оновлювати в міру зміни продукту. Звичайну мову легко редагувати. Модульна конструкція

робить зміни в коді автоматизації більш безпечними. Сценарії також можна відфільтрувати за назвою тегу, щоб вирішити, що запускатиметься, а що ні.

4.1.3 Опис об'єктів веб-сторінки

Page Factory в Selenium – це вбудоване розширення Page Object Model, але воно дуже оптимізоване. Це клас, де ми використовуємо анотацію @FindBy, щоб знайти метод WebElement та initElementts() для автоматичної ініціалізації веб-елементів.

Клас Page Factory надає кілька анотацій для пошуку елемента на веб-сторінці, але найпоширенішою є анотація @FindBy, яка використовується для легкого пошуку елементів за допомогою локаторів.

Анотація @FindBy може приймати в якості атрибутів ідентифікатор, ім'я, ім'я тега, ім'я класу, часткове посилання, посилання на текст та розширені стратегії пошуку, такі як XPath, селектори CSS.

На рисунку В.14 додатку В, нижче представлений приклад використання анотації @FindBy та селекторів Xpath та CSS в DefaultMiniBasketComponent класі.

Для того щоб WebDriver знав, до якого елементу слід застосовувати ту чи іншу дію, потрібно описати елементи, які знаходяться на сторінці з допомогою одного з двох способів: XPath або CSS.

Отримавши конкретний опис елемента зі сторінки, необхідно створити клас, з назвою сторінки, що тестується, і описати в ньому елементи.

XPath – це техніка в Selenium, яка дозволяє орієнтуватися в структурі HTML веб-сторінки. Це синтаксис пошуку елементів на веб-сторінках. Використання UXPath у Selenium допомагає знаходити елементи, які не знаходять локатори, такі як ID, клас чи ім'я. XPath у Selenium можна використовувати як у документах HTML, так і в XML [21].

CSS – це окрема мова зі своїм форматом і правилами, але він створений спеціально для HTML. За допомогою CSS, можна описати зовнішній вигляд сторінки і окремих елементів. Наприклад, колір тексту, розмір шрифту, розміри і відступи картинок, рамки навколо елементів, заокруглення в фотографіях тощо [22].

Для вибору елемента на сторінці за допомогою CSS необхідно скласти CSS селектор. Для отримання якісного селектора необхідно відкрити в браузері інструменти розробника, навести курсор на цікавий для нас елемент, у вікні інструментів розробника подсвітиться код, який відповідає за положення даного елемента на сторінці. Після цього необхідно визначити ключові атрибути елемента (клас предок, наявність дочірніх класів, на відмінність специфічних атрибутів тощо) і сформувати CSS селектор.

4.1.4 Структура проектування класів

Структура проекту основується на Page Object підході до проектування коду для автотестів.

Page Object є широковідомим та популярним у використанні в автоматизації шаблоном проектування, який дозволяє розмежовувати логіку тестових сценаріїв від їх програмної реалізації у вигляді коду. За допомогою нього сторінки тестового продукту можна моделювати у вигляді об’єктів. Як результат кожен такий об’єкт буде представляти окремий клас з повним описом конкретної веб-сторінки або її певної частини чи компоненту (наприклад меню), що дозволятиме взаємодіяти з її програмним інтерфейсом. Зазвичай він складається з конструктора, який приймає в якості параметра об’єкт Selenium WebDriver, елементів з локаторами та методів взаємодії з ними, які можуть повертати об’єкти інших класів. Так як одні й ті ж об’єкти сторінок можна буде використовувати в різних програмних реалізаціях, то використанням такого підходу, обсяг коду, який буде повторюватися, буде значно

зменшуватися. Основною перевагою такого методу проектування класів є те, що якщо користувацький інтерфейс буде зазнавати якихсь змін, то в існуючі реалізації доведеться прикласти мінімум змін і зробити виправлення тільки в одному певному місці, а не у всіх випадках, де відбувається виклик та використання функціоналу.

4.1.5 Інтеграція Cucumber зі Spring

Як було раніше зазначено, одним з інструментів для створення фреймворку був обраний Spring. Для того, щоб впровадити інтеграцію Cucumber-Spring, потрібно додати залежності Maven: cucumber-spring.

Оскільки ми хочемо використовувати контекст Spring та ін’єкцію залежностей, то до кожного Page object класу додамо анотацію @Component.

Spring реалізує Dependency Injection, тобто механізм впровадження залежностей, завдяки якому класи, які помічені анотацією @Component не відповідають за створення власних залежностей, а отримують їх ззовні від Spring.

Анотація @Component створює бін, клас, створений Spring та який керується ним, і розміщує його в ApplicationContext – сховище для зберігання бінів.

Використання анотації представлено у додатку Ж на прикладі програмної реалізації DefaultCategoryListingPage класу. Цей клас абстрагує всі дії зі сторінкою на якій знаходяться різні категорії продуктів, тому нам не потрібно дублювати їх в рамках наших тестів на Cucumber.

Щоб отримати бін доданого класу до ApplicationContext сховища, ми використовуємо анотацію @Autowired. Завдяки залежності cucumber-spring та механізму впровадження залежностей, Spring шукає потрібний клас в ApplicationContext сховищі та дає можливість його використовувати. Нижче на рисунку В.15 додатку В представлена реалізацію використання @Autowired анотації в Step Definition. Використовуючи доданий екземпляр впровадженого класу ми можемо оперувати методами, які знаходяться в ньому.

Також у фремворкі є клас ContextConfig, який має альтернативний спосіб створення бін класів за допомогою @Bean анотації (додаток В, рисунок В.16). В ньому містяться методи, які створюють екземпляр класів, що будуть надалі використовуватися в потрібній реалізації. До того ж на кожен метод, де є анотація @Bean додається анотація @Lazy, яка говорить про те, ще бін впровадженого класу буде створений тільки при явному зверненні до екземпляру цього класу [22].

Однією із проблем використання об’єктів сторінок у Java є питання про те, як створювати їх екземпляри. Створення всіх сторінок відбувається в класах об’єктів сторінки. Кожен метод повертає екземпляр об’єкта сторінки, що представляє екран, на якому буде знаходитись додаток після його виконання. Незалежно від того, в тестовому класі чи на об’єктах сторінки нам потрібно в якийсь момент викликати конструктор для всіх об’єктів сторінки в нашому коді. Це одна клопота, яка зникне завдяки введенню залежності Spring. Конструктори не успадковуються в Java, тому це потрібно для кожного об'єкта сторінки. Врешті-решт, ми не хочемо заморочуватися такими технічними завданнями під час написання тестів. Якщо ми хочемо передавати інші об'єкти між сторінками, це стає ще більш заплутаним (нам потрібно керувати декількома конструкторами). І саме додавання ін’єкції залежності Spring, мабуть, одна з найкращих.

Представляючи Spring, ми перетворюємо об'єкти сторінки, тест і WebDriver у компоненти Spring і даємо контейнеру Spring керувати ними. Це означає, що Spring буде створювати екземпляри цих класів і надавати екземпляр, коли це буде потрібно.

Конфігурація Java для цього досить проста. Ми використовуємо @ComponentScan для вказівки пакетів, які Spring буде сканувати. Він знайде всі класи, анотовані @Component, і використає конструктори за замовчуванням для створення їх екземплярів.

За замовчуванням усі компоненти Spring є singletons, що означає, що вони створюються один раз під час запуску контексту Spring, а потім повторно використовуються, коли вони потрібні. Об'єкти сторінки не мають стану, тому це добре підходить [23].

Звичайно, використання ресурсів Spring не обмежується лише об'єктами сторінок або WebDriver. Це вигідний спосіб додати до нашого коду також інші служби (наприклад, для обробки тестових даних, викликів API тощо).

На додачу до цього, ми можемо використовувати Spring для багатьох інших випадків використання, що є ще однією причиною того, що, незважаючи на дещо збільшений відбиток від його бібліотек, ми віддаємо перевагу Spring перед більш легким або навіть доморощеним підходом до ін’єкцій.

4.1.6 Додаткові класи для Selenium WebDriver

Selenium WebDriver володіє потужним API за допомогою якого стає можливим кодування команд для роботи з браузером. Основні команди API Selenuim WebDriver:

а) get() – метод дозволяє перейти по URL на веб сторінку;

б) findElement() – метод дозволяє WebDriver виявити елементна сторінці за допомогою заздалегідь відомих XPath або CSS;

в) click() – метод дозволяє натиснути на будь-який об'єкт на сторінці;

г) sendKeys() – метод дозволяє ввести будь-якої текст в полевведення. Цей метод можна викликати для будь-якого елемента на сторінці,наприклад для симулювання натискання гарячих клавіш;

д) clear() – метод дозволяє очистити текстове поле.

Звернення до WebDriver з використанням даних команд в такому вигляді робить код тестів дуже громіздким і важко читається, тому було прийняте рішення написати класи і методи, що спрощують взаємодію з API. WebDriverFactory – клас містить методи ініціалізації драйвера,потрібного для конкретного браузера, а також методи, які спрощують виклик деяких часто повторюваних функцій WebDriver. Наприклад метод closeDriverRunnable() завершує роботу поточного драйвера і виводить поточне повідомлення в лог і консоль.

4.1.7 Конфігурація запуску тестів

Для того, щоб запустити тест в IDEA, потрібно створити конфігурацію запуску. Для цього потрібно обрати пункт Edit Configurations в меню або в меню «Run». У вікні натиснути плюс в лівому верхньому кутку і вибрати конфігурацію Cucumber java. Як результат створиться безіменна конфігурація. Результат налаштування конфігурації запуску представлений на рисунку В.17 додатку В.

В Main class буде міститися клас нашої програми, що містить метод main, який потрібно виконати при запуску програми, і даємо конфігурації осмислене ім'я.

У текстовому полі Glue вказується ім'я пакету, де зберігаються визначення stepdef.

Working directory має містити робочий каталог, який буде використовуватися для запуску програми. Цей каталог є відправною точкою для всіх відносних вхідних та вихідних шляхів. За замовчуванням поле містить каталог, в якому знаходиться файл проекту.

У Use classpath of module оберемо модуль, шлях класу якого слід використовувати для запуску програми.

Shorten command line, в цьому полі обирається метод, який буде використовуватися для скорочення командного рядка, якщо шлях до класу стає занадто довгим або у програми є багато аргументів віртуальної машини, які перевищують обмеження довжини командного рядка операційної системи. У даній програмі значення none: тобто IntelliJ IDEA не скоротить довгий шлях до класу. Якщо командний рядок перевищує обмеження операційної системи, IntelliJ IDEA не зможе запустити програму та відобразить повідомлення із пропозицією вказати метод скорочення.

4.1.8 Характеристика результатної інформації

Результуюча інформація, яка надходить з фреймворку, який розробляється, є відомостями про прогони тестів, представлені в різних формах для різних категорій користувачів. Логування інформації відбувається в консоль, або у вигляді запису в файли (рисунок В.18, додаток В).

Логування можна розділяти на рівні, що відрізняються вищою або нижчою детальністю дій, які будуть логуватися для різних цілей – починаючи з цілей налагодження тестів, і закінчуючи метою перегляду короткої інформації про те, які випробування і методи були запущені. Кінцевими користувачами сгенерованої інформації будуть, в основному, розробники автотестів і тестувальники, що володіють просунутим рівнем знань в мові програмування, на якому реалізований фреймворк. Логування у фреймворку здійснюється практично кожним класом, використовуючи об'єкт Logger бібліотеки slf4j.

4.1.9 Система звітів по виконаним тестам

Існує безліч цікавих платформ тестування для різних мов програмування. Але, на жаль, не всі з них можуть створити та забезпечити результати проходження тесту дійсно у зручному, читабельному та інформативному вигляді. В якості інструменту для генерації звітів буде використовуватися Allure Reporter від Qameta розробників.

Як було вже раніше зазначено, у проекті є файл allure.properties, де вказується необхідна директорія, куди буде генеруватися звіт з результатами виконаного тесту, щоб уникнути створення пропущених цільових результатів.

Створення будь-якого звіту відбувається у два етапа. Першим кроком являється те, що в той час коли виконується тест, бібліотека під назвою адаптер,

приєднана до тестової основи, зберігає інформацію про виконані тести у файлах XML. Під час формування звіту (другий крок) файли XML розширення перетворюються у звіт у вигляді HTML.

Налаштування вихідного каталогу знаходиться у класі CucumberReportGenerator який містить конфігурації та шлях до директорії куди буде генеруватися звіт, а також метод generateReport(), який і буде виконувати саму генерацію.

Щоб створити звіт HTML і автоматично відкрити його у веб-браузері, потрібно виконати такі команди в командному рядку в консолі на рівні папки target в директорії проекту: allure generate allure-results -o allure-report – звіт буде сформовано у папці allure-report. Процес генерації Allure звіту представлений на рисунку В.19 додатку В.

Коли звіт сформовано, ми можемо відкрити його у своєму системному браузері за замовчуванням прописавши: allure open allure-report.

Структура звітування Allure чудово працює з будь-якими тестовими системами, такими як TestNG, JUnit тощо. Нижче наведено приклад реалізації звітів Allure у Selenium за допомогою JUnit, Java та Maven.

Типовий звіт складається з вкладки «Overview», навігаційної панелі, декількох вкладок для різних видів представлення даних тесту та сторінок тестових кейсів для кожного окремого тесту, які представлені на рисунку Г.1 додатку Г. Кожен звіт Allure підтримується деревоподібною структурою даних, яка представляє процес виконання тесту. Різні вкладки дозволяють перемикатися між поданнями оригінальної структури даних, надаючи інший погляд.

На сторінці огляду розміщено кілька віджетів за замовчуванням, що представляють основні характеристики проекту та тестового середовища.

Вкладка «Categories», яка забражена на рисунку Г.2 додатку Г дає вам можливість створити власну класифікацію дефектів, яка застосовуватиметься до результатів тесту.

За замовчуванням існує дві категорії дефектів:

– дефекти продукту (невдалі тести);

– тестові дефекти (непрацюючі тести).

На вкладці «Suites» можна знайти стандартне структурне представлення виконаних тестів, згрупованих за наборами та класами (додаток Г, рисунок Г.3).

Графіки дозволяють бачити різну статистику, зібрану з даних тесту: розподіл статусів або діаграми тяжкості та тривалості (додаток Г, рисунок Г.4).

Вкладка «Timeline», яка зображена на рисунку Г.5 додатку Г, візуалізує ретроспективу виконання тестів, адаптери Allure збирають точні терміни тестів, і тут на цій вкладці вони розташовані відповідно до їх послідовної або паралельної структури синхронізації.

Для підходу, керованого поведінкою, ця вкладка групує результати тестування за тегами Epic, Feature та Story.

Вкладка «Behaviors» представляє деревоподібний макет результатів тесту, згрупований за різними пакетами (додаток Г, рисунок Г.6).

На рисунку Г.7 додатку Г представлений звіт за результатом успішного проходження певного тестового сценарію.

Ми також повинні зробити знімок екрана про помилку тесту та прикріпити звіт за допомогою анотації @Attachment. Ми використовуємо Junit прослуховувач для того, щоб робити знімки екрана при падінні тесту, в даному випадку вся логіка конфігураціїї скріншотів у Allure звіті знаходиться у класі GlobalHooks.

Наведемо приклад програмної реалізації допоміжних методів, які будуть надавати додаткову інформацію за допомогою анотації @Attachment до згенерованого Allure звіту про поточне посилання на сторінку, де сталося падіння тесту, отримання логування браузера та скріншот з місця падіння, і також точний час падіння тесту (додаток Е).

Результат конфігурації анотації @Attachment можемо спостерігати нижче на рисунку Г.8 додатку Г.

Ця владка з додатковою інформацією розташована в меню «Suites» під візуалізацією певного тестового сценарію, результати виконання якого потрібно проаналізувати. Кожне значення містить в собі файли з детальної інформацією про проходження тесту.

ВИСНОВКИ

Завданням цієї кваліфікаційної роботи було дослідження методів автоматизації веб-додатків e-commerce систем та розробка архітектури фреймворку автоматизації тестування e-commerce веб-додатків. Розроблене рішення дозволить скоротити як час на проведення тестування, так і кількість необхідних для проведення тестування фахівців, що, безсумнівно, сприятиме збільшенню ефективності компанії, яка впроваджуватиме цю систему в діяльність відділу тестування програмного забезпечення.

В результаті роботи була створена система, яка об'єднує компоненти для роботи з Selenium WebDriver, для роботи з GUI-інтерфейсом веб-додатків, для формування, генерації і зберігання тестових даних, для роботи з логами, а також компонент для створення звітів по тестуванню за допомогою зовнішнього інструменту Allure, який за допомогою свого інтерфейсу, надає зручність та ефективність для аналізу результатів виконаних тестів. Особливо якщо кількість тестових сценаріїв будуть з часом збільшуватися і розширюватися в окремі тестові набори та сортуватися за окремою функціональністю.

При розробці були враховані особливості веб-додатків, такі як: робота з конкретними браузерами (версіями браузерів), робота під конкретними операційними системами, для чого був обраний універсальний і розширюваний інструмент Selenium WebDriver.

Також враховуючи той факт, що в якості тестового продукту був обраний веб-додаток e-commerce сайт, то впровадження Page Object шаблону та Spring інструменту є немаловажливим. Адже, якщо б фреймворк не використовував шаблон Page Jbject, тобто тестові сценарії, методи, їх реалізація, локатори не були б структуровано реалізовані в різні класи, а наприклад усі локатори були б упаковані до самого тестового класу і якщо ми б хотіли повторно використовувати їх у кількох тестах, нам потрібно б було їх дублювати (або ще гірше, посилатися на них у різних тестових класах). Очевидно, що це могло б стати великою проблемою

для пітримки великих тестових наборів. А так як у фреймворці впроваджена Page Object реалізація, то ми успішно розмістили елементи інтерфейсу та всі взаємодії з ними в окремі об'єкти Java. Це інкапсулює технічні деталі тесту та робить (функціональні) етапи тесту більш чіткими. Ми можемо легко повторно використовувати об’єкти сторінки під час декількох тестів. Якщо макет нашої програми змінюється, в ідеалі нам потрібно лише торкнутися об’єктів сторінки. Якщо ми хочемо змінити або додати тести, наші існуючі об’єкти сторінки вже стануть хорошою відправною точкою.

До того ж, так як в якості тестового продукту була обрана e-commerce система, веб-додаток мережі онлайн магазинів, який по своїй суті представляє собою великий та складний механізм з широким спектром функціоналу та вимогами до нього. Тому всі ті переваги, які містить в собі BDD підхід до розробки авто тестів є найоптимальнішим для тестування даного проекту. Адже всі тестові сценарії мають таку структуру, яка дозволяє бізнес-вимогам перетворитись на тестові кейси, які переглядаються бізнесом та користувачами за потреби на мові, яка зарозуміла навіть нетехнічним спеціалістам.

Таким чином, була розроблена максимально універсальна система, яка забезпечує максимально зручну можливість розширення під конкретні системи і вимоги.

У роботі був проведений аналіз вже існуючих фреймворків автоматизованого тестування, описані переваги і недоліки обраного фреймворку в порівнянні з ними.

У підсумку можна зробити висновок, що розроблене рішення має достатню кількість достоїнств, в порівнянні з існуючими розробками, що дозволяє рекомендувати рішення до розгляду питання про впровадження. Розроблений фреймворк може бути використаний широким колом як індивідуальних розробників програмного забезпечення, так і корпоративними користувачами, такими як: відділи/управління тестування в рамках організаційних структур відділів / управлінь розробки програмного забезпечення.

Подальший розвиток програмного фреймворку може припускати підтримку і удосконалення наявної реалізації архітектури, а також, розширення новим

тестовим функціоналом, бібліотеками та інструментами, які можуть в майбутньому ще більш спростити роботу з тестовими скриптами. Такі активності будуть включати в себе розробку більш універсальних сценаріїв та тестових даних, оптимізація і прискорення роботи в ході запуска тестів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. Москва. МИФИ, 2006. – 157 с.
2. Баранцев А. Онлайн-курс Программирование на Python для тестировщиков [Електронний ресурс]: Москва, 2010-2018. / URL: https://trainings/schedule?task=3&cid=233 (дата звернення: 27.02.2021)
3. Vechur O., Chayka A., Chemodakov Y. The component method of scene analysis and object recognition. Second IEEE International Wokshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003, p.197-200.
4. Turevska O., Shubin I. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. Information Technologies in Innovation Business Conference (ITIB), 2015, с. 93-96.
5. Vechur O., Shevchenko D. Evaluation and optimization of software product infrastructure. East European Scientific Journal. Printed in the «Jerozolimskie 85/21, 02-001 Warsaw, Poland», 2020.
6. Карбасов Д., Пасевич К. Тестирование ПО с использованием инструментов HP Mercury [Електронний ресурс]: Москва, 2008-2018. / URL: http://softwaretesting.ru/library/vendors/162-hp-mercury#6 (дата звернення 23.02.2021)
7. Савкин В. Принципы управления качеством программ [Електронний ресурс]: Москва. Открытые системы, 2008-2018. URL: http://www.osp.ru/os/2008/06/5344965 (дата звернення: 27.02.2021)
8. Портал Про Тестинг. Тестирование Программного Обеспечения / URL: http://www.protesting.ru (дата звернення: 27.02.2021)
9. Selenium IDE. Documentation / URL: https://www.selenium.dev/documentation/en/webdriver/ (дата звернення: 28.02.2021)

10. Katalon Automation Recorder Quickstart / URL: https://www.katalon.com/resources-center/blog/katalon-automation-recorder/ (дата звернення: 10.03.2021)

11. Micro Focus or one of its affiliates. UFT One / URL: https://www.microfocus.com/en-us/products/uft-one/overview (дата звернення: 10.03.2021)

12. Watir / URL: http://watir.com/ (дата звернення: 12.03.2021)

13. SmartBear / URL: https://smartbear.com/ (дата звернення: 12.03.2021)

14. Robot Framework / URL: https://robotframework.org/#learning (дата звернення: 13.03.2021)

15. TIOBE Software BV / URL: https://www.tiobe.com/tiobe-index/ (дата звернення: 13.03.2021)

16. Вечур О., Дударь З., Городищер Р. Застосування функціонального підходу в паралельному програмуванні. East European Journal of Enterprise Technologies, том4, випуск 2, PC Technology Center, 2010, p.17-20.

17. SAP HANA / URL: https://www.sap.com/products/hana.html (дата звернення: 13.03.2021)

18. Automated testing with Selenium and Cucumber / URL: https://cucumber.io/docs/guides/browser-automation/ (дата звернення: 14.03.2021)

19. Dor Ma'ayan. The Quality of Junit Tests Conference: SQUADE@ICSE, Gothenburg, 2018 / URL: https://www.researchgate.net/publication/323847238_The_Quality_of_Junit_Tests (дата звернення: 15.03.2021)

20. Leshchynskyi V. Principles of explanation in e-commerce system based on sales dynamics. COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES KHARKIV, APRIL 2020, p.76-77.

21. B. Douglas Blansit. An Introduction to Cascading Style Sheets (CSS) / URL: https://www.tandfonline.com/doi/figure/10.1080/15424060802453811 (дата звернення: 18.03.2021)

22. Page object models / URL: https://www.selenium.dev/documentation/en/guidelines_and_recommendations/page_object_models/ (дата звернення: 04.04.2021)

23. Spring Framework / URL: https://spring.io/projects/spring-framework (дата звернення: 04.04.2021)