МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІМ. Ю.М. ПОТЕБНІ ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему Особливості використання фреймворку Selenium при тестуванні Веб-застосунків

Виконав: студент 2 курсу, групи 8.1212-іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
С.С. Затоковий
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. Коломоєць Г.П.
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дісітел
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення_____
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_ __ Т.В. Критська_
"_01_" ___вересня____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Затоковий Сергій Сергійович
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості використання фреймворку Selenium при тестуванні Веб-застосунків

керівник роботи _____доцент, к.ф.-м.н. Коломоєць Г.П.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___ЗНУ 1577-с від 09.10.2023 р.
2. Строк подання студентом кваліфікаційної роботи_____28.11.2023_____
3. Вихідні дані магістерської роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розпізнавання мов та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) 15 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	різвище, ініціали та посада онсультанта	ідпис, дата авдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів магістерської роботи	Строк виконання тапів магістерської роботи	Примітка
1	наліз предметної області	02.09-05.09.23	виконано
2	ормулювання основної задачі дипломної роботи узгодження її з науковим керівником	06.09-10.09.23	виконано
3	орівняння можливостей та засобів фреймворків	14.09-26.09.23	виконано
4	ослідження фреймворків для тестування Веб-стосунків	27.09-01.10.23	виконано
5	згодження подальших дій з науковим керівником	02.10-03.10.23	виконано
6	изначення функціональності, яка буде стуватися, розробка тестових вимог.	04.10-10.10.23	виконано
7	озгортання тестового середовища та створення	11.10-12.10.23	виконано
8	творення та прогонка димових тестів (тестів новної функціональності), на кожному з реймворків	10.10-21.10.23	виконано
9	озробка тестових сценаріїв та проведення стування обраного Веб-застосунку	22.10-29.10.23	виконано
10	наліз результатів	30.10-04.11.23	виконано
11	аписання результатів дослідження	06.11-12.11.23	виконано
12	формлення звіту	14.11-10.12.23	виконано

Студент _____ С.С. Затоковий
(підпис) (прізвище та ініціали)

Керівник роботи _____ Г. П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок : 89

Рисунків: 5

Таблиць: 1

Джерел: 21

Затоковий С. С. Особливості використання фреймворку Selenium при тестуванні Веб-застосунків: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Коломоєць Г.П. Запоріжжя : ЗНУ, 2023. 89 с.

Робота присвячена розгляду ключових аспектів використання фреймворку Selenium для автоматизації тестування Веб-застосунків. У роботі розглядаються основні можливості фреймворку, його взаємодія з браузерами.–Досліджується архітектура Selenium та основні засоби технології Selenium WebDriver. Аналізуються стратегії локаторів для ефективного вибору елементів на веб-сторінках та методи управління очікуванням, спрямовані на обробку асинхронних операцій Веб-застосунків.

Також в роботі розглядаються фреймворки автоматизації тестування Playwright та Robot Framework та виконується їх порівняльне дослідження з Selenium WebDriver для визначення переваг та недоліків кожного.

За результатами проведенного дослідження надані рекомендації щодо застосування Selenium WebDriver для автоматизованого тестування Веб-застосунків.

Ключові слова: інструменти тестування, застосунок, інтерфейс, тестовий сценарій, Java, Android, Appium, Katalon, Espresso, Instagram.

SUMMARY

Pages: 89

Figures: 5

Tables: 1

Sources: 21

Zatokovyi S. S. Peculiarities of using the Selenium framework in testing Web applications: master's qualification thesis of specialty 121 "Software engineering" Academic Supervisor Kolomoets G.P. Zaporizhzhia: ZNU, 2023. 89.

The work is devoted to the consideration of key aspects of using the Selenium framework to automate testing of Web applications. The work discusses the main features of the framework, its interaction with browsers. Selenium architecture and the main means of Selenium WebDriver technology are studied. The strategies of locators for effective selection of elements on web pages and methods of expectation management aimed at processing of asynchronous operations of Web applications are analyzed.

The paper also examines the testing automation frameworks Playwright and Robot Framework and performs their comparative study with Selenium WebDriver to determine the advantages and disadvantages of each.

Based on the results of the study, recommendations were given for the use of Selenium WebDriver for automated testing of Web applications.

Keywords: testing tools, application, interface, test scenario, Java, Android, Appium, Katalon, Espresso, Instagram.

ЗМІСТ

ВСТУП

Актуальність теми

Зростання популярності Веб-застосунків веде за собою і зростання вимог до якості продукту. Автоматизовані тести Веб-застосунків дозволяють виконувати повторювані тестові сценарії швидше та ефективніше, якщо порівнювати з ручним тестуванням. Вони дозволяють зменшити час для тестування продукту та забезпечити більш швидкий цикл розробки. Також автоматизовані тести виконуються з більш високою точністю аніж ручні. Враховуючи ці моменти, можна вважати дослідження технологій автоматизованого тестуванн Веб-застосунків актуальним.

Мета і завдання дослідження

Метою роботи порівняльне дослідження вільно розповсюджуваних популярних фреймворків аавтматизованого тестування Веб застосунків:~~и~~ Selenium WebDriver, Robot Framework та Playwright з метою надання рекомендацій щодо їх використання у певних ситуаціях.

Об’єкт дослідження

Об’єктом дослідження є Веб-сайт Steam, який надає послуги дистрибуції багатокористувацьких ігор і спілкування гравців.

Предмет дослідження

Предметом дослідження є засоби автоматизованого тестування Веб-застосунків: Selenium WebDriver, Robot Framework та Playwright.

Методи дослідження

Для досягнення мети дослідження були виконані наступні завдання:

1. Виконаний огляд інформаційних джерел.
2. Визначення критеріїв порівняння досліджуваних фреймворків.
3. Вибір застосунку, що буде тестуватися, та вивчення його основної функціональності.
4. Розробка тест-плану та тестових сценаріїв.
5. Встановлення та налаштування фреймворків автоматизованого тестування.
6. Виконання тестів та аналіз.
7. Формулювання висновків та рекомендацій.

Наукова новизна одержаних результатів

До наукової новизни роботи можна віднести аналіз результатів порівняльного дослідження тестових фреймворків Selenium Web Driver, Robot Framework та Playwright.

Практичне значення одержаних результатів

Результати дослідження надають ~~об'єктивну~~ інформацію про переваги та недоліки Selenium WebDriver порівняно з іншими популярними фреймворками автоматизованого тестування, що є практично значущим для розробників та тестувальників Веб-застосунків.

Апробація одержаних результатів

Результати дослідження були представлені на XVI науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023»[**Помилка! Джерело**

посилання не знайдено.] , а також на XXVIII науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету.

Глосарій

Автоматизоване тестування — це процес використання програмних засобів для виконання тестів на автоматизовану перевірку функціональності, надійності та продуктивності мобільних застосунків.

Веб-застосунки — це програмне забезпечення, яке використовується через веб-браузер та доступне за допомогою Інтернету. Вони дозволяють користувачам виконувати різноманітні завдання та функції, такі як перегляд інформації, спілкування, робота з даними, онлайн-торгівля, ігри та багато іншого.

Selenium — це набір інструментів та бібліотек для автоматизації веб-браузерів. Він надає програмні інтерфейси для взаємодії з веб-браузерами, що дозволяє автоматизувати дії, які зазвичай виконуються користувачем на веб-сторінках.

Selenium WebDriver — це одна з основних складових бібліотеки Selenium. Він є інструментом автоматизації веб-браузерів, який надає програмні інтерфейси для взаємодії з веб-браузерами безпосередньо з допомогою коду.

Playwright — це бібліотека автоматизації з відкритим вихідним кодом для тестування браузерів та очищення веб-сторінок, розроблена Microsoft.

Robot Framework — фреймворк для розробки приймальних автотестів Це keyword-driven testing фреймворк, який надає табличне форматування.

IntelliJ IDEA — інтегрованою середовищем розробки для розробки програмного забезпечення на мовах програмування Java, Kotlin, Groovy, Scala та інших.

Python — високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним керуванням пам'яттю

Node (Node.js) — програмна платформа, заснована на движку V8, перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення.

Test Runner Frameworks - інструменти або середовища, які надають можливість запускати та виконувати тестові сценарії, керувати порядком їх виконання та збирати результати виконання.

JUnit — фреймворк для модульного тестування мовою Java.

Steam — онлайн-сервіс, широко поширений серед комп'ютерних ігор та програм, що підтримується компанією Valve.

Test Suite — поняття, що використовується в контексті тестування програмного забезпечення. Він вказує на групу тестів, які об'єднуються разом для виконання певної функціональності або тестування певної частини програми.

Smoke-тести — відомі як "перевірочні" тести або "швидкі" тести, є першим рівнем тестування програмного забезпечення. Вони призначені для швидкого перевіряння основних функцій та стабільності системи після внесення змін або під час випуску нової версії.

Фреймворк автоматизації тестування (Test automation framework) — це сукупність інструментів, бібліотек та прийомів, що допомагають забезпечити структуру та організацію автоматизованих тестів.

РОЗДІЛ 1 АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ ТА СУЧАСНІ ІНСТРУМЕНТИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

1.1 Огляд автоматизованого тестування Веб-застосунків

Автоматизоване тестування є складовою процесу розробки програмного забезпечення, що забезпечує покращення якості продукту, ефективності розробки а також зменшення витрат часу та ресурсів.

У атоматизованому тестуванні є багато можливостей, умов, особливостей для тестування програмного продукту, що є критично важливою частиною процесу розробки для забезпечення високої якості веб-додатків. Це включає в себе автоматизацію тестів як інтерфейсу, функціональності, продуктивності та безпеки веб-застосунків[22].

А також для автоматизованого тестування є багато різних інструментів для реалізації всіх умов та можливостей тестування але і вибір конкретного інструменту може залежати від різних факторів, таких як тип застосунку, технології використовуваного стеку, мова програмування та конкретні особисті потреби. Всі інструменти мають свої переваги та недоліки з використанням, це як правило специфічні особливості кожного, швидкодія, доступність, легкість розуміння та написання тестів і багато чого іншого. Але у кожної людини є своє бачення на переваги та недоліки, бо хтось буде як перевагу враховувати швидкість написання та виконання тестів, не враховуючи що інструмент має вузьку спільоту для обміну знаннями, та мати маленькі ресурси документацій та навчальних матеріалів.

Тому використання автоматизації тестування Ве-застосунків має свої переваги та недоліки. Автоматизація стала дуже популярною, та вона є досить універсальним рішенням для підтримки якості програмного забезпечення. Але багато чого залежить від обраного інструменту, через що треба зважити всі за та проти кожного і обрати самий універсальний[22].

1.1.1 Види автоматизованого тестування Веб-застосунків

Автоматизоване тестування Веб-застосунків включає у себе багато різноманітних видів тестування, які спрямовані на перевірку різних аспектів функціональності, продуктивності, а також безпеки, взаємодії та інших характеристик застосунків[18].

Нижче вказано перелік деяких видів автоматизованого тестування Веб-застосунків:

1. Тестування інтерфейсу (UI тестування): Перевірка коректності та ефективності взаємодії користувача з інтерфейсом веб-застосунку.
2. Тестування функціональності: Перевірка того, чи виконуються функціональні вимоги застосунку.
3. Тестування регресії: Виявлення можливих помилок або невідповідностей у функціональності після внесення змін в код або додавання нового функціоналу.
4. Тестування продуктивності: Оцінка продуктивності веб-застосунку під час роботи з різними рівнями навантаження.
5. Тестування кросс-браузерної сумісності: Перевірка того, як веб-застосунок працює на різних веб-браузерах.

Як правило видів тестування на багато більше аніж приведених вище. На кожному виді тестування спеціалізується свій інструмент, який необхідно правильно обрати для умпішного тестування.

1.1.2 Сучасні підходи реалізовані в автоматизації

Автоматизоване тестування включає в себе використання різних підходів та методів для покращення ефективності тестування програмного забезпечення.

1. Тестування на основі ключових слів (Keyword-driven testing): цей підхід використовує ключові слова, щоб описати тести та їхні дії. Тестувальник може створювати скрипти тестування, використовуючи ці ключові слова, що дозволяє зменшити кількість коду та спростити процес тестування.
2. Тестування на основі моделей (Model-based testing): цей підхід використовує моделі програми, щоб автоматично створювати тести. Це дозволяє ефективно виявляти помилки та пропуски, що можуть бути пропущені вручну.
3. Тестування на основі скріншотів (Screenshot-based testing): цей підхід використовує знімки екрану для автоматизованого тестування. Це дозволяє тестувальникам швидко перевірити, чи відображає програма коректний інтерфейс користувача та чи працює вона на різних пристроях.
4. Тестування на основі ключових моментів (Keynote-driven testing): цей підхід використовує ключові моменти програми, щоб автоматично створювати тести. Ключові моменти - це функції та функціональні можливості програми, які є важливими для її коректної роботи.
5. Тестування на основі масштабованих тестових платформ (Scaled Testing Platforms): цей підхід використовує великі тестові платформи для автоматизованого тестування програмного забезпечення. Це дозволяє тестувати програмне забезпечення на різних пристроях та у разних умовах, що забезпечує високу якість тестування та покриття всіх можливих сценаріїв використання програмного забезпечення.
6. Тестування на основі розумних контрактів (Smart Contract Testing): цей підхід використовує технологію блокчейн та розумні контракти для автоматизованого тестування децентралізованих програм. Тестувальники можуть використовувати розумні контракти, щоб перевірити, чи працює програма належним чином та чи відповідає вона вимогам.

7. Тестування на основі машинного навчання (Machine Learning Testing): цей підхід використовує машинне навчання для автоматизованого тестування програмного забезпечення. Алгоритми машинного навчання можуть бути використані для автоматичного виявлення помилок та прогнозування можливих проблем у програмі.
8. Тестування на основі імітації (Simulation-based Testing): цей підхід використовує імітацію реального середовища для автоматизованого тестування програмного забезпечення. Це дозволяє тестувальникам використовувати програмне забезпечення в різних умовах та на різних пристроях, щоб переконатися, що вона працює належним чином.

Ці підходи до автоматизованого тестування дозволяють ефективно виявляти помилки та проблеми в програмному забезпеченні, що дозволяє розробникам та тестувальникам підтримувати високу якість програмного забезпечення[18].

1.1.3 Задачі автоматизованого тестування

Автоматизоване тестування має на меті підвищити ефективність тестування програмного забезпечення та зменшити витрати на тестування. Ось декілька типових задач автоматизованого тестування:

1. Функціональне тестування
2. Тестування навантаження
3. Тестування відповідності
4. Регресійне тестування
5. Тестування інтерфейсу користувача
6. Тестування безпеки
7. Тестування продуктивності

Розглянемо пункт "Функціональне тестування". Це один із видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним замовнику функціоналом.

Залежно від мети, функціональне тестування ми проводимо на основі функціональних вимог, зазначених у специфікації вимог. При цьому для тестування створюються тестові випадки (test cases), створення яких враховує пріоритетність функцій ПЗ, які необхідно покрити тестами. Таким чином ми можемо переконатися в тому, що всі функції продукту, що розробляється, працюють коректно при різних типах вхідних даних, їх комбінацій, кількості і т.д[22].

А також на основі бізнес-процесів, які має забезпечити додаток. У цьому випадку, нас цікавить не так працездатність окремих функцій ПЗ, як коректність

виконуваних операцій, з точки зору сценаріїв використання системи. Таким чином, тестування в даному випадку буде грунтуватися на варіантах використання системи (use cases)[22].

Друга задача автоматизації полягає в тестуванні навантаження, це задача що імітує роботу певної кількості бізнес-користувачів на якомусь загальному (розділюваному ними) ресурсі.

Мета якого є в тому щоб переконатися що застосунок однаково якісно працює як з одним, так і з безліччю користувачів. При цьому час запиту від юзера до сервера і назад не має змінюватися. Щоправда, в реальному житті він все одно змінюється. Тому необхідно перевірити, що зміни не настільки критичні, щоб дратувати користувачів.

Також дозволяє перевірити працездатність застосунку під значним трафіком. Сайт чи застосунок під навалою великої кількості користувачів не повинен постійно перезавантажуватись та крутити спінери. Яскравий приклад — коли інтернет-магазин розпочинає акцію зі знижками на всі товари. В такому випадку трафік може зрости, припустимо, з 10000 до мільйона користувачів. А це може критично вплинути на перфоманс. Спеціалісти з навантажувального тестування мають перевірити, як сайт поводиться за таких умов[18].

Та дає можливість оцінити якість роботи застосунку при нормальному навантаженні. Наведу приклад з власної практики. Маємо хмарний застосунок на мікросервісах, в одному з яких на тестах знайдено проблему. Цей мікросервіс перезавантажувався за тиждень п’ять разів без зрозумілих причин. Проаналізувавши ситуацію, ми виявили, що розробники так «оптимізували» код, що кожен запит від клієнта до сервера зберігався в оперативній пам’яті. Для сервісу було виділено 2 ГБ: по одному для нього та для кешу. І хоча кожен запит займав дуже мало місця, після мільйона таких запитів кеш переповнювався. Через

це сервіс не міг обробляти значний обсяг даних і був змушений перезавантажуватися для очищення кешу[18].

Третій пункт задач автоматизації – тестування відповідності. Який надає процес перевірки відповідності продукту або послуги певним вимогам, правилам, стандартам або специфікаціям.

Такий тип тестування зазвичай використовується для перевірки, чи відповідає продукт чи послуга певним нормам, які можуть бути встановлені регулюючими органами, законодавством, клієнтами, контрактами або внутрішніми стандартами компанії.

Наприклад, якщо компанія розробляє програмний продукт, то вона може проводити тестування відповідності, щоб переконатися, що продукт відповідає стандартам безпеки, які були встановлені регулюючими органами або законодавством.

Тестування відповідності може забезпечити підтвердження, що продукт чи послуга відповідає вимогам і може бути безпечно використовуваним відповідно до призначення.

Четвертий пункт – регресійне тестування - тип тестування, який використовується для перевірки того, чи не відбулися зміни у функціональності продукту після внесення змін. Зазвичай регресійне тестування виконується для перевірки, чи не виникли нові помилки після внесення змін у коді, дизайні, або інших елементах продукту.

У процесі регресійного тестування, тестувальний персонал виконує певний набір тестів, які були використані раніше для тестування продукту, або створює нові тести для перевірки нової функціональності. Ці тести можуть бути виконані автоматично за допомогою тестових скриптів або вручну[18].

Регресійне тестування може бути дуже важливим для забезпечення якості продукту, оскільки воно дозволяє виявити помилки, які можуть виникнути через

зміни, що внесені у продукт, і впевнитися в тому, що вони були виправлені. Це допомагає зберегти час і ресурси на подальше тестування, а також забезпечує високу якість продукту.

Пятий пункт який бере за увагу тестування інтерфейсу користувача. процес перевірки функціональності та придатності інтерфейсу користувача відповідно до заданих вимог та очікувань.

Автоматизуване тестування інтерфейсу користувача процес, коли використовуються спеціальні програмні засоби для автоматизації тестування різних аспектів інтерфейсу користувача, таких як клікання, введення даних, навігація тощо.

Це дозволяє зменшити час тестування та забезпечити більшу точність.

Включає в себе велику кількість інструментів тестування UI в які входять тестувальні фреймворки, системи моніторингу, інструменти запису дій користувача, тестувальні середовища, функціональність, зручність використання, оформлення, сумісність.

Шостий пункт є тестування безпеки який відповідає за процес перевірки програмного забезпечення на вразливості та можливості несанкціонованого доступу до системи або даних. Його мета полягає в забезпеченні високої безпеки програмного забезпечення та запобіганні можливих кібератак. Маємо кілька видів тестування безпеки, таких як:

1. Penetration testing - процес етичного вторгнення в систему з метою виявлення слабкостей і вразливостей. Цей вид тестування дозволяє встановити, як ефективно працюють засоби захисту системи та знайти шляхи підвищення її безпеки.
2. Vulnerability scanning - автоматизований процес сканування системи з метою виявлення вразливостей та потенційних загроз безпеці. Цей вид

тестування є швидким та ефективним способом виявлення слабкостей, але не дає повної карти загроз та вразливостей.

3. Security auditing - процес виявлення і аналізу загроз безпеці системи, що виконується з метою оцінки рівня безпеки. Цей вид тестування дозволяє зрозуміти загальний рівень безпеки системи, оцінити ризики та розробити план заходів з підвищення безпеки.
4. Code review - аналіз вихідного коду програмного забезпечення з метою виявлення потенційних вразливостей та слабкостей безпеки. Цей вид тестування дозволяє знайти загрози безпеці на ранніх етапах розробки та внести необхідні зміни в код програми.

Правильно проведене тестування безпеки є важливою складовою у забезпеченні високої безпеки системи та захисту від кібератак.

І остання задача – тестування продуктивності – це комплекс типів тестування, метою якого є визначення працездатності, стабільності, споживання ресурсів та інших атрибутів якості додатка в умовах різних сценаріїв використання та навантажень. Тестування продуктивності дозволяє знаходити можливі вразливості та недоліки в системі з метою запобігання їх негативному впливу на роботу програми в умовах використання[22].

1.2 Види фреймворків автоматизированного тестування Веб-застосунків

Одними із найчастіших питань є те, яку мову програмування вибрати для вивчення, а також який інструмент для автоматизації краще. Ці питання насправді варто розглядати в іншій площині, а мова програмування є більше наслідком, а не причиною, а інколи і взагалі для автоматизації і не потрібне[14].

При всьому різноманітті того, що вважатиметься автоматизацією Веб-додатків, можна виділити категорії:

- Функціональне чи нефункціональне тестування;
- Рівні: Модульне тестування, Інтеграційне, Сумісність і т.д;
- Методи: біла скринька, чорна скринька;
- Рівень у мережній моделі: frontend, backend, database;
- Браузери: Chrome, Opera, Edge, Firefox etc;
- Масштабованість: Single-Threaded, Multi-Threaded;
- Програма, скрипт, фреймворк чи бібліотека;
- Створення тестів, як приклад: нативна мова програмування, універсальна мова програмування , тестова модель і ключові слова чи рекордери (Capture & Playback);

Кожен інструмент створений і підходить для певних йому завдань, а також поєднує кілька критеріїв описаних вище. Саме тому немає кращого інструменту та єдино правильної мови програмування, але можна виділити яйсь інструмент з його особливостями на фоні інших.

1.2.1 Selenium WebDriver

Основним з обраних мною фреймворків автоматизованого тестування є Selenium Web Driver.

Selenium WebDriver — є одним із компонентів фреймворку Selenium, який використовується для автоматизації тестування веб-додатків. Він надає розширений набір функцій для взаємодії з веб-елементами, виконання різних дій на веб-сторінках та отримання результатів.

Основна мета Selenium WebDriver полягає у створенні автоматизованих тестів, які можуть відтворювати дії користувача на веб-сторінках. Він дозволяє робити різні дії, такі як клікання на елементи, введення тексту, відправлення форм, скролінг сторінки, отримання атрибутів елементів та багато іншого. Взаємодія з веб-елементами здійснюється за допомогою різних локаторів, таких як CSS-селектори, XPath, ідентифікатори елементів тощо[14].

Selenium WebDriver підтримує багато мов програмування, включаючи Java, Python, C#, Ruby та інші, що дозволяє розробникам писати тести на своїй улюбленій мові.

Використання WebDriver з TestNG, Junit або іншими тестовими фреймворками дозволяє створювати структуровані тести з налагоджуваною звітністю, можливістю групування тестів, параметризацією та багато іншого. Selenium WebDriver є потужним інструментом для автоматизації тестування веб-додатків, що дозволяє прискорити процес тестування, покращити його якість та забезпечити надійність веб-застосунків.

Selenium IDE (Integrated Development Environment) — інструмент для автоматизованого тестування веб-застосунків, який дозволяє записувати, редагувати та відтворювати тестові сценарії безпосередньо в браузері. Selenium IDE надає зручний інтерфейс користувача та базується на Selenium WebDriver для виконання тестів. Основні характеристики Selenium IDE[14]:

- Selenium IDE надає можливість записувати тестові дії, які виконуються в браузері. Включаючи кліки, введення тексту, вибори елементів, наведення курсора миші, очікування та інші дії.
- Дозволяє редагувати записані тестові сценарії прямо в інтерфейсі Selenium IDE та відтворювати їх для перевірки правильності виконання.

- Надає підтримку різних браузерів як Chrome, Firefox, Microsoft Edge, інші.
- Також надає можливість використовувати змінні для зберігання значень та використання їх у тестових сценаріях.
- Містить різні команди які дозволяють керувати тестовими сценаріями, такі як умовні оператори, цикли, обробка винятків та інші.
- Дозволяє експортувати тестові сценарії до різних форматів, таких як Java, C#, Python, інші. Це робить його доречним для використання в різних середовищах та з різними тестовими стеками.
- Підтримує розширення та плагіни, що дозволяють розширювати його функціональність та пристосовувати під конкретні потреби проекту.
- Дозволяє додавати команди ассертів для перевірки правильності результатів та відповідності очікуванням.

Selenium WebDriver та Selenium IDE взаємоповязані один з одним і є потужними інструментами для швидкого створення та виконання тестів. Selenium IDE створений для простих тестів,а для більш складних тестових сценаріїв та автоматизації слід використовувати Selenium WebDriver[14].

1.2.2 Playwright

Playwright — це фреймворк для автоматизованого тестування веб-застосунків, який дозволяє тестувати веб-застосунки на різних браузерах, платформах та мовах програмування.

Playwright підтримує всі сучасні рендерингові двигуни, включаючи Chromium, WebKit та Firefox. Він також підтримує автоматизоване тестування

мобільних веб-застосунків. Playwright має багато корисних функцій, таких як автоматичне очікування, Web-first assertions, трасування та багато іншого[3].

Цей фреймворк можна використовувати з TypeScript, JavaScript, Python, .NET, Java.

Основні характеристики та можливості Playwright:

- Playwright підтримує різні види браузерів як Chrome, Firefox та WebKit, що дозволяє тестувати веб-додатки на різних платформах та браузерах.
- Має ассинхрону модель виконання з чого виходить що всі операції у Playwright є асинхронними і це дозволяє ефективно керувати асинхронним JavaScript кодом та взаємодіяти з асинхронними фреймворками.
- Playwright підтримує як headless (безголовний) режим, так і повноекранный режим для виконання тестів у відкритому браузері.
- Має інструменти для відлагодження та виявлення помилок, що допомагають швидко локалізувати та виправляти проблеми в тестових сценаріях.
- Підтримує JavaScript та TypeScript для написання тестових сценаріїв, а також є можливість використання Python, Java та C#.
- Playwright є відкритим проектом, з великою кількістю відкритого коду , що розвивається активною спільнотою.

Playwright універсальний інструмент та добре підходить для вимог до тестування сучасних веб-додатків та має функціонал, який дозволяє легко і ефективно автоматизувати різноманітні сценарії взаємодії з веб-додатками[3].

1.2.3 Robot Framework

Robot Framework - це високорівневий фреймворк для автоматизованого тестування та автоматизації виконання тестових сценаріїв.

Одним із основних переваг Robot Framework є його простота в читанні та написанні тестових сценаріїв, яка в основному забезпечується використанням ключових слів та природною мовою для опису тестових сценаріїв[4].

Нижче описані основні характеристики та можливості:

- Robot Framework використовує синтаксис ключових слів, що робить його досить зрозумілим та доступним для тестувальників та розробників.
- Тестові сценарії легко читаємі, оскільки вони написані на природній мові, що полегшує розуміння вимог та результатів тестування.
- Надає можливість використання табличних даних та файлів конфігурації для організації та параметризації тестових сценаріїв.
- Robot Framework має широкий спектр вбудованих та сторонніх бібліотек, які включають в себе функції та ключові слова для різних видів автоматизації.
- Robot Framework легко інтегрується з різними інструментами для автоматизації, такими як Selenium для веб-тестування, Appium для мобільного тестування, та іншими.
- Robot Framework є відкритим проектом з активною спільнотою користувачів та розробників.
- Має функціонал який генерує докладні HTML-звіти та логи, що допомагають в аналізі та відлагодженні помилок.

- Здатність виконувати тести паралельно, що підвищує ефективність виконання тестових наборів
- Має багато вбудованих бібліотек для взаємодії з різними технологіями, такими як HTTP, REST, SOAP, бази даних, мережеві пристрої, інші.
- Як унікальність Robot Framework має свою власну мову, яка є простою та легко читабельною. А також надає можливість написання тестів за допомогою Python, Java, C#, JavaScript та інші.

Robot Framework є ефективним інструментом для тестування та автоматизації особливо коли треба створювати та управляти тестовими сценаріями швидко та ефективно. А також маючи низький рівень програмування має свою власну мову, яке є простою для розуміння[4].

1.3 Порівняння можливостей та засобів фреймворків

Порівняння фреймворків насправді є дуже обширним завданням, так як існує багато різних фреймворків для різних мов програмування та завдань.

Для порівняння можливостей фреймворків можна використовувати такі критерії:

1. Підтримка мов програмування;
2. Можливості інтеграції-з іншими інструментами розробки;
3. Підтримка різних браузерів;
4. Зручність використання;
5. Наявність графічного інтерфейсу;
6. Підтримка різних типів тестів;
7. Масштабованість;
8. Підтримка спільноти;

9. Вартість.

Ці критерії будуть використовуватися для аналізу особливостей використання Selenium за допомогою порівнняням їх з іншими фреймворками як Playwright та Robot Framework.

1.3.1 Порівняння Selenium WebDriver, Playwright та Robot Framework

Selenium як фреймворк здатний автоматизувати та контролювати веб-браузери та взаємодіяти з елементами інтерфейсу користувача, і це найпопулярніший фреймворк у галузі на сьогодні. У пакеті Selenium є кілька інструментів, зокрема[17]:

- Selenium WebDriver : WebDriver надає гнучку колекцію API з відкритим кодом, які можна використовувати для легкого тестування веб-програм
- Selenium IDE: цей інструмент для запису та відтворення дозволяє швидко розробляти тести як для інженерів, так і для нетехнічних користувачів
- Selenium Grid: Grid дозволяє поширювати та запускати тести паралельно на кількох машинах

Вплив Selenium виходить навіть за рамки основної структури, оскільки ряд інших популярних інструментів, таких як Appium і WebDriverIO , були створені безпосередньо на основі API Selenium.

Playwright розроблено для наскрізного автоматизованого тестування веб-додатків. Він є кросплатформним, кросбраузерним і мовним, а також містить такі корисні функції, як автоматичне очікування. Він спеціально розроблений для сучасного Інтернету і зазвичай працює дуже швидко, навіть для складних проектів тестування.

Хоча Playwright набагато новий, ніж Selenium, швидко набирає обертів і має все більше прихильників. Частково через свій молодий вік він підтримує менше браузерів/мов, ніж Selenium, але тим самим він також включає нові функції та можливості, які більше відповідають сучасному Інтернету[3].

Архітектура:

- Архітектура Selenium використовує API WebDriver для взаємодії між веб-браузерами та драйверами браузерів. Він працює шляхом перекладу тестових випадків у JSON і надсилання їх до браузерів, які потім виконують команди та надсилають відповідь HTTP.
- Архітектура Playwright використовує з’єднання WebSocket, а не WebDriver API і HTTP. Це залишається відкритим протягом тесту, тому все надсилається за одним з’єднанням. Це одна з причин, чому швидкість виконання Playwright має тенденцію бути вищою.

З цього можна зробити висновок, що швидкість Playwright трохи вища за швидкість Selenium.

Robot Framework - це фреймворк автоматизації тестування, який використовується для виконання різних типів тестування, таких як тестування прийняття, кінцеве тестування тощо.

Таблиця 1 – Порівняння функцій фреймворків:

Критерії	Selenium	Playwright	Robot Framework
Браузери	Chrome, Safari, Firefox, Opera, Edge та IE.	Chromium, Firefox і WebKit (Playwright тестує проекти браузерів , а не стандартні браузери)	Може використовувати SeleniumLibrary для взаємодії з браузерами, які підтримуються Selenium.
Мови програмування	Java, Python, C#, Ruby, Perl, PHP і JavaScript.	Java, Python, .NET C#, TypeScript і JavaScript.	Підтримує Java та Python. Має власну мову програмування.
Підтримка Test Runner Frameworks	Jest/Jasmine, Mocha, WebDriver IO, Protractor, TestNG, JUnit і Nunit.	Jest/Jasmine, AVA, Mocha та Vitest.	Robot test, Jenkins, Pytest-robot
Інтеграція з CI	Так	Так	Так
Передумови	Прив’язки Selenium, драйвери та	NodeJS.	Robot Framework та встановлені

	автономний сервер Selenium		Selenium бібліотеки
Підтримка спільноти	Велика налагоджена колекція документації та варіантів підтримки..	Менший, але зростаючий набір ресурсів спільноти.	Має гарну активну спільноту, але менше чим у Selenium та Playwright
Зручність	Має велику спільноту, що полегшує знаходження ресурсів та підтримку для новачків.	Створений для того, щоб бути простим та ефективним у використанні, навіть для новачків.	Через невелику спільноту є ускладнення у використанні.
Швидкодія	Середня.	Швидкий.	Залежить від оптимізації структури тестів.
Розширення	Має гарні розширення для використання.	Немає	Бібліотеки сторонніх розробників, Custom Libraries
Асинхронний підхід	Немає	Має асинхронний API	Немає

Інструменти відлагодження	Має зручні засоби для відлагодження	Має вбудовані інструменти для відлагодження	Має вбудовані інструменти для відлагодження

Обираючи між Selenium, Playwright та Robot Framework важливо враховувати вимоги проекту, рівень зручності для команди, а також специфічні особливості кожного фреймворка. Описані фреймворки є потужними та ефективними, але можуть бути більш або менш зручними в залежності від контексту та завдань, які потрібно реалізувати.

1.4 Висновок до розділу 1

Автоматизоване тестування веб-застосунків є невід'ємною частиною сучасного процесу зі створення програмного забезпечення, спрямованою на покращення якості та надійності. Застосування відповідних інструментів для автоматизації тестів значно полегшує процес розробки, та зменшує кількість помилок, а також прискорює видачу нових функцій[22].

У першому розділі виконане порівняльне дослідження за наявними інформаційними джерелами популярних фреймворки автоматизованого тестування веб-застосунків: Selenium, Playwright та Robot Framework. Проведене порівняння можливостей та засобів цих фреймворків дозволило виокремити їхні переваги та особливості.

Порівнюючи Selenium та Playwright, визначили що обидва фреймворки мають свої переваги та недоліки. Selenium є широко використовуваним, має велику спільноту та підтримує багато мов програмування. З іншого боку, Playwright вирізняється своєю швидкодією та асинхронним підходом.

Також ми розглянули порівняння Selenium з іншим популярним фреймворком – Robot Framework. Кожен з них має свої унікальні риси, але варто враховувати специфічні потреби та характеристики проекту при виборі між ними.

Враховуючи різноманіття вимог та контекстів проектів, важливо обирати той фреймворк, який найкращим чином відповідає конкретним потребам та сприяє успішному впровадженню автоматизації тестування.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ ФРЕЙМВОРКУ SELENIUM

2.1 Дослідження Selenium IDE

Selenium IDE (Інтегроване середовище розробки) є найпростішим інструментом у Selenium Suite. Це надбудова, яка дуже швидко створює тести за допомогою функції запису та відтворення. Ця функція схожа на функцію QTP. Його легко встановити та легко освоїти. Через свою простоту Selenium IDE краще використовувати лише як інструмент для допомоги або створення прототипів, а не як загальне рішення для розробки та підтримки складних тест наборів.

Використовуючи Selenium IDE можна не мати попередніх знань у програмуванні, але повинні мати знання принаймні з HTML, JavaScript і DOM, щоб використовувати цей інструмент у повній мірі. Також можна мати знання JavaScript які можуть знадобиться для використання команди Selenese « runScript».

Selenium IDE підтримує режим автозаповнення під час створення тестів. Ця функція служить двом цілям:

- Допомога тестеру швидше вводити команди.
- Заборона користувачеві вводити недійсні команди.

Він має рядок меню який розташований у верхній частині IDE. Найпоширенішими меню є меню «Файл», «Правка» та «Параметри».

Меню «Файл»:

- Містить параметри для створення, відкриття, збереження та закриття тестів.
- Тести зберігаються у форматі HTML .

- Найкориснішим варіантом є «Експорт» , оскільки він дозволяє вам перетворити тестові приклади Selenium IDE у формати файлів, які можна запускати на WebDriver. Можна експортувати лише в формати .cs, .java, .py, .rb

Меню редагування (*Рисунок 1*)**:**

- Містить такі звичайні параметри, як Скасувати, Повторити, Вирізати, Копіювати, Вставити, Видалити та Вибрати все.
- Двома найважливішими параметрами є « Вставити нову команду » та « Вставити новий коментар ». Щойно вставлену команду чи коментар буде розміщено поверх поточного вибраного рядка.

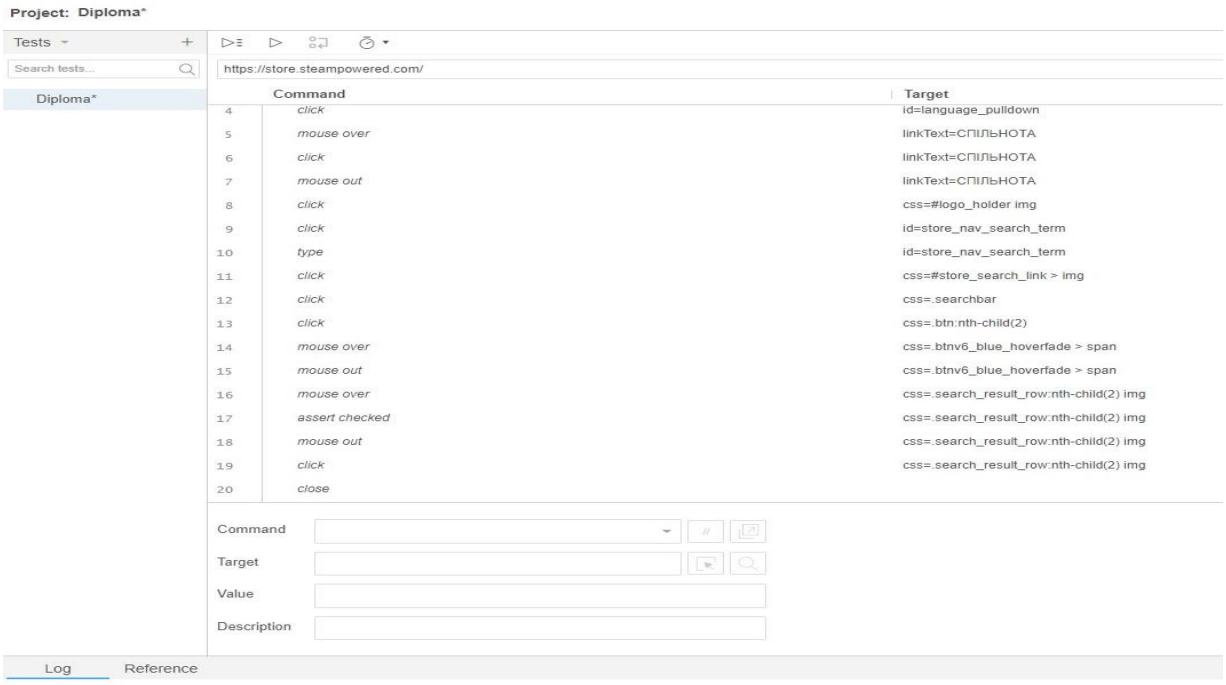

Рисунок 1. Меню редагування

Меню параметрів:

Забезпечує інтерфейс для налаштування різних параметрів Selenium IDE. Є параметр який називається «Формат буфера обміну». Дозволяє скопіювати команду Selenese з редактора та вставити її як фрагмент коду. Формат коду відповідає параметру, який ви вибрали тут у списку Формат буфера обміну. Наприклад, вибравши Java/JUnit 4/WebDriver як формат буфера обміну, кожна команда Selenese, яку копіюємо з редактора Selenium IDE, буде вставлена як код Java.

Також важлива частина це « Діалогове вікно параметрів» Selenium IDE яке надає:

- Значення часу очікування за замовчуванням. Це стосується часу, протягом якого Selenium має чекати, поки певний елемент з’явиться або стане доступним, перш ніж він згенерує помилку.
- **Розширення Selenium IDE** . Тут ви вказуєте розширення, які хочете використовувати для розширення можливостей Selenium IDE.
- Запам'ятати базову URL-адресу. Selenium IDE може запам’ятовувати базову URL-адресу кожного разу, коли ви її запускаєте.
- Автозапуск запису. Цей пункт Selenium IDE одразу запише дії вашого браузера після запуску.
- Конструктори локаторів. Можна вказати порядок, у якому повинні генеруватися локатори під час запису.

Якщо підбити підсумок то інструмент для автоматизації тестування веб-додатків Selenium IDE може бути хорошим вибором. Однак більшу гнучкість та можливості моужть дати інші інструменти, такі як Selenium WebDriver, а Selenium IDE використати як додатковий помічник.

2.1.2 Дослідження Selenium WebDriver

WebDriver керує браузером нативно, як це робив би користувач локально або на віддаленій машині за допомогою сервера Selenium, що означає стрибок вперед у плані автоматизації браузера. Selenium WebDriver відноситься як до мовних прив’язок, так і до реалізацій окремого керуючого коду браузера.

WebDriver розроблений як простий і лаконічний інтерфейс програмування. Є компактно об’єктно-орієнтованим API, який ефективно керує браузером.

Selenium підтримує автоматизацію всіх основних браузерів на ринку за допомогою WebDriver. WebDriver — це API та протокол, який визначає нейтральний щодо мови інтерфейс для керування поведінкою веб-браузерів. Кожен браузер підтримується певною реалізацією WebDriver, яка називається драйвером. Драйвер — це компонент, який відповідає за делегування до браузера та обробляє обмін даними між Selenium і браузером.

Це відокремлення є частиною свідомих зусиль, щоб постачальники веб-переглядачів взяли на себе відповідальність за впровадження для своїх браузерів. Selenium використовує сторонні драйвери, де це можливо, але також надає власні драйвери, які підтримуються проектом, для випадків, коли це не реально. Фреймворк Selenium об’єднує всі ці елементи разом через інтерфейс користувача, який дозволяє прозоро використовувати різні серверні модулі браузера, забезпечуючи кросбраузерну та кросплатформенну автоматизацію.

Налаштування Selenium значно відрізняється від налаштувань інших комерційних інструментів. Перш ніж почати писати код Selenium, вам потрібно встановити бібліотеки прив’язок мови для вибраної мови, браузера, який ви хочете використовувати, і драйвер для цього браузера.

В Selenium WebDriver є велика кількість цікавого, в деяких випадках унікального функціоналу.

Сеанси драйвера:

Сеанс створюється автоматично шляхом ініціалізації нового об’єкта класу Driver. Кожна мова дозволяє створити сеанс з аргументами з одного з цих класів (або еквівалентів):

- Параметри для опису типу сеансу, який ви хочете; значення за замовчуванням використовуються для локального, але це необхідно для віддаленого
- Певна форма конфігурації HTTP-клієнта (реалізація залежить від мови)

Локальний драйвер - основний унікальний аргумент для запуску локального драйвера містить інформацію про запуск необхідної служби драйвера на локальній машині.

Стратегії очікування:

Можливо, найбільш поширеною проблемою при автоматизації браузера є забезпечення того, щоб веб-застосунок могла виконувати певну команду Selenium за бажанням. Процеси часто опиняються в стані змагання , коли іноді браузер переходить у правильний стан першим (все працює, як задумано), а іноді код Selenium виконується першим (все працює не за призначенням). Це одна з основних причин невдалих результатів тестів.

Подібним чином у багатьох односторінкових програмах елементи динамічно додаються до сторінки або змінюють видимість на основі клацання. Елемент має бути присутнім і відображеним на сторінці, щоб Selenium міг з ним взаємодіяти.

Першим рішенням, до якого звертаються багато людей, є додавання оператора сну, щоб призупинити виконання коду на встановлений період часу. Оскільки код не може точно знати, скільки часу йому потрібно чекати, це може

вийти з ладу, якщо він не спить достатньо довго. З іншого боку, якщо встановлено занадто високе значення та оператор сну додається скрізь, де це потрібно, тривалість сеансу може стати непомірно високою. Selenium надає два різні механізми для синхронізації, які є кращими.

Неявні очікування:

Selenium має вбудований спосіб автоматичного очікування елементів, який називається неявним очікуванням . Неявне значення очікування можна встановити або за допомогою можливості тайм-аутів у параметрах браузера, або за допомогою методу драйвера. Це глобальне налаштування, яке застосовується до кожного виклику розташування елемента протягом усього сеансу. Значення за замовчуванням 0— це означає, що якщо елемент не знайдено, він негайно поверне помилку. Якщо встановлено неявне очікування, драйвер чекатиме тривалість наданого значення перед поверненням помилки. Але треба відмітити, що як тільки елемент знайдено, драйвер поверне посилання на елемент, і код продовжить виконання, тому неявне значення очікування не обов’язково збільшить тривалість сеансу.

Явні очікування:

Цикли, додані до коду, які опитують програму щодо певної умови, щоб оцінити її як істинну, перш ніж вона виходить із циклу та продовжує до наступної команди в коді. Якщо умова не виконується до визначеного значення тайм-ауту, код видасть помилку тайм-ауту. Оскільки існує багато способів, за допомогою яких програма не перебуває в бажаному стані, тож явні очікування є чудовим вибором для визначення точної умови очікування в кожному потрібному місці. Ще одна приємна особливість полягає в тому, що за замовчуванням клас Selenium Wait автоматично очікує на існування зазначеного елемента.

Клас Wait може бути створений із різними параметрами, які змінять спосіб оцінки умов:

- Зміна частоти оцінювання коду (інтервал опитування)
- Визначення винятків, які мають оброблятися автоматично
- Зміна загальної тривалості тайм-ауту
- Налаштування повідомлення про час очікування

Двонаправлена функціональність:

Selenium співпрацює з постачальниками браузерів для створення протоколу WebDriver BiDirectional як засобу забезпечення стабільного кросбраузерного API, який використовує двонаправлену функціональність, корисну як для автоматизації браузера загалом, так і для тестування. Раніше користувачі, яким потрібен був такий функціонал мусили використовувати Chrome DevTools Protocol з усіма його перевагами та недоліками.

Традиційна модель WebDriver із строгими командами запиту/відповіді буде доповнена можливістю потокової передачі подій від агента користувача до керуючого програмного забезпечення через WebSockets, що краще відповідає подієвій природі DOM браузера. Оскільки прив’язувати свої тести до певної версії будь-якого браузера не дуже гарна ідея, проект Selenium рекомендує використовувати WebDriver BiDi, де це можливо.

Взаємодія з веб-елементами:

Є лише 5 основних команд, які можна виконати над елементом:

- click (для будь-якого елемента)
- send keys (лише для текстових полів і редагованих елементів вмісту)
- clear (лише для текстових полів і редагованих елементів вмісту)
- submit (застосовується до елементів форми)
- select

Support features - основні бібліотеки Selenium намагаються бути низькорівневими та неупередженими. Класи підтримки кожною мовою надають

продумані оболонки для спільних взаємодій, які можна використовувати для спрощення деяких дій.

Waiting with Expected Conditions - очікувані умови використовуються з очевидним очікуванням. Замість визначення блоку коду, який виконуватиметься за допомогою лямбди, можна створити метод очікуваних умов для представлення загальних речей, які очікуються. Деякі методи приймають локатори як аргументи, інші — елементи.

Command Listeners - це дозволяє виконувати спеціальні дії щоразу, коли надсилаються певні команди Selenium.

Working with select list elements - об’єкт Select надає ряд команд, які дозволять взаємодіяти з елементом <select>. Функція поділяється на 2 типи, як один вибір - стандартний розкривний об’єкт, де можна вибрати один і лише один параметр. А також множинний вибір - список вибору дозволяє вибирати та скасовувати вибір кількох параметрів одночасно. Це стосується лише <select>елементів з multiple атрибутом.

Дослідження Selenium WebDriver свідчить про його високуефективність у сфері автоматизованого тестування веб-застосунків. Володіє дуже великим функціоналом, що робить його універсальним інструментом для розробників. Вбудовані засоби для відлагодження та багатофункціональність забезпечують зручність в розробці та підтримці тестових сценаріїв.

2.2 Конфігурація та взаємодія з фреймворком Selenium

2.2.1 Встановлення

Щоб встановити Selenium WebDriver та Selenium IDE, спочатку потрібно встановити деякі засоби, такі як веб-браузер та мова програмування, яку треба

буде використовувати для написання автотестів. В нашому випадку це буде Google Chrome як браузер та мова програмування Java[5].

Встановлення Selenium WebDriver:

1. Вибір та встановлення мови програмування:
   - Визначаємося з мовою програмування (у нашому випадку це Java)
   - Встановлюємо мову програмування.
2. Встановлення веб-браузера:
   - Встановілюємо веб-браузер з яким будемо взаємодіяти через Selenium WebDriver (у нашому випадку це Chrome)
3. Встановлення Selenium WebDriver:
   - Завантажуємо та встановлюємо Selenium WebDriver[17] для потрібної мови програмування.
4. Завантажуємо драйвер браузера:
   - Завантажуємо відповідний драйвер браузера та включаємо його в системний шлях.

Встановлення Selenium WebDriver:

1. Встановлення веб-браузера:
   - Встановлюємо підтримуваний веб-браузер для Selenium IDE (Google Chrome)
2. Встановлення розширення Selenium IDE:
   - Заходимо у веб-магазин Chrome та встановлюємо розширення Selenium IDE.
3. Запуск Selenium IDE:
   - Після встановлення розширення, запускаємо Selenium IDE з панелі інструментів браузера.

Тепер є можливість приступити до розгортання тестового середовища та створення і прогонці димових тестів. Будемо користатися документацією для отримання деталей щодо використання цих інструментів у обраному середовищі програмування.

2.2.2 Команди та локатори

Можливості автоматизації функціонального тестування Selenium станом на зараз мають велику популярність. Неможливо уявити спеціаліста з тестування, або розробника програмного забезпечення який, як мінімум, не чув про цей інструмент.З кожним днем Selenium стає більш популярний, тому щодня все нові та нові фахівці з тестування починаю освоювати його[5].

В Selenium існує три типи команди:

- Дії – функціональна дія над тестованим веб-додатком у браузері. Наприклад, заповнення полів, натискання на кнопку та інші;
- Перевірки – виконання перевірок на сторінці, що тестується. Наприклад, перевірка того, що певне поле форми має вказане значення або перевірка заголовка вікна;
- Очікування – організація як, скільки і яку подію Selenium чекатиме (очікування завантаження сторінки, ajax тощо).

Дії (Actions). Набір дій в Selenium досить широкий, основні (які використовуються) дії[5]:

- `open` – відкрити сторінку у браузері за певною адресою. Синтаксис команди – `Open(string url.`
- `click` – натиснути елемент сторінки. Синтаксис команди – `Click(string locator).`

- `type` – введіть значення в текстове поле сторінки. Синтаксис команди – `Type(string locator, string value).`
- `select` – вибрати значення зі списку, що випадає. Синтаксис команди - `Select (string selectLocator, string optionLocator`). Як опція для вибору елемента можна використовувати такі локатори: `label, value, id, index`.
- `selectWindow` – переключити фокус інше вікно. Синтаксис команди – `SelectWindow(string windowID)`
- `getTitle` – повертає Title для поточної сторінки. Синтаксис команди – `GetTitle().`
- `getValue` – повертає значення елемента сторінки. Синтаксис команди – `GetValue(string locator).`
- `goBack` – вернуться на предыдущую страницу. Синтаксис команды – `GoBack().`
- `close` – закрити поточне вікно. Синтаксис команди – `Close().`

При використанні команд Selenium спочатку варто навчитися працювати з локаторами. Локатор (locator) – це рядок, що унікально ідентифікує елемент веб-сторінки. Тобто те, за допомогою чого шукається, шукаються елементи на сторінці. Для того, щоб виконати дію на сторінці Selenium, необхідно знати, з яким елементом на сторінці йому потрібно виконати необхідну дію. Для цього служать локатори[5].

Типи локаторів у Selenium:

- id – як локатор використовується атрибут id (унікальний ідентифікатор) елемента сторінки;
- name - як локатор використовується атрибут name елемента сторінки;

- identifier – використовується атрибут id елемента, якщо з id елемент не знайдено, то пошук буде вестися по атрибуту name;
- dom - пошуку елемента відбувається за виразом DOM;
- xpath – використовується для пошуку елемента за виразом XPath;
- link – пошук посилань із зазначеним текстом;
- css – даний тип локаторів ґрунтується на описах таблиць стилів (CSS).

Перевірка (checks). Перевірка одна з головних складових при написанні тесту. У Selenium IDE виділяється два типи перевірок – assert та verify. Різниця між ними полягає в тому, що якщо тест "паде" при виконанні перевірки assert, то тест припиняє свою роботу і позначається як failed. А якщо не виконується перевірка verify, тест відзначить дану перевірку як failed, але продовжить свою роботу[5].

Команди, представлені для Selenium IDE:

- `verifyLocation/assertLocation` – перевірити адресу поточної сторінки.
- `verifyTitle/assertTitle` – перевірити значення Title сторінки.
- `verifyValue/assertValue` – перевірити значення елемента сторінки.
- `verifyTextPresent / assertTextPresent` – перевірити, чи сторінка містить вказаний у команді текст.
- `verify Element Present / assertElement Present` – перевірити, чи є на сторінці вказаний елемент.

Очікування (wait). При виконанні деяких команд необхідно чекати завантаження сторінки, або її певних елементів. Для цього призначені команди-очікування.

- `WaitForCondition` – очікування на виконання певної події на сторінці, вказаної в параметрі script (наприклад, завантаження певного елемента сторінки, або сторінки в цілому).
- `WaitForFrameToLoad` – чекає на завантаження фрейму на сторінці вказану кількість часу.
- `WaitForPageToLoad` – чекає на завантаження сторінки вказану кількість часу.
- `WaitForPopUp` – чекає на появу PopUp елемента сторінки вказану кількість часу.

Набір команд у Selenium можна розширювати за допомогою додавання готових бібліотек, фреймворків або написання своєї власної бібліотеки команд. Існуючі команди Selenium також доступні для редагування.

2.3 Висновок до розділу 2

Фреймворк Selenium є важливим інструментом у сфері автоматизації тестування веб-застосунків, який надає потужні можливості для веб-розробників та QA інженерів. Складаючись з різних компонентів, зокрема Selenium IDE та Selenium WebDriver, фреймворк дозволяє ефективно тестувати веб-інтерфейси, відтворюючи взаємодію користувачів з веб-сторінками[5].

Selenium IDE, розширення для браузерів, є інтуїтивно зрозумілим інструментом для створення простих тестових сценаріїв. Його графічний інтерфейс дозволяє записувати дії користувача в браузері, генеруючи автоматично тестові скрипти, які можна виконати для перевірки функціональності веб-сайтів. Selenium IDE ідеально підходить для швидкого створення та перевірки простих тестових сценаріїв, але має обмеження у складності та гнучкості тестів.

Selenium WebDriver є більш потужним інструментом, що дозволяє створювати складніші та гнучкіші тестові сценарії. WebDriver працює безпосередньо з веб-браузерами, надсилаючи їм команди та контролюючи їх поведінку. Це дає змогу імітувати більш складні взаємодії користувачів, такі як перетягування елементів, скролінг сторінок, робота з вкладками браузера та інші динамічні дії.

РОЗДІЛ 3 РОЗРОБКА ТЕСТОВИХ ВИМОГ ДЛЯ ВЕБ-ЗАСТОСУНКУ

3.1 Характеристика Веб-застосунку Steam

Steam (*Рис.*2)— онлайн-сервіс цифрового розповсюдження комп'ютерних ігор та програм, розроблений та підтримуваний компанією Valve. Steam виконує роль засобу технічного захисту авторських прав, платформи для розрахованих на багато користувачів ігор і потокового мовлення, а також соціальної мережі для гравців. Steam містить Веб-додаток, мобільний додаток та комп'ютерний клієнт[21].

Програмний клієнт Steam також забезпечує встановлення та регулярне оновлення ігор, хмарні збереження ігор, текстовий та голосовий зв'язок між гравцями.

На початок 2021 через Steam пропонувалося до продажу понад 50 тисяч ігор, і ця кількість збільшувалась щорічно на 8—10 тисяч ігор. У 2020 році кількість активних користувачів сервісу, які використовували його хоча б один раз на місяць, перевищувала 120,4 мільйонів.З 2016 року було затверджено премію в галузі комп'ютерних ігор Steam Awards, в рамках якої відбираються лауреати голосуванням серед користувачів служби цифрової дистрибуції[12].

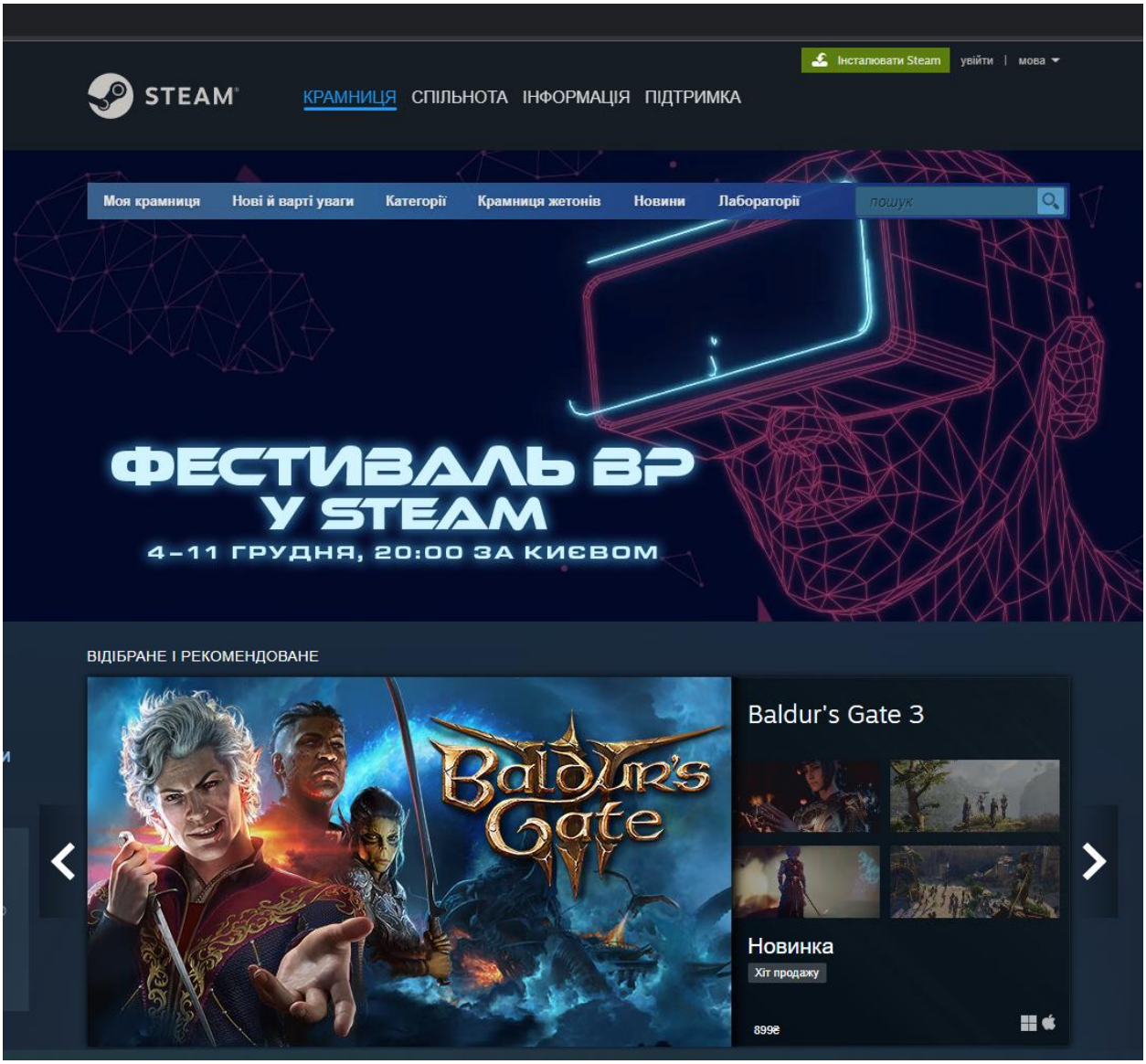

Рис. 1 — Тестовий застосунок

3.1.1 Вплив

Поява Steam сильно вплинула розробки інди-ігор, творці яких часто не могли знайти собі видавця. Поширення нестандартних ігор на дисках з погляду видавців — ризикований крок, адже заздалегідь невідомо, як покупці поставляться до гри. Метод цифрової дистрибуції, що дозволяє відмовитися від витрат на друк дисків, документації та коробок, а також від витрат на продаж

через торгові мережі, дозволяє видавати такі ігри з меншим фінансовим ризиком[12].

Після того, як Valve почала розповсюджувати через Steam ігри Darwinia та Red Orchestra, їх розробники отримали відразу кілька пропозицій від роздрібних видавців, оскільки ці ігри продемонстрували свою популярність.

Також поширення Steam вплинуло на цифрову дистрибуцію в цілому. Відразу після виходу Half-Life 2 (що вимагає обов'язкової установки Steam) цей сервіс привернув увагу багатьох гравців. Steam став першим вдалим проектом у цій галузі (близько 25% копій Half-Life 2 було продано через Steam). Незабаром після цього про створення своїх подібних сервісів заявили кілька великих компаній, таких як Sega, EA Games та 3D Realms[12].

Дрібніші компанії, такі як Stream Theory і Virgin Games, також анонсували свої проекти.Проте, деякі з них не довели власні проекти до завершення. Sega стала продавати свої ігри в Steam, а сервіс Triton компанії Game xStream, що розповсюджував ігри 3D Realms, був закритий (ігри 3D Realms також перейшли в Steam).

Electronic Arts теж розповсюджує деякі свої ігри через Steam, незважаючи на наявність власного торгового майданчика Origin (крім цифрової дистрибуції цей сервіс призначений і для онлайн-продажів ігор на фізичних носіях)[12].

3.1.2 Можливості

Steam виступає в ролі технічного засобу захисту авторських прав (DRM) (причому навіть якщо використовується коробкова версія гри, немає необхідності постійно вставляти диск з грою)[12].

Має такі можливості:

- Можливість купити гру для іншої людини як подарунок або подарувати комусь «зайву» гру, куплену повторно у складі збірки.
- Надання регіональних цін — у різних регіонах ціна може відрізнятися від стандартної або бути різною для назв однієї гри, що відрізняються.
- У Steam реалізовано регіональний захист через те, що в різних регіонах коробкові видання Steam-ігор мають різну вартість. Іншими словами, ігри, куплені в Україні (у коробковому варіанті), не працюватимуть за її межами. У той же час цей підхід дозволяє видавцям зробити доступною будь-яку офіційну озвучку, оскільки видавці можуть не побоюватися реекспорту ігор.
- Активація ключів ігор. CD-ключ - це своєрідний код для активації гри в сервісі Steam, який є набором з 13 символів, 18 символів, 25 символів і 26 символів, які в свою чергу складаються з латинських букв і цифр.
- Здійснення повернення яких можливе лише протягом 14 днів і за умови, що користувач провів у грі не більше двох годин.
- Потокове завантаження - Steam підтримує потокове завантаження контенту. Це дозволяє розподілити пріоритети завантаження вмісту. Таким чином, спочатку завантажується частина гри, яка потрібна для запуску. Інші файли — у фоновому режимі.
- Режим Big Picture – 10 вересня 2012 року вступив до бета-тесту, а 3 грудня вийшов офіційно. Big Picture – це режим роботи Steam, оптимізований для великих екранів телевізорів та керування геймпадом.
- Steam Cloud - Дана функція дозволяє зберігати ігрові дані (такі як особисті конфігурації, налаштування клавіатури та миші, лого-спреї для мультиплеєрних ігор, файли збережень) на серверах Valve.

- Steam Інвентар - Дана функція вперше з'явилася в Steam в ході класових оновлень ігор, де за виконання певних досягнень або просто в процесі гри гравцеві видавалася вдосконалена зброя, яка складалася в особливий рюкзак, асоційований з ідентифікатором Steam.
- Майстерня Steam - це галерея предметів, зроблена компанією Valve, яка розрахована на створення та розгляд предметів, створених для модифікації різних ігор користувачами.
- Steam Community Market - 14 грудня 2012 року вступив у бета-тест, який дозволяє користувачам продавати внутрішньоігрові предмети. Продавець сам може виставляти ціну кожного предмета, аж до 300 $. Гроші на рахунку стиму мають значення як і справжніх грошей, але вивести їх можна лише за допомогою сторонніх сайтів.
- Система досягнень - У деяких іграх існують досягнення - необов'язкові ігрові завдання, при виконанні кожного з яких у грі зберігається інформація. При цьому, якщо Steam запущено в онлайн-режимі, досягнення також зберігається на обліковому записі Steam, що дозволяє відновити отримані раніше досягнення при перевстановленні гри.

3.1.3 Соціальні функції

Steam Community — 12 вересня 2007 року було випущено оновлення для Steam (офіційно назване Steam Community), яке полегшить спілкування між користувачами сервісу. Steam Community дозволяє створювати свої персональні web-сторінки в системі Steam, створювати групи та вступати до них. У кожній групі є чат, кількість учасників у якому відображається на сторінці групи. Статус кожного учасника відображається на його сторінці: У

мережі, Відключений від мережі, Відійшов, Зайнятий, У грі (із зазначенням назви гри) та Сплю (недоступний для самостійного вибору статус, який автоматично виставляється після двогодинної бездіяльності клавіатури та миші). Також на персональній сторінці відображається статистика часу гри за останні два тижні, окремо для кожної гри[12].

Час, проведений за No-Steam грою, у статистиці не відображається. Чат Steam був значно перероблений до виходу Steam Community. Було додано можливість голосового спілкування, а можлива кількість учасників зросла до кількох десятків (раніше було можливим спілкування лише один на один)[12].

Головним оновленням Steam-чату є можливість використовувати його в будь-якій грі, навіть якщо вона не продається в Steam: якщо в меню Steam дозволено відповідну опцію, певною комбінацією клавіш (за замовчуванням Shift+TAB) відкривається напівпрозоре меню Steam Overlay - інтерфейс Steam Community поверх зображення ігри[12].

Так як в наш час інтернет є у кожної людини і його використання робить житя легче, то Веб-застосунки відіграють дуже важливу роль у використанні інтернет мережі. Завдяки Веб-застосункам ми можемо робити багато речей такі як спілкуватися, робити покупки, Шукати інформацію, читати що у людей на думці, грати в ігри та багато іншого. Steam це та платформа яка може підримувати більшість перелічених вище класифікацій Веб-застосунків. Бо Steam є популярною платформою для гри в комп'ютерні ігри та цифрової дистрибуції вмісту[12].

3.2 Визначення тестуємої функціональності Веб-застосунку

3.2.1 Функціональність Веб-застосунку

У Steam є дуже великий набір функціональності жо яких відносяться:

- Steam Workshop: Це платформа для спільноти геймерів, де вони можуть створювати та ділитися власними модифікаціями (модами) до ігор. Інші гравці можуть завантажувати ці моди та використовувати їх у своїх іграх.
- Steam Chat: Це інструмент для обміну повідомленнями та спілкування з друзями на платформі Steam. Ви можете створювати індивідуальні чати, групові розмови та відправляти повідомлення зображеннями та емодзі.
- Steam Cloud: Це сервіс зберігання у хмарі, який дозволяє гравцям зберігати свої геймплейні дані та налаштування на серверах Steam. Це означає, що ви можете мати доступ до своїх збережених ігор з будь-якого комп'ютера, на якому ви увійшли в свій обліковий запис Steam.
- Каталог ігор: Steam має великий каталог ігор, який охоплює різні жанри та платформи. Користувачі можуть переглядати, шукати і придбавати ігри прямо з Веб-застосунку.
- Керування бібліотекою ігор: Користувачі можуть керувати своєю бібліотекою ігор через веб-застосунок Steam. Вони можуть встановлювати, оновлювати, видаляти та керувати налаштуваннями своїх ігор.
- Steam Community: Це спільнотний портал в Steam, де користувачі можуть обговорювати ігри, залишати відгуки, ділитися враженнями, створювати групи та спілкуватися з іншими гравцями.
- Система досягнень: Багато ігор в Steam мають систему досягнень, яка стимулює гравців до досягнення певних цілей або виконання завдань у грі. Це надає додатковий інтерес та мотивацію для гри.

- Ігровий обмін та ринок: Користувачі можуть обмінюватися ігровими предметами з іншими гравцями або продавати їх на ігровому ринку Steam. Це дозволяє створити власну колекцію предметів або отримати користь від продажу.
- Реєстрація та вхід в систему: Користувачі мають змогу реєстрації нового облікового запису, а також змогу входу в систему Steam як зареєстрований користувач[12].

3.2.2 Функіональність для тестування

Для тестування функціональності Веб-застосунку методом дослідження було обрано:

1. Пошук та перегляд гри (*Рисунок 3*) - тестування функціональності пошуку гри за допомогою поля пошуку, за власною назвою. Аналіз знайдених результатів в процессі пошуку, для порівнення результатив актуального та фактичного пошуку гри.
2. Додавання гри до кошику (*Рисунок 4*) – тестування функціональності додавання гри до кошику, через натискання кнопки «До кошику» та перевірки що гра успішно додана, через порівняння за назвою гри.
3. Видалення гри з кошику (*Рисунок 5*) - тестування функціональності видалення гри з кошику через кнопку «Вилучити» з перевіркой того, що гра успішно видалена завдяки перевірці того що елемент з корзиною не відображається на головній сторінці.

Тестування буде виконуватись за допомогою інструментів Selenium WebDriver, Playwright та Robot Framework через порівняння результатів тестування, визначення плюсів та мінусів, а також особливостей Selenium.

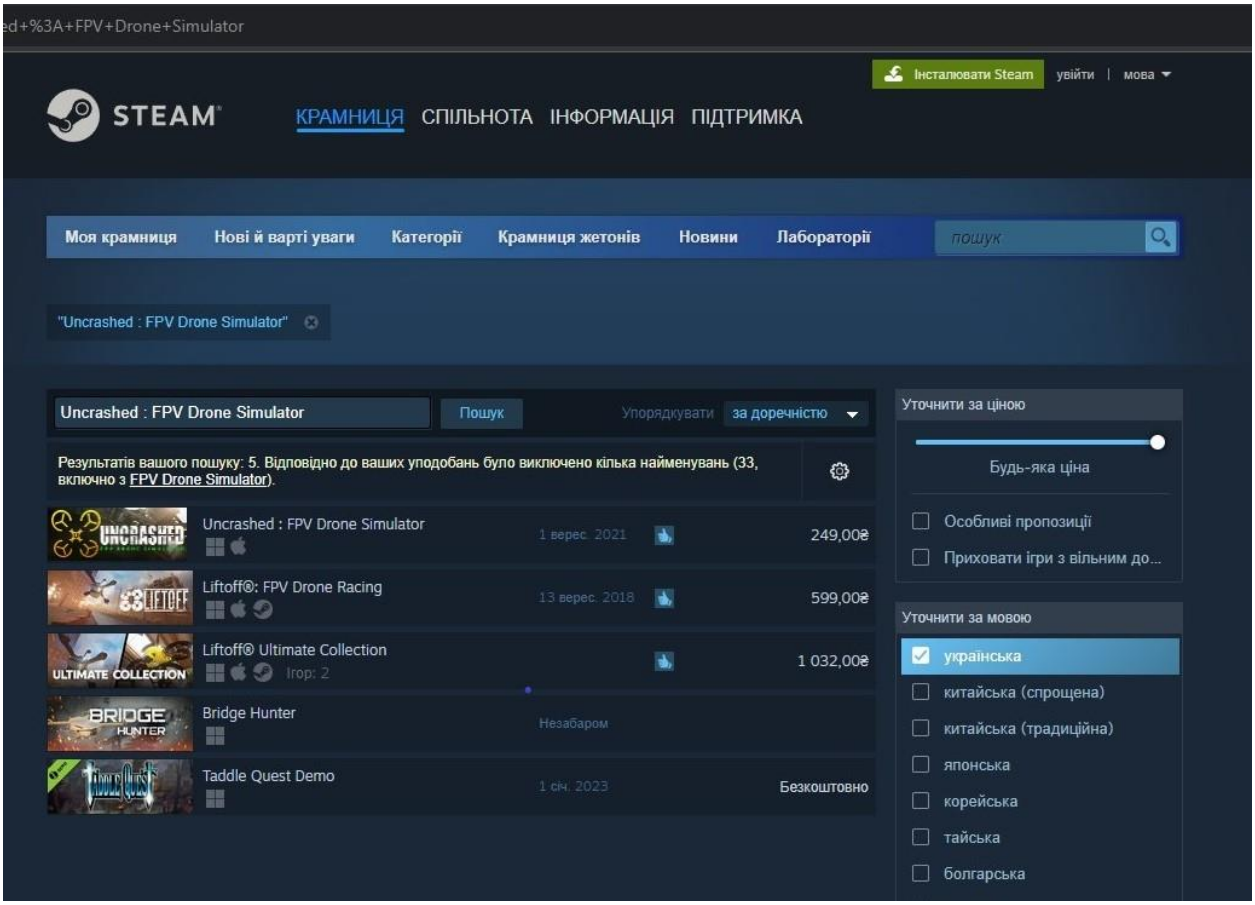

Рис. 2 – Пошукова сторінка

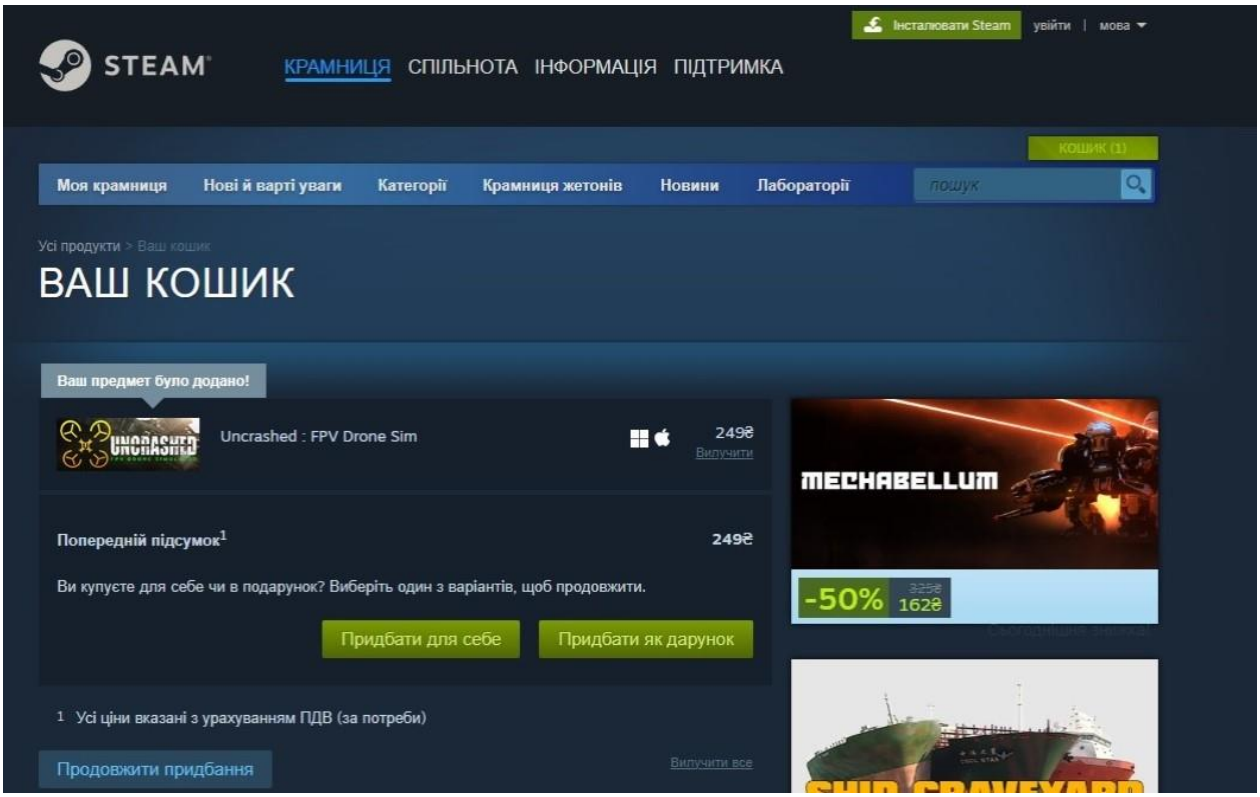

Рис. 3 – Сторінка кошику з доданою грою

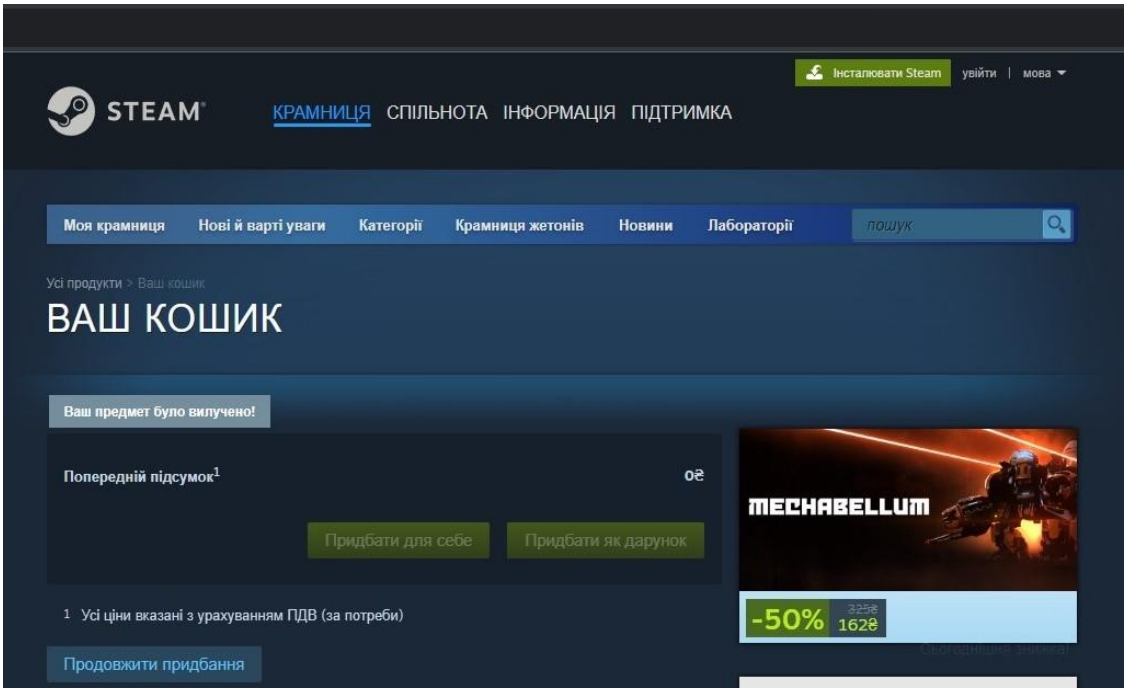

Рис. 4 – Сторінка кошику з видаленою грою

3.2.3 Розробка тестових вимог

Для розробки тестових вимог до функціональності Веб-застосунку Steam, ми можемо розглянути три основні функції: пошук гри, додавання гри до кошику, та видалення гри з кошику. Ці функції важливі для забезпечення гарного користувацького досвіду та ефективності застосунку.

1. Пошук гри та перевірка результатів пошуку:

Адаптивність результатів пошуку: Система має адаптувати результати пошуку згідно з історією користувача або його уподобаннями.

Релевантність результатів: Відображені результати мають бути високо релевантні запиту, з мінімальною кількістю нерелевантних результатів.

Пагінація результатів: Якщо результатів пошуку багато, вони мають бути розбиті на сторінки для зручності користувача.

Обробка помилкових запитів: Система має коректно обробляти помилкові або невизначені запити, надаючи повідомлення про відсутність результатів.

Реакція на зміну параметрів пошуку: Під час зміни параметрів фільтрації чи сортування, система має швидко реагувати та оновлювати результати.

2. Додавання гри до кошику та перевірка результатів:

Відображення опцій для додавання: Різні версії гри (наприклад, стандартна, розширена) мають бути доступні для додавання в кошик.

Перевірка обмежень на додавання: Наприклад, перевірка обмежень на вікову категорію або регіональні обмеження перед додаванням до кошику.

Оновлення інтерфейсу при додаванні: Інтерфейс користувача має відображати зміни відразу після додавання товару до кошику.

3. Видалення гри з кошику та перевірка результатів:

Перевірка функціоналу масового видалення: Можливість вибрати кілька товарів у кошику та видалити їх одночасно.

Оновлення пропозицій та рекомендацій: Після видалення гри з кошику, система може оновлювати рекомендації та спеціальні пропозиції для користувача.

Перевірка збереження стану кошика: Після видалення гри з кошику то, елемент кошика має зникнути з головної сторінки.

Після розробки цих вимог, важливо провести грунтовне тестування, щоб переконатися, що всі аспекти функціональності Веб-застосунку працюють належним чином. Selenium є ідеальним інструментом для цього, оскільки він дозволяє автоматизувати тестові сценарії та ефективно виявляти помилки.

3.3 Розгортання тестового середовища

В процесі автоматизованого тестування Веб-застосунків, ефективне розгортання засобів тестування визначається не лише вибором відповідних інструментів, але й правильною конфігурацією та налагодженням у відповідності до специфікацій та вимог проекту. У даному розділі розглядається важливий етап дослідження - розгортання обраних засобів тестування для Веб-затосунків.

Для тестування було обрано такі інструменти – Selenium WebDriver, Playwright та Robot Framework. Ці популярні інструменти забезпечують надійне та ефективне тестування Веб-застосунків і в цьому розділі буде описано розгортання для обраного раніше Веб-застосунку.

Перед налаштуванням робочого середовища та інструментів для автоматизації тестування оголосимо характеристики пристрою що буде використовуватись для роботи.

Налаштування робочого середовища, написання коду та виконання тестів буде реалізовано на особистому комп’ютері з наступними характеристиками:

1. Шестиядерний процесор Intel(R) Core(TM) i5-8500 з тактовою частотою 3.00 ГГц.

2. Графічний процесор NVIDIA GeForce GTX 1080.
3. 16 ГБ оперативної пам’яті.
4. Операційна система Windows 10.

Мета даного розділу - надати детальне інформацію про процес розгортання засобів тестування та визначити, як обрані інструменти можуть бути оптимально інтегровані в розробку Веб-застосунку. Через вивчення цього етапу дослідження ми отримаємо розуміння практичного використання обраних засобів тестування у реальних умовах та їхнього впливу на якість та продуктивність.

3.3.1 Налаштування середовища розробки для фреймворку Selenium WebDriver

Для тестування інструментів необхідно встановити для кожного з них свої середовища розробки, набір інструментів та бібліотек.

Для правильного та ефективного виконання тестування за допомогою Selenium WebDriver розглянемо процес налаштування середовища розробки для фреймворку з використанням IntelliJ IDEA як середовищем розробки та мовою програмування Java[7].

1. Встановлення Java Development Kit (JDK):
    - Завантажуємо та встановлюємо Java Development Kit (JDK) з офіційного сайту Oracle
    - Встановлюємо системну змінну JAVA_HOME, яка вказує на каталог, де розташовано JDK, налаштовуємо змінні оточення JAVA_HOME та PATH.
2. Встановлення IntelliJ IDEA:

- Завантажуємо та встановлюємо IntelliJ IDEA з офіційного сайту JetBrains.
- Запускаємо IntelliJ IDEA і налаштуємо його для використання JDK, яке було раніше встановлено.

3. Завантаження та розпакування ChromeDriver:
   - Обираємо версію, яка відповідає версії вашого браузера Google Chrome.
   - Завантажуємо веб-драйвер (ChromeDriver) з офіційного сайту ChromeDriver.
   - Розпаковуэмо завантажений архів ChromeDriver в зручне для нас місце на комп'ютері.
4. Створення Java проекту в IntelliJ IDEA:
   - Відкриваємо IntelliJ IDEA і обираємо "Create New Project".
   - Обераємо "Java" в списку проектів та під’єднуємо SDK, яке було встановлено.
   - Надаємо ім'я "DiplomaProject" для проекту , встановлюємо потрібне місцезнаходження та натискаємо "Створити проект".
5. Додавання Selenium WebDriver до проекту:
   - Використовуємо систему керування залежностями Maven для додавання Selenium WebDriver. IntelliJ IDEA інтегрується з Maven, що дозволяє легко додавати залежності.

Відкриваємо файл pom.xml у кореневому каталозі проекту. Та додаємо залежність для Selenium WebDriver:

```
<dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
```

```
            <version>LATEST</version>
            <scope>test</scope>
</dependency>
```

Після підключення залежностей, зазвичай, IntelliJ IDEA автоматично завантажить необхідну бібліотеку Selenium останньої версії. Якщо цього не сталося, використовуємо опцію "Reload Project"[7].

Також підключаємо залежність для Junit для забезпечення якості коду та автоматичного виконання тестів:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter-api</artifactId>
          <version>5.10.1</version>
    <scope>test</scope>
</dependency>
```

6. Створення тестового класу[7]:
   - Створюємо новий Java-клас для нашого тесту через правий клік на папці src/test/java і оберіть "New -> Java Class".
   - Надаємо класу ім'я "Test Suit" та натискаємо "OK".
7. Написання пробного тесту з використанням Selenium WebDriver:

Імпортуємо бібліотеки:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
```

Пишемо тест:

```
public class TrialTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.example.com");
        driver.quit();
    }
}
```

8. Запуск тесту:

Виконуємо тест, вибравши його в IntelliJ IDEA та натиснувши Ctrl + Shift + F10 або використовуючи опції в меню.

Наступними кроками буде написання тестів і виконання їх через описане вище середовище[7].

3.3.2 Налаштування середовища розробки для фреймворку Playwright

1. Встановлення Node.js:
   - Завантажуємо та встановлюємо Node.js з офіційного сайту Node.js.
   - Підключаємо слідкуючи за інструкціями для потрібної операційної системи.
2. Створення Java проекту в IntelliJ IDEA:
   - Відкриваємо вже завчасно встановлений IntelliJ IDEA і створюємо новий проект.

- Вибераємо "Java" в списку проектів та встановлене раніше JDK.

3. Додавання Залежностей Playwright:
   - Додаємо залежність Playwright до проекту. Додаємо наступний код до файлу pom.xml:

```
<dependency>
    <groupId>com.microsoft.playwright</groupId>
    <artifactId>playwright</artifactId>
    <version>1.40.0</version>
     <scope>test</scope>
</dependency>
```

   - Використваємо опцію "Reload" у нашому IDE для оновлення проекту.

4. Налаштування Playwright у Коді:
   - Створюємо екземпляр Playwright та браузер у коді перед написанням тестових сценаріїв.

Імпортуємо бібліотеки:

```
import com.microsoft.playwright.Browser;
import com.microsoft.playwright.BrowserContext;
import com.microsoft.playwright.Page;
import com.microsoft.playwright.Playwright;
```

Пишемо екземпляр:

```
public class PlaywrightExample {
    public static void main(String[] args) {
        try (Playwright playwright = Playwright.create())
{
```

```
            Browser                browser                =
playwright.chromium().launch();
            BrowserContext context = browser.newContext();
            Page page = context.newPage();
            browser.close();
        }
```

Ці кроки дозволили налаштувати середовище розробки для фреймворку Playwright з використанням IntelliJ IDEA та мови програмування Java. Наступними кроками буде написання тестів і виконання їх через описане вище середовище[4].

3.3.3 Налаштування середовища розробки для фреймворку Robot Framework

Налаштування середовища для фреймворку Robot Framework у комбінації з мовою програмування Java може бути реалізовано без використання інтегрованих середовищ розробки (IDE), але все ще треба встановити необхідні компоненти[4].

1. Встановлення Java Development Kit (JDK):
   - Завантажуємо та встановлюємо Java Development Kit (JDK) з офіційного сайту Oracle
   - Встановлюємо системну змінну JAVA_HOME, яка вказує на каталог, де розташовано JDK, налаштовуємо змінні оточення JAVA_HOME та PATH.
2. Встановлення Robot Framework:
   - Встановлюємо Robot Framework за допомогою pip (пакетний менеджер Python).

```
pip install robotframework
```

3. Встановлення залежностей с Framework для Java:
   - Встановлюємо бібліотеку “robotframework-java”:

`pip install robotframework-java` - це дозволяє використовувати Java-код в тестових сценаріях Robot Framework.

4. Налаштування змінних середовища:
   - Додаємо шлях до ‘java’ та ‘robot’ у системну змінну PATH. Це дозволяє запускати Java та Robot Framework з командного рядка.
5. Створення та Запуск тестового сценарію:
   - Створюємо Robot Framework тестовий сценарій у текстовому файлі з розширенням ‘.robot’.
   - Створюємо пробний файл example_test.robot:
   - Запускаємо тестовий сценарій з командного рядка:

```
robot example_test.robot
```

Зауваження:

Python тут є тому, що Robot Framework, основний фреймворк для автоматизованого тестування, написаний на мові програмування Python. Робот пропонує власну DSL (Domain-Specific Language) для написання тестових сценаріїв, але при цьому використовується велика кількість бібліотек та компонентів, які реалізовані на Python[4].

Java-бібліотеки для Robot Framework зазвичай використовують обгортки Python, тобто їхні Java-версії взаємодіють з Python-кодом Robot Framework. Це відбувається за допомогою бібліотек, таких як robotframework-java для взаємодії Java-коду та jrobotremoteserver для комунікації між Python та Java.

3.4 Висновок до розділу 3

Загальна характеристика Веб-застосунку Steam, як великий та багатофункціональний веб-застосунок, є втіленням складності сучасних веб-платформ. Ця платформа не просто дозволяє користувачам купувати та завантажувати ігри, але й створює ціле співтовариство, де гравці можуть взаємодіяти, ділитися враженнями та навіть створювати власний контент. Ця багатошаровість створює унікальні виклики для тестування, вимагаючи глибокого розуміння не лише технічних аспектів застосунку, але й потреб та поведінки користувачів.

В цьому розділі ми розібрали особливості розробки тестових вимог для обраного Веб-застосунку, а також розгортання тестового середовища.

Тому під час тестування Steam важливо не лише виявити потенційні помилки та вдосконалити функціональність, але також забезпечити позитивний досвід користувача. Збалансованість між технічною надійністю та зручністю інтерфейсу є ключовою для успіху веб-застосунків, подібних до Steam. Тестування з використанням Selenium ідеально підходить для досягнення цих цілей, оскільки воно дозволяє автоматизувати та деталізувати процес перевірки різних аспектів веб-застосунку, від базових функцій до складних користувацьких взаємодій.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ЗАСОБІВ SELENIUM ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ

4.1 Розробка тестових сценаріїв

Створюємо перелік тестових сценаріїв, забезпечуючи глибоке покриття функціональності Веб-застосунку Steam. Ці сценарії допоможуть переконатися, що застосунок працює коректно в різноманітних ситуаціях**.**

1). Тестовий сценарій для “Пошук Гри”:

Сценарій: Пошук гри за допомогою її назви.

Попередні умови: Користувач знаходиться на головній сторінці Steam.

Кроки:

1. Ввести в пошукове поле назву гри.
2. Натиснути кнопку пошуку.
3. Перевірити знайдені результати.

Очікуваний результат: Результат пошуку відповідає заданій назві гри.

2). Тестовий сценарій “Додавання гри до кошику”:

Сценарій: Перевірка можливості додавання гри до кошику.

Попередні умови: Потрібна гра вже успішно відкрита.

Кроки:

1. Натискаємо кнопку “Додати до кошику”.
2. Перевіряємо повідомлення, про додавання гри у кошик.
3. Повертаємось на головну сторінку
4. Перевіряємо що присутній елемент кошику на головній сторінці
5. Натискаємо на кошик
6. Перевіряємо що гра успішно відображається

Очікуваний результат: Обрана гра успішно відображається в кошику.

3). Тестовий сценарій для “Видалення Гри з Кошику”:

Сценарій: Перевірка видалення гри з кошику.

Попередні умови: Гра додана до кошику.

Кроки:

1. Перевіряємо що елемент кошику відображається на головній сторінці
2. Переходимо до кошику
3. Натискаємо кнопку “Вилучити”
4. Перевіряємо наявність повідомлення шо гра вилучена з кошика

5. Повертаємось на головну сторінку
6. Перевіряємо що елемента кошику не відображається на головній сторінці

Очікуваний результат: Гра успішно видалена, елемент кошику не відображається на головній сторінці.

Під час розробки тестових сценаріїв важливо враховувати не лише основні сценарії використання, але й крайні випадки та потенційні помилки. Це забезпечує всебічне тестування та допомагає покращити якість застосунку.

4.2 Проведення тестування Веб-застосунку

Для тестування обраного Веб-застосунку було як описанно вище обрано Selenium WebDriver та Playwright фреймворки з використанням мови програмування Java.

4.2.1 Тестування з Selenium WebDriver

Підключення бібліотек для написання тестів:

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
```

Створення публічного класу тесту з назвою: TestSuitDiploma

```
public class TestSuitDiploma {
private WebDriver driver;
private Map<String, Object> vars;
JavascriptExecutor js;
…}
```

Під’єднуємо driver для запуску браузера

```
public void setUp() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<String, Object>();
        }
```

Створюємо тест перевірки пошуку гри:

```
public void findGameTest() {
        driver.get("https://store.steampowered.com/");
        driver.manage().window().setSize(new    Dimension(1920,
1040));
driver.findElement(By.id("store_nav_search_term")).click();
driver.findElement(By.id("store_nav_search_term")).sendKeys("U
ncrashed        :        FPV        Drone        Simulator");
driver.findElement(By.cssSelector("#store_search_link        >
img")).click();
        {
```

```
        WebElement                  element                  =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(1) .search_name"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(1) .title"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(2) .title"));
```

```
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(3) .title"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(4) .title"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
```

```
        {
        WebElement                    element                    =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.cssSelector(".search_result_row:nth-
child(5) .title"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }

assertThat(driver.findElement(By.cssSelector(".search_result_r
ow:nth-child(1) .title")).getText(), is("Uncrashed : FPV Drone
Simulator"));

driver.findElement(By.cssSelector(".search_result_row:nth-
child(1) .title")).click();

assertThat(driver.findElement(By.id("appHubAppName")).getText(
), is("Uncrashed : FPV Drone Simulator"));
```

```
        {
        WebElement                    element                    =
driver.findElement(By.cssSelector("#userReviews                  >
.user_reviews_summary_row:nth-child(1)
.game_review_summary"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.cssSelector("#userReviews                  >
.user_reviews_summary_row:nth-child(2) > .summary"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
```

```
        WebElement                    element                    =
driver.findElement(By.cssSelector("#userReviews                  >
.user_reviews_summary_row:nth-child(1)"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                    element                    =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        driver.close();
        }
```

Створюємо тест перевірки Додавання до кошику:

```
public void addToCartTest() {

driver.get("https://store.steampowered.com/app/1682970/Uncrash
ed__FPV_Drone_Simulator/");
        driver.manage().window().setSize(new    Dimension(1920,
1040));

driver.findElement(By.cssSelector("#btn_add_to_cart_598887    >
span")).click();
        {
        WebElement                    element                    =
driver.findElement(By.cssSelector(".cart_item_img img"));
```

```
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        driver.findElement(By.cssSelector("#logo_holder
img")).click();
        {
        List<WebElement>              elements              =
driver.findElements(By.id("cart_link"));
        assert(elements.size() > 0);
        }
        driver.findElement(By.id("cart_link")).click();
        assertThat(driver.findElement(By.linkText("Uncrashed :
FPV Drone Sim")).getText(), is("Uncrashed : FPV Drone Sim"));
        driver.findElement(By.cssSelector("#logo_holder
img")).click();
        driver.close();
        }
```

Створюємо тест перевірки Видалення гри з кошику:

```
public void deleteFromCart() {
        driver.get("https://store.steampowered.com/");
        driver.manage().window().setSize(new    Dimension(1920,
1040));
        {
        WebElement                  element                  =
driver.findElement(By.cssSelector("#logo_holder img"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
        }
        {
        WebElement                  element                  =
driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
        }
        {
        List<WebElement>               elements               =
driver.findElements(By.id("cart_link"));
        assert(elements.size() > 0);
        }
        driver.findElement(By.id("cart_link")).click();
        driver.findElement(By.linkText("Вилучити")).click();

assertThat(driver.findElement(By.cssSelector(".cart_status_mes
sage")).getText(), is("Ваш предмет було вилучено!"));
```

```
        driver.findElement(By.cssSelector("#logo_holder
img")).click();
        driver.close();
        }
        }
```

Визиваємо закриття браузера:

```
public void tearDown() {
        driver.quit();
        }
```

4.2.2 Тестування з Playwright

Підключення бібліотек для написання тестів:

```
import com.microsoft.playwright.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
```

Створення публічного класу тесту з назвою TestSuitDiploma:

```
public class TestSuitDiploma {
     private Browser browser;
private BrowserContext context;
private Page page;
```

…}

Під’єднуємо driver для запуску браузера:

```
 public void setUp() {
 browser      =      Playwright.create().chromium().launch(new
BrowserType.LaunchOptions().setHeadless(false));
context = browser.newContext();
page = context.newPage();
}
```

Створюємо тест перевірки пошуку гри:

```
public void findGameTest() {
        page.navigate("https://store.steampowered.com/");
        page.setViewportSize(1920, 1040);
        page.click("#store_nav_search_term");
        page.type("#store_nav_search_term",  "Uncrashed  :  FPV
Drone Simulator");
        page.click("#store_search_link > img");
        page.hover(".search_result_row:nth-child(1)
.search_name");
        page.hover(".search_result_row:nth-child(1) .title");
        page.hover(".search_result_row:nth-child(2) .title");
        page.hover(".search_result_row:nth-child(3) .title");
        page.hover(".search_result_row:nth-child(4) .title");
        page.hover(".search_result_row:nth-child(5) .title");
     assertEquals(page.innerText(".search_result_row:nth-
child(1) .title"), "Uncrashed : FPV Drone Simulator");
        page.click(".search_result_row:nth-child(1) .title");
```

```
        assertEquals(page.innerText("#appHubAppName"),
"Uncrashed : FPV Drone Simulator");
        page.hover("#userReviews                                    >
.user_reviews_summary_row:nth-child(1) .game_review_summary");
        page.hover("#userReviews                                    >
.user_reviews_summary_row:nth-child(2) > .summary");
        page.hover("#userReviews                                    >
.user_reviews_summary_row:nth-child(1)");
        page.close();
    }
```

Створюємо тест перевірки Додавання до кошику:

```
public void addToCartTest() {

page.navigate("https://store.steampowered.com/app/1682970/Uncr
ashed__FPV_Drone_Simulator/");
        page.setViewportSize(1920, 1040);
        page.click("#btn_add_to_cart_598887 > span");
        page.hover(".cart_item_img img");
        page.click("#logo_holder img");
        assertTrue(page.locator("#cart_link").count() > 0);
        page.click("#cart_link");
        assertEquals(page.locator("text=Uncrashed : FPV Drone
Sim").innerText(), "Uncrashed : FPV Drone Sim");
        page.click("#logo_holder img");
        page.close();
    }
```

Створюємо тест перевірки Видалення гри з кошику:

```
public void deleteFromCart() {
        page.navigate("https://store.steampowered.com/");
        page.setViewportSize(1920, 1040);
        page.hover("#logo_holder img");
        assertTrue(page.locator("#cart_link").count() > 0);
        page.click("#cart_link");
        page.click("text=Вилучити");
        assertEquals(page.innerText(".cart_status_message"),
"Ваш предмет було вилучено!");
        page.click("#logo_holder img");
        page.close();
    }
```

Визиваємо закриття браузера:

```
public void tearDown() {
        browser.close();
    }
```

4.3 Аналіз результатів

Після написання та виконаання автоматизованого тестування Веб-застосунка Steam за допомогою фреймворка Selenium WebDriver, хотів би зробити результати пройденого шляху. Фреймворк Selenium - це потужний інструмент для автоматизації тестування веб-застосунків, який дозволяє розробникам та тестувальникам ефективно створювати та виконувати тести для веб-інтерфейсів.

Після порівняння Selenium WebDriver з Playwright та Robot Framework, можу сказати, що складно виділити явні їх переваги один на одним. Але більш успішно виходило написання та виконання тестів на Selenium, завдяки його великій та активній спільноті, та завдяки відкритим документаціям.

4.4 Надання рекомендацій

Насправді вибір інструмента для автоматизації, залежить від вимог для тестування та особистих переваг. Але я рекомендую використання Selenium WebDriver для написання автоматизованих тестів.

Якщо ви плануєте використовувати Selenium для автоматизації тестування веб-застосунків, ось деякі рекомендації, які можуть бути корисними:

- Перш ніж розпочати роботу з Selenium, важливо вивчити основні концепції автоматизації тестування, такі як локатори, робота з елементами сторінки, очікування (waits) та інші.
- Виберіть мову програмування, з якою вам комфортно працювати. Selenium підтримує кілька мов, таких як Java, Python, C#, Ruby і JavaScript. Або проаналізуйте вимоги для тестування вашого проекту, та зробіть вибір підходящої мови програмування.

- Дотримуйтесь хороших практик програмування, таких як структурування коду, використання функцій, коментарі та іменування змінних. Це полегшить розуміння та підтримку вашого коду.
- Розробляйте тести так, щоб вони були модульними і незалежними. Це дозволить вам легко виявляти і виправляти помилки, а також забезпечить стабільність вашого тестового набору.
- Якщо ви використовуєте систему управління конфігурацією або інші інструменти тестування (наприклад, TestNG, JUnit, Jenkins), вивчіть, як Selenium може бути інтегрований з ними для поліпшення управління тестами та звітності.
- Аналізуйте ваш застосунок для розуміння його архітектури це допомагає ефективно визначити, як створювати тести та які частини функціоналу тестувати.
- Докладно вивчайте документацію Selenium для того, щоб мати повне розуміння можливостей та особливостей фреймворку.
- Використовуйте можливості відладки для аналізу та виправлення помилок у вашому коді.
- Використовуйте очікування перед взаємодією з елементами, щоб уникнути проблем через недостатній час завантаження сторінки чи елементів.

Якщо дотриматись вище поданих рекомендацій, то Selenium перетвориться в ваших руках на справжню чарівну палицю, для створення автоматизованих тестів, вам буде легко вливатись та вдосконалюватись у цьому напрямку.

ВИСНОВКИ

Автоматизоване тестування Веб-застосунків у наш час є важливим і актуальним компонентом сучасного процесу розробки програмного забезпечення

сприяє високій якості продукту, економить час і ресурси у довгостроковій перспективі.

Обрані мною інструменти були для того, щоб зробити порівняння за допомогою переваг та недоліків відносно нових, популярних та безкоштовні open-source фреймворків Playwright та Robot Framework з вже потужним досвідченим Selenium WebDriver і дізнатися чи має він переваги над новим поколінням інструментів.

У процесі дослідження і порівняння обраних інструментів було виявлено що кожен з них має свою переваги та недоліки.

Selenium WebDriver його переваги полягають в тому, що він має широку підтримку браузерів, гарну мовну залежність, має простий та зрозумілий інтерфейс для взаємодії з елементами веб-сторінок, та є популярним інструментом через що має активну спільноту розробників. Але як мінус має нестабільність під час оновлення браузерів, залежність від інтерфейсу користувача та не самий швидкий у виконанні тестів.

Playwright позитиві сторони це - має гарну швидкодію, підтримує немало мов програмування, має широкі можливості для тестування. З недоліків виділяється: складність налаштування, має меншу активну спільноту чим Selenium, що може ускладнити процес вивчення інструмента, обмежена підтримка для деяких браузерів та має труднощі з інтеграцією в деякі середовища.

Robot Framework проявив себе з такими перевагами як простота вивчення та використання, велика кількість бібліотек та співпраця з іншими інструментами. А з недоліків виявилась: обмежена функціональність, нестабільні оновлення, невелика активна спільнота, що ускладнює процес вивчення та неоптимальна підтримка для деяких технологій.

Всі інструменти мають свої плюси та мінуси, але все залежить від потреб виконання тестів та особистого бажання, все індивідуально.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Appel F. Testing with Junit. Packt Publishing, Limited, 2015. URL: https://vdoc.pub/documents/testing-with-junit-7di3vneu7hs0 (дата звернення: 09.04.2023)
2. Armstrong D. Learn XPath Fast: A Beginner-Friendly, Exercise-based Course for People Who Want to Use XPath in Selenium, SQL Server, XQuery or Anywhere Else. Independently Published, 2020. URL: https://www.amazon.com/Learn-XPath-Fast-beginner-friendly-exercise-based (дата звернення: 05.06.2023)
3. Bahmutov G., Kinsbruner E. a Frontend Web Developer's Guide to Testing: Explore Leading Web Test Automation Frameworks and Their Future Driven by Low-Code and AI. Packt Publishing, Limited, 2022. URL: https://www.scribd.com/document/665246379/A-Frontend-Web-Developer-s-Guide-to-Testing-Explore-Leading-Web-Test-Automation-Frameworks-Eran-Kinsbruner-Z-Library (дата звернення: 21.10.2023)
4. Bisht S. Robot Framework Test Automation. Packt Publishing, 2013. 98 p. URL: https://vdoc.pub/documents/robot-framework-test-automation-6ceg2def1m20 (дата звернення: 11.02.2023)
5. Cocchiaro C., Chaubal P. A. Selenium Framework Design in Data-Driven Testing: Build Data-Driven Test Frameworks Using Selenium WebDriver, AppiumDriver, Java, and TestNG. Packt Publishing, Limited, 2018. 354 p. URL: https://dokumen.pub/selenium-framework-design-in-data-driven-testing-9781788473576.html (дата звернення: 01.06.2023)
6. Goel S., Vartak K. Selenium with Support of both TestNG and Cucumber Frameworks. *International Journal of Computer Applications*. 2018. Vol. 180, no. 51. P. 1–5. URL: https://doi.org/10.5120/ijca2018917334 (дата звернення: 02.06.2023).

7. Gundecha U. Selenium Testing Tools Cookbook. Packt Publishing, Limited, 2012. 326 p. URL: https://vdoc.pub/documents/selenium-testing-tools-cookbook-7redl6llkka0 (дата звернення: 03.01.2023).
8. Massol V. JUnit in action. Greenwich, Conn : Manning, 2004. 360 p. URL: https://vdoc.pub/documents/junit-in-action-4gr2oobp4k50 (дата звернення: 10.10.2023)
9. Menon V. TestNG Beginner's Guide. Packt Publishing, 2013. 276 p. URL:http://chenweixiang.github.io/docs/TestNG_Beginner_Guide.pdf (дата звернення: 22.06.2023)
10. Müller M. Testing with Selenium. *Practical JSF in Java EE 8*. Berkeley, CA, 2018. P. 69–82. URL: https://doi.org/10.1007/978-1-4842-3030-5_7 (дата звернення: 23.06.2023).
11. Robillard M. P. Introduction to Software Design with Java. Cham : Springer International Publishing, 2019. URL: https://doi.org/10.1007/978-3-030-24094-3 (дата звернення: 14.06.2023).
12. Опис тестового застосунку. URL: https://ru.wikipedia.org/wiki/Steam
13. Russell J. P. Java Programming for the Absolute Beginner (For the Absolute Beginner (Series).). Course Technology PTR, 2002. 528 p. URL: https://archive.org/details/javaprogrammingf0000russ/page/n9/mode/2up (дата звернення: 13.06.2023)
14. Selenium as a Free Tool to Test for Java Web Application / J. P. R. d. Santos et al. *International Journal of Advanced Engineering Research and Science*. 2020. Vol. 7, no. 4. P. 135–139. URL: https://doi.org/10.22161/ijaers.74.15 (дата звернення: 21.06.2023).
15. Solutions I. P. S. Java Programming. Pinnacle Software Solutions, Inc, 2000. 430 p.

16. UI web automation testing using testng log4j. *Journal of Xidian University*. 2020. Vol. 14, no. 7. URL: https://doi.org/10.37896/jxu14.7/102 (дата звернення: 02.06.2023).
17. Завантаження Selenium WebDriver URL: https://www.selenium.dev/downloads/
18. Киричек Г., Тягунова М., Курач А. Автоматизоване тестування Веб-платформ з використанням Java та Selenium. Information technology and society. 2022. № 1. С. 31–37.
URL: https://doi.org/10.32689/maup.it.2022.1.4 (дата звернення: 24.06.2023).
19. Затоковий С.С. магістрант, Кломоєць Г.П., канд. ф.-м. наук, доц. — науковий керівник. Особливості використання фреймворку Selenium при тестуванні Веб-застосунків. *Молода наука-2023* : зб. наук. праць студентів, аспіра., докторантів і молодих вчених. Запоріжжя : ЗНУ, 2023. Т. 5. С. 87-89
20. ОРІВНЯННЯ ПОПУЛЯРНИХ ТЕСТОВИХ ФРЕЙМВОРКІВ JUNIT И TESTNG / О. Каратанов та ін. *Молодий вчений*. 2021. № 5 (93). С. 164–170. URL: https://doi.org/10.32839/2304-5809/2021-5-93-31 (дата звернення: 25.06.2023).
21. Тестовий застосунок Steam URL: https://store.steampowered.com/
22. Fewster M., Graham D. Experiences of Test Automation: Case Studies of Software Test Automation. Pearson Education, Limited, 2021. URL: https://vdoc.pub/documents/experiences-of-test-automation-case-studies-of-software-test-automation-75dtgnk512i0
23. Затоковий С.С магістрант, Кломоєць Г.П., канд. ф.-м. наук, доц. — науковий керівник. Особливості використання фреймворку Selenium при тестуванні Веб-застосунків. Матеріали III всеукраїнської науково-практичної конференції за участю молодих науковців. Запоріжжя : ЗНУ, 2023. С. 130-131

Декларація

академічної доброчесності

здобувача вищої освіти ЗНУ

Я__Затоковий Сергій Сергійович__, студент___2__курсу,
форми здобуття освіти __денна_____,
Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ
Спеціальності 121 Інженерія програмного забезпечення,
адреса електронної пошти_ipz18bd-03@zsea.edu.ua_____,
підтверджую, що написана мною кваліфікаційна робота на тему «**Особливості використанння фреймворку Selenium при тестуванні Веб-застосунків**»
відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;
- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет-системи, а також на архівування роботи в базі даних цієї системи.

Дата_____ Підпис_____ С.С. Затоковий
(студент)

Дата_____ Підпис_____ Г.П. Коломоєць
(науковий керівник)