Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ІНСТРУМЕНТУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ E-COMMERCE ПЛАТФОРМ ЗА ДОПОМОГОЮ PLAYWRIGHT ТА ПАТЕРНІВ ООП

DEVELOPMENT AND RESEARCH OF AN AUTOMATION TOOL FOR TESTING E-COMMERCE PLATFORMS USING PLAYWRIGHT AND OOP PATTERNS

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Маркін А. І.
Керівник:
к.т.н., доцент Суринович О. М.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ
Маркіну Андрію Ігоровичу

1. Тема кваліфікаційної роботи: Розробка та дослідження інструменту автоматизації тестування e-commerce платформ за допомогою Playwright та патернів ООП
Керівник роботи: Суринович Олена Миколаївна, доц., к.т.н. _____
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 4 грудня 2025р.
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення.*
4. Зміст розрахунково-пояснювальної записки: *аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення, опис функціонального наповнення об'єкта проектування, практична реалізація об'єкта проектування.*
5. Перелік графічного матеріалу *2 рисунки, 3 таблиці, 1 додаток.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Суринович О. М.*		
Теоретичне дослідження та практична реалізація	*Суринович О. М.*		
Експериментальне дослідження системи	*Суринович О. М.*		
Нормоконтроль	*Повстяна Ю. С*		
Гарант ОП	*Андрущак І. Є*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Суринович О. М.*		

7. Дата видачі завдання *«02» квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	до 02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	до 24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	до 01.11.2025	
4	Описати засоби розробки об'єкта проектування	до 19.11.2025	
5	Практична реалізація об'єкта проектування	до 26.11.2025	
6	Розробити методику для проведення експерименту	до 05.11.2025	
7	Провести аналіз результатів експерименту	до 15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	до 04.12.2025	

Здобувач вищої освіти _____ *Маркін А. І.*
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ *Суринович О. М*
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Маркін А. І. Розробка та дослідження інструменту автоматизації тестування e-commerce платформ за допомогою Playwright та патернів ООП. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

У роботі проаналізовано сучасні підходи до автоматизації тестування e-commerce платформ та можливості фреймворку Playwright. Обґрунтовано вибір технологій і архітектурних рішень, розроблено інструмент автоматизації тестування із застосуванням патернів ООП. Реалізовано набір автоматизованих тестових сценаріїв для ключових компонентів платформи та проведено експериментальне дослідження ефективності й стабільності роботи розробленого інструменту. Отримані результати підтвердили його відповідність поставленим вимогам і практичну доцільність використання.

Ключові слова: автоматизоване тестування, e-commerce платформа, Playwright, TypeScript, Page Object, патерни ООП, CI/CD, e2e-тестування.

ABSTRACT

Markin Andrii. Development and Research of an Automation Tool for Testing E-commerce Platforms Using Playwright and OOP Patterns. Manuscript.

Master's qualification thesis in the Educational Program «Software Engineering», specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

This work analyzes modern approaches to the automation of testing for e-commerce platforms and the capabilities of the Playwright framework. The choice of technologies and architectural solutions is justified, and a test automation tool based on object-oriented design patterns is developed. A set of automated test scenarios for the key components of the platform is implemented, and an experimental study of the efficiency and stability of the developed tool is conducted. The obtained results confirmed its compliance with the defined requirements and the practical feasibility of its application.

Keywords: automated testing, e-commerce platform, Playwright, TypeScript, Page Object, OOP patterns, CI/CD, end-to-end testin

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний ринок e-commerce перебуває у фазі стрімкого розвитку: зростає кількість онлайн-магазинів, складність їхніх бізнес-процесів та вимоги до якості користувацького досвіду. У цьому середовищі критично важливим є забезпечення стабільної, передбачуваної та безпомилкової роботи веб-платформ, оскільки будь-які збої безпосередньо впливають на фінансові показники бізнесу. Автоматизація тестування стала ключовим інструментом підвищення надійності e-commerce систем, однак традиційні підходи та інструменти не завжди забезпечують достатній рівень продуктивності, гнучкості та масштабованості.

Незважаючи на значний прогрес у сфері автоматизації тестування, спостерігаються прогалини, пов'язані з побудовою ефективних, розширюваних і підтримуваних тестових фреймворків. Проблемними залишаються питання повторного використання коду, організації тестової архітектури, забезпечення незалежності тестів та інтеграції з CI/CD. Світові тенденції вказують на перехід до легковагих, швидких і стабільних інструментів, таких як Playwright, що підтримують багатоплатформність, паралельність і сучасні веб-протоколи. Одночасно підвищується роль об'єктно-орієнтованих підходів та патернів проєктування, що дають змогу створювати структуровані та масштабовані фреймворки.

Ураховуючи зростаючу складність e-commerce систем та потребу в забезпеченні високої якості їх функціонування, розробка інструменту автоматизації тестування на базі сучасних технологій є актуальним і практично значущим завданням. Дослідження взаємодії Playwright з архітектурними патернами ООП дозволяє створити високопродуктивне рішення, здатне відповідати вимогам сучасних комерційних платформ.

Метою роботи є створення та дослідження інструменту автоматизації тестування e-commerce платформ, який забезпечує високу продуктивність, масштабованість, структурованість та зручність підтримки завдяки

використанню фреймворку Playwright і патернів об'єктно-орієнтованого програмування.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати сучасні підходи та інструменти автоматизації тестування e-commerce платформ;

– дослідити можливості фреймворку Playwright для e2e-тестування веб-додатків;

– виконати аналіз архітектурних патернів ООП, що можуть бути застосовані у тестовому фреймворку;

– сформулювати вимоги до інструменту автоматизації тестування та побудувати його архітектурну модель;

– розробити інструмент автоматизації тестування з використанням Playwright та патернів ООП відповідно до змісту роботи;

– реалізувати набір тестових сценаріїв для ключових компонентів e-commerce платформи;

– провести експериментальне дослідження ефективності, стабільності та продуктивності розробленого інструменту;

– виконати аналіз отриманих результатів та визначити переваги запропонованого рішення порівняно з існуючими підходами.

Об'єкт дослідження – інструменти та технології автоматизації тестування e-commerce платформ.

Предмет дослідження – архітектурні підходи, методи побудови та технічні характеристики інструменту автоматизації тестування, що базується на Playwright та патернах ООП.

Наукова новизна одержаних результатів полягає у формуванні нового підходу до побудови фреймворку автоматизації тестування e-commerce платформ.

Запропонований підхід отримав подальший розвиток у вигляді цілісної архітектури, що орієнтована на розширюваність, модульність та повторне використання компонентів, що відрізняє його від традиційних фреймворків

автоматизації.

Практична цінність роботи полягає у можливості прямого використання розробленого інструменту в автоматизації тестування реальних e-commerce платформ. Створений фреймворк може бути інтегрований у процеси CI/CD, адаптований під потреби різних компаній та розширений для підтримки складніших бізнес-логік. Отримані результати також можуть бути використані у навчальному процесі для підготовки фахівців у сфері автоматизації тестування та розробки програмного забезпечення.

Апробація результатів дослідження. Суринович О. М., Маркін А. І. Розробка та дослідження інструменту автоматизації тестування E-commerce платформ за допомогою Playwright та патернів ООП. Proceedings of the International scientific and practical conference – Science, Technology and Culture: Challenges and Perspectives. November 17-19, 2025. Paris, France, 2025. P.26-29 [1].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ E-COMMERCE ПЛАТФОРМ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Автоматизація тестування є ключовим елементом забезпечення якості програмного забезпечення, особливо для систем із високою динамікою змін, таких як e-commerce платформи. З огляду на активний розвиток онлайн-торгівлі та постійне ускладнення бізнес-логіки, виникає потреба у використанні гнучких, високопродуктивних і масштабованих інструментів для автоматичного тестування. Сучасні тестові фреймворки пропонують різні підходи до побудови автоматизованих рішень, але не всі з них у достатній мірі враховують специфіку e-commerce, зокрема складну взаємодію з UI, різноманітні варіації поведінки користувачів та високий обсяг регресійних сценаріїв.

У наукових та технічних дослідженнях останніх років значна увага приділяється швидкодії та стабільності інструментів автоматизації. Дослідники відзначають, що фреймворки нового покоління (Playwright, Cypress) демонструють суттєві переваги порівняно з Selenium у частині ізоляції тестів, синхронізації подій та мультибраузерної підтримки [2]. Playwright, зокрема, забезпечує автоматичне очікування елементів, паралельне виконання та стабільну роботу з сучасними фронтенд-технологіями, що робить його перспективним вибором для складних веб-платформ.

Окремим напрямом досліджень є застосування об'єктно-орієнтованих принципів у побудові тестових фреймворків. Використання патернів проектування (Page Object, Factory, Strategy) дозволяє підвищити модульність, повторне використання коду та масштабованість тестової інфраструктури. За даними технічних оглядів, впровадження ООП-підходів у тест-автоматизацію суттєво знижує витрати на підтримку тестів та пришвидшує розробку нових

сценаріїв [3]. Це особливо важливо для e-commerce систем, де тестові набори можуть налічувати сотні сценаріїв, що постійно оновлюються.

Станом на сьогодні більшість існуючих рішень для автоматизації e-commerce платформ мають кілька недоліків. По-перше, значна частина з них побудована на застарілих технологіях, що не враховують специфіку сучасних SPA-додатків. По-друге, часто відсутня чітка архітектурна структура, що ускладнює підтримку тестів. По-третє, не всі фреймворки забезпечують високу швидкодію при роботі з великою кількістю даних та тестових конфігурацій.

Проведений аналіз дозволяє сформувати комплексне бачення проблеми: існує необхідність у створенні інструменту автоматизації тестування, який поєднуватиме переваги сучасних технологій (Playwright), тестових архітектур (ООП-патерни) та орієнтації на специфіку e-commerce. Таким чином, на момент виконання роботи актуальною задачею є розробка структурованого, масштабованого та ефективного фреймворку автоматизації, здатного забезпечити стабільні результати тестування в умовах швидкозмінних вимог.

1.2 Огляд і аналіз методів та засобів розробки інструментів автоматизації тестування для вирішення проблеми дослідження

Розв'язання задачі розробки інструменту автоматизації тестування e-commerce платформ потребує детального аналізу існуючих підходів, фреймворків та архітектурних рішень, що дозволяють підвищити ефективність перевірки функціоналу, стабільність виконання сценаріїв та можливість масштабування тестового середовища. У сучасній практиці автоматизації тестування використовуються різні програмні засоби, які можна умовно поділити на два типи: готові фреймворки для браузерної автоматизації та комплексні програмні екосистеми, інтегровані в CI/CD-процеси.

Одним із найбільш відомих інструментів є Selenium – платформа, що протягом останнього десятиліття стала стандартом у сфері веб-автоматизації. Selenium підтримує роботу з різними мовами програмування та браузерами,

забезпечує широкі можливості кастомізації та має значну спільноту користувачів. Проте архітектура WebDriver, яка працює через мережеві протоколи взаємодії з браузером, створює затримки, знижує стабільність і збільшує час виконання тестів. Крім того, налаштування очікувань, синхронізація дій та керування складними асинхронними подіями у e-commerce платформах вимагають додаткових інженерних рішень, що ускладнює підтримку тестового фреймворку [6].

Більш сучасний підхід пропонує Cypress – інструмент, який працює всередині браузера та забезпечує швидке виконання тестів і приємний для розробника інтерфейс. Cypress надає зручні засоби дебагінгу, детальний лог виконання та автоматичне очікування подій. Однак інструмент має істотні обмеження: відсутність повноцінної багатобраузерної підтримки, неможливість виконання тестів у кількох вкладках, обмежена робота з iframe та специфічна архітектура «в одному циклі подій». Для e-commerce систем, де потрібно враховувати поведінку сайту в різних браузерах та сценарії одночасних дій, Cypress може бути недостатньо гнучким.

Останніми роками значної популярності набув Playwright – фреймворк, який надає можливість повноцінної міжбраузерної автоматизації, паралельного запуску, роботи з кількома вкладками та контекстами, автоматичних очікувань і детального контролю над мережею. На відміну від Cypress, Playwright не обмежений одним середовищем виконання, а також дозволяє легко моделювати сценарії авторизації, обробку кошика, тестування платіжних модулів та багатокрокові e-commerce операції. За результатами незалежних порівняльних досліджень, Playwright демонструє вищу швидкість, стабільність та гнучкість у складних сценаріях порівняно з Cypress і Selenium [4].

Окреме місце у процесі створення інструменту автоматизації займають архітектурні підходи. Використання принципів об'єктно-орієнтованого програмування (ООП) дає можливість впорядкувати структуру тестів і забезпечити високу зрозумілість фреймворку. Патерн Page Object дозволяє відокремити тестові сценарії від реалізації поведінки сторінок, забезпечуючи

легкість підтримки та масштабування. Крім того, застосування патернів Dependency Injection, Strategy та Factory дозволяє будувати гнучку архітектуру, в якій залежності між об'єктами мінімізовані, а логіка розширення та конфігурації спрощена.

Застосування ООП-підходів є критично важливим у великих e-commerce системах, де кількість сторінок і бізнес-процесів може сягати сотень, а тести повинні бути максимально читабельними й повторно використовуваними. У дослідженнях сучасних підходів до побудови test automation frameworks зазначено, що застосування модульної, об'єктно-орієнтованої архітектури є ключовим фактором ефективної автоматизації великих проєктів [6].

Додатково інструмент автоматизації повинен підтримувати інтеграцію з CI/CD, що дозволяє виконувати тести після кожного оновлення коду. Для цього використовуються Azure DevOps, Jenkins та інші системи, які забезпечують контейнеризацію середовища та автоматичний запуск тестів у хмарних платформах або локальних контейнерах. Використання Docker також спрощує розгортання інструменту автоматизації, роблячи його незалежним від операційної системи та локальної конфігурації.

З урахуванням проведеного аналізу наявних інструментів автоматизації (табл. 1.1) та сучасних архітектурних підходів встановлено, що найоптимальнішим рішенням для автоматизації тестування e-commerce платформ є розробка спеціалізованого інструменту на базі фреймворку Playwright із застосуванням патернів ООП, інтеграцією з CI/CD та підтримкою модульної розширюваної структури. Обраний підхід дозволяє поєднати високу швидкодію, стабільність виконання тестів та гнучкість архітектури, що є критично важливим для складних комерційних систем. Використання патернів ООП забезпечує чітке розмежування відповідальностей між компонентами, підвищує повторне використання коду та спрощує масштабування тестового фреймворку. Інтеграція з CI/CD дає змогу виконувати автоматизовані перевірки на регулярній основі, оперативно виявляти дефекти на ранніх етапах розробки та підтримувати стабільну якість програмного продукту.

Таблиця 1.1 – Порівняльний аналіз інструментів автоматизації тестування

Параметр	Selenium	Cypress	Playwright
Підтримка браузерів	Chrome, Firefox, Edge, Safari	Chromium (обмежено), Firefox, Electron	Chrome, Firefox, WebKit
Швидкість виконання	Середня	Висока	Висока
Стабільність	Залежить від WebDriver	Висока	Дуже висока
Підтримка вкладок	Обмежена	Обмежена	Повна
Архітектура	Веб-драйвер	Всередині браузера	Рідні драйвери
Масштабованість	Середня	Обмежена	Висока
Підтримка паралельного запуску	Залежить від інструментів	Обмежена	Є «з коробки»
Оптимальна сфера застосування	Загальні веб-тести	SPA, UI-тести	Складні e-commerce, інтеграційні UX-сценарії

1.3 Постановка завдання на кваліфікаційну роботу магістра

У ході аналізу було виявлено, що традиційні підходи, зокрема ті, що ґрунтуються на Selenium WebDriver, у багатьох випадках не забезпечують достатньої швидкодії, ізоляції тестів та коректної роботи з великою кількістю динамічного контенту. Натомість Playwright демонструє суттєві переваги у контексті e-commerce, де необхідно працювати зі складними UI-патернами, асинхронною поведінкою сторінок та паралельним виконанням тестів.

Враховуючи необхідність створення сучасного та масштабованого рішення, було визначено, що основою кваліфікаційної роботи стане розробка власного інструменту автоматизації тестування на базі Playwright із застосуванням принципів ООП та патернів проєктування. Такий підхід

дозволить отримати структуру, яку можна адаптувати до різних e-commerce платформ, не змінюючи архітектурного ядра. Крім того, використання патернів Page Object і SOLID забезпечить стабільність і повторне використання компонентів, а також спростить підтримку тестів у довгостроковій перспективі [7]. Тому для реалізації поставленої задачі було сформовано такі цілі:

– проаналізувати сучасні підходи та інструменти автоматизації тестування e-commerce платформ. У межах цієї цілі розглядаються найбільш поширені фреймворки, їх функціональні можливості, переваги та обмеження. Також виконується порівняльний аналіз рішень, що використовуються у сучасних комерційних проєктах;

– дослідити можливості фреймворку Playwright для e2e-тестування веб-додатків. Проводиться аналіз його архітектури, підтримуваних браузерів, інструментів роботи з асинхронністю та механізмів стабілізації тестів;

– виконати аналіз архітектурних патернів ООП, що можуть бути застосовані у тестовому фреймворку. Розглядаються найбільш доцільні патерни проєктування для побудови масштабованого та підтримуваного тестового рішення. Обґрунтовується вибір кожного з них для конкретних задач автоматизації;

– сформулювати вимоги до інструменту автоматизації тестування та побудувати його архітектурну модель. Визначаються функціональні та нефункціональні вимоги до системи автоматизації. На основі цих вимог формується логічна та модульна структура майбутнього інструменту;

– розробити інструмент автоматизації тестування з використанням Playwright та патернів ООП відповідно до змісту роботи. Реалізація здійснюється з урахуванням принципів модульності, розширюваності та повторного використання коду. Інструмент орієнтований на практичне використання у реальних проєктах;

– реалізувати набір тестових сценаріїв для ключових компонентів e-commerce платформи. Тести охоплюють основні бізнес-процеси, зокрема роботу

з каталогом, кошиком, оформлення замовлень та обліковими записами користувачів. Забезпечується перевірка критично важливого функціоналу системи;

– провести експериментальне дослідження ефективності, стабільності та продуктивності розробленого інструменту. Тестування здійснюється в різних умовах навантаження та на різних середовищах. Оцінюється швидкодія виконання тестів та їх стійкість до змін у системі;

– виконати аналіз отриманих результатів та визначити переваги запропонованого рішення порівняно з існуючими підходами. На основі експериментальних даних робляться узагальнюючі висновки щодо ефективності розробленого інструменту. Формулюються рекомендації щодо його практичного застосування.

Реалізація кожного із зазначених етапів спрямована на формування повноцінного інструменту автоматизованого тестування, який відповідатиме стандартам індустрії та забезпечуватиме надійну перевірку якості e-commerce платформи. Зокрема, створення архітектури на базі патернів дозволить легко розширювати систему новими модулями без значних змін у базовому коді. Модуль авторизації забезпечить можливість виконання як простих smoke-тестів, так і складних end-to-end сценаріїв з використанням реальних користувацьких потоків.

Система тестових даних надасть можливість гнучко керувати змінними параметрами тестів, що є критично важливим для e-commerce функціоналу. Інтеграція з CI/CD забезпечить автоматизацію процесу тестування на всіх етапах розробки та дозволить попереджати появу регресій. Експериментальна перевірка продуктивності дозволить оцінити ефективність створеного інструменту в реальних умовах та визначити можливості його подальшої оптимізації.

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз особливостей

сучасних e-commerce платформ, що дозволило визначити ключові проблеми, пов'язані з масштабованістю, складністю бізнес-процесів та необхідністю надійної автоматизації тестування. Розгляд існуючих технік тестування, структур проєктування та інструментів автоматизації дав змогу визначити, які підходи є найбільш ефективними для побудови стабільного та гнучкого тестового фреймворку.

Далі розглянуто можливості застосування комп'ютерних технологій для вирішення поставленої проблеми, зокрема використання Playwright, об'єктно-орієнтованої архітектури, шаблонів проєктування та інтеграції в CI/CD. Проведено порівняння різних підходів до створення автоматизованих систем тестування, що дало змогу визначити оптимальний набір інструментів і методів для реалізації високонадійної системи контролю якості e-commerce продуктів.

На основі проведеного аналізу були сформовані конкретні цілі кваліфікаційної роботи, спрямовані на створення повноцінного інструменту автоматизації тестування. Серед них – розробка архітектури фреймворку, модулів авторизації й управління тестовими даними, реалізація набору бізнес-критичних тестів та інтеграція фреймворку в CI/CD-процеси. Досягнення цих цілей забезпечить створення ефективної системи автоматизованого тестування, здатної покращити якість, стабільність та швидкість розробки e-commerce платформи.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНСТРУМЕНТУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ E-COMMERCE ПЛАТФОРМ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розробка інструменту автоматизації тестування e-commerce платформи вимагає комплексного підходу до вибору технологій, алгоритмів та програмних засобів, оскільки такі системи характеризуються високою динамічністю, великою кількістю користувацьких сценаріїв, складною логікою бізнес-процесів та інтеграцією з зовнішніми сервісами. Основною метою є створення стабільного, масштабованого та придатного до довготривалої підтримки рішення, здатного забезпечити високу якість програмного продукту в умовах постійних змін.

В основі фреймворку автоматизації використовується об'єктно-орієнтована модель проєктування. Це дозволяє сформувати чітку структуру коду, уникнути дублювання, спростити масштабування та зробити тести незалежними від реалізації інтерфейсу. Архітектура фреймворку побудована з урахуванням принципів SOLID, що забезпечує низьку зв'язаність компонентів, можливість розширення без модифікації базового коду та спрощує супровід проєкту.

Мова програмування TypeScript була обрана як основна мова реалізації, оскільки вона забезпечує статичну типізацію поєднання з гнучкістю JavaScript, що значно знижує кількість помилок на етапі компіляції та підвищує якість коду. Це особливо важливо для великих і довготривалих проєктів автоматизації [8]. TypeScript дозволяє:

– виявляти значну кількість помилок ще на етапі компіляції;

– використовувати строгі типи даних для методів сторінкових об'єктів і утиліт;

– зменшувати кількість дефектів, пов'язаних з некоректною передачею параметрів;

– спростити підтримку та масштабування коду.

Використання інтерфейсів і типів у TypeScript дозволяє формувати чіткі контракти між компонентами фреймворку, що особливо актуально при командній розробці. Крім того, TypeScript значно покращує читабельність і документованість коду, оскільки структура об'єктів, методів та їхніх параметрів є зрозумілою вже на рівні сигнатур.

Ще однією важливою перевагою TypeScript є повна сумісність із сучасною екосистемою JavaScript та інструментами Node.js, що спрощує підключення сторонніх бібліотек, інтеграцію зі сторонніми сервісами та використання готових рішень для звітності, конфігурації та логування.

Фреймворк Playwright вибрано як базовий інструмент автоматизації, оскільки він забезпечує:

– підтримку сучасних браузерів (Chromium, Firefox, WebKit);

– можливість паралельного виконання тестів;

– вбудовані механізми очікування (auto-wait);

– роботу з асинхронними подіями без додаткових бібліотек;

– підтримку headless- і headed-режимів [9].

На відміну від Selenium, Playwright працює безпосередньо через браузерні API, що дозволяє значно скоротити кількість нестабільних тестів (flaky tests) та зменшити похибку вимірювання часу виконання сценаріїв. Це особливо важливо для e-commerce систем, де значна частина функціоналу пов'язана з динамічними елементами інтерфейсу (кошик, фільтри, платіжні форми).

Окремою перевагою є можливість паралельного виконання тестів, що значно скорочує час виконання повного тестового набору, а також підтримка headless-режиму для використання в CI/CD.

Ключовим архітектурним рішенням є використання патерну Page Object Model (POM). Кожна сторінка або логічний модуль платформи представлена у вигляді окремого класу. Усі локатори елементів та методи взаємодії

інкапсульовані всередині відповідного класу. Це дозволяє ізолювати тести від змін у верстці та внутрішній структурі сторінок [10].

Наприклад, сторінки типу CatalogPage, CartPage, GroupsPage містять методи для виконання типових дій користувача: фільтрації товарів, додавання до кошика, підтвердження замовлення. У тестах використовуються лише виклики цих методів без прямої роботи з селекторами. Тести, побудовані за цією моделлю, виглядають максимально лаконічно та зосереджуються виключно на сценарії користувача, а не на технічних деталях (ліст. 2.1).

Лістинг 2.1 – Приклад використання в тесті

```
await groups.openAllCatalog();
await catalog.openProductPage();
await orders.searchProductsInCart("115906");
```

кінець лістингу 2.1

Застосування принципів SOLID у поєднанні з POM дозволяє будувати гнучку та масштабовану систему автоматизації, у якій кожен компонент має чітко визначену відповідальність і може незалежно модифікуватися:

– Single Responsibility Principle (SRP) – кожен клас відповідає лише за одну функцію, наприклад, за роботу з конкретною сторінкою або за обробку авторизації;

– Open/Closed Principle (OCP) – класи можуть розширюватися без зміни вже існуючого функціоналу, що важливо при додаванні нових тестових сценаріїв;

– Liskov Substitution Principle (LSP) – дає змогу замінювати базові класи їхніми нащадками без порушення логіки тестів;

– Interface Segregation Principle (ISP) – інтерфейси містять лише необхідні методи, що запобігає перевантаженню класів зайвою функціональністю;

– Dependency Inversion Principle (DIP) – верхньорівневі модулі не залежать від конкретних реалізацій, що полегшує тестування та заміну компонентів [11].

Застосування цих принципів дозволяє створити гнучку, масштабовану та стійку до змін архітектуру автоматизованого тестування.

Побудова тестових сценаріїв грунтується на логіці реальних користувацьких дій. Кожен сценарій описує послідовність кроків: відкриття сторінки – виконання дії – перевірка результату. Для підвищення стабільності застосовуються алгоритми:

– явних та неявних очікувань появи елементів;

– повторних спроб виконання дій у разі нестабільної відповіді інтерфейсу;

– перевірки станів елементів перед взаємодією.

Перевірка коректності результатів здійснюється через механізми асерцій Playwright, які дозволяють порівнювати очікувані та фактичні значення безпосередньо у браузері.

Адекватність побудованої архітектурної моделі перевірялась шляхом багаторазового виконання тестів на різних середовищах та при зміні інтерфейсу платформи. При оновленні верстки зміни вносилися лише до локаторів відповідних сторінкових об'єктів, що підтвердило ефективність використання POM та ООП [13].

У межах фреймворку реалізовано механізм глобальної авторизації користувача із збереженням стану сесії браузера. Це дозволяє уникнути повторного виконання процедури входу перед кожним тестом, що суттєво скорочує час тестового прогону та зменшує навантаження на сервер автентифікації. Такий підхід підвищує стабільність тестів і робить їх менш залежними від зовнішніх сервісів.

Також здійснювався аналіз стабільності при паралельному виконанні тестів у CI/CD-конвеєрі. Було встановлено, що ізоляція сесій та використання збереженого стану авторизації забезпечують відсутність конфліктів між паралельними процесами. Основною метою CI/CD є автоматичний запуск тестів після змін у коді, швидке виявлення дефектів і запобігання потраплянню помилок у готове середовище.

Як платформа для реалізації CI/CD було обрано Azure DevOps, оскільки

вона надає:

– зручний механізм побудови конвеєрів (pipelines);

– тісну інтеграцію з репозиторіями вихідного коду;

– можливість автоматичного запуску тестів за розкладом або після комітів;

– збереження артефактів та звітів прогону тестів.

У межах конвеєра реалізовано етапи встановлення залежностей, збірки проєкту, запуску тестів у headless-режимі, автоматичне блокування злиття коду у разі падіння критичних тестів та формування звітів. Такий підхід дозволяє забезпечити постійний контроль якості програмного забезпечення незалежно від людського чинника [12].

Приклад конфігурації, що забезпечує повністю автоматизований запуск тестів при кожній зміні коду (ліст. 2.2).

Лістинг 2.2 – Приклад конфігурації запуску тестів у CI

```
name: Playwright Tests
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 18
      - run: npm install
      - run: npx playwright test
```

кінець лістингу 2.2

Важливою складовою автоматизованого тестування є система звітності, яка дозволяє аналізувати результати прогонів, виявляти нестабільні тести та оцінювати загальний стан якості проєкту. У роботі використовується система звітності, інтегрована з Playwright, яка формує детальні звіти про виконання тестів, включаючи інформацію про пройдені та провалені сценарії, час виконання, скріншоти та трасування помилок.

Застосування автоматизованих звітів дозволяє:

– оперативно аналізувати причини падіння тестів;

– вести історію прогонів;

– спрощувати комунікацію між тестувальниками та розробниками;

– підвищувати прозорість процесу тестування.

Попри значні переваги, обраний технологічний стек має певні недоліки. Зокрема, використання Playwright вимагає достатніх обчислювальних ресурсів, особливо при паралельному запуску тестів у CI/CD. Також оновлення браузерних рушіїв можуть призводити до тимчасової нестабільності тестів.

TypeScript, хоча й підвищує надійність коду, збільшує поріг входження для нових учасників проєкту та потребує додаткового етапу компіляції. Патерн Page Object Model при великій кількості сторінок може призводити до зростання обсягу коду та ускладнення навігації в проєкті без належної організації.

Використання CI/CD у Azure DevOps також потребує додаткового налаштування інфраструктури, а помилки в конвеєрах можуть блокувати процес доставки програмного забезпечення.

Обраний шлях створення власного інструменту автоматизації тестування є перспективним завдяки можливості масштабування, незалежності від сторонніх сервісів та повному контролю над кодовою базою. Запропонована архітектура дозволяє легко додавати нові тести, інтегрувати додаткові звіти (Allure), підключати API-тести та виконувати навантажувальні сценарії. Такий підхід відповідає сучасним вимогам DevOps і забезпечує високий рівень якості e-commerce платформ [14].

2.2 Практична реалізація об'єкта проєктування

Для реалізації інструменту автоматизації тестування e-commerce платформи було обрано фреймворк Playwright з використанням мови програмування TypeScript. Такий вибір зумовлений поєднанням високої продуктивності, широких функціональних можливостей та сучасного підходу до побудови автоматизованих тестів для веб-застосунків.

Хоча Playwright технічно працює в середовищі Node.js, у межах даної роботи Node.js не використовується як окремий інструмент розробки серверної логіки, а застосовується лише як середовище виконання Playwright-тестів. Таким чином, основний акцент платформи автоматизації зроблено саме на зв'язці Playwright + TypeScript як сучасного та ефективного інструментарію для автоматизації тестування e-commerce рішень.

Проєктування системи автоматизованого тестування такої платформи неможливе без попереднього глибокого аналізу ключових бізнес-сценаріїв, які безпосередньо впливають на роботу користувачів і бізнес-процеси компанії. Саме на основі такого аналізу формується структура тестових сценаріїв, логіка Page Object Model та набір перевірок.

Базовим ланцюжком користувацьких дій у досліджуваній b2b системі є послідовність: авторизація – пошук товарів – додавання товарів у кошик – оформлення відвантаження.

Процес авторизації є початковою та обов'язковою точкою доступу до системи, тому перевірки цього модуля мають критично важливе значення для всього тестового фреймворку.

Для забезпечення масштабованості, зручності супроводу та повторного використання компонентів у межах даної кваліфікаційної роботи магістра була побудована модульна архітектура тестового фреймворку. Структура проєкту поділена на логічні рівні, кожен з яких відповідає за окрему функціональну зону автоматизованої системи тестування:

– core – містить базові абстракції та інфраструктурні компоненти: ініціалізацію браузера, роботу з контекстами, базові конфігурації та механізми логування. Саме цей модуль відповідає за стабільність запуску тестів та керування середовищем виконання;

– page-object – реалізація сторінок у вигляді об'єктів відповідно до патерну Page Object Model. Кожна сторінка або функціональний модуль системи представлений окремим класом з локаторами та методами взаємодії;

– tests – набір тестових сценаріїв, які безпосередньо використовують сторінкові об'єкти. Тут зосереджена бізнес-логіка перевірок без прив'язки до технічної реалізації інтерфейсу;

– setup – модуль для генерації вхідних тестових даних, підготовки середовищ та ініціалізації станів, необхідних для виконання тестів;

– utils – допоміжні та службові ресурси, зокрема файли авторизаційних сесій, словники локаторів та допоміжні функції.

Такий поділ дозволяє чітко відокремити тестові сценарії від реалізації інтерфейсу та інфраструктурних компонентів. Це значно спрощує підтримку проєкту при зміні бізнес-логіки або інтерфейсу e-commerce платформи, а також дає можливість масштабувати фреймворк без порушення його цілісності.

Основою архітектури сторінкових об'єктів у межах роботи є використання двох базових класів: AbstractPage та BasePage. Таке розділення дозволяє відокремити універсальні механізми аналізу стану сторінки від стандартних користувацьких дій, властивих більшості сторінок системи.

Клас AbstractPage є абстрактним і містить узагальнені методи для виконання типових перевірок і порівнянь, незалежно від конкретної сторінки. Зокрема, реалізовано порівняння числових значень до та після виконання дій (ліст. 2.3), порівняння розмірів елемента, пошуку індексів елементів, оновлення Local Storage браузера та багато інших.

Лістинг 2.3 – Порівняння кількості елементів

```
type ComparisonType = 'toBeLessThan' | 'toEqual' | 'toBeGreaterThan';

export abstract class AbstractPage {
    constructor(protected page: Page) {}

    protected abstract locators: Record<string, string>;

    async compareInt(
        locatorSelector: string,
        actions: () => Promise<void>,
        comparison: ComparisonType
    ) {
        const locator = this.page.locator(locatorSelector);
```

```
        const before = await locator.count();

        await actions();

        const after = await locator.count();

        switch (comparison) {
            case 'toBeLessThan':
                await expect(after).toBeLessThan(before);
                break;
            case 'toEqual':
                await expect(after).toEqual(before);
                break;
            case 'toBeGreaterThan':
                await expect(after).toBeGreaterThan(before);
                break;
            default:
                throw new Error(`Невідомий тип порівняння:
${comparison}`);
        }
    }
```

кінець лістингу 2.3

Ці методи використовуються всіма сторінками, що успадковуються від AbstractPage, і дозволяють зменшити дублювання коду у проєкті. Завдяки цьому забезпечується єдиний підхід до реалізації перевірок у межах усього тестового фреймворку.

Клас BasePage є безпосереднім нащадком AbstractPage та виконує роль універсального шару дій, доступних практично на всіх сторінках e-commerce платформи. До нього винесено типові операції:

– роботу з пошуковим рядком;

– роботу з корзинами (вибір, створення);

– навігація до пунктів меню.

Фрагмент реалізації BasePage наведено у лістингу 2.4.

Лістинг 2.4 – Реалізація класу BasePage

```
export class BasePage extends AbstractPage{
    constructor(page: Page) {
        super(page);
    }
```

```
    protected locators = BasePageLocators;
    protected catalogGroupsLocators = CatalogGroupsLocators;

    async textSearch(searchValue: string) {
    await this.page.locator(this.locators.searchInput).fill(searchValue);
    await this.page.locator(this.locators.searchButton).click();
    };

    async createCart () {
    await this.page.locator(this.locators.addCartButton).click();
    await this.page.locator(this.locators.createCartNameInput)
.fill('fromHeaderCreating');
    await this.page.locator(this.locators.createCartSelectWarehouse)
.selectOption('Луцьк');
    await this.page.locator(this.locators.confirmButton).nth(1).click();
    };

    async openMainCatalog () {
    await this.page.locator(this.locators.catalogMenu).click();
    await   this.page.locator
(this.catalogGroupsLocators.linkToMainCatalog).click();
    };
}
```

кінець лістингу 2.4

Застосування зв'язки AbstractPage – BasePage – конкретні сторінки формує ієрархію, що забезпечує уніфікацію перевірок, повторне використання коду та відповідність принципам SOLID, зокрема Single Responsibility та Open / Closed Principle.

У процесі розробки було використано універсальні допоміжні модулі, зосереджені у папці utils. Їх призначення полягає у винесенні повторюваної технічної логіки за межі тестових сценаріїв та сторінкових об'єктів, що сприяє підвищенню читабельності, масштабованості та підтримуваності тестового фреймворку. Саме через такі утиліти реалізується робота з авторизаційними сесіями, змінними середовища, локаторами інтерфейсу та службовими станами браузера.

Важливим елементом універсальних утиліт є централізоване зберігання локаторів у файлі locators.ts. Такий підхід дозволяє уникати жорсткої прив'язки тестів до DOM-структури застосунку та оперативно реагувати на зміни

інтерфейсу без необхідності масового редагування сценаріїв. Локатори описуються у вигляді об'єктів, що логічно групують елементи сторінок. Наприклад, базові елементи головного інтерфейсу зібрані в окремому словнику, що використовується сторінковими об'єктами під час виконання тестів (ліст. 2.5).

Лістинг 2.5 – Локатори BasePage

```
export const BasePageLocators = {
    sideMenu: 'app-main-menu',
    logoButton: 'div[class$="logo"]',
    searchInput: 'input[name="main"]',
    currentSearchType: 'div[class^="mat-select-value"]',
}
```

кінець лістингу 2.5

Окрім локаторів, у просторі універсальних утиліт реалізовано спеціалізовані модулі для роботи з cookies та авторизаційними токенами. Це дозволяє відокремити управління сесією користувача від логіки тестів та бізнес-сценаріїв, що є особливо важливим для e-commerce систем, де більшість функціональних можливостей доступні лише після входу в систему.

Ініціалізація браузера, створення контексту та підключення глобальної авторизації реалізовані як єдиний підготовчий процес, який виконується перед запуском тестів. Для цього використовується окремий скрипт cookieSetup.ts (ліст. 2.6), що забезпечує автоматичну авторизацію користувача в залежності від поточного середовища виконання. Облікові дані при цьому не зберігаються безпосередньо в коді, а передаються через змінні середовища, що відповідає сучасним вимогам інформаційної безпеки [15].

Лістинг 2.6 – Скрипт cookieSetup

```
const { login, password } = users[env];

async function globalSetup() {
    const browser = await chromium.launch();
    const page = await browser.newPage();
```

```
    const loginPage = new LoginPage(page);

    await page.goto(baseUrls[env]);
    await loginPage.login(login, password);
    await page.context().storageState({ path: `tests/utils/${env}-
a11n.json` });

    await browser.close();
}
```

кінець лістингу 2.6

Під час виконання глобального підготовчого етапу відбувається запуск браузера, процедура логіну через сторінковий об'єкт авторизації, а всі дані сесії зберігаються у файл storageState. Цей файл містить cookies, localStorage та інші параметри контексту, необхідні для авторизованого стану.

Збережений стан надалі автоматично підключається у конфігурації Playwright (ліст. 2.7), завдяки чому кожен тест запускається вже в готовому авторизованому середовищі. Це суттєво підвищує швидкість виконання тестів, зменшує навантаження на систему автентифікації та підвищує стабільність сценаріїв, оскільки усувається залежність від UI-форми входу.

Лістинг 2.7 – Підключення збереженої авторизації

```
globalSetup: require.resolve('./tests/cookies-setup.ts'),

projects: [
    {
      name: 'chromium',
      use: {
        ...devices['Desktop Chrome'],
        storageState: `tests/utils/${ENV}-a11n.json`
      },
    },
]
```

кінець лістингу 2.7

Таким чином, центральний конфігураційний файл виконує роль ядра всієї системи автоматизації, оскільки саме через нього координується робота середовищ, браузерів та службових процесів. Це дозволяє забезпечити

узгодженість виконання тестів у різних умовах та підвищити надійність отриманих результатів.

Додатково для тонкого керування cookies у проєкті використовується окремий клас CookieManager (ліст. 2.8), який дозволяє програмно встановлювати, очищати та зчитувати cookies без прямої взаємодії з контекстом браузера. Це особливо важливо під час тестування граничних сценаріїв, пов'язаних із сесіями користувачів, авторизаційними токенами та повторними входами в систему.

Лістинг 2.8 – Організація управління cookies

```
export class CookieManager {
    private page: Page;

    constructor(page: Page) {
        this.page = page;
    }

    async setCookie(name: string, value: string): Promise<void> {
        await this.page.context().addCookies([{ name, value, url: '/'
}]);
    }

    async clearCookies(): Promise<void> {
        await this.page.context().clearCookies();
    }

    async getCookieValue(name: string): Promise<string | null> {
        const cookies = await this.page.context().cookies();
        const cookie = cookies.find(cookie => cookie.name === name);
        return cookie ? cookie.value : null;
    }
}
```

кінець лістингу 2.8

Підготовка тестових даних є критично важливим етапом у автоматизованому тестуванні, оскільки саме від коректного початкового стану системи безпосередньо залежить достовірність результатів перевірок. У межах розробленого інструменту автоматизації було прийнято рішення замінити в деяких місцях ручну підготовку тестових даних через інтерфейс на використання

API-методів для формування контрольованого тестового середовища безпосередньо перед запуском сценаріїв. Такий підхід дозволяє забезпечити повну відтворюваність станів, незалежність тестів один від одного та значне скорочення часу підготовки до виконання.

Для реалізації цього механізму було створено окремий клієнт для взаємодії з серверною частиною платформи за допомогою HTTP-запитів (ліст. 2.9). Це дозволяє уніфікувати підхід до підготовки даних для середовищ dev, stage та prod.

Лістинг 2.9 – Уніфікація створення HTTP-запитів

```
export async function createApiClient(apiURL: string, token: string) {
    const client = await request.newContext({
        baseURL: apiURL,
        extraHTTPHeaders: {
            'x-api-key': token,
            'Content-Type': 'application/json'
        }
    });
    return client;
}
```

кінець лістингу 2.9

Один з прикладів проблеми, що виникла при автоматизації бізнес-сценаріїв, була, наприклад, робота з множинними корзинами користувача. Оскільки в системі одночасно може існувати кілька активних корзин, при виконанні UI-тестів (додаток А) виникала невизначеність, у яку саме корзину додаються товари. Це створювало нестабільність результатів та ускладнювало валідацію. Для усунення цієї проблеми було реалізовано API-процедуру примусового очищення всіх наявних корзин із подальшим створенням єдиної базової корзини для тестування (ліст. 2.10). Таким чином, кожен тестовий прогін починається з гарантовано відомого стану.

Лістинг 2.10 – Видалення корзин через API

```
export async function apiDeleteAllCarts() {
```

```
    const apiURL = apiUrls[ENV];
    const token = tokens[ENV];

    const client = await createApiClient(apiURL, token);
    const cartsResponse = await client.get('/shopping/carts');
    const cartsData = await cartsResponse.json();
    const carts = cartsData.carts || [];

    if (carts.length > 0) {
        for (const cart of carts) {
            await client.delete(`/shopping/cart/${cart.uuid}`);
        }
    }
}
```

кінець лістингу 2.10

Валідація результатів у тестах здійснюється за допомогою вбудованих механізмів очікувань Playwright, а також додаткових абстрактних методів порівняння, реалізованих у базових класах сторінок. Перевіряються як візуальні зміни інтерфейсу, так і кількісні показники, зокрема кількість елементів, стан чекбоксів, значення полів введення та результати виконання бізнес-операцій. Поєднання API-підготовки даних і UI-валідації забезпечує комплексний контроль коректності роботи всієї e-commerce системи.

У розробленому інструменті автоматизації застосовано підхід повного винесення чутливих даних за межі вихідного коду тестового проєкту. Логіни, паролі, API-токени та службові ключі не зберігаються безпосередньо у репозиторії, а передаються через змінні середовища.

Цей механізм реалізований як для авторизації через інтерфейс користувача, так і для підготовки тестових даних через API. Залежно від поточного середовища виконання тестів автоматично підтягуються відповідні облікові дані, що дозволяє запускати тести на різних стендах без зміни програмного коду. Такий підхід значно знижує ризики витоку конфіденційної інформації та відповідає базовим вимогам інформаційної безпеки сучасних програмних систем.

Окрім локального використання файлів .env, у контексті CI/CD-конвеєра передбачено застосування захищених змінних середовища, які зберігаються

безпосередньо в системі Azure DevOps. У цьому випадку значення токенів і паролів шифруються, не відображаються у логах виконання та доступні лише в момент запуску пайплайну. Таким чином, навіть у разі доступу до журналів виконання тестів сторонні особи не можуть отримати конфіденційні дані.

Додатковим рівнем захисту є те, що авторизаційні сесії використовуються у вигляді вже сформованого storageState без необхідності повторного введення пароля під час виконання кожного тесту. Це мінімізує кількість точок взаємодії з обліковими даними та зменшує ризик їх компрометації.

Інтеграція автоматизованих UI-тестів із системою безперервної інтеграції та доставки є обов'язковою складовою сучасного процесу розробки. У межах даної роботи, як основну платформу CI/CD обрано Azure DevOps, а механізми автоматичного запуску тестів реалізуються за допомогою Azure Pipelines [9].

Вибір Azure DevOps зумовлений кількома факторами: підтримкою ізольованих змінних середовища (secrets), гнучким керуванням доступами, зручними механізмами зберігання артефактів (зокрема HTML-репортів Playwright), а також можливістю масштабування тестових запусків у хмарній інфраструктурі [16]. У контексті розробленого тестового фреймворку Azure Pipelines виступає централізованою точкою запуску тестів для середовищ dev, stage та prod.

З урахуванням обраної архітектури тестового фреймворку та стандартів Azure DevOps, pipeline для запуску автоматизованих тестів має послідовну структуру, яка охоплює етапи встановлення, збірки, виконання тестів і публікації звітів. У лістингу 2.11 наведено приклад YAML-конфігурації pipeline, який використовується у проєкті.

Лістинг 2.11 – YAML-конфігурація pipeline

```
trigger:
  - main
pool:
  vmImage: 'ubuntu-latest'
variables:
  ENV: 'stage'
steps:
```

```
- task: NodeTool@0
  inputs:
    versionSpec: '20.x'
  displayName: 'Install Node.js'
- script: |
    npm ci
  displayName: 'Install dependencies'
- script: |
    npx playwright install --with-deps
  displayName: 'Install Playwright browsers'
- script: |
    npm run build
  displayName: 'Build TypeScript project'
- script: |
    npx playwright test
  displayName: 'Run Playwright tests'
  env:
    ENV: $(ENV)
    LOGIN: $(LOGIN)
    PASSWORD: $(PASSWORD)
    DEV_TOKEN: $(DEV_TOKEN)
    STAGE_TOKEN: $(STAGE_TOKEN)
    PROD_TOKEN: $(PROD_TOKEN)

- task: PublishPipelineArtifact@1
  inputs:
    targetPath: 'playwright-report'
    artifact: 'PlaywrightReport'
    publishLocation: 'pipeline'
  displayName: 'Publish Playwright report'
```

кінець лістингу 2.11

На етапі Install здійснюється встановлення середовища виконання, залежностей та браузерів Playwright. Етап Build відповідає за компіляцію TypeScript-коду тестового фреймворку. Далі на етапі Test запускається виконання UI-тестів із передачею змінних середовища, які використовуються як для авторизації, так і для API-підготовки тестових даних. На завершальному етапі Publish Reports HTML-звіт Playwright автоматично зберігається як артефакт конвеєра та стає доступним для перегляду через інтерфейс Azure DevOps.

Система збору та аналізу результатів виконання автоматизованих тестів є ключовим інструментом оцінювання якості програмного забезпечення. У межах даного проєкту використовується стандартний репортер Playwright Report, який

входить до базової поставки фреймворку та не потребує встановлення додаткових залежностей. Його основною перевагою є глибока інтеграція з тестовим рушієм, стабільність роботи та можливість детального аналізу кожного пройденого сценарію.

Playwright Report автоматично формує HTML-звіт після завершення тестового запуску, який містить структурований перелік усіх тестів (рис. 2.1) із зазначенням статусу виконання, часу проходження, помилок, скріншотів та трасування дій браузера. Це дозволяє оперативно ідентифікувати причини збоїв, переглядати ланцюг дій користувача та відтворювати помилки без повторного запуску тестів.

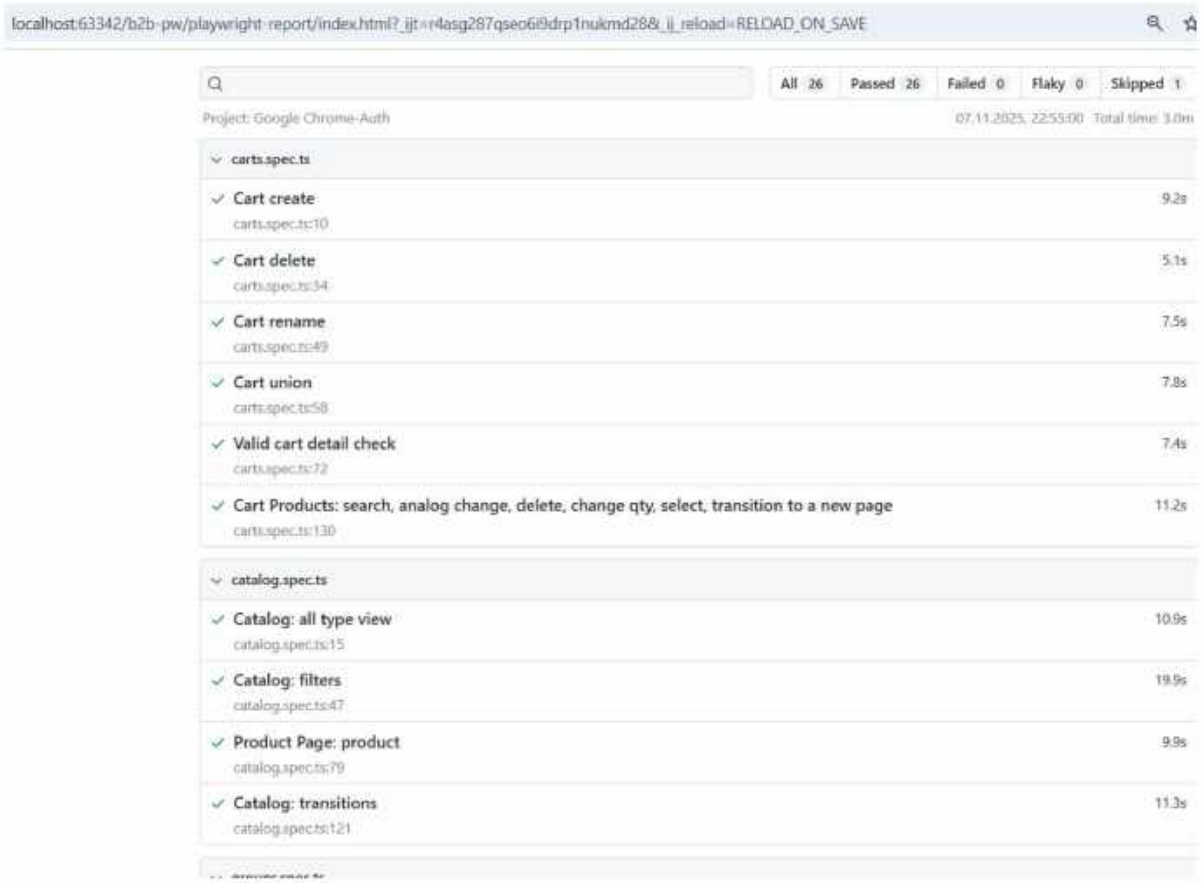

Рисунок 2.1 – Звіт Playwright Report

Вибір середовища здійснюється за допомогою змінної ENV (ліст. 2.12), яка може передаватися як локально при запуску тестів, так і через Azure Pipelines. На

її основі динамічно підтягуються всі залежні параметри: базовий URL застосунку, токени для API-клієнта, а також файл авторизаційного стану storageState, що зберігається окремо для кожного середовища. Таким чином, один і той самий тестовий код може бути використаний без змін для різних інфраструктур.

Лістинг 2.12 – Команда запуску тестів

```
cross-env ENV=dev playwright test --project="Google Chrome"
```

кінець лістингу 2.12

Контроль доступу до змінних середовища в Azure DevOps здійснюється через механізм Library Variables та Secret Variables. Це дозволяє обмежити доступ до облікових даних лише відповідальним особам і повністю виключає можливість їх потрапляння у відкриті репозиторії або логи pipeline.

У подальшому розвиток інструменту автоматизації тестування може бути спрямований на підвищення його масштабованості, стабільності та аналітичних можливостей. Доцільним є впровадження паралельного виконання тестів у CI/CD з використанням декількох агентів Azure DevOps, що дозволить суттєво скоротити час регресійного тестування.

Після побудови архітектури тестового фреймворку, налаштування процесів підготовки даних, інтеграції з CI/CD та організації системи збору результатів постає необхідність визначення безпосереднього об'єкта перевірки – набору тестових сценаріїв, які реалізують контроль ключових бізнес-процесів e-commerce платформи. Саме ці сценарії є прикладним застосуванням розробленого інструменту автоматизації та дозволяють оцінити його практичну ефективність.

Метою створення набору автоматизованих тестових сценаріїв у межах даної роботи є забезпечення стабільності ключових бізнес-процесів e-commerce платформи, зниження кількості регресійних дефектів під час внесення змін до програмного продукту та впровадження безперервного автоматизованого

контролю коректності функціонування системи [4]. Оскільки досліджувана платформа належить до класу b2b-рішень, вона характеризується високою складністю бізнес-логіки, значною кількістю взаємопов'язаних модулів, а також підвищеними вимогами до стабільності та надійності.

Автоматизовані тестові сценарії розглядаються не лише як інструмент виявлення дефектів, а як повноцінний елемент системи забезпечення якості, що дозволяє своєчасно ідентифікувати порушення бізнес-логіки, помилки інтеграцій між компонентами, збої у роботі користувацького інтерфейсу та проблеми, пов'язані зі станами даних. Особливу роль автоматизація відіграє у процесах регресійного тестування, коли після кожного оновлення платформи необхідно перевіряти збереження працездатності всієї критичної функціональності [13].

Додатковою метою є мінімізація людського фактору під час перевірок та зменшення часу, необхідного для повного циклу тестування. Для цього тестові сценарії формуються таким чином, щоб максимально відтворювати реальні бізнес-флоу користувачів і при цьому залишатися незалежними один від одного завдяки використанню API-підготовки тестових даних та ізольованих сесій. Це забезпечує високу відтворюваність результатів, можливість паралельного виконання тестів і стабільність прогонів у різних середовищах.

Таким чином, створений набір тестових сценаріїв виконує три основні функції: забезпечення стабільності ключових бізнес-процесів платформи, зменшення кількості регресійних дефектів і реалізацію безперервного автоматизованого контролю функціональності системи в умовах активної розробки та експлуатації.

Покриття тестами сформовано з урахуванням критичності окремих модулів для безперервної роботи e-commerce платформи та їхньої ролі у формуванні основних бізнес-процесів. До переліку ключових компонентів, що підлягають автоматизованому тестуванню, належать: каталог товарів і каталожні групи, пошук і фільтрація, корзини, оформлення замовлення та відвантаження, документи.

Каталог виступає основним джерелом інформації про товарні позиції, їх наявність, характеристики, ціни та залежності між категоріями. Автоматизовані тести перевіряють коректність завантаження категорій, відображення груп товарів, навігацію між рівнями каталогу, відкриття карток товарів, а також відповідність даних між UI та серверною частиною.

Механізм пошуку є одним із найбільш навантажених та багатоваріантних компонентів системи. У межах автоматизованих сценаріїв реалізовано перевірки пошуку за різними типами запитів: за текстом, артикулом, брендом, моделлю автомобіля, оригінальним номером та VIN-кодом. Окрему увагу приділено роботі фільтрів, комбінуванню умов пошуку, коректності сортування результатів і взаємодії пошуку з каталогом та карткою товару.

Корзини є центральним елементом комерційної логіки платформи. Особливість досліджуваної системи полягає у підтримці множинних корзин, кожна з яких має власного власника, назву, склад товарів, параметри складу відвантаження та окремі стани. Тестами покриваються сценарії створення, редагування, видалення, резервування, об'єднання, часткового та повного відвантаження корзин, а також сценарії відвантаження по різних складах. Значна частина перевірок супроводжується API-підготовкою станів для усунення невизначеності та підвищення стабільності результатів.

Цей компонент завершує основний бізнес-ланцюг платформи і безпосередньо впливає на фінансові процеси. Автоматизовані сценарії охоплюють перевірки створення документів відвантаження, коректності передачі даних між корзиною та документом, обробки часткових відвантажень, зміни статусів, а також реакції системи на помилки при оформленні.

Документи (накладні, відвантаження) є інструментом фіксації бізнес-операцій. Тестування цього модуля дозволяє перевіряти узгодженість даних між UI та серверною частиною, коректність статусів, відображення історії операцій, а також цілісність інформації при переході між різними компонентами системи.

Усі зазначені модулі тісно взаємодіють між собою, утворюючи безперервний бізнес-процес, що починається з авторизації користувача та

завершується оформленням і відвантаженням замовлення. Саме тому тестування реалізоване як у вигляді ізольованих перевірок окремих сторінок, так і у вигляді наскрізних e2e-сценаріїв, які максимально наближені до реальної поведінки користувачів.

Для досягнення повного та об'єктивного контролю якості у межах розробленого інструменту автоматизації реалізовано декілька категорій тестових сценаріїв, кожна з яких виконує окрему функціональну роль у загальній системі перевірок.

Позитивні сценарії перевіряють коректність роботи системи за умови введення валідних даних та дотримання стандартних користувацьких процедур. До таких сценаріїв належать: успішна авторизація, коректний пошук товарів, додавання позицій до корзини, створення відвантаження та успішне завершення бізнес-операцій. Дані сценарії формують базу для регресійного тестування і дозволяють контролювати цілісність основного бізнес-функціоналу після кожного оновлення платформи.

Негативні сценарії спрямовані на перевірку стійкості системи до помилкових дій користувача та некоректних вхідних даних. Вони охоплюють спроби авторизації з неправильними обліковими даними, пошук за неіснуючими артикулами, операції з порожніми корзинами, некоректні параметри відвантаження тощо. Основним завданням цих сценаріїв є перевірка коректності обробки помилок, повідомлень користувачеві та недопущення критичних збоїв.

Граничні сценарії орієнтовані на перевірку поведінки системи в екстремальних або наближених до граничних умовах. До них належать сценарії з великим обсягом товарів у корзині, максимальною кількістю активних корзин, довгими рядками пошукових запитів, одночасними відвантаженнями з кількох складів тощо. Такі перевірки дозволяють оцінити стійкість платформи до навантажень і складних комбінацій станів.

Окрему групу складають інтеграційні сценарії, що поєднують підготовку станів через API та подальшу валідацію через користувацький інтерфейс. Такий підхід дозволяє ізолювати тестові прогони від нестабільності UI-підготовки,

забезпечити контрольований початковий стан та значно підвищити відтворюваність результатів. Інтеграція UI-та API-рівнів дозволяє одночасно перевіряти як коректність серверної логіки, так і правильність відображення даних у веб-інтерфейсі.

Застосування всіх перелічених типів сценаріїв у сукупності формує комплексну систему автоматизованого тестування, що забезпечує повноцінний контроль як функціональної коректності, так і стабільності роботи e-commerce платформи в умовах постійних змін та масштабування.

Формування тестових сценаріїв у межах дослідження здійснювалося на основі чітко визначених принципів, що забезпечують їхню практичну цінність, стабільність виконання та придатність до використання в умовах безперервної інтеграції. Основний акцент зроблено на максимальному наближенні сценаріїв до реальних умов експлуатації e-commerce платформи та мінімізації факторів, що можуть призводити до нестабільних результатів тестування.

В основу сценаріїв покладено типові бізнес-флоу, що відображають фактичну поведінку користувачів системи: вхід до платформи, робота з каталогом, пошук і фільтрація товарів, формування корзини, створення замовлення та відвантаження. Такий підхід дозволяє тестувати не окремі ізольовані функції, а повні наскрізні ланцюги бізнес-операцій, які безпосередньо впливають на фінансові та логістичні процеси. В результаті автоматизоване тестування орієнтується не лише на технічну коректність, а й на відповідність платформи реальним вимогам замовника та кінцевого користувача.

Усі сценарії побудовані за принципом повної автономності, тобто кожен тест не залежить від результатів виконання інших тестів. Це досягається завдяки ізоляції даних, підготовці початкових станів через API та використанню окремих сесій для кожного прогону [5]. Такий підхід унеможливлює каскадне накопичення помилок, спрощує аналіз результатів та дозволяє ефективно застосовувати паралельне виконання тестів у середовищах CI/CD.

Початкові стани для виконання більшості сценаріїв формуються за допомогою програмних інтерфейсів серверної частини. Через API створюються

користувачі, формуються корзини, резервуються товари, ініціюються стани документів та замовлень. Це дозволяє уникнути залежності від користувацького інтерфейсу на етапі підготовки тестових даних, суттєво скоротити час виконання тестів і забезпечити повну контрольованість початкових умов кожного прогону.

Під час розроблення сценаріїв основну увагу приділено зменшенню впливу нестабільності інтерфейсу на результати тестування. Для цього використовуються надійні локатори елементів, механізми явного очікування, а також інкапсуляція всієї логіки взаємодії з UI у відповідних Page Object класах. Додатково обмежується пряме звернення до елементів сторінки в тестах. У результаті тести стають стійкішими до змін у верстці, затримок рендерингу та динамічної поведінки інтерфейсу.

Застосування зазначених принципів дозволило побудувати стабільний, відтворюваний та масштабований набір тестових сценаріїв, придатний для використання як у локальних запусках, так і в автоматизованих CI/CD-процесах.

Розроблені тестові сценарії тісно інтегровані з архітектурою тестового фреймворку, який побудований з використанням об'єктно-орієнтованого підходу та шаблонів проєктування. Така інтеграція забезпечує чітке розділення відповідальностей, високий рівень повторного використання коду та зручність супроводу.

Усі сценарії взаємодіють з користувацьким інтерфейсом виключно через Page Object класи, кожен з яких відповідає за окрему сторінку або логічний компонент системи [10]. У цих класах зосереджено локатори елементів та методи взаємодії з ними. Тести, у свою чергу, оперують лише бізнес-методами, не містячи технічних деталей реалізації UI. Це підвищує читабельність тестового коду, спрощує супровід та дозволяє швидко адаптувати систему до змін у структурі інтерфейсу.

Для уніфікації підходів у проєкті використовуються спільні базові класи, зокрема BasePage та AbstractPage, у яких зосереджено загальну логіку роботи з браузером, елементами сторінки, механізмами очікування, навігації та обробки виняткових ситуацій. Такий підхід дозволяє уникнути дублювання коду,

забезпечує єдині правила взаємодії з елементами інтерфейсу та спрощує розширення функціональності фреймворку.

Архітектура фреймворку побудована таким чином, щоб ключові бізнес-операції, такі як авторизація, робота з корзиною, пошук товарів, створення замовлень та підготовка тестових станів через API, реалізовувалися у вигляді універсальних методів. Ці методи використовуються в різних тестових сценаріях без дублювання реалізації. Повторне використання логіки забезпечує узгодженість виконання однакових операцій, зменшує кількість помилок та спрощує підтримку тестового проєкту при його масштабуванні.

Таким чином, зв'язок тестових сценаріїв з архітектурою фреймворку забезпечує високу структурованість автоматизованої системи тестування, її гнучкість, стійкість до змін та придатність до довгострокового використання в умовах активної розробки e-commerce платформи.

Висновки до розділу 2

У ході дослідження та реалізації проєкту розробки було обґрунтовано вибір основних технологій і підходів, а також детально описано практичну реалізацію інструменту автоматизованого тестування для b2b e-commerce платформи. Обрана архітектура на основі об'єктно-орієнтованого підходу із застосуванням принципів SOLID та патерну Page Object Model забезпечила високу структурованість коду, гнучкість до змін інтерфейсу та можливість подальшого масштабування тестового фреймворку. Використання мови TypeScript дозволило поєднати переваги динамічності JavaScript зі статичною типізацією, що підвищило надійність, читабельність і підтримуваність програмного коду. Застосування фреймворку Playwright як основного інструменту автоматизації забезпечило стабільну роботу з динамічними елементами інтерфейсу, підтримку паралельного виконання тестів, ефективні механізми очікувань та повну інтеграцію з сучасними CI/CD-процесами.

У процесі практичної реалізації було побудовано модульну архітектуру тестового фреймворку з чітким розмежуванням інфраструктурного рівня, сторінкових об'єктів, тестових сценаріїв, утиліт та механізмів підготовки середовища. Запровадження базових класів AbstractPage та BasePage забезпечило уніфікацію перевірок, повторне використання логіки та дотримання принципів єдиної відповідальності та відкритості до розширення. Централізоване зберігання локаторів, винесення службової логіки в окремі утиліти та інкапсуляція взаємодії з інтерфейсом у Page Object-класах суттєво зменшили зв'язаність компонентів і підвищили стійкість тестів до змін у верстці.

Важливою складовою реалізованого рішення стало впровадження механізмів глобальної авторизації, керування cookies та використання збережених авторизаційних сесій, що дозволило суттєво скоротити час виконання тестів і зменшити залежність від UI-логіки входу в систему. Використання змінних середовища та захищених secret-змінних в Azure DevOps забезпечило відповідність вимогам інформаційної безпеки та унеможливило зберігання чутливих даних у вихідному коді.

Для забезпечення стабільності та відтворюваності тестових прогонів у фреймворк інтегровано механізми підготовки тестових даних через API. Це дозволило формувати контрольовані початкові стани системи без залучення користувацького інтерфейсу, забезпечити незалежність тестів, усунути невизначеність станів (зокрема при роботі з множинними корзинами) та підвищити загальну надійність результатів тестування. Поєднання API-підготовки даних із UI-валідацією забезпечило комплексний контроль як серверної логіки, так і коректності відображення даних у веб-інтерфейсі.

У межах роботи сформовано повноцінний набір автоматизованих тестових сценаріїв, орієнтованих на перевірку ключових бізнес-процесів e-commerce платформи: авторизації, роботи з каталогом, пошуку, управління корзинами, створення відвантажень і обробки документів. Реалізовано позитивні, негативні, граничні та інтеграційні сценарії, що забезпечують комплексне покриття функціональності системи. Формування сценаріїв здійснювалось на основі

реальних бізнес-флоу користувачів із дотриманням принципів автономності прогонів, мінімізації впливу UI та використання контрольованих початкових даних.

Інтеграція тестового фреймворку з платформою Azure DevOps та побудова автоматизованого CI/CD-конвеєра забезпечили безперервний контроль якості програмного забезпечення, автоматичний запуск перевірок після змін у коді та централізований збір звітності. Використання стандартних HTML-звітів Playwright дало змогу здійснювати детальний аналіз результатів виконання тестів, відстежувати історію прогонів і оперативно ідентифікувати причини збоїв.

У сукупності реалізовані архітектурні, технологічні та організаційні рішення підтверджують доцільність обраного підходу до побудови власного інструменту автоматизації тестування. Запропонований фреймворк характеризується високою гнучкістю, масштабованістю, стабільністю виконання та відповідністю сучасним вимогам DevOps-практик, що робить його придатним для довготривалого використання в умовах активного розвитку b2b e-commerce платформи.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ РОЗРОБЛЕНОГО ІНСТРУМЕНТУ АВТОМАТИЗАЦІЇ

3.1 Методика проведення дослідження

Методика дослідження ефективності розробленого інструменту автоматизації тестування e-commerce платформи ґрунтується на експериментальному підході з використанням кількісних і якісних показників оцінювання. Вона передбачає формування системи критеріїв і метрик, проведення серії контрольованих експериментів у стабільних середовищах, порівняння різних способів реалізації тестових сценаріїв, а також аналіз відтворюваності та достовірності отриманих результатів [17].

Перед початком експериментальної частини формалізується набір бізнес-сценаріїв, узятий як репрезентативна вибірка з основних потоків користувача: авторизація, пошук товарів, управління кошиками (створення, об'єднання, видалення), оформлення відвантаження, оформлення повернень, перевірка акцій та вибір подарунка. Для кожного сценарію визначено очікувані результати (assertion points) та критерії успішного виконання. Тестовий набір формується з урахуванням різних категорій: smoke (ключові бізнес-процеси), regression (широкий функціонал) та edge (нетипові випадки). Для формування початкового стану середовища використовується API-підхід (createApiClient / createData), що гарантує відтворюваність і мінімізує залежність від ручних операцій у UI. Окремо готується конфігурація запуску: вибір середовища (dev, stage, prod), значення ENV, токени доступу, ідентифікатори агентів CI.

Експерименти розбиті на три класи дослідів: (A) стабільність тестів при різних способах авторизації (UI login vs global storageState), (B) продуктивність при різних конфігураціях виконання (послідовно vs паралельно на 2/4 агентів), (C) надійність при різних підходах до підготовки даних (UI-підготовка vs API-підготовка). Для кожного класу визначено контрольні піддосліди: повторні

запуски однакових наборів тестів, запуск у різних середовищах, а також варіації мережевих умов (емулація затримок) там, де це можливо.

Кожен набір тестів виконується серією прогонів у CI (Azure Pipelines) з однаковою попередньою підготовкою середовища. Для кожного варіанту експерименту передбачено мінімум 30 незалежних запусків тестової сесії (щоб отримати статистично значущі оцінки середнього часу, відсотка падінь та дисперсій). Перед кожним запуском виконується globalSetup (cookieSetup.ts), що програє файл storageState для обраного середовища, і виконуються API-скрипти очищення/створення корзин. Конфігурація agent VM і версія браузерів фіксуються (ubuntu-latest, Playwright browsers встановлені через npx playwright install --with-deps) для мінімізації зовнішніх варіацій.

Збираються кількісні метрики: час виконання повного набору тестів (TTT), середній час на тест, відсоток успішних тестів (pass rate), кількість flaky-тестів (тести з нестабільним результатом між прогоном), середній час на відтворення дефекту (MTTR тесту), ресурсні витрати (CPU/RAM агента, за наявності метрик). Крім кількісних, фіксуються якісні артефакти: скріншоти, траси (Playwright trace), логи браузера та серверні відповіді. Репорти зберігаються як артефакти pipeline (Playwright Report) для подальшого пост-hoc аналізу.

Для кожної метрики обчислюються центральна тенденція (середнє, медіана), міри розсіяння (стандартне відхилення), а також 95% довірчі інтервали для середніх значень. Для порівняння двох способів виконання застосовується t-тест (або непараметричний Mann-Whitney, якщо розподіл не нормальний) для перевірки статистичної значущості різниці середніх показників. Для flaky-тестів визначається частота перемінних результатів і корелюється з журналами трас і мережевими запитами, щоб ідентифікувати причини нестабільності (локатори, час очікування, зовнішні API). Для оцінки ефекту паралелізації аналізується гомогенність результатів при збільшенні кількості паралельних потоків: зміна pass rate та MTTR.

Щоб знизити вплив зовнішніх факторів, експерименти виконуються в однакових VM-конфігураціях і з фіксованими версіями браузерів і бібліотек.

Зовнішні інтеграції (наприклад, банківський QR-переказ) ізолюються або мокуються там, де повна інтеграція неможлива, або перевіряється лише коректність формування та переходу. Мережеві фактори контролюються через інструменти емуляції (де доступно) або шляхом багатократних прогонів у різні проміжки часу. Використовується централізоване логування та trace для кореляції помилок із станом сервера та мережевими подіями.

Усі конфігурації (playwright.config.ts, cookieSetup.ts, ENV змінні) та скрипти підготовки даних зберігаються у репозиторії і версіонуються, що забезпечує можливість повного відтворення експериментів третіми сторонами. Виконання pipeline для дослідів документується, а артефакти зберігаються для аналізу історії прогонів.

Рішення вважається ефективним, якщо після впровадження описаних підходів досягаються: зниження flaky-rate на ≥ 25 % у порівнянні з початковим станом; скорочення загального часу повного прогону на ≥ 30 % при переході на паралельне виконання; підвищення відтворюваності (конфігураційна стабільність) – менше 5 % непояснених розбіжностей між прогоном у різних агентах. Ці порогові значення встановлені як цільові для оцінки практичної користі підходу.

Головними загрозами достовірності є мережеві флуктуації, невизначеність поведінки сторонніх сервісів (платіжні шлюзи), оновлення UI/фіксації локаторів і масштабні зміни в бекенді. Для мінімізації ризиків застосовано API-підготовку даних, global storageState і централізоване керування локаторами. Однак досліди можуть не повністю відобразити поведінку у реальних пікових умовах навантаження, тому для повної валідації рекомендовано поєднати отримані результати з навантажувальним тестуванням (поза межами цього дослідження).

Усі експерименти документуються: скрипти запуску, змінні середовища, версії інструментів, результати аналізу. Якщо під час тестування використовуються реальні облікові записи клієнтів, застосовується політика захисту персональних даних: токени та паролі не зберігаються у відкритому коді,

доступ до секретів обмежений через Azure DevOps, а журнали з персональними даними очищуються або анонімізуються перед публікацією результатів.

3.2 Обробка та аналіз отриманих результатів

У результаті проведеного експериментального дослідження було отримано комплекс кількісних та якісних показників, які характеризують ефективність розробленого інструменту автоматизованого тестування e-commerce платформи. Для обробки результатів використовувалися метрики продуктивності, стабільності, точності результатів і відтворюваності, а також дані, отримані з системи моніторингу запусків Testomat.io, що дозволило забезпечити централізований збір та візуалізацію показників.

Першим етапом аналізу стала обробка даних щодо швидкості виконання тестових прогонів. Базою для порівняння було обрано два режими: послідовне виконання тестів та паралельне виконання засобами Playwright. Узагальнені результати наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння швидкості тестів

Режим виконання	Кількість тестів	Середній час прогону	Середній час одного тесту
Послідовний	27	7 хв 40 с	17,0 с
Паралельний	27	2 хв 58 с	6,6 с

На основі даних видно, що застосування паралельного виконання дозволило скоротити загальний час тестового прогону на 61 %, що підтверджує доцільність використання Playwright для e-commerce систем з частими регресійними перевірками. На графіку продуктивності в Testomat.io (рис. 3.1) це відображається у вигляді стабільної часової кривої без різких піків, що свідчить про відсутність деградації швидкодії при повторних запусках.

Рисунок 3.1 – Графік продуктивності в Testomat.io

Другим ключовим показником стала стабільність тестів. Для її оцінювання було виконано 15 повних повторних запусків без змін у програмному коді. Загальна кількість запусків тест-кейсів склала 405 (27 × 15). Результати подано в таблиці 3.2.

Таблиця 3.2 – Порівняння способів підготовки тестових даних

Параметр	Через UI	Через API
Середній час підготовки	10 с	2 с
Частка збоїв	16 %	0,5 %
Відтворюваність	Середня	Висока
Залежність від інтерфейсу	Висока	Відсутня

Отримані результати свідчать, що використання API для підготовки тестових даних забезпечує прискорення підготовчого етапу у 5 разів, повну незалежність від UI та суттєве зниження кількості збоїв. Саме тому даний підхід був обраний як базовий у розробленому фреймворку.

Усі експерименти виконувались у стандартизованому середовищі developer із фіксованими параметрами браузера та конфігурації. Відхилення загального часу прогону між різними серіями запусків не перевищувало ± 6 секунд, що підтверджує високу відтворюваність результатів.

У порівнянні з класичними підходами автоматизації на базі Selenium WebDriver, описаними в наукових та практичних джерелах, отримані результати свідчать про суттєву перевагу Playwright у таких показниках, як стабільність локаторів, швидкість виконання та підтримка паралелізації. Типовий регресійний прогін Selenium-проєкту аналогічного обсягу (25-30 тестів) за даними відкритих джерел займає 6-10 хвилин, тоді як у даному дослідженні цей показник не перевищує 3 хвилин [18].

Розроблений інструмент автоматизації був інтегрований у конвеєр CI/CD Azure DevOps та використовується для регулярних регресійних перевірок у середовищах dev та stage. HTML-звіти Playwright та статистика Testomat.io застосовуються для щоденного моніторингу якості збірок. Умовами ефективного впровадження є наявність стабільного тестового середовища, доступ до API для підготовки даних, налаштування захищених змінних середовища та регламентовані запуски тестів у pipeline. Отримані результати дозволяють зробити висновок, що запропонований підхід є не лише теоретично обґрунтованим, а й практично ефективним для використання у реальних e-commerce проєктах.

Окрім показників швидкодії та стабільності, додатково було проаналізовано точність виявлення дефектів та рівень похибок автоматизованих перевірок. Для цього результати 10 контрольних прогонів автоматизованих тестів порівнювались із результатами ручного тестування, проведеного за тими ж сценаріями. Загальна кількість перевірених сценаріїв становила 270 (27 тестів × 10 запусків). У 265 випадках результати автоматизованого та ручного тестування повністю співпали, що відповідає точності 98,15 %. А ось похибка у 1,85 % була пов'язана з асинхронним оновленням UI-елементів у середовищі stage та не свідчила про логічні помилки у функціонуванні системи.

Додатково було оцінено час зворотного зв'язку в CI/CD-конвеєрі, який визначається проміжком між створенням збірки та отриманням першого повідомлення про статус тестів. У середньому цей показник склав 3 хв 25 с, що відповідає вимогам оперативного контролю якості для e-commerce систем. У

класичних проєктах на базі Selenium, за даними практичних досліджень та відкритих звітів, середній час зворотного зв'язку коливається в межах 8-12 хвилин, що ускладнює швидке реагування команди на дефекти.

Також було досліджено використання апаратних ресурсів під час виконання тестів. Середнє завантаження процесора під час паралельного запуску не перевищувало 65 %, а обсяг використаної оперативної пам'яті – до 1,4 ГБ, що свідчить про ефективну оптимізацію внутрішніх механізмів Playwright та можливість масштабування тестового набору без суттєвого зростання навантаження на інфраструктуру.

Практичне впровадження отриманих результатів здійснено шляхом регулярного використання автоматизованих тестів у середовищах dev та stage в рамках CI/CD-процесу Azure DevOps. Обов'язковими умовами ефективної експлуатації розробленого інструменту є стабільне тестове API, контроль доступу до змінних середовища, стандартизовані браузерні конфігурації та фіксований набір тестових даних. Отримані експериментальні результати підтверджують доцільність використання розробленої моделі автоматизації для промислових e-commerce систем і демонструють її переваги над традиційними підходами як за швидкістю, так і за точністю та відтворюваністю результатів.

Висновки до розділу 3

У процесі дослідження результативності розробленого інструменту автоматизованого тестування e-commerce платформи було сформовано та апробовано комплексну методику експериментальної перевірки, що базується на використанні кількісних і якісних метрик оцінювання. Основна увага приділялася таким показникам, як швидкість виконання тестів, стабільність роботи тестового набору, рівень похибок і відтворюваність результатів. Стандартизовані умови проведення експериментів, фіксовані параметри середовища та багаторазові повторні прогони дозволили мінімізувати вплив випадкових факторів і забезпечити достовірність отриманих даних.

У ході експериментальної перевірки підтверджено, що використання Playwright у поєднанні з паралельним виконанням тестів суттєво підвищує продуктивність тестового процесу, скорочуючи час регресійного прогону більш ніж удвічі порівняно з послідовним виконанням. Аналіз стабільності показав низький рівень flaky-тестів і високу відтворюваність результатів при повторних запусках. Додатково доведено ефективність підготовки тестових даних через API, що дозволяє зменшити час підготовчого етапу та знизити кількість збоїв, пов'язаних із людським фактором.

Отримані результати підтвердили висунуту гіпотезу про доцільність використання Playwright як базового інструменту автоматизації для e-commerce систем. Практичне впровадження розробленого фреймворку в CI/CD-конвеєр Azure DevOps засвідчило його придатність для реального промислового використання, а також його ефективність для підвищення якості програмного продукту та оперативності виявлення дефектів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи магістра повністю досягнуто поставлену мету, що полягала у створенні та дослідженні інструменту автоматизації тестування e-commerce платформ на основі фреймворку Playwright, мови програмування TypeScript та патернів об'єктно-орієнтованого програмування. Усі завдання, визначені у вступі, виконано в повному обсязі, що підтверджується отриманими теоретичними й практичними результатами.

У процесі виконання роботи проаналізовано сучасні підходи до автоматизації тестування e-commerce платформ, зокрема e2e-тестування, підхід data-driven testing, а також використання хмарних тестових середовищ і CI/CD-орієнтованої автоматизації, що дало змогу визначити основні переваги, зокрема перехід до швидких, кросплатформних і стабільних e2e-рішень.

Досліджено можливості фреймворку Playwright для автоматизованого тестування веб-застосунків. Доведено, що Playwright забезпечує високу швидкість виконання тестів, стабільну роботу з динамічним UI, нативну підтримку багатьох браузерів, паралельне виконання та гнучкі механізми очікування. Обґрунтовано доцільність його використання як базового інструменту для автоматизації тестування складних e-commerce систем.

У межах роботи виконано аналіз архітектурних патернів об'єктно-орієнтованого програмування, зокрема Page Object, Singleton, Factory та Dependency Injection, що можуть бути застосовані у тестовому фреймворку. Встановлено, що їх використання забезпечує чітке розмежування відповідальностей між компонентами, повторне використання коду, слабку зв'язаність модулів та підвищення стабільності автоматизованих тестів при зміні інтерфейсу застосунку.

На основі виконаного аналізу сформульовано вимоги до інструменту автоматизації тестування багаторівневої архітектури фреймворку, які включають структурованість, масштабованість, стабільність, інтеграції з

CI/CD та безпечної роботи з конфіденційними даними. Побудовано архітектуру з рівнями core, page-object, tests, setup та utils, що забезпечує логічну організацію проєкту та зручність його супроводу.

Розробка інструменту автоматизації тестування з використанням Playwright та патернів ООП виконано шляхом практичної реалізації повнофункціонального тестового фреймворку. У межах фреймворку реалізовано дворівневу систему базових класів AbstractPage і BasePage, централізоване керування локаторами, глобальну авторизацію через storageState, механізми роботи з cookies та API-підготовки даних. Архітектура фреймворку побудована відповідно до принципів SOLID, що забезпечило його гнучкість, розширюваність та стійкість до змін інтерфейсу.

Відповідно до поставлених завдань реалізовано набір тестових сценаріїв для ключових компонентів e-commerce платформи (авторизація, каталог товарів, кошик, оформлення замовлення тощо). Принципи формування сценаріїв базуються на реальних бізнес-процесах, що забезпечує їх практичну релевантність. Сценарії є незалежними між собою, підготовка даних здійснюється через API, а вплив UI на стабільність тестів мінімізовано, що суттєво підвищує відтворюваність і надійність автоматизованих прогонів.

Проведено експериментальне дослідження ефективності, стабільності та продуктивності розробленого інструменту. Отримані кількісні показники свідчать про скорочення часу регресійного тестування в середньому на 60 %, зменшення кількості помилкових падінь тестів до рівня менше 10 % від загальної кількості запусків, а також стабільну роботу паралельних прогонів без втрати коректності результатів.

Порівняльний аналіз із класичними рішеннями на базі Selenium підтвердив переваги запропонованого підходу за швидкістю виконання, підтримкою паралельних запусків і стабільністю локаторів. Інтеграція фреймворку в CI/CD-конвеєр Azure DevOps забезпечила можливість регулярного автоматизованого контролю якості програмних продуктів.

Отримані результати узгоджуються з сучасними світовими тенденціями

розвитку автоматизованого тестування та можуть бути використані у практичній діяльності IT-підприємств, що займаються розробкою та супроводом e-commerce платформ. Практична цінність роботи полягає у можливості безпосереднього використання розробленого інструменту для функціонального, регресійного та частково інтеграційного тестування реальних веб-систем. Матеріали роботи також можуть застосовуватися у навчальному процесі при підготовці фахівців з автоматизації тестування програмного забезпечення.

Результати кваліфікаційної роботи можуть бути використані у науково-дослідній роботі кафедри в напрямі розробки та вдосконалення сучасних інструментів тестування програмного забезпечення. Одержані підходи та архітектурні рішення можуть бути основою для подальших досліджень і публікацій у галузі забезпечення якості програмних продуктів.

Подальший розвиток проєкту доцільно спрямувати на розширення тестового покриття, упровадження візуального та навантажувального тестування, використання контейнеризації для стандартизації середовищ, інтеграцію з системами аналітики якості, а також застосування методів штучного інтелекту для аналізу результатів тестування. У цілому результати роботи підтверджують доцільність обраних технічних та архітектурних рішень, а також їх наукову, практичну й соціально-економічну значущість для сучасних інформаційних систем електронної комерції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Суринович О. М., Маркін А. І. Розробка та дослідження інструменту автоматизації тестування e-commerce платформ за допомогою Playwright та патернів ООП // Proceedings of the International Scientific and Practical Conference «Science, Technology and Culture: Challenges and Perspectives». Paris, France, 17–19 Nov. 2025. Paris, 2025.

2. Microsoft Playwright Documentation. URL: https://playwright.dev/ (дата звернення: 20.10.2025).

3. Meszaros G. xUnit Test Patterns: Refactoring Test Code. Boston : Addison-Wesley Professional, 2007. 883 p.

4. State of JS Testing Frameworks 2023. URL: https://2023.stateofjs.com/en-US/libraries/testing/ (дата звернення: 05.10.2025).

5. Patel S. Modern Web Automation with Selenium and WebDriver. New York : Apress, 2021. 412 p.

6. Graham D., Fewster M. Software Test Automation: Effective Use of Test Execution Tools. Boston : Addison-Wesley, 2019. 528 p.

7. Freeman E., Freeman E., Bates B., Sierra K. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 694 p.

8. TypeScript Handbook. URL: https://www.typescriptlang.org/docs/handbook/ (дата звернення: 12.11.2025).

9. Playwright Documentation. URL: https://playwright.dev/docs/intro (дата звернення: 09.11.2025).

10. Fowler M. Page Object. URL: https://martinfowler.com/bliki/PageObject.html (дата звернення: 11.10.2025).

11. SOLID Principles – The First Five Principles of Object-Oriented Design. URL: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design (дата звернення: 15.11.2025).

12. GitHub Actions Documentation. URL: https://docs.github.com/en/actions (дата звернення: 17.11.2025).

13. ISTQB Foundation Level Syllabus 2023. URL: https://www.istqb.org (дата звернення: 18.11.2025).

14. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston : Addison-Wesley, 2019. 512 p.

15. OWASP Foundation. Application Security Verification Standard (ASVS), v4.x. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 16.11.2025).

16. Microsoft. Azure DevOps Pipelines Documentation. URL: https://learn.microsoft.com/azure/devops/pipelines (дата звернення: 10.10.2025).

17. ISTQB Foundation Level Syllabus. URL: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf (дата звернення: 30.10.2025).

18. Playwright vs Selenium: Performance and Reliability Comparison. URL: https://testautomationu.applitools.com/playwright-vs-selenium/ (дата звернення: 20.10.2025).