

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ЗАСОБІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ
ВЕБЗАСТОСУНКІВ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕСТІВ
RESEARCH OF AUTOMATION TOOLS FOR TESTING WEB
APPLICATIONS AND SOFTWARE IMPLEMENTATION OF TESTS

Виконав: студент(ка) 2 курсу, групи КНТ-223м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ЯВОРСЬКИЙ Б.В.

(прізвище та ініціали)

Керівник ШИТІКОВА О.В.

(прізвище та ініціали)

Рецензент КОЦУР М.І.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЯВОРСЬКОГО Богдана Вікторовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження засобів автоматизації тестування вебзастосунків та програмна реалізація тестів. Research of Automation Tools for Testing Web Applications and Software Implementation of Tests

керівник проєкту (роботи) к.т.н., ШИТІКОВА Олена Вікторівна к.т.н.,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 02 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково—пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Проєктування та основні рішення щодо реалізації. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ШИТІКОВА О.В., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “09” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Дослідження існуючих технологій автоматизації тестування вебзастосунків.	4–5 тижні	Розділ 2
4	Проектування алгоритмів роботи програми та розробка тест-дизайну.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

Богдан ЯВОРСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Олена ШИТІКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
93 с., 5 табл., 45 рис., 3 дод., 24 джерела.

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ,
КОМП'ЮТЕРНА ПРОГРАМА, PLAYWRIGHT, JAVASCRIPT.

Об'єкт дослідження — процес автоматизації тестування вебзастосунків.

Предмет дослідження — засоби (фреймворки) автоматизації тестування вебзастосунків.

Мета роботи — дослідження сучасних інструментів автоматизації тестування вебзастосунків та програмна реалізація тестів.

Матеріали, методи та технічні засоби: типізоване програмування, фреймворк Playwright, мова програмування TypeScript, персональний комп'ютер з процесором з частотою не менше 2.5 ГГц під управлінням операційної системи Microsoft Windows 10 (64bit).

Результати. Створено програмні автотести, що запускаються у вбудованому графічному інтерфейсі користувача фреймворку Playwright, який тестує інтернет магазин.

Висновки. Розроблено автотести для тестування вебзастосунку, які автоматизують тестування основного функціоналу інтернет магазину та прискорює процес відслідковування помилок.

Галузь використання — розробка та забезпечення якості вебзастосунків.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 93 pages, 5 tables, 45 figures, 3 appendixes, 24 sources.

TEST AUTOMATION, QUALITY ASSURANCE, COMPUTER SOFTWARE, PLAYWRIGHT, JAVASCRIPT.

The object of research is the process of automating the testing of web applications.

The subject of the research is tools (frameworks) for automating the testing of web applications.

The purpose of the work is the study of modern tools for automating the testing of web applications and the software implementation of tests.

Materials, methods and technical means: typed programming, the Playwright framework, the TypeScript programming language, a personal computer with a processor with a frequency of at least 2.5 GHz and running the Microsoft Windows 10 (64bit) operating system.

The results. Created software autotests that run in the built—in graphical user interface of the Playwright framework, which tests the online store.

Conclusions. Autotests have been developed for testing the web application, which automate the testing of the main functionality of the online store and accelerate the process of error tracking.

Field of use – development and quality assurance of web applications.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Аналіз тестування	12
1.1.1 Ручне (мануальне) тестування	12
1.1.2 Автоматизоване тестування.....	14
1.1.3 Переваги та недоліки типів тестування	15
1.2 Огляд існуючих фреймворків автоматизації.....	18
1.2.1 Selenium.....	18
1.2.2 Cypress	19
1.2.3 Playwright	19
1.2.4 NightwatchJS	20
1.2.5 Appium.....	21
1.2.6 Puppeteer.....	21
1.2.7 Katalon Studio.....	22
1.3 Постанова завдання.....	22
1.3.1 Функціональні вимоги.....	22
1.3.2 Вимоги до інтерфейсу	23
1.3.3 Експлуатаційні вимоги	23
1.4 Висновки за розділом 1	23
2 Матеріали і методи.....	25
2.1 Вибір та обґрунтування фреймворку	25
2.1.1 Selenium.....	25
2.1.2 Cypress	27
2.1.3 Playwright	28
2.1.4 NightwatchJS	30
2.1.5 Appium.....	30
2.1.6 Puppeteer.....	31

2.1.7 Katalon Studio.....	32
2.1.8 Порівняння та обґрунтування вибору.....	33
2.2 Вибір мови програмування	34
2.3 Вибір середовища розробки.....	38
2.4 Висновки за розділом 2	42
3 Проєктування та основні рішення щодо реалізації	43
3.1 Тест-дизайн та розробка тест-кейсів.....	43
3.1.1 Тест-дизайн	43
3.1.2 Розробка тест-кейсів	44
3.2 Розробка алгоритмів роботи автотестів.....	46
3.3 Підготовка та налаштування середовища розробки.....	50
3.4 Розробка програмної частини тестів.....	53
3.4.1 Тестовий вебзастосунок для реалізації автотестів	53
3.4.2 Написання коду	54
3.5 Висновки за розділом 3	59
4 Експлуатація, тестування та експериментальне дослідження програми	60
4.1 Умови застосування.....	60
4.2 Умови передачі автотестів	60
4.3 Характеристики програмної частини автотестів	61
4.4 Інструкція з використання інтерфейсу користувача	61
4.5 Тестування автотестів у інтерфейсі користувача	67
4.6 Висновки за розділом 4	72
Висновки	73
Перелік джерел посилань	74
Додаток А Технічне завдання	76
Додаток Б Текст програми	79
Додаток В Слайди презентації.....	86

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	— Application Programming Interface;
HTTP	— Hypertext Transfer Protocol;
IDE	— Integrated Development Environment,
IOS	— Iphone Operation System;
QA	— Quality Assurance;
ROI	— Return On Investments;
SDK	— Software Development Kit;
VS Code	— Visual Studio Code;
IT	— інформаційні технології;
OC	— операційна система;
ПЗ	— програмне забезпечення.

ВСТУП

У сучасному світі вебзастосунки є невід’ємною частиною повсякденного життя. Вони забезпечують доступ до інформації, комунікації, розваг та виконання професійних задач через інтернет. Згідно зі статистикою, у 2023 році понад 5 мільярдів людей користуються Інтернетом, що становить близько 63% населення світу. Значна частина цього трафіку припадає на взаємодію з вебзастосунками — від соціальних мереж до електронної комерції та дистанційного навчання. Такі системи не тільки полегшують взаємодію між користувачами та компаніями, але й сприяють зростанню економіки та глобалізації суспільства [1].

Сучасні вебзастосунки розробляються самими різноманітними методологіями, технологіями та рівнями спеціалістів. Тестування ж вебзастосунків займає дуже велику частину розробки. Їм займається відділ QA, що повинен забезпечувати якість програмного забезпечення і зокрема вебзастосунків. Тестування глобально поділяється на ручне та автоматизоване.

Автоматизоване тестування продовжує змінювати правила в галузі тестування програмного забезпечення. У 2024 році Headspin.io очікує ще більшого впровадження інструментів і фреймворків автоматизованого тестування. Організації використовують автоматизацію для виконання повторюваних завдань, що дозволяє тестувальникам зосередитися на більш складних сценаріях. Завдяки вдосконаленню таких інструментів, як Selenium, Cypress і TestComplete, автоматизоване тестування стає доступнішим і ефективнішим, забезпечуючи ширше охоплення тестуванням і швидші цикли зворотного зв’язку [2].

Об’єкт дослідження — процес автоматизації тестування вебзастосунків.

Предмет дослідження — засоби (фреймворки) автоматизації тестування вебзастосунків.

Тому метою цієї дипломної кваліфікаційної роботи буде дослідження сучасних інструментів автоматизації тестування вебзастосунків та програмна реалізація тестів.

Для досягнення поставленої мети необхідно:

- дослідити інструменти автоматизації тестування;
- обґрунтувати вибір фреймворку;
- виконати застосування обраної технології для програмної реалізації автоматичних тестів вебзастосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Ефективність і надійність вебзастосунків є критично важливими, адже навіть незначні помилки можуть призвести до фінансових втрат, втрати репутації чи компрометації даних користувачів. Наприклад, згідно з опитуванням Uptime щодо стійкості центрів обробки даних за 2022 рік, за останні три роки проблеми, пов'язані з мережею, були основною причиною всіх інцидентів простою ІТ-служб — незалежно від ступеня серйозності [3].

Збої, пов'язані з проблемами програмного забезпечення, мережі та систем, зростають через ускладнення зі збільшенням використання хмарних технологій, програмно-визначених архітектур і гібридних, розподілених архітектур [3].

Переважає більшість збоїв, що стаються через людські помилки, пов'язані з ігноруванням або неадекватними процедурами. За останні три роки майже 40% організацій зазнали серйозних збоїв, спричинених людською помилкою. З цих інцидентів 85% це недотримання персоналом процедур або недоліки в самих процесах і процедурах [3].

Зовнішні ІТ-провайдери викликають більшість великих громадських збоїв. Чим більше робочих навантажень передано зовнішнім постачальникам, тим більше ці оператори спричиняють резонансні публічні збої. На сторонніх комерційних ІТ-операторів (включаючи хмарних, хостингових, колокейшн, телекомунікаційних провайдерів тощо) припадає 63% усіх публічно повідомлених збоїв, які Uptime відстежує з 2016 року. У 2021 році комерційні оператори спричинили 70% усіх збоїв [3].

Тривалий простой стає все більш поширеним явищем у публічно зареєстрованих збоях. За останні п'ять років проміжок між початком великого громадського відключення та повним відновленням значно розтягнувся. Майже 30% цих збоїв у 2021 році тривали більше 24 годин, що викликає занепокоєння порівняно з лише 8% у 2017 році [3].

1.1 Аналіз тестування

1.1.1 Ручне (мануальне) тестування

Ручне тестування – це тип тестування програмного забезпечення, при якому тестовий кейс виконується тестувальником вручну без допомоги будь-яких автоматизованих інструментів. Компанії використовують ручне тестування як метод виявлення помилок або проблем у своєму програмному забезпеченні. Хоча дехто описує це як просту або примітивну форму тестування, в кінцевому підсумку це дозволяє встановити функціональність програми без використання сторонніх інструментів тестування [4].

У всіх формах тестування програмного забезпечення є певні ручні аспекти, оскільки є деякі особливості програми, які просто неможливо протестувати без певного ручного втручання [4].

Існує кілька етапів, на яких розробники використовують ручне тестування, перший – на етапі розробки базового функціоналу. Коли основна функціональність програмного забезпечення знаходиться в стадії розробки, розробники тестують роботу кожної частини програми вручну, оскільки це швидше, ніж створювати тестові кейси для досить простих частин коду [4].

Ручне тестування також переважає на останніх етапах розробки, коли програма має створений інтерфейс користувача. Тестування інтерфейсу передбачає перевірку того, як реальний користувач реагує на те, як розроблено меню і як працює система [4].

Оскільки це пов'язано з великою кількістю якісних даних і особистої думки, а не з чистими кількісними показниками, ручне тестування є ідеальним варіантом для отримання більш глибокого розуміння продукту.

Існує кілька різних аспектів програмного забезпечення, які перевіряють ручні тести, кожен з яких краще перевіряти за допомогою ручного тестування завдяки специфічним завданням тестів [4].

Тестування базового функціоналу це одна з найбільш ранніх частин процесу тестування програмного забезпечення яка розглядає базову функціональність програмного забезпечення. На цьому етапі розробник або тестувальник переглядає один з функціональних модулів коду і оцінює, чи працює він так, як очікувалося. Прикладом цього є програмне забезпечення для роботи з базами даних, коли тестувальники вводять частину даних у функцію і вже знають очікуваний результат. Якщо вони збігаються, тест пройдено успішно. Тестування на цьому етапі процесу закладає міцний фундамент для подальшої роботи компанії [4].

Тестування інтерфейсу фокусується як на тому, як працює інтерфейс, так і на тому, чи є він зручним для користувача, включаючи те, чи може користувач взаємодіяти з усіма функціями і чи є меню естетично привабливим. Інтерфейс користувача – це користувацький інтерфейс програмного забезпечення, тобто меню, кнопки та інтерактивні можливості, доступні користувачеві. На цьому етапі ручне тестування є необхідністю, оскільки якісна інформація, наприклад, чи добре виглядають інтерфейси, не може бути отримана автоматизованою програмою [4].

Дослідницьке тестування – це тестування, яке потрібно виконати лише один або два рази, отримавши таку назву, оскільки воно є частиною “дослідження” програмного забезпечення на наявність будь-яких несподіваних функцій або помилок. Ручне тестування краще підходить для цього випадку, оскільки написання коду для тестового кейсу займає багато часу, а людина, яка вручну заходить у програму і перевіряє її, витрачає менше часу. Наприклад, коли розробник хоче перевірити, чи правильно інтегрована певна функція, за допомогою одного тесту, який перевіряє правильність переміщення даних у програмі [4].

1.1.2 Автоматизоване тестування

Мета будь-якого тестування — отримати максимум інформації про якість додатку в рамках ресурсів проєкта. Як правило, ці ресурси — час, бюджет, компетенції команди, програмне і апаратне забезпечення — обмежені. Одним зі способів їх оптимізації є автоматизація тестування [5].

З практичної точки зору автоматизація тестування означає, що ряд задач, які виконувались вручну інженерами, передається програмно-апаратним комплексам. Це передбачає, що програмне рішення виконає певну задачу з прийнятною якістю і меншими витратами. В першу чергу, часу. Програма не витрачає його на перерви та каву, вона не втомлюється під кінець робочого дня і в теорії може працювати в режимі 24x7x365, хоча на практиці більше використовують програми на короткий час за запитом [5].

Програми роблять лише те, для чого були створені: вони слідують закладеним у них алгоритмом. Рутинні операції вони відпрацьовують набагато краще за людину. Наприклад, завиграшки виконують сценарій з тисячами різних наборів даних. Для людини це доволі складно: виконання рутинних операцій впродовж тривалого часу викликає втому і нудьгу. А вони призводить до помилок і уповільнення темпу роботи [5].

Втім, програми не володіють інтуїцією і не спроможні генерувати творчі рішення. Завдяки делегуванню роботи програмі у інженерів вивільняється час для вирішення більш квалікованих задач. Наприклад, можна запровадити у процес розробки новий кейс або інструмент тестування. Це надасть додаткову інформацію про якість додатку та знизить ймовірність появи помилок, які не були виявлені раніше [5].

Важливим фактором автоматизації є підвищення надійності. Достатньо один раз запрограмувати і налагодити алгоритм, який реалізує програма, і він виконуватиметься завжди однаково, з одним і тим самим результатом. Програма не переплутає і не забуде ввести вхідні дані, не

пропустить перевірку потрібного параметру і в неї не «замілюється око» від нескінченних повторів. Якщо трапиться збій — його буде зафіксовано [5].

З технічної точки зору, будь-які завдання тестування можуть бути автоматизовані. Або, принаймні, більшість з них. Питання в тому, яких витрат це вимагатиме. Деякі види тестування простіше, дешевше і швидше виконати вручну. Наприклад, оцінити зручність графічного користувацького інтерфейсу: людина може на око оцінити розташування елементів керування, їх компонування, кольорову гамму, логічність інтерфейсу [5].

Так, більша частина таких перевірок може бути реалізована програмно. Ось тільки час, витрачений на створення відповідного алгоритму, його валідацію, імплементацію, виконання та аналіз отриманих результатів, може в багато разів перевищувати час і вартість виконання такої перевірки вручну. Щоб уникнути цього, на етапі підготовки автоматизації тестування підраховують економічну доцільність автоматизації ROI (Return On Investments) [5].

Однак для багатьох видів тестування автоматизація є економічно обґрунтованою. В першу чергу, для таких трудомістких і відносно довготривалих видів тестування, як регресійне та димове тестування. Їх особливість — в багаторазовому повторі великої кількості тестів в однакових умовах. Тому їх легко налагодити і потім виконувати раз за разом [5].

1.1.3 Переваги та недоліки типів тестування

Люди звикли протипоставляти ручне тестування автоматизованому. Обидва засоби перевірки програмного забезпечення мають переваги та недоліки.

Основні переваги автоматизації тестування — швидкість та відсутність людського фактору зробили широким його використання. Здається, що автоматизація застосовується для виконання все більшої

кількості задач. Може здаватися, що вона може повністю витіснити роботу мануальних тестувальників у найближчому майбутньому [6].

Але спеціалісти, які здійснюють тестування вебсайтів, десктопне та мобільне тестування, стверджують, що цього ніколи не станеться, тому що в деяких аспектах жоден інструмент не може зрівнятися з інженером з тестування [6].

Переваги ручного тестування ПЗ за автоматизоване:

- тестувальники можуть вирішити перевірити області або виконати тести, які не враховані та не автоматизовані, і таким чином збільшити тестове покриття та знайти помилки, які б не знайшов інструмент;

- тестувальники бачать дефекти, які не були передбачені в автоматизованих тестах. Все передбачити неможливо — інженери з тестування можуть імпровізувати, а інструменти такого не роблять;

- інструменти не мають почуттів, які мають кінцеві користувачі, тому вони не можуть розраховувати на попередній досвід та інтуїцію. Тільки люди можуть повністю оцінити програму, а інструмент лише визначає, чи проходить програма тести [6].

Автоматизоване тестування передбачає таку взаємодію між програмним продуктом та інструментами тестування, за якої втручання людини зводиться до мінімуму. Інструменти автоматизації корисні для багатьох видів діяльності. Вони дозволяють заощадити масу часу, виконуючи регресійне, модульне, навантажувальне та деякі інші види тестування [6].

Процеси, які можуть бути автоматизованими:

- регресійне тестування;
- димове тестування;
- тестування продуктивності;
- тестування навантаження;
- тестування методом білого ящика;
- аналіз покриття коду.

Але найбільшою помилкою розробників є автоматизація не тих процесів під час розробки програмного забезпечення. Не все можна запрограмувати та виконати автоматично [6].

Що ніколи не слід автоматизувати:

- нестабільне програмне забезпечення. Якщо програмний продукт знаходиться на стадії розробки та піддається змінам, автоматизоване тестування не буде доречним;

- сценарії тестування, які рідко використовуються. Не слід автоматизувати ті сценарії, що будуть виконуватися рідко;

- аналіз програмного кода і документації. Автоматизація цих процесів може в подальшому тільки додати клопоту.

Зазвичай для автоматизації найбільше підходять задачі, що повторюються. Саме тому деякі компанії вважають за краще спочатку провести ручне тестування, щоб визначити ті процеси, які виконуватимуться неодноразово, для їх подальшої автоматизації [6].

Переваги автоматизованого тестування ПЗ за ручне:

- написаний скрипт можна виконувати скільки завгодно разів;
- автотести виконуються з більшою точністю, так як людський фактор виключений;

- час виконання автотестів набагато менший, ніж виконання тих самих перевірок вручну;

- тестовий прогін можна запланувати на будь-який час, запустити за розкладом, щоб він виконувався без участі людини.

Обидва види тестування мають як переваги, так і недоліки. На реальних проєктах часто використовується комбінація ручного та автоматизованого тестування, до того ж рівень автоматизації залежить як від типу проєкта, так і від особливостей виробничих процесів в компанії. Комбінація обох видів — ідеальний спосіб отримати від тестування максимальний результат [6].

1.2 Огляд існуючих фреймворків автоматизації

У цьому розділі узагальнено розглянемо сучасні і найбільш популярні фреймворки автоматизації тестування вебзастосунків. Більш детальне заглиблення у переваги і недоліки вибраних фреймворків буде у розділі 2 з обґрунтуванням вибору інструментів тестування ПЗ.

1.2.1 Selenium

Selenium існує у світі автоматизації протягом тривалого часу. Це інструмент автоматизації з відкритим вихідним кодом, який полегшує автоматизацію тестування для сучасних вебдодатків за допомогою нещодавньої версії Selenium 4. Функції Selenium 4 включають помітну інтеграцію chrome CDP і багато іншого, щоб зробити тестування ефективнішим. Ось чому багато QA оновлюють Selenium 3 до Selenium 4 [7].

Компоненти автоматизації тестування Selenium:

- Selenium IDE: Selenium IDE допомагає записувати та відтворювати ваші тести;
- Selenium Grid: допомагає запускати тести на кількох машинах паралельно;
- Selenium WebDriver: Selenium WebDriver найбільш корисний для запуску тестів Selenium у браузері;
- Selenium RC: Selenium RC допомагає писати тестові приклади різними мовами програмування для автоматизації тестів інтерфейсу користувача для вебдодатків на будь-якому вебсайті HTTP. Нині він не дуже використовується.

Без сумніву, Selenium є найпопулярнішим інструментом на ринку автоматизації тестування. Це найкращий вибір, якщо ваша програма велика і містить поєднання старих і нових технологій. Якщо вашій організації

потрібна система автоматизації для інтеграції з браузером і мобільними пристроями, то Selenium, швидше за все, буде кращим [7].

Як згадувалося раніше, Selenium потребує обслуговування, а це означає, що вам потрібні спеціальні ресурси для автоматизованого тестування. Використання найкращих переваг Selenium також вимагає продвинутого досвіду програмування, на відміну від будь-якої іншої структури [7].

1.2.2 Cypress

Cypress змінив уявлення про систему автоматизації інтерфейсу користувача, надавши багато функцій. Це один із популярних інструментів автоматизації тестування на ринку. Cypress — інструмент із відкритим вихідним кодом, який надає Cypress Dashboard як професійну послугу з платним пакетом [7].

Cypress надає зручні для розробника функції, такі як легке налаштування, інсталяція, налагодження, спеціальне вікно Cypress для виконання тестів тощо [7].

Cypress добре підходить для сучасних вебдодатків, таких як React, Vue, Angular тощо. Не можна заперечувати той факт, що це зручний інструмент для розробників. Cypress агресивно надає функції, але має свої обмеження. Якщо ви знайомі з Javascript, ви можете пропустити `async/await` у Cypress [7].

1.2.3 Playwright

Playwright був запущений у 2020 році, і за кілька років він подорослішав і залучив багатьох користувачів. Це через їхню агресивну стратегію випуску та добре задокументований код. Playwright — це інструмент із відкритим кодом. Однак він прив'язаний до бренду Microsoft.

Це один із піонерів безголового тестування браузерів, що робить його досить популярним [7].

Незважаючи на те, що Playwright є новим інструментом, він має багатий набір функцій, він підтримує кілька мов, але план зв'язування мови досі незрозумілий. Платформа автоматизації інтерфейсу користувача Playwright за замовчуванням постачається зі своїми тестовими бібліотеками, тому, якщо ви плануєте інтегрувати інші бібліотеки, крім стандартних, наприклад Jest, Jasmine і Mocha, вам можуть знадобитися досвідчені знання для їх налаштування [7].

Якщо ваша організація надто стурбована тестуванням рідних мобільних додатків, Playwright не розпочинає над цим роботу. Але даний фреймворк тестування має менше обмежень і підтримує всі сучасні фреймворки, React, Angular тощо [7].

1.2.4 NightwatchJS

NightwatchJS — це фреймворк на основі NodeJS, створений за допомогою NodeJS, який використовує протокол Webdriver. BrowserStack підтримує Nightwatch. Nightwatch можна використовувати для наскрізного тестування, тестування API, модульного тестування та інтеграційного тестування за допомогою комбінації служб NodeJS і NightWatchJS API [7].

NightwatchJS розвивається та додає нові функції, тому його варто розглянути. Однак ви повинні переконатися, що обмеження не блокують вашу організацію. Оскільки NightwatchJS створено на основі Selenium, складності в Selenium все ще виникають [7].

1.2.5 Appium

Appium — це платформа автоматизації інтерфейсу користувача з відкритим вихідним кодом, яка підтримує власну автоматизацію мобільних додатків. За допомогою Appium можна тестувати власні мобільні програми Android, iOS і Windows SDK. Appium має хороший графічний інтерфейс, який допомагає перевіряти та ефективно писати код [7].

Appium — найпопулярніший фреймворк для мобільного тестування. Однак налаштування вимагає досвіду кодування. Тож новачок може відчувати складнощі. Appium можна інтегрувати з Selenium; ви можете безперешкодно змусити фреймворк працювати на комп'ютері та мобільному пристрої за допомогою налаштованого фреймворку. Якщо у вас уже є готовий фреймворк на основі Selenium і ви шукаєте інструмент для розширення фреймворку для мобільних додатків, то Appium — найкращий вибір [7].

1.2.6 Puppeteer

Puppeteer — це платформа з відкритим вихідним кодом, якою керує Google. Puppeteer створено за допомогою NodeJS і підтримує мови програмування Javascript/Typescript. Puppeteer постачається з ліцензією Apache 2.0. Puppeteer — найкращий інструмент щодо швидкості та стабільності, але його основна проблема полягає в тому, що він офіційно підтримує лише браузері на базі Chromium [7].

Підтримка Puppeteer у Firefox все ще знаходиться в експериментальному стані. Puppeteer надає оболонку для протоколу Chrome DevTools, вам потрібно інтегрувати бібліотеки тверджень, такі як Jest Mocha тощо. Реалізація Puppeteer вимагає хороших знань мови програмування Javascript [7].

1.2.7 Katalon Studio

Katalon Studio не є ПЗ з відкритим кодом. Однак у нього є безкоштовна версія; Оскільки багато ручних тестувальників вважають його корисним завдяки функції запису та безпроблемній інтеграції, ми згадуємо його тут. Katalon Studio підтримує системи на базі Windows, macOS і Linux. У безкоштовній версії Katalon доступні обмежені функції. Зверніть увагу, що Katalon є безкоштовним інструментом, а не з відкритим кодом. На відміну від проєкту з відкритим кодом, функція та обмеження контролюються Katalon. Це може змінитися в будь-який час [7].

1.3 Постанова завдання

Головним завданням дипломної кваліфікаційної роботи є дослідження сучасних фреймворків автоматизації тестування і використання відповідного ПЗ для реалізації автоматизованого тестування вебзастосунку.

1.3.1 Функціональні вимоги

Для системи автоматизації тестування вебзастосунку будуть висунуті такі вимоги з функціональності:

- система повинна проходити тестування сайту покроково;
- система повинна відображати статус тестів (пройдені/провалені);
- система повинна дати змогу відстежити у якій частині вебзастосунку проблема;
- система повинна дати змогу відстежити особисту недосконалість що призвела до проваленого тесту;
- система повинна проводити тестування на основних сучасних рушіях браузерів (Firefox, Chromium та Webkit);

— система повинна давати змогу улюбий момент запустити себе і відтворити тестування вебзастосунку при виконанні вимог.

1.3.2 Вимоги до інтерфейсу

Для системи автоматизації тестування будуть висунуті такі вимоги:

- система повинна мати вбудований інтерфейс користувача для полегшення опанування;
- інтерфейс повинен давати широкий спектр можливостей для дебагінгу ПЗ самих автотестів;
- інтерфейс повинен бути гнучким і адаптованим під різні роздільні здатності екрану.

1.3.3 Експлуатаційні вимоги

Вимоги будуть сформовані здебільшого від вимог середовища у якому воно має бути запусчено:

Мінімальні вимоги до програмно-технічних засобів:

- ОС Windows 10/11 (64-bit);
- процесор з тактовою частотою не менше за 2.5 ГГц;
- наявність оперативної пам'яті не менше 8 Гб;
- місце на диску не менше за 100 Мб;
- .NodeJS останньої версії;
- наявність екрану;
- наявність пристроїв типу клавіатура і миша.

1.4 Висновки за розділом 1

З метою дослідження сучасних систем автоматизації тестування вебзастосунків було проведено розгляд актуальності тестування, двох

основних підходів до тестування (ручного та автоматизованого), популярних на даний момент серед тестувальників фреймворків автоматизації.

На основі розглянутого матеріалу були висунуті функціональні, нефункціональні вимоги, а також вимоги до інтерфейсу для майбутньої системи автоматизованого тестування вебзастосунку.

2 МАТЕРІАЛИ ТА МЕТОДИ

У другому розділі буде проведено вибір та обґрунтування фреймворку, мови програмування та середовища розробки з приведенням порівняльних таблиць.

2.1 Вибір та обґрунтування фреймворку

Для початку створення автотестів головне визначитись з фреймворком що буде використаний. Зазвичай у роботі розробники вибирають фреймворк в залежності від проєкту на який треба внедрити автоматизоване тестування. В конкретному випадку тестовий сайт не має якихось спеціальних факторів, а також бюджету для реалізації немає. Враховуючі ці фактори можна провести аналіз і порівняння популярних фреймворків для вибору.

Спочатку розглянемо переваги і недоліки кожного з фреймворків, загальну кількість користувачів та рейтинг на платформі GitHub (зірочки від користувачів), що не відображає фактичну популярність але непогано показує відношення спеціалістів до цього застосунку.

2.1.1 Selenium

На сьогодні Selenium має близько 31 тисячі зірочок та 300 тисяч користувачів [8].

Переваги Selenium:

- Selenium це інструмент із відкритим вихідним кодом, який економить витрати на ліцензії;
- підтримує кілька мов програмування, таких як Java, Javascript, C#, Python, Ruby тощо;
- підтримує всі основні браузерни Chrome, Edge, Firefox і Safari;

- підтримує кілька операційних систем Windows, Mac і ОС на базі Linux;
- сильна підтримка спільноти, оскільки Selenium використовується багатьма QA у всьому світі;
- підтримує паралельне виконання за допомогою Selenium Grid, що скорочує час виконання тесту.
- Selenium має велику кількість плагінів і пакетів спільноти, доступних на ринку;
- структуру можна розширити до мобільного тестування [7].

Недоліки Selenium:

- Selenium надає API високого рівня, реалізація залежить від користувача. Це означає, що спеціалісту потрібно створити структуру відповідно до конкретних потреб. Новачки у автоматизованому тестуванні зтикаються з труднощами під час налаштування надійної системи;
- спеціальна технічна підтримка SLA недоступна; оскільки Selenium є відкритим кодом, необхідно покладатися на підтримку спільноти;
- Selenium має залежність від Webdriver, оскільки працює з драйверами для браузера (наприклад, для запуску тестів у Chrome потрібен драйвер Chrome). Це сповільнює виконання тесту, а узгодження версій браузера та драйвера іноді складно;
- керування залежностями, як згадувалося раніше, selenium надає лише API високого рівня для керування діями браузера. Необхідно покладалися на пакети сторонніх розробників для конфігурацій тестового середовища та звітування. Що в свою чергу збільшує затрати на обслуговування;
- проблема з сучасними фреймворками: Selenium може дуже добре працювати з додатком, створеним із застарілою архітектурою. Для сучасних фреймворків веброзробки, таких як React, Angular, Vue тощо, він може працювати не дуже добре;

— на відміну від сучасної системи автоматизації тестування, Selenium не є дуже агресивним у випуску функцій; він постійно прислухається до голосу користувача та впроваджує нові функції [7].

2.1.2 Cypress

Cypress на сьогоднішній момент має близько 47 тисяч зірочок та понад одного мільйона користувачів [9].

Переваги Cypress:

- налаштування користувацького інтерфейсу фреймворку є дуже простим;
- підтримує різні типи тестування, Cypress підтримує модульне, інтеграційне, компонентне, тестування API та наскрізне тестування;
- функція Cypress Time Travel забезпечує миттєвий знімок кожного кроку тесту;
- Cypress записує відео тестів, можна налаштувати опцію знімка екрана;
- не потрібно явно вказувати затримки перед командами. Cypress автоматично чекає перед виконанням будь-яких дій;
- тестування Cypress надійніше і швидше. У Cypress немає концепції драйвера браузера, він безпосередньо взаємодіє з браузером, тому тести швидші та стабільніші;
- вікно Interactive Test Runner у Cypress допомагає пришвидшити розробку та налагодження;
- добре структурована документація в Cypress дозволяє відносно легко вивчати нові функції та концепції;
- спільнота Cypress зростає, тому можна також сподіватися на хорошу підтримку фреймворку в майбутньому [7].

Недоліки Cypress:

- перехресне обмеження походження: офіційно задокументовано, що Cypress підтримує лише ті самі вихідні URL-адреси в тесті. Наприклад, якщо ваш вебсайт має автентифікацію Google, URL-адреса вашого вебсайту переспрямується. Оскільки гугл має інший домен, Cypress не дозволяє таке переспрямування. Це одна з головних причин, чому багато організацій вагаються приймати Cypress;

- Cypress підтримує лише мову програмування JavaScript/Typescript. Багато інженерів з автоматизації тестування працюють на Java або C#. Через це організаціям потрібно інвестувати в розширення набору навичок своїх спеціалістів;

- Cypress не заохочує шаблони об'єктів сторінки, хоча моделі об'єктів сторінки можуть бути створені в Cypress певною мірою, Cypress не рекомендує цього;

- налаштування паралельного тестування є складним у Cypress;
- фреймворк не підтримує кілька вкладок і вікон;
- немає прямої підтримки фреймворку BDD, як Cucumber;
- використовуючи Cypress, зазвичай треба додавати пакети сторонніх розробників, що збільшує кошти на обслуговування, і структура поступово стає нестабільною [7].

2.1.3 Playwright

Playwright на момент написання має близько 68 тисяч зірочок та 139 тисяч користувачів [10].

Переваги Playwright:

- підтримує кілька мов програмування, таких як Javascript/Typescript, Python, Java, C# (.Net);

- підтримує тестування кількох різних браузерів одночасно, таких як Firefox, Edge, Chrome і Safari;

- немає залежності від вебдрайвера, оскільки Playwright безпосередньо виконує команду в браузері. Таким чином, тести є значно швидшими та стабільнішими;

- тестувати дуже зручно з вебпереглядач Safari на Windows із інтерфейсом користувача Playwright. Зазвичай браузери Safari можна тестувати лише з macOS, але Playwright надає версію WebKit із виправленнями для Windows, яка більше схожа на Safari;

- підтримує всі селектори;
- підтримує сучасні фреймворки, такі як React, Angular і Vue;
- Playwright codegen, тестовий записувач для вебтестів. Дозволяє записувати послідовність дій для тестів через інтерфейс, неймовірно зручна функція для новачків;

- Playwright пропонує налагодження різними способами. Тести можна налагодити за допомогою Playwright Inspector, Playwright Trace Viewer, Browser Developer Tools, Visual Studio Code debugger, Verbose API logs;

- підтримка кількох бібліотек тверджень, за замовчуванням поставляється з програмою тестування Playwright, однак ви можете налаштувати його за допомогою Jest, Jasmine Mocha тощо [7].

Недоліки Playwright:

- хоча Playwright підтримує Safari, це версія з відкритим кодом WebKit, а не справжній Safari;

- він не підтримує мобільні програми;

- ви можете розширити Playwright для Cucumber, але прямої підтримки немає. Це можна зробити за допомогою сторонніх плагінів або пакетів [7].

2.1.4 Nightwatch

Nightwatch на момент написання має близько 12 тисяч зірочок та 149 тисяч користувачів [11].

Переваги NightwatchJS:

- NightwatchJS помірно легко встановити та налаштувати;
- тестові скрипти NightwatchJS більш читабельні;
- його можна використовувати для різних видів тестування;
- він підтримує тестування кількох браузерів, таких як Chrome, Firefox, Edge;
- NightwatchJS підтримує page object pattern [7].

Обмеження NightwatchJS:

- оскільки NightwatchJS використовує Selenium у серверній частині, будь-які обмеження Selenium також діятимуть у NightwatchJS.

2.1.5 Appium

Станом на сьогодні Appium має близько 19 тисяч зірочок та 6 тисяч користувачів [12].

Переваги Appium:

- Appium це бібліотека з відкритим вихідним кодом, яка економить витрати на ліцензію;
- Cross-Platform Support Appium Забезпечує кросплатформну підтримку Safari в IOS і Chrome, а також у вбудованому браузері для Android, і тест бездоганно працює на всіх пристроях;
- Appium підтримує кілька мов програмування, таких як Java, Ruby, C#, PHP і Python;
- фреймворк Appium можна інтегрувати з фреймворком Selenium;
- підтримує реальні пристрої, емулятори та симулятори;

- Appium надає інструмент запису, за допомогою якого початківці можуть легко розпочати тестування мобільних програм;
- Appium підтримує хмарне тестування, таке як Browserstack тощо;
- Appium можна інтегрувати з інструментами DevOps [7].

Недоліки Appium:

- налаштування структури Appium вимагає багато зусиль і досвіду кодування;
- інтеграція звітів складна порівняно з іншими фреймворками;
- Appium підтримує тільки Android версії 4.3 і вище;
- підтримка Appium для гібридних програм є складною; перемикання з браузера на рідну програму вимагає написання складного коду [7].

2.1.6 Puppeteer

Puppeteer на момент написання має близько 89 тисяч зірочок та 490 тисяч користувачів [13].

Переваги Puppeteer:

- Puppeteer підтримує безголове тестування браузера;
- він підтримує виконання спеціального JavaScript, що полегшує взаємодію з динамічним вмістом;
- Puppeteer часто використовується для наскрізного тестування, інтеграційного тестування та функціонального тестування;
- Puppeteer надає детальні повідомлення про помилки та можливості налагодження [7].

Недоліки Puppeteer:

- Puppeteer не містить вбудованих функцій керування тестами, таких як організація тестів, звітність або відстеження результатів;

- він працює в однопоточному середовищі, тому потребує додаткового налаштування або конфігурації, як BrowserStack Automate, для запуску паралельних тестів;
- перехресне тестування за допомогою Puppeteer може бути складним, оскільки він не підтримує інші браузері, такі як Firefox, Safari або Edge [7].

2.1.7 Katalon Studio

Переваги Katalon Studio:

- він підтримує тестування вебдодатків, мобільних додатків (iOS і Android) і тестування API;
- Katalon Studio містить вбудовану IDE з такими функціями, як редактори коду, інструменти налагодження та звіти про виконання тестів;
- він забезпечує функцію запису та відтворення, що дозволяє користувачам фіксувати взаємодії;
- Katalon інтегрується з різними інструментами CI/CD [7].

Недоліки Katalon Studio:

- Katalon має високу складність для навчання;
- хоча Katalon пропонує безкоштовну версію, більшість розширених функцій та корпоративних можливостей доступні лише за платними ліцензіями;
- важко керувати тестовими даними у великому масштабі;
- значна залежність від вбудованих функцій Katalon може призвести до проблем, якщо знадобиться інтеграція з іншими інструментами чи системами [7].

2.1.8 Порівняння та обґрунтування вибору

Одразу виключаються з порівняння фреймворки Appium, Puppeteer Katalon Studio.

Appium не підходить під вимоги завдання через те що цей фреймворк спроектований працювати тільки з мобільними браузерами, також надто складне налаштування і низьку швидкість виконання тестів.

Puppeteer виключається через те що може працювати виключно з браузерами на рушії Chromium. На даний момент тестування у компаніях вимагає покривати якумога більше різних браузерів. Також підтримка мов програмування надто вузька. Підтримується тільки JavaScript, що може сильно ускладнити адаптацію під інструмент спеціаліста з автоматизації тестування програмного забезпечення.

Katalon Studio не враховується по причині того що по цьому фреймворку надто низька кількість матеріалів для навчання, а також аудиторія що їм користується, це сильно ускладнює використання. Багато функцій закриті за пейволом.

У таблиці 2.1 приведено ітогове порівняння фреймворків.

Таблиця 2.1 – Порівняння фреймворків

Фреймворк	Selenium	Cypress	Playwright	Nightwatch
Доступні мови	Java, Python, C#, Ruby	JavaScript	JavaScript, Python, C#, Java	JavaScript
Підтримка браузерів	Chrome, Firefox, Safari, Edge	Chrome, Firefox, Edge	Chrome, Firefox, Safari, Edge	Chrome, Firefox, Safari, Edge
Тестування API	-	+	+	+

Продовження таблиці 2.1

Фреймворк	Selenium	Cypress	Playwright	Nightwatch
Кількість матеріалу	Велика	Середня	Середня (зростає)	Мала
Простота використання	Складно	Просто	Дуже просто	Середньо
Швидкість і стабільність	Повільно, нестабільно	Швидко, нестабільно	Дуже швидко, стабільно	Відносно швидко, стабільно

Після уважного порівняння аналогів можна прийти до чіткого висновку що Playwright це інструмент автоматизації тестування, який повністю задовольняє поставленим вимогам до фреймворку, що були зазначені у розділі 1.3. Цей фреймворк або має переваги над зазначеними аналогами у одній з категорій або нічим не поступається.

Проведений аналіз показав невелику кількість недоліків Playwright в порівнянні з аналогами, а також зростання популярності серед спеціалістів. Тому на даний момент можна констатувати що цей фреймворк є чудовим інструментом для написання автотестів вебзастосунків.

2.2 Вибір мови програмування

Фреймворк підтримує такі мови програмування як JavaScript/TypeScript, Python, Java та C#. У цьому підрозділі буде розглянуто кожна з них та вибрано одну для реалізації автотестів.

JavaScript — це динамічно типізована мова програмування, яка є стандартом для роботи з вебдодатками. Вона виконується у браузері та на сервері через Node.js. Популярність мови робить її невіддільною частиною сучасної веброзробки.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має кілька властивостей, притаманних функціональним мовам, — функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання — що додає мові додаткову гнучкість. JavaScript має C-подібний синтаксис [14].

Особливості:

- гнучкість: JavaScript дозволяє швидко створювати інтерактивні вебінтерфейси;
- велика екосистема: Є безліч бібліотек і фреймворків що засновані на цій мові;
- асинхронне програмування: Дозволяє ефективно обробляти операції з браузером;
- легкість інтеграції: Ідеально підходить для тестування фронтенду через доступ до DOM.

Недоліки:

- відсутність суворої типізації може ускладнювати підтримку великих проєктів. Усувається модифікацією мови під назвою TypeScript.

Python — Це мова об'єктноорієнтованого програмування (ООП). Програмне забезпечення на Python оформлене у вигляді моделей, які можуть бути зібраними в пакеті. Тип та структуру кожного об'єкта можна запитувати під час виконання програми. Для кожного з об'єктів можна отримати всю інформацію про його внутрішньої структури. вибір того:

- у мови логічний синтаксис, завдяки чому вихідний код легко читати та розуміти;
- гнучкість та масштабованість Python дозволяє адаптувати високорівневу логіку та розширити складні застосунки, як тільки виникне така необхідність;
- розробка на Python у більшості випадків проходить швидше, ніж на інших об'єктно орієнтованих мовах програмування;

— Python це інтерпретована мова програмування. Що означає що код можна написати в будь-якому текстовому файлі на будь-якій платформі, а потім успішно запустити;

— у Python колосальна спільнота однодумців. Тож будь-які складнощі конкретних розробників можуть вирішуватись колективно [15].

Проте є кілька особливостей, які можна віднести до недоліків:

— повільність (ця мова програмування хоч і універсальна, проте повільніша за інші об'єктно орієнтовані мови програмування);

— велика кількість ресурсів, більшість для роботи та «прив'язаність» до системних бібліотек;

— менш ефективна для високонавантажених завдань у порівнянні з компільованими мовами;

— не така природна інтеграція з браузерними технологіями, як у JavaScript [15].

Java — це компільована об'єктно-орієнтована мова програмування, відома своєю стабільністю і використанням у корпоративному середовищі.

Широко застосовується для серверних і мобільних додатків.

Особливості:

— підходить для великих корпоративних систем;

— статична типізація допомагає уникати помилок у великих проєктах [16].

Недоліки:

— складність і громіздкість у порівнянні з іншими мовами;

— повільний процес написання коду через суворий синтаксис.

C# — це об'єктно-орієнтована мова програмування, розроблена Microsoft, яка є частиною екосистеми .NET. Підходить для створення десктопних, серверних і мобільних додатків.

Особливості:

— інтеграція з Windows: Підходить для тестування програм, розроблених під Microsoft-технології;

— асинхронне програмування через `async/await` дозволяє доволі ефективно працювати з браузерами.

Недоліки:

— основний акцент на Microsoft-екосистему, що може бути обмеженням для кросплатформених проєктів;

— складність і громіздкість у порівнянні з Python та JS [16].

У таблиці 2.2 приведено порівняння мов програмування доступних для розробки автотестівна цьому фреймворке.

Таблиця 2.2 — Порівняння мов програмування

Мова	JavaScript/ TypeScript	Python	Java	C#
Підтримка у фреймворку	Найкраща (рідна мова)	Підтримка через обгортку	Обмежена підтримка	Обмежена підтримка
Легкість у навчанні	Легка	Найлегша для початківців	Важка, вимагає знань та досвіду	Важка через специфіку .NET
Швидкість розробки	Висока	Висока, обмежена	Середня	Середня
Ефективність у тестах	Прямий доступ до DOM та API	Не пристосований для роботи з веб	Акцент на серверних додатках	Оптимізація під Microsoft
Швидкість тестів	Велика	Помірна	Середня	Середня
Інструментарій	Багато готових рішень	Середня	Акцент на серверних застосунках	Обмежена

Для реалізації програмної частини автотестів було вибрано TypeScript. Основні причини вибору це:

- JavaScript та TypeScript природно інтегрується з браузерами та має прямий доступ до DOM. Це спрощує створення тестів для вебінтерфейсів;
- Playwright створено з фокусом на JavaScript/TypeScript, що забезпечує повну підтримку функціоналу та найновіших можливостей фреймворка;
- гнучкість і швидкість: Асинхронні операції JavaScript дозволяють швидко й ефективно працювати з браузерами та виконувати паралельні тести;
- велика спільнота розробників, документація та інтеграція з іншими інструментами (CI/CD, репортери) роблять JavaScript кращим вибором;
- можливість використання TypeScript для статичної типізації забезпечує баланс між швидкістю розробки та масштабованістю [16].

2.3 Вибір середовища розробки

Playwright, як універсальний інструмент для автоматизації браузерів і end-to-end тестування, сумісний із багатьма середовищами розробки. Списком перелічені найбільш популярні середовища для роботи з ним:

Visual Studio Code Рекомендується як одне з найкращих середовищ завдяки відмінній підтримці JavaScript, TypeScript, відлагодження та наявності розширень для Playwright. Забезпечує функції автозаповнення коду, вбудований термінал та зручні можливості для дебагінгу. Також існують спеціальні розширення, як-от Playwright Test Runner, для покращення досвіду тестування.

JetBrains IDE WebStorm: підходить для розробки на JavaScript і TypeScript, пропонує інтегроване відлагодження та управління тестами.

Легкі редактори, такі як Atom та Notepad++ можуть бути використані, але мають суттєві недоліки, розглянемо їх детальніше.

Atom — легкий текстовий редактор з відкритим вихідним кодом, створений GitHub. Має потужну систему розширень, які можна використовувати для підтримки мов програмування, форматування коду та інтеграції із системами контролю версій (наприклад, Git). Також підтримує плагіни для роботи з JavaScript і TypeScript, які можуть бути корисними для роботи з Playwright.

Обмеження:

- відсутність повноцінного інтегрованого налагоджувача (debugger) робить налагодження складнішим;
- розширення для роботи з тестовими фреймворками менш розвинені, ніж у VS Code або JetBrains IDEs;
- Atom офіційно припинив отримувати оновлення у 2022 році, що може вплинути на його підтримку сучасних інструментів.

Atom може бути використаний для базового редагування коду та запуску Playwright через CLI, але для масштабних проєктів або складних тестів він менш зручний.

Загалом, вибір середовища залежить від ваших потреб і стилю роботи. Однак Visual Studio Code є найбільш рекомендованим через його інтеграцію з Playwright і підтримку TypeScript

Notepad++ це легкий та швидкий текстовий редактор для Windows, популярний серед новачків та тих, хто шукає мінімалістичні інструменти. Підтримує плагіни для роботи з різними мовами програмування, включаючи JavaScript.

Обмеження:

- не підтримує повноцінну інтеграцію з TypeScript чи Playwright, що ускладнює розробку складних сценаріїв;
- відсутність вбудованих засобів налагодження та тестування;

— не пропонує таких інструментів, як автозавершення для складних бібліотек чи інтеграція з CI/CD.

Notepad++ підходить лише для простих сценаріїв або для редагування коду, але не рекомендується як основне середовище для роботи з Playwright.

У таблиці 2.3 будуть порівняні середовища для розробки.

Таблиця 2.3 — Порівняння середовищ розробки

Параметр	VS Code	WebStorm	Atom	Notepad++
Підтримка мови	Відмінна	Відмінна	Середня	Середня
Інтеграція з фреймворком	Спеціалізовані	Можливість налаштування	Відсутня специфічна	Відсутня специфічна
Налагодження	+	+	-	-
Автозаповнення коду	+	+	-	-
Інтеграція з гіт	+	+	Через плагіни	-
Простота використання	Легка	Середня	Легка	Легка
Актуальність та підтримка	Широка	Широка	Підтримку припинено	Активна

Для розробки автотестів із Playwright Atom і Notepad++ можна використовувати лише для базового редагування коду. Однак, якщо потрібне продуктивне середовище з багатими функціями налагодження, автоматизації та інтеграції, краще віддати перевагу Visual Studio Code або іншим IDE з підтримкою Playwright.

WebStorm підходить для великих проєктів і командної роботи, але може бути складним для новачків через більшу складність інтерфейсу і потребу в налаштуваннях.

Visual Studio Code це найкращий вибір для більшості розробників, оскільки пропонує просте налаштування, потужні можливості, специфічну підтримку Playwright, і широкий вибір розширень.

Visual Studio Code – це редактор коду для різних мов програмування. Він відносно трохи важить, гнучкий та зручний. У ньому можна писати, форматовувати та редагувати код різними мовами [17].

Visual Studio Code не варто плутати з Visual Studio — це IDE, дуже потужна і масштабна, але водночас важка. Назви схожі, тому що в обох засобів розробки один і той же творець, але продукти різні за своєю суттю. VS Code менше важить, простіше в освоєнні і підходить у тому числі для розробників-початківців [17].

Редактор коду існує для всіх популярних операційних систем: Windows, Linux та MacOS. Він безкоштовний, на відміну більшості версій IDE Visual Studio [17].

З Visual Studio Code працюють програмісти різними мовами. Наприклад, ним активно користуються веброзробники, які пишуть HTML/CSS, JavaScript, PHP. Але редактор підтримує набагато більше мов: Python, Go, Ruby, C#, TypeScript тощо. Він працює і з розширеннями та фреймворками для популярних мов – наприклад, з React JS та Vue.js, з мовами стилів SCSS та LESS, які доповнюють CSS. Теоретично користуватися VS Code може розробник практично будь-якою з сучасних мов. Але на практиці його застосовують там, де не потрібні потужності повноцінної IDE [17].

VS Code дозволяє легко писати, форматовувати та редагувати код різними мовами. З його допомогою можна швидко створити проєкт та структуру файлів у ньому, він підсвічує синтаксис коду та допомагає автоматично правити помилки. У ньому є можливості для налагодження та

запуску коду деякими мовами. Редактор легко розширюється, тому до перерахованих функцій можна додати нові – досить просто завантажити потрібне доповнення з офіційного каталогу. Доповнення також розповсюджуються безкоштовно [17].

У Visual Studio Code також дуже легко встановити Playwright і розробники цього фреймворку використовують його за замовчуванням [18].

2.4 Висновки за розділом 2

У другому розділі пояснювальної записки було описано переваги і недоліки майже всіх сучасних фреймворків автоматизації тестування вебзастосунків, порівняно їх та обгрунтовано вибір фреймворку Playwright для реалізації програмної частини автотестів. Також було проведено вибір і обгрунтування мови програмування JavaScript/TypeScript. Було описано середовище розробки Visual Studio Code.

Обрані інструменти розробки дадуть можливість без суттєвих проблем швидко приступити до реалізації програмної частини не витрачаючи багато часу на навчання користуванням.

3 ПРОЄКТУВАННЯ ТА ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ

3.1 Тест-дизайн та розробка тест-кейсів

3.1.1 Тест-дизайн

Тест-дизайном називають процес розробки методів та технік тестування. Його мета – оптимізувати перевірку великого обсягу функцій вебзастосунку. Завдання – виявити максимальну кількість помилок за мінімальний час. Для цього спеціалісти складають докладну документацію з унікальними сценаріями тестування [21].

Тест-дизайн спирається на загальні принципи та логіку тестування та враховує особливості продукту. Він включає:

- оцінка обсягу даних;
- визначення найбільш сприятливих способів тестування;
- вибір тестових даних;
- опис критеріїв успішного проходження тестів.

Фахівці виділяють три основні етапи тестування: підготовка, тестування та аналіз результатів. Ми загостримо увагу на першому пункті у цьому розділі пояснювальної записки.

Підготовка — тестувальники працюють з наявною в документації інформацією про вебдодаток та шукають додаткові дані. На цьому етапі їм важливо максимально точно визначити потенційно проблемні галузі та функції продукту. Мета етапу — зрозуміти, за яких умов проводитиметься тестування, та коректно оцінити необхідний на перевірку час [21].

Для дипломної роботи достатньо скласти тести які відповідають потребам тестування основних (критичних) функцій сайту. Так як необхідний час був виділений у календарному плані.

3.1.2 Розробка тест-кейсів

В рамках проектування тест-дизайну дипломної роботи були розроблені тест-кейси функціональної частини вебсайту, що включають перевірку наступних функцій:

- логіна на сайт для користувачів;
- відображення помилки при невалідному логіні;
- перевірка чи видимі елементи сайту для користувачів;
- перевірка чи відкриваються елементи що за дизайном повинні бути клікабельними;
- перевірка чи правильні назви та ціни мають товари у магазині;
- перевірка чи правильну ціну мають товари при переході до оплати;
- перевірка чи коректно проходить процес замовлення товару у магазині;
- перевірка чи коректно відображається кількість товарів на візуальних елементах сайту;
- перевірка чи вводяться правильно дані у поля для вводу.

Вказані тести вирішено було створити і автоматизувати по причині того що перевірка основного (критичного) функціоналу для будь-якого інтернет-магазину.

Також ці тести будуть незмінні впродовж всього життєвого циклу розробки і тестування вебзастосунку та їх треба повторювати після кожного оновлення функціоналу сайту, це називається регресивним тестуванням.

Основні тести які вирішено реалізувати у програмному коді для перевірки функціоналу інтернет магазину вказані у таблиці 3.1.

Таблиця 3.1 — Перелік тест-кейсів позитивного тестування

№	Опис тесту	Дія користувача	Очікуваний результат
1	Тест позитивного логіну	Користувач вводить правильні дані в поля та натискає кнопку увійти	Сайт повинен увійти в аккаунт користувача
2	Тест чи клікабельна назва товару	Користувач натискає на назву товару чи картинку	Сайт повинен відобразити що відкрити подробиці про товар
3	Тест чи видні картинки товарів користувачу	Користувач дивиться на картинки кожного товару	Сайт повинен коректно відображати свою картинку для кожного товару
4	Тест видалення предмету з кошика	Користувач додає товар у кошик і видаляє його	Сайт повинен коректно додавати і видаляти товар з кошика
5	Перевірка назв та цін кожного товару	Користувач дивиться на назви та ціни товарів	Сайт повинен відображати коректні назви та ціни товарів
6	Тест покупки	Користувач додає товар у кошик і проходить всі кроки купівлі	Сайт повинен дати можливість користувачу купити товар

В таблиці 3.2 показаний перелік тестів негативного тестування, вирішених автоматизувати.

Таблиця 3.2 — Перелік тест-кейсів негативного тестування

№	Опис тесту	Дія користувача	Очікуваний результат
1	Тест логіну з некорректними даними	Користувач вводить неправильні дані в поля та натискає кнопку увійти	Сайт повинен відобразити помилку і не дати увійти
2	Тест логіну з некорректним логіном і коректним паролем	Користувач вводить неправильні дані в поле логіну і правильні у поле пароля, після чого натискає кнопку увійти	Сайт повинен відобразити помилку і не дати увійти
3	Тест логіну з коректним логіном і некорректним паролем	Користувач вводить правильний логін в поле та неправильний пароль, після чого натискає кнопку увійти	Сайт повинен відобразити помилку і не дати увійти
4	Тест покупки товару юзером з проблемним аккаунтом	Користувач додає товар у корзину і проходить всі кроки купівлі	Сайт повинен коректно додавати і видаляти товар з корзини

3.2 Розробка алгоритмів роботи автотестів

У цьому розділі будуть зазначені алгоритми спроектованих тестів.

З інтерфейса користувача можна буде запускати як оба тест-кейса одночасно так і окремі тести поодиночно. Основна властивість як тест-кейсів так і тестів — це відображати чи успішно вони пройшли чи провалені що й зазначено на рисунку 3.1.

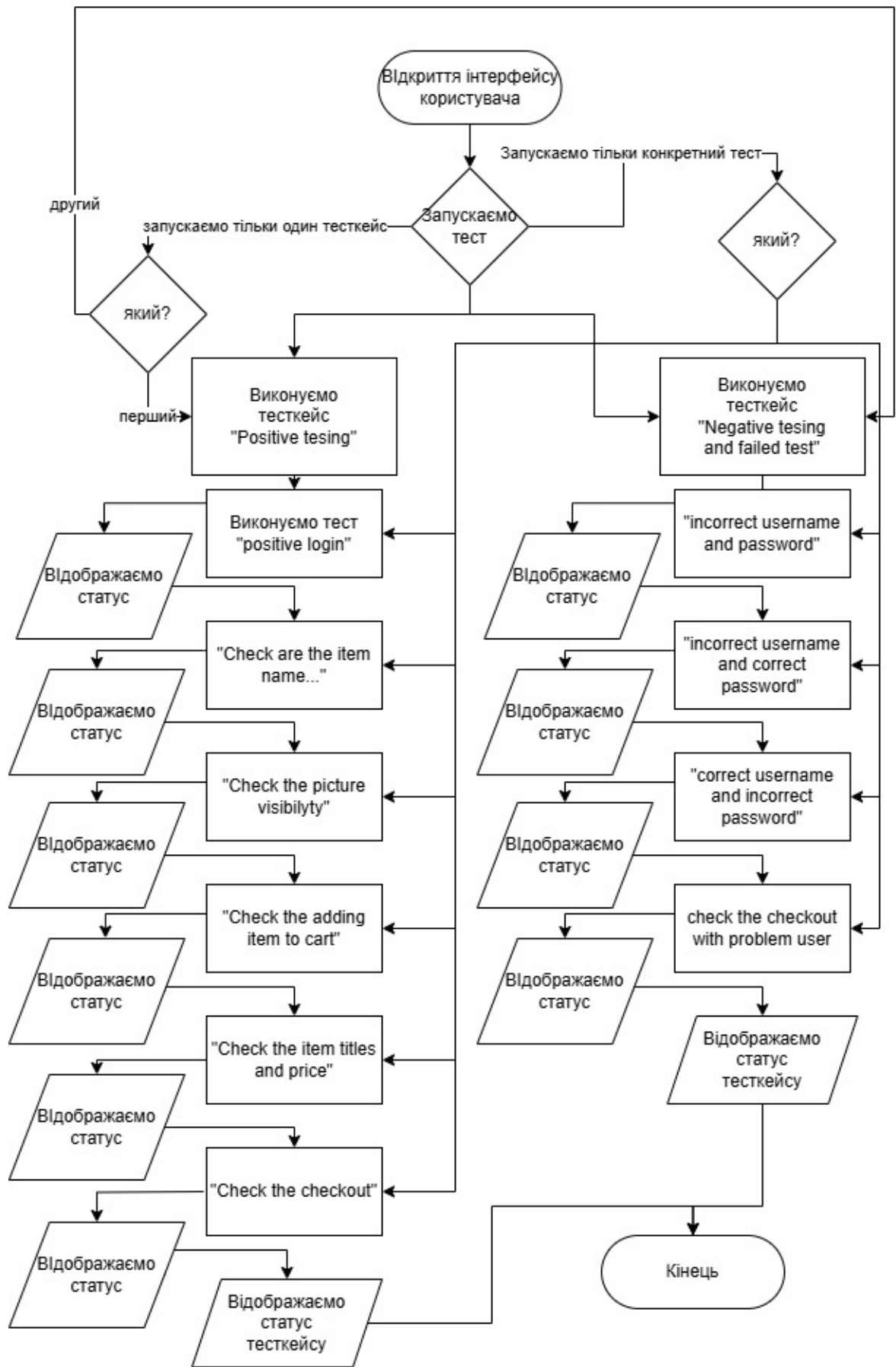


Рисунок 3.1 — Схема роботи програми автотестів

Далі показана більш конкретна схема роботи тест-кейсу, для прикладу взято другий. У другому тест-кейсі проводиться негативне, тобто з невалідними вхідними даними, тестування логіну на сайті.

Алгоритм роботи тест-кейсу показана на рисунку 3.2.

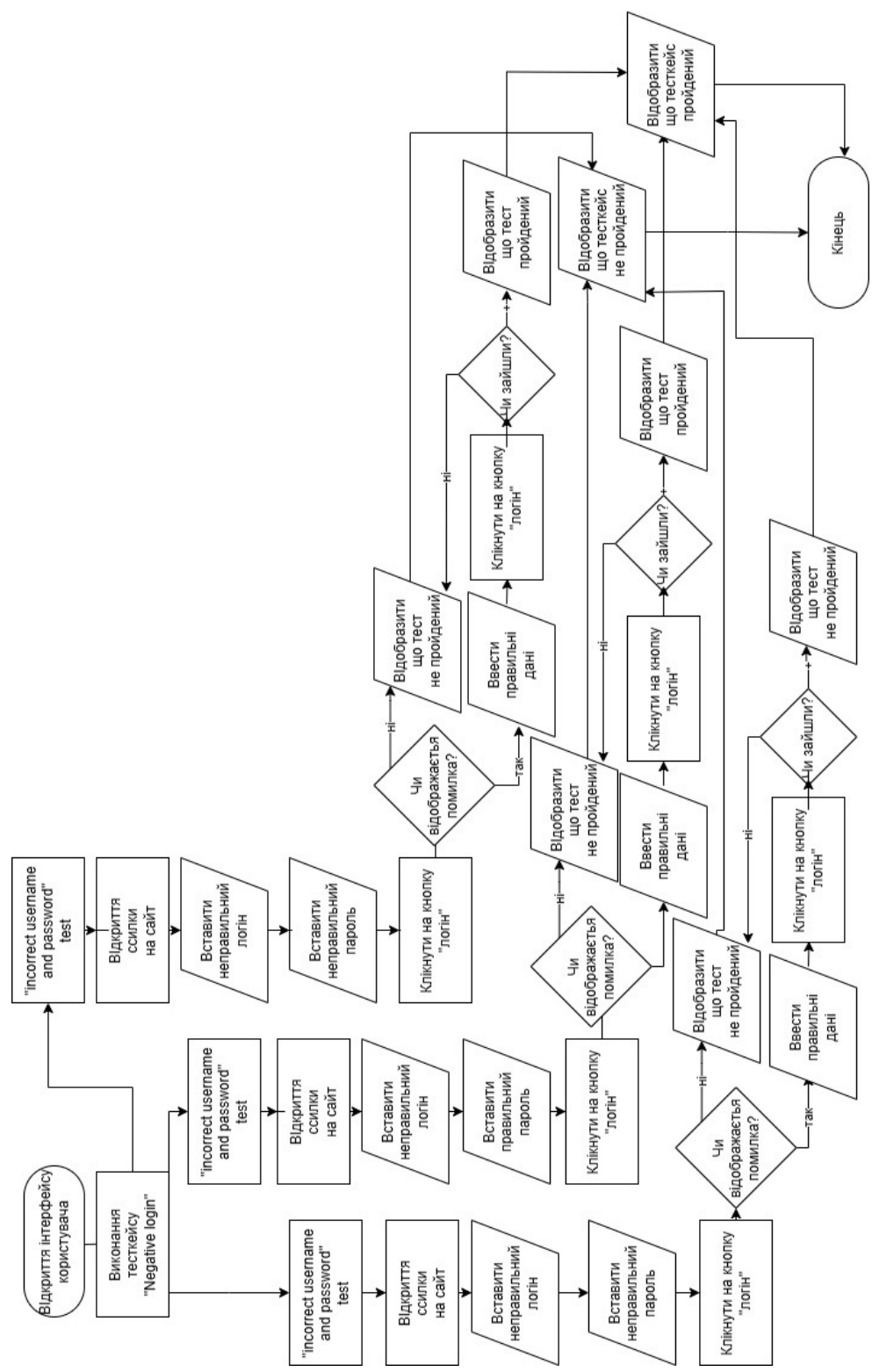


Рисунок 3.2 — Алгоритм роботи тест-кейсу негативного тестування логіну

Також було спроектовано тест покупки товару. Алгоритм показаний на рисунку 3.3.

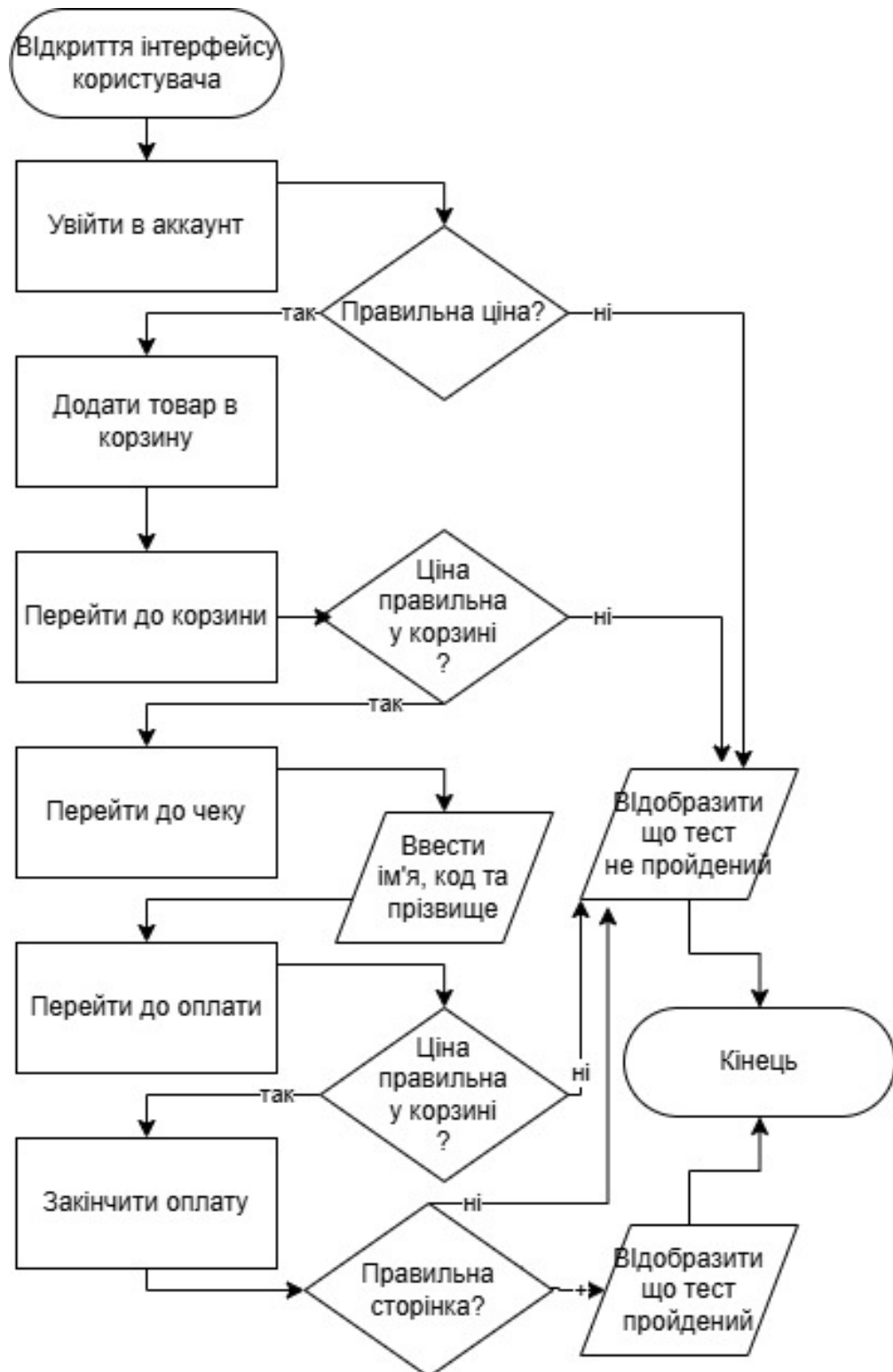


Рисунок 3.3 — Алгоритм роботи тесту покупки товару

3.3 Підготовка та налаштування середовища розробки

Для початку треба завантажити та встановити середовище розробки Visual Studio Code з офіційного сайту [19].

Сторінка показана на рисунку 3.4.

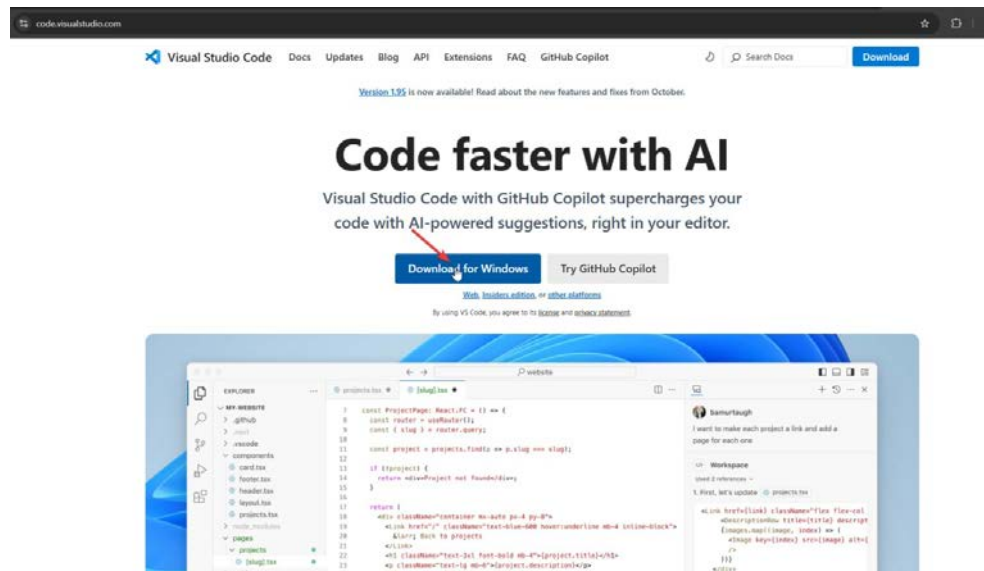


Рисунок 3.4 — Скачування середовища розробки

Краще встановлювати в стандартну директорію, це гарантує більш стабільну роботу та відсутність зайвих помилок. Шлях до директорії не повинен мати символів кирилиці.

На наступному кроці треба встановити бібліотеку Node.js для системи. Скачати його треба з офіційного сайту і уважно встановити [20].

Сторінка показана на рисунку 3.5.

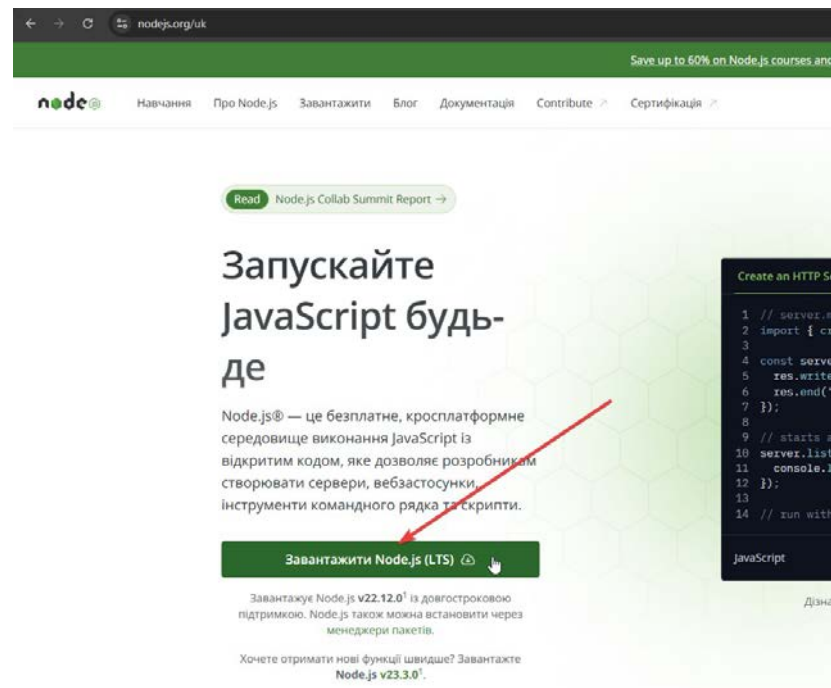


Рисунок 3.5 — Скачування бібліотеки Node.js

Після цього треба встановити розширення Playwright Test for VSCode [22].

Сторінка скачування продемонстрована на рисунку 3.6.

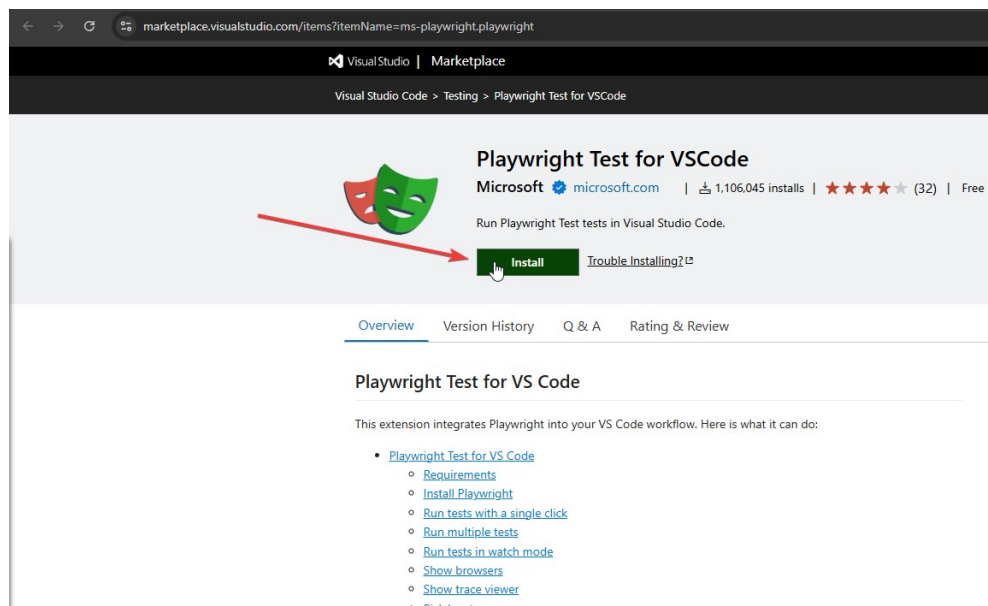


Рисунок 3.6 — Сторінка розширення фреймворку для середовища

Наступним кроком потрібно ввести команду “>Install Playwright” у поле пошуку Visual Studio Code як показано на рисунку 3.7.

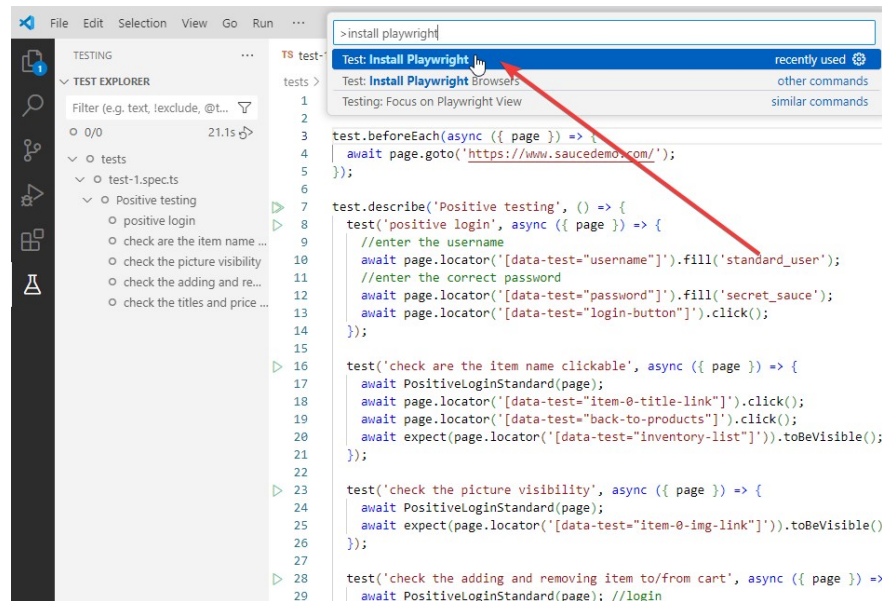


Рисунок 3.7 — Команда встановлення

Після натиснення на команду, контекстне меню запропонує вибрати браузерери котрі треба тестувати, у цьому випадку треба вибрати всі.

Контекстне меню продемонстроване на рисунку 3.8.

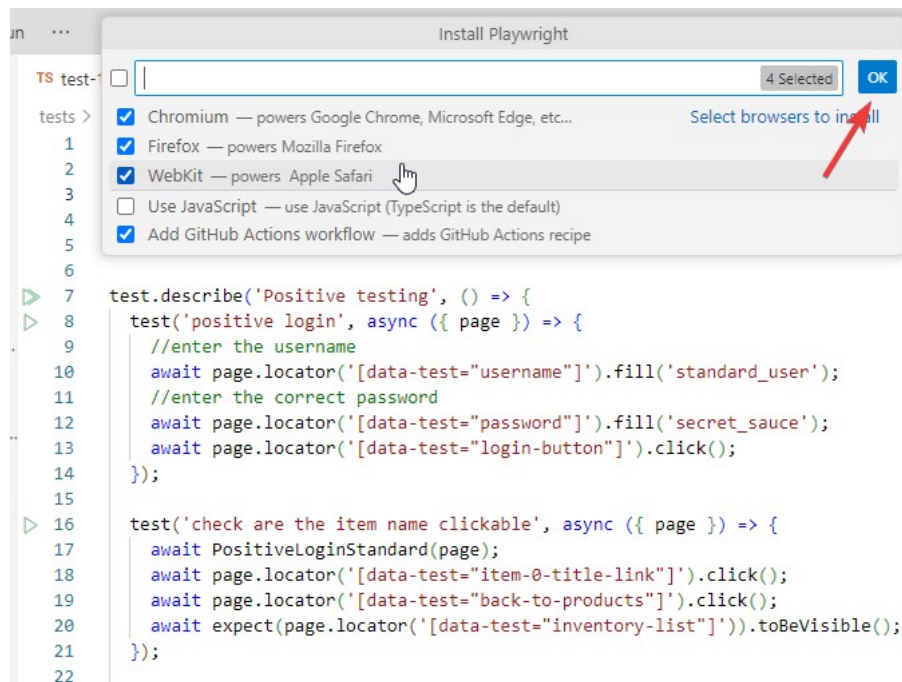


Рисунок 3.8 — Встановлення фреймворку

Далі необхідно створити новий файл тесту у директорії tests всередині директорії куди встановлений Playwright.

Тепер середовище і фреймворк повністю налаштовані і готові до роботи. Можна приступати до програмної реалізації тестів.

3.4 Розробка програмної частини тестів

3.4.1 Тестовий вебзастосунок для реалізації автотестів

Для того щоб розробити автоматичні тести треба мати вебзастосунок, котрий буде тестуватися. Для тестування я обрав простий тестовий вебзастосунок від компанії SauceLabs [23].

На рисунку 3.9 показана сторінка логіна тестового сайту.

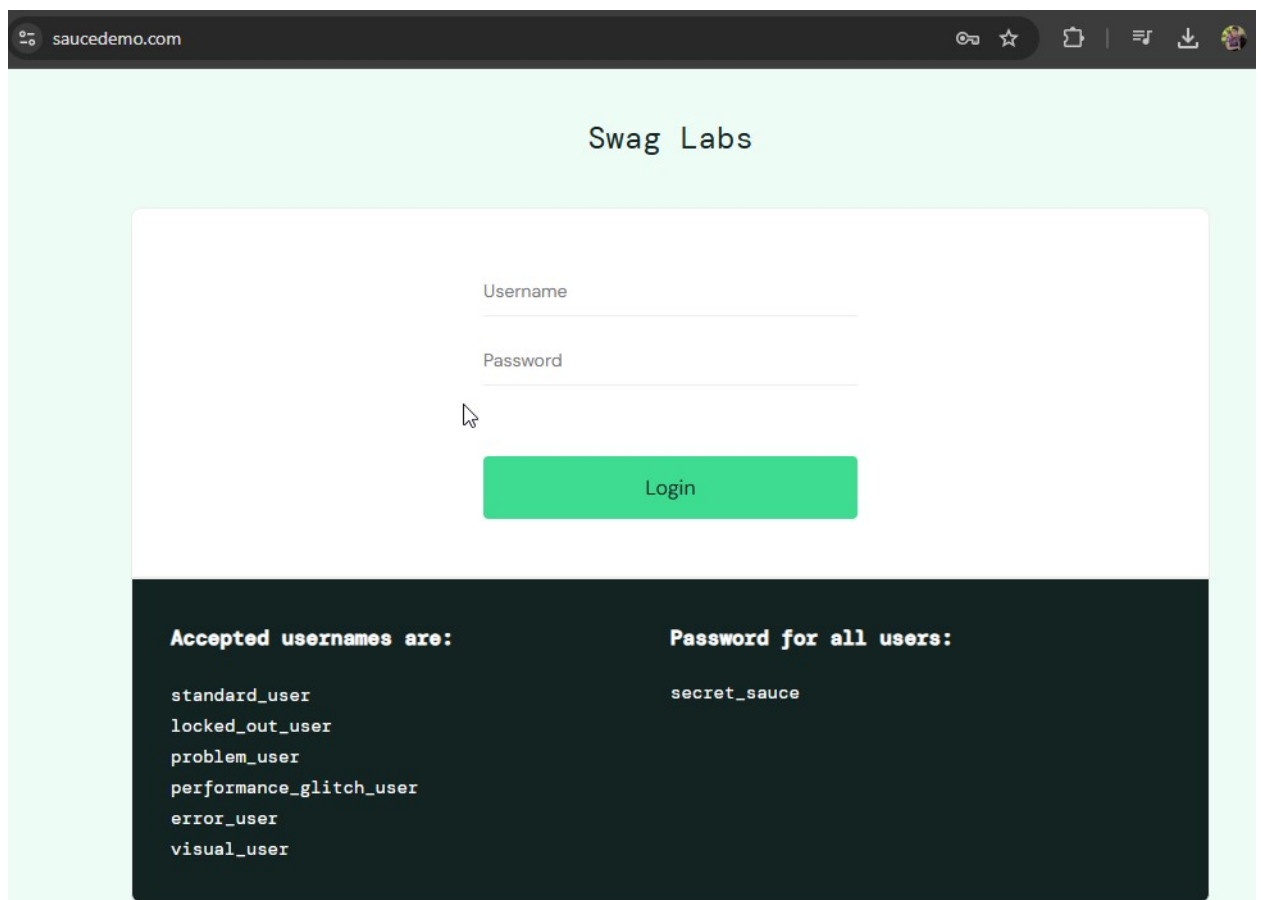


Рисунок 3.9 — Сторінка логіна тестового сайту

На рисунку 3.10 показана сторінка тестового магазину після логіну.

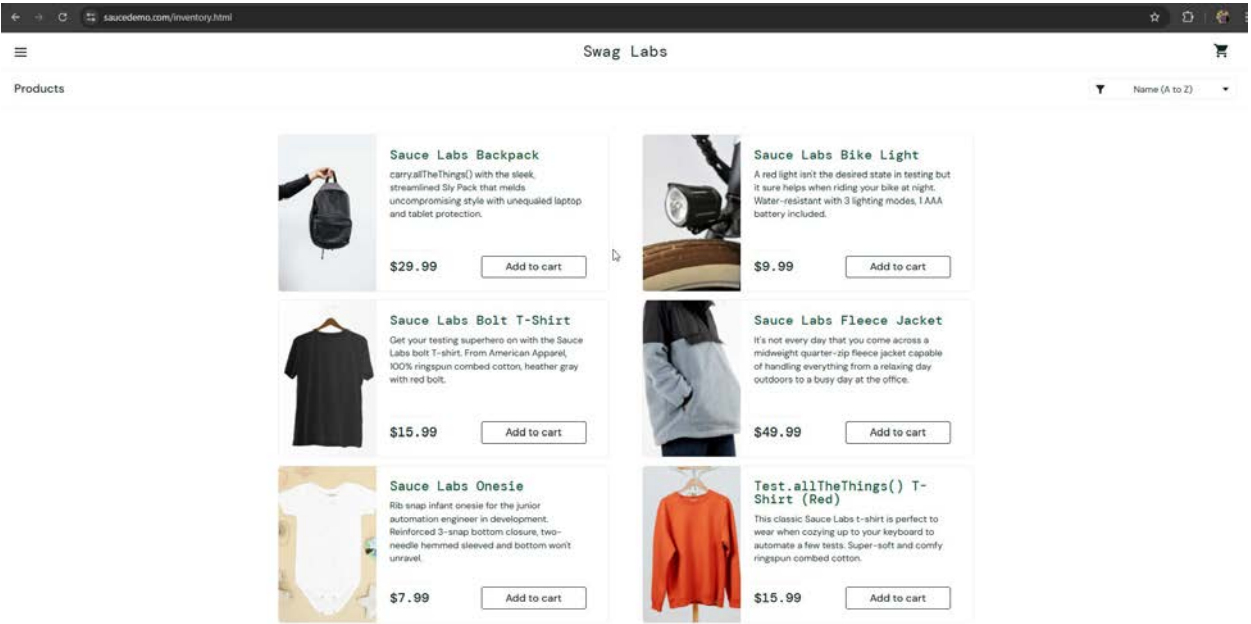


Рисунок 3.10 — Сторінка тестового магазину

3.4.2 Написання коду

Після обрання тестового вебзастосунку можна починати реалізувати програмну частину у середовищі Visual Studio Code.

Для початку треба імпортувати бібліотеку, після чого ініціювати глобальну змінну `page` що тримає у собі посилання на сайт, код що відповідає за це показаний на рисунку 3.11.

```

TS test-1.spec.ts ×
tests > TS test-1.spec.ts > ...
1  import { test, expect } from '@playwright/test';
2
3  test.beforeEach(async ({ page }) => {
4    |  await page.goto('https://www.saucedemo.com/');
5  });

```

Рисунок 3.11 — Глобальна змінна `page`

Далі необхідно створити структуру що нагадує тест-кейс, що буде у собі мати безпосередньо тести, тести як і тест-кейс мають назву для того щоб легко відображатись у інтерфейсі користувача.

Перший тест буде містити у собі просту перевірку можливості логіну на сайт за допомогою коректно введених даних, можна звернути увагу що назва тест-кейсу і тесту відображається у структурі вкладки «Testing», що з'явилася у середовищі після встановки фреймворку.

Код з тест-кейсом «Positive testing», тестом «positive login» та вкладкою «Testing» зображений на рисунку 3.12.

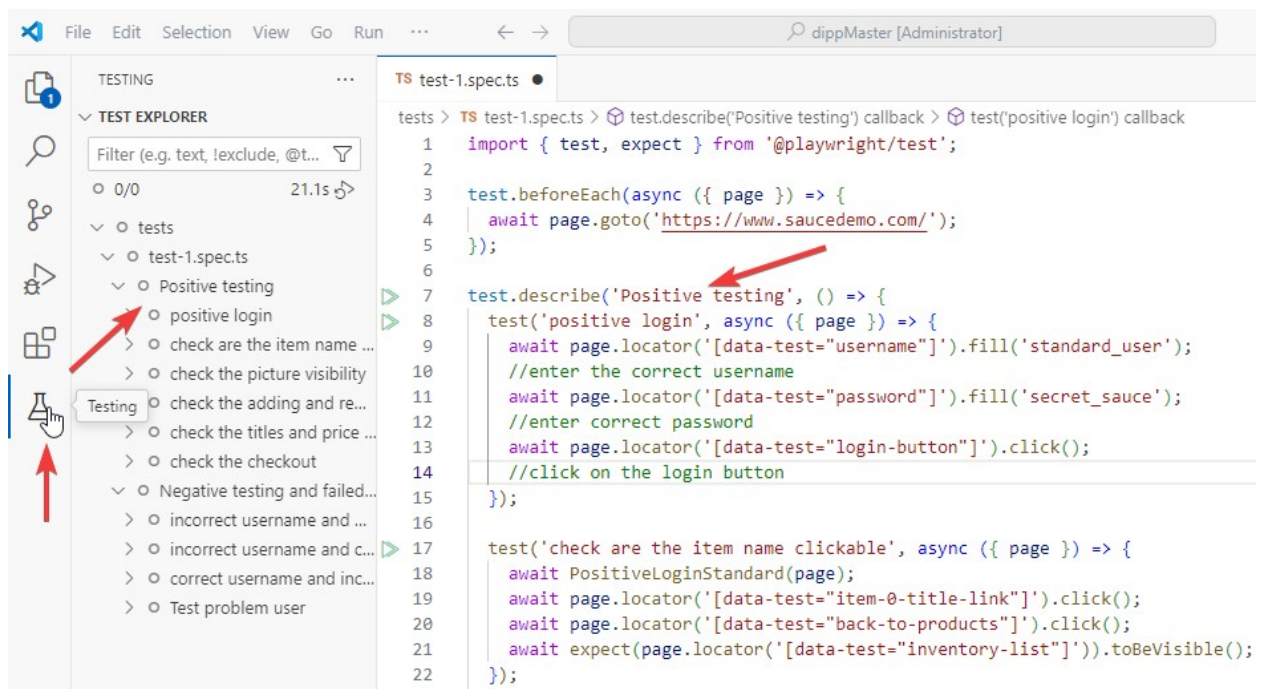


Рисунок 3.12 — Тест-кейс, тест та їх відображення у вкладці «Testing»

Для того щоб можна було звертатись до елементів сайту у коді треба подивитись на назви цих елементів у інструментах розробника браузера.

На рисунку 3.13 показана назва поля ім'я користувача по котрому програма звертається до нього на 9 строці.

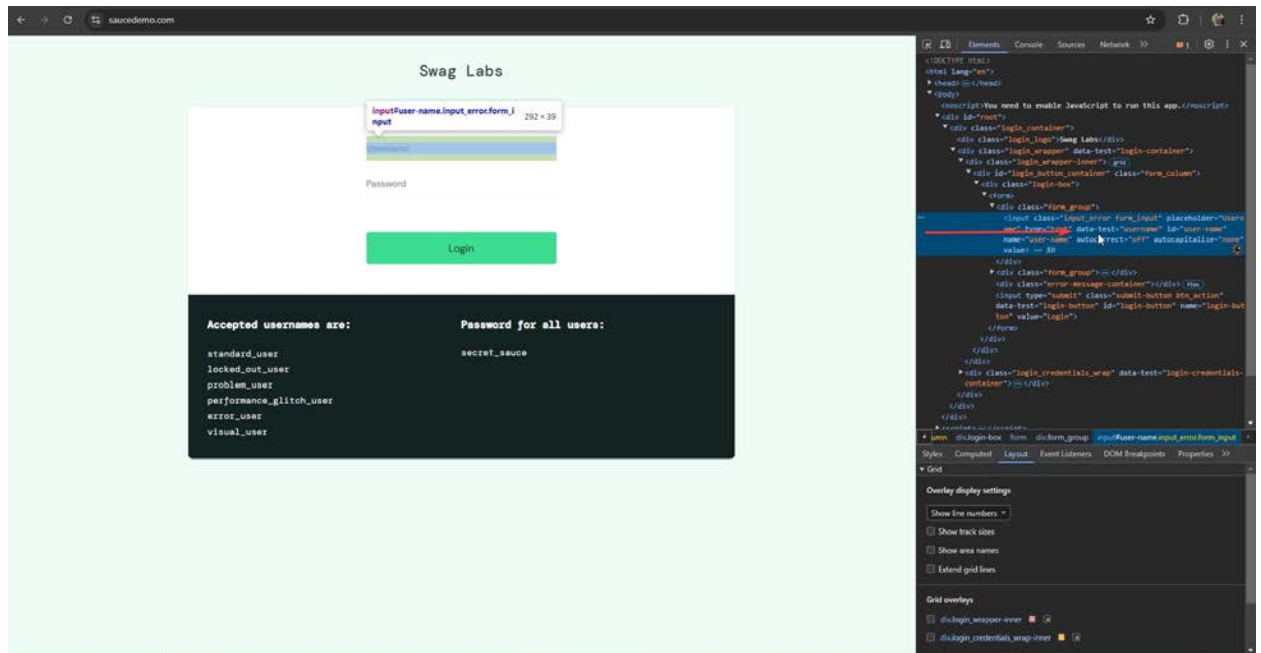


Рисунок 3.13 — Назва елементу у інструментах розробника Chrome

Так як тести проходять всі з нульового початку, тобто з запуску пустого браузера, для зручності треба написати функцію для швидкого логіну яку можна буде використовувати у кожному тесті. Додана функція `fill()` бере і вписує вказаний текст до елементу котрий вказаний локатором.

Функція логіну показана на рисунку 3.14.

```

209
210 async function PositiveLoginStandard(page) {
211   //enter the username
212   await page.locator('[data-test="username"]').fill('standard_user');
213   //enter the correct password
214   await page.locator('[data-test="password"]').fill('secret_sauce');
215   await page.locator('[data-test="login-button"]').click();
216 }
217
218

```

Рисунок 3.14 — Функція логіну

Для того щоб викликати функцію треба всього лише написати її ім'я та передати в якості аргументу глобальну змінну `page`, що виконує перехід по посиланню на сайт.

Також опишемо тест що має перевірити чи видаляються товари з корзини і чи коректно після цього користувач переходить знову до покупок.

- спочатку проходить стандартний логін на сайт;
- натискає на кнопку додавання товару до корзини;
- перевіряє чи іконка поряд з корзиною відображає 1 товар;
- клікає на корзину для того щоб перейти у корзину;
- натискає на кнопку для видалення товару з корзини;
- натискає на кнопку для продовження покупок;
- перевіряє чи є заголовок, котрий є тільки на тій сторінці котру ми очікуємо побачити.

Додана функція ToContainText до локатору перевіряє чи є заданий текст у елементі, і якщо є то тест проходить до виконання наступного кроку.

Варто зазначити що невиконання хоч одного з кроку моментально прерве тест і поверне його статус як провалений, тож наступні кроки не будуть виконуватись. Тож якщо хоч одна функція не знайде елементу з локатором який у ній заявлений – тест автоматично провалиться.

Код тесту видалення товару показаний на рисунку 3.15.

```

28
29 test('check the adding and removing item to/from cart', async ({ page }) => {
30     await PositiveLoginStandard(page); //login
31     await page.locator('[data-test="add-to-cart-sauce-labs-bike-light"]').click(); //add to cart
32     await expect(page.locator('[data-test="shopping-cart-badge"]').toContainText('1'));
33     //are cart display 1 item icon?
34     await page.locator('[data-test="shopping-cart-link"]').click();
35     //go to cart
36     await page.locator('[data-test="remove-sauce-labs-bike-light"]').click();
37     //remove
38     await page.locator('[data-test="continue-shopping"]').click();
39     //go back to the products tab
40     await expect(page.locator('[data-test="inventory-list"]').toBeVisible());
41     //are we in corr tab?
42 });
43

```

Рисунок 3.15 — Тест видалення товару з корзини

Ще важливий тест це перевірка назв всіх товарів та їх ціни. Код цього тесту приведений на рисунку 3.16.

```

43
44 test('check the titles and price of items', async ({ page }) => {
45   await PositiveLoginStandard(page);
46   //login
47   await expect(page.locator('[data-test="item-4-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Backpack');
48   //are the Sauce Labs Backpack item has correct title?
49   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$29.99');
50   //and correct price?
51   await expect(page.locator('[data-test="item-0-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Bike Light');
52   //are the Sauce Labs Bike Light item has correct title?
53   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$9.99');
54   //and correct price?
55   await expect(page.locator('[data-test="item-1-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Bolt T-Shirt');
56   //are the Sauce Labs Bolt T-Shirt has correct title?
57   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$15.99');
58   //and correct price?
59   await expect(page.locator('[data-test="item-5-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Fleece Jacket');
60   //are the Sauce Labs Fleece Jacket item has correct title??
61   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$49.99');
62   //and correct price?
63   await expect(page.locator('[data-test="item-2-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Onesie');
64   //are the Sauce Labs Onesie item has correct title?
65   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$7.99');
66   //and correct price?
67   await expect(page.locator('[data-test="item-3-title-link"] [data-test="inventory-item-name"]')).toContainText('Sauce Labs Red T-Shirt');
68   //are the Sauce Labs Red T-Shirt item has correct title?
69   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$15.99');
70   //and correct price?
71 });
72

```

Рисунок 3.16 — Тест назв всіх товарів і їх цін

Код тесту покупки товару, схема котрого була показана на рисунку 3.3, показаний на рисунку 3.17.

```

72
73 test('check the checkout', async ({ page }) => {
74   await PositiveLoginStandard(page);
75   // login
76   await expect(page.locator('[data-test="inventory-list"]')).toContainText('$15.99');
77   //check the price of item in catalog
78   await page.locator('[data-test="add-to-cart-test\\allthethings\\(\\)\\-t-shirt-\\(red\\)"]').click();
79   //add item to cart
80   await page.locator('[data-test="shopping-cart-link"]').click();
81   //go to cart
82   await expect(page.locator('[data-test="inventory-item-price"]')).toContainText('$15.99');
83   //check the price in the cart
84   await page.locator('[data-test="checkout"]').click();
85   //go to the checkout page
86   await page.locator('[data-test="firstName"]').fill('testfirst1');
87   //enter user's first name
88   await page.locator('[data-test="lastName"]').fill('testlast1');
89   //enter user's last name
90   await page.locator('[data-test="postalCode"]').fill('69069');
91   //enter user's postcode
92   await page.locator('[data-test="continue"]').click();
93   //go to the payment
94   await expect(page.locator('[data-test="title"]')).toContainText('Checkout: Overview');
95   //check are we on the expected page?
96   await expect(page.locator('[data-test="inventory-item-price"]')).toContainText('$15.99');
97   //check the item price
98   await expect(page.locator('[data-test="subtotal-label"]')).toContainText('Item total: $15.99');
99   //check the item price in checkout before taxes
100  await page.locator('[data-test="finish"]').click();
101  //click finish
102  await expect(page.locator('[data-test="complete-header"]')).toContainText('Thank you for your order!');
103  //check are we on the expected page?
104  });
105  });
106

```

Рисунок 3.17 — Тест покупки товару

Також варто розглянути один з негативних тестів тест-кейсу що був показаний на рисунку 3.2, там використовується функція `toBeVisible` що перевіряє чи видимий для користувача елемент що зазначений локатором.

Основна задача цього тесту це ввести неправильні дані у поля логіна і пароля, перевірити чи видна помилка і чи правильний текст вона має у собі.

Після чого він вводить правильні дані щоб перевірити чи правильно логін працює після неправильної попитки.

Тест показаний на рисунку 3.18.

```

107 test.describe('Negative testing and failed tests', () => {
108   test('incorrect username and password data login', async ({ page }) => {
109     await page.locator('[data-test="username"]').fill('standard_user1');
110     //enter the incorrect username
111     await page.locator('[data-test="password"]').fill('secret_sauce1');
112     //enter the incorrect password
113     await page.locator('[data-test="login-button"]').click();
114     //click on the login button
115     await expect(page.locator('[data-test="error"]')).toBeVisible();
116     //check the visibility of error
117     await expect(page.locator('[data-test="error"]')).toContainText('Epic sadface: Username and password
118     //check the error text
119     await page.locator('[data-test="password"]').fill('secret_sauce');
120     //enter correct password
121     await page.locator('[data-test="username"]').fill('standard_user');
122     //enter the correct username
123     await page.locator('[data-test="login-button"]').click();
124     //check the correct login after incorrect
125     await expect(page.locator('[data-test="title"]')).toContainText('Products');
126     //are we login succesfully?
127   });
128 }

```

Рисунок 3.18 — Негативний тест логіну та помилки

3.5 Висновки за розділом 3

У третьому розділі було спроектовано основні функції тестів, пройдений етап встановлення необхідного ПЗ та налаштування середовища розробки Visual Studio Code. Було обрано тестовий сайт від компанії SauceLabs. Було створено і описано код що виконує поставлені задачі по тестуванню основних функцій вебзастосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Умови застосування

Програма тестує сайт saucedemo.com на трьох основних браузерях, Google Chrome, Firefox, Safari(Webkit) [23].

Перевіряються основні, критичні функції роботи вебмагазину:

- логін;
- додання та видалення товару до кошика;
- купівлю товару;
- відображення картинок, назв та ціни товарів;
- зміни інтерфейсу від функціональних кнопок.

Автотести для запуску потребують ПК з наступними характеристиками та ПЗ:

- ОС Windows 10/11 (64-bit);
- процесор з тактовою частотою не менше за 2.5 ГГц;
- наявність оперативної пам'яті не менше 8 Гб;
- місце на диску не менше за 100 Мб;
- Visual Studio Code останньої версії;
- .NodeJS останньої версії;
- наявність екрану;
- наявність пристроїв типу клавіатура і миша.

4.2 Умови передачі автотестів

Автотести передаються безкоштовно, може поширюватись через хмарні технології та любых способів передачі файлів.

Достатньо передати файл тестів формату .ts та розташувати його у директорію test з встановленим фреймворком Playwright.

Вміст директорії з встановленим фреймворком показаний на рисунку 4.1.

Name	Date modified	Type
.github	06.12.2024 20:50	File folder
node_modules	06.12.2024 20:50	File folder
playwright-report	10.12.2024 16:11	File folder
test-results	10.12.2024 16:11	File folder
tests	09.12.2024 22:51	File folder
tests-examples	09.12.2024 22:51	File folder
.gitignore	06.12.2024 20:50	Git Ignore Source
package	06.12.2024 20:50	JSON Source File
package-lock	06.12.2024 20:50	JSON Source File
playwright.config	06.12.2024 20:50	JavaScript File

Рисунок 4.1 — Вміст директорії з фреймворком

Після цього треба відкрити файл у середовищі розробки Vsdual Studio Code та почати розробку/зміни.

4.3 Характеристики програмної частини автотестів

Тести написані на мові програмування TypeScript що є модифікацією мови JavaScript з використанням фреймворку Playwright версії v1.49.1 для запуску тестів і використання вбудованого інтерфейсу користувача. Проєкт створено у середовищі розробки Visual Studio Code версії 1.95.3.

4.4 Інструкція з використання інтерфейсу користувача

Для того щоб відкрити інтерфейс користувача треба відкрити термінал у середовищі розробки та вписати команду показану на рисунку 4.2.

```
PS C:\Users\anddx\Documents\dippMaster> npx playwright test --ui
PS C:\Users\anddx\Documents\dippMaster> npx playwright test --ui
PS C:\Users\anddx\Documents\dippMaster> npx playwright test --ui
PS C:\Users\anddx\Documents\dippMaster> npx playwright test --ui
```

Рисунок 4.2 — Команда для вмикання вбудованого інтерфейсу користувача

Після її виконання відкриється вікно з інтефейсом користувача для дебагінгу Playwright. Вікно показане на рисунку 4.3.

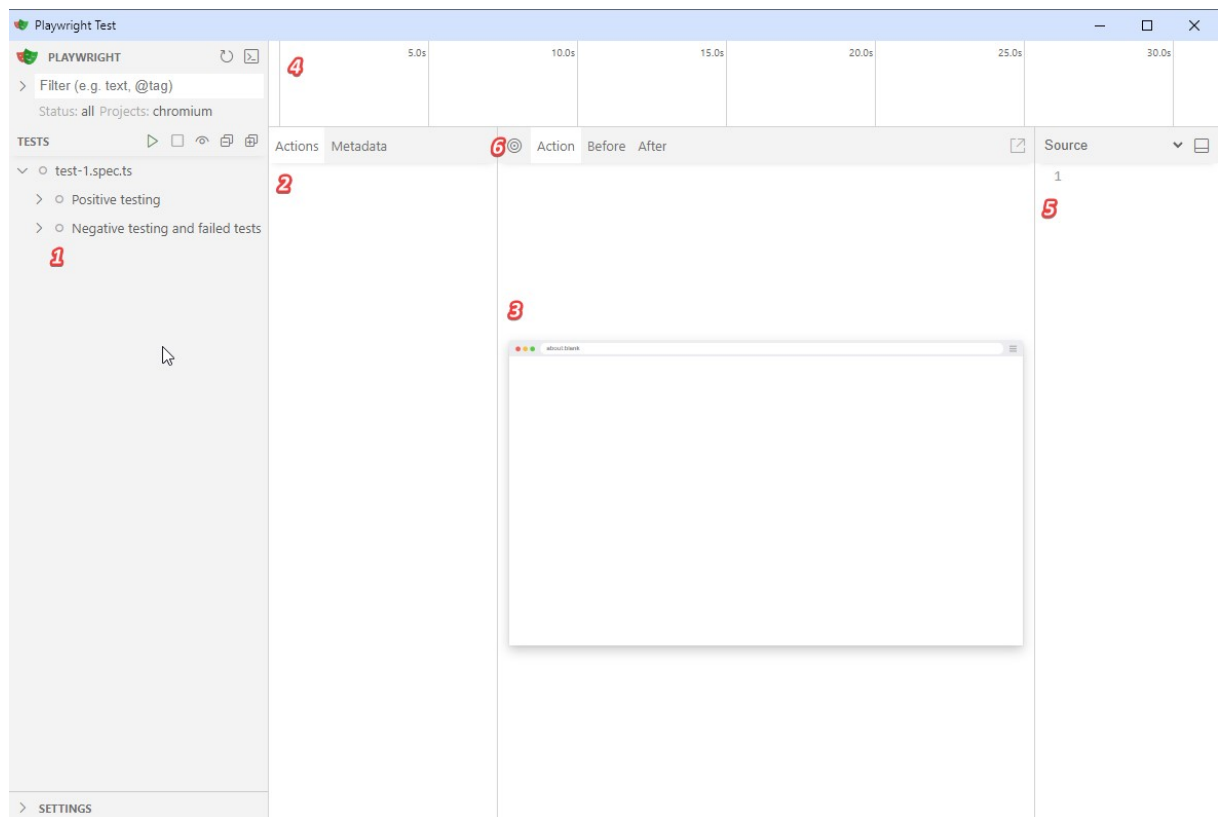


Рисунок 4.3 — Вікно інтерфейсу користувача після відкриття

Розглянемо по порядку елементи що вказано на скріншоті.

- список тест-кейсів та тестів у файлі тестів;
- упорядкований список команд у кожному тесті;
- віконце для відображення дій у браузері під час виконання тестів;
- посекундний список з скріншотами вкладки браузера;
- віконце з кодом яке співпадає з виділеним тестом у вікні №2;
- утіліта для визначення ім'я елемента на сайті, за яким до нього можна звернутись у коді.

Поряд з списком тест-кейсів є кружки для відображення статусу виконання тестів, вони можуть бути:

- сірими (не виконувались);
- зеленими з галочкою (успішно пройшли);
- червоні (провалені).

Розгорнутий список тест-кейсу «Negative testing and failed tests» з всіма статусами можна побачити на рисунку 4.4.

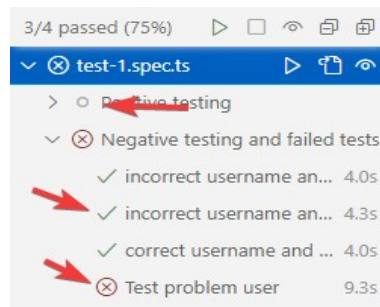


Рисунок 4.4 — Список тест-кейсу з статусами виконання

При натисненні на тест відкриється вся докладна інформація про тест. Розберем більш ретельно кожен з елементів, почнемо з деталей дій тесту.

Список дій тесту невалідних даних у логіні показаний на рисунку 4.5.

Actions	Metadata
✓ Passed	4.1s
▼ Before Hooks	3.5s
▼ beforeEach hook	3.5s
> fixture: browser	165ms
> fixture: context	44ms
> fixture: page	209ms
page.goto https://www.saucedemo.com/	3.0s
locator.fill locator("[data-test='username']")	45ms
locator.fill locator("[data-test='password']")	39ms
locator.click locator("[data-test='login-button']")	78ms
expect.toBeVisible locator("[data-test='error']")	26ms
expect.toContainText locator("[data-test='error...")	27ms
locator.fill locator("[data-test='password']")	26ms
locator.fill locator("[data-test='username']")	30ms
locator.click locator("[data-test='login-button...")	193ms
expect.toContainText locator("[data-test='title']")	13ms
▼ After Hooks	173ms
fixture: page	0ms
fixture: context	0ms

Рисунок 4.5 — Меню дій тесту невалідних даних у логіні

У цьому списку детально показана кожна дія з можливістю натиснення для додаткових деталей у інших блоках. Також є такі дані:

- назначені дії до початку дій тесту;
- статус або дії після кінця тесту;
- час виконання або чекання для виконання кожної дії;
- статус виконання всього тесту;
- час виконання всього тесту разом.

Наступне це меню відображення вікна браузера на котрому можна побачити яка дія що робить у фіксований момент.

На рисунку 4.6 зображена перевірка чи співпадає текст помилки з записаним у тесті.

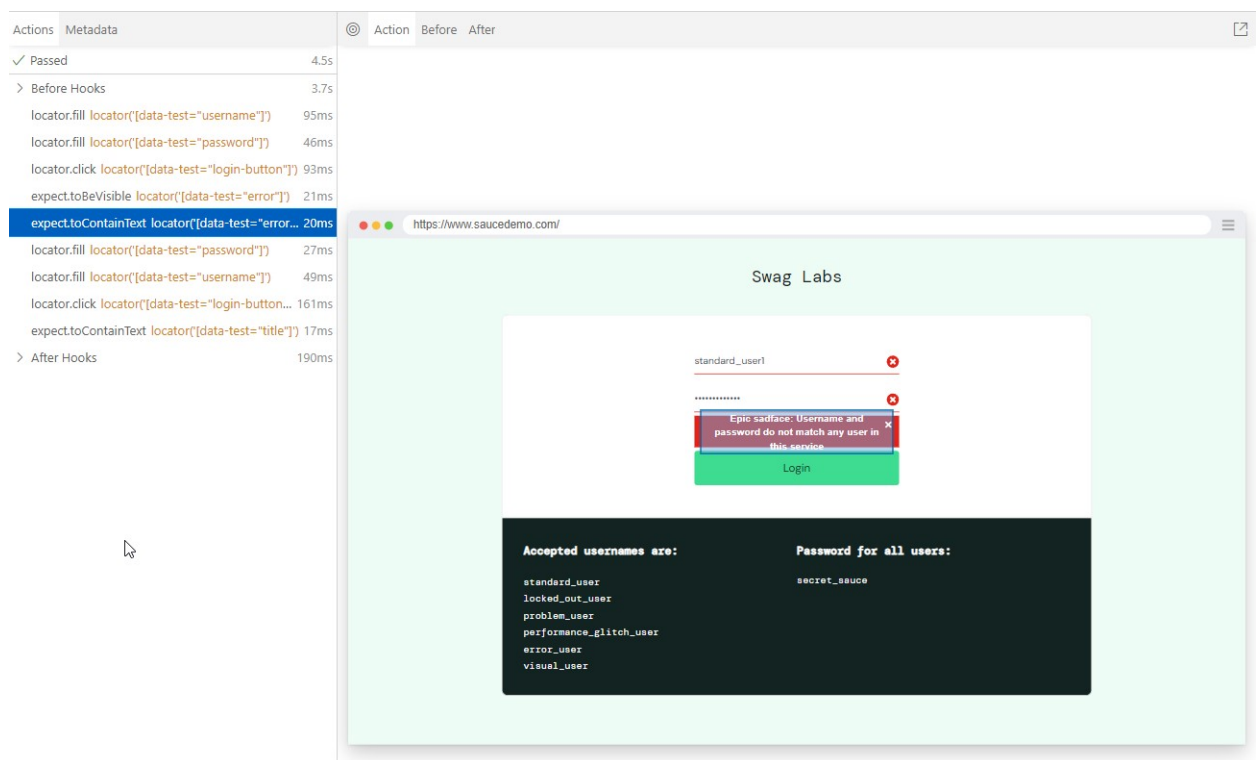


Рисунок 4.6 — Перевірка співпадіння тексту у помилці

Також у сусідньому блоці показана строчка коду на якій виконується перевірка, це 117 строка, інтерфейс одразу виділяє її. Також є кнопка «Open in VS Code», що відкрив цю строку у середовищі. Це можна побачити на рисунку 4.7.

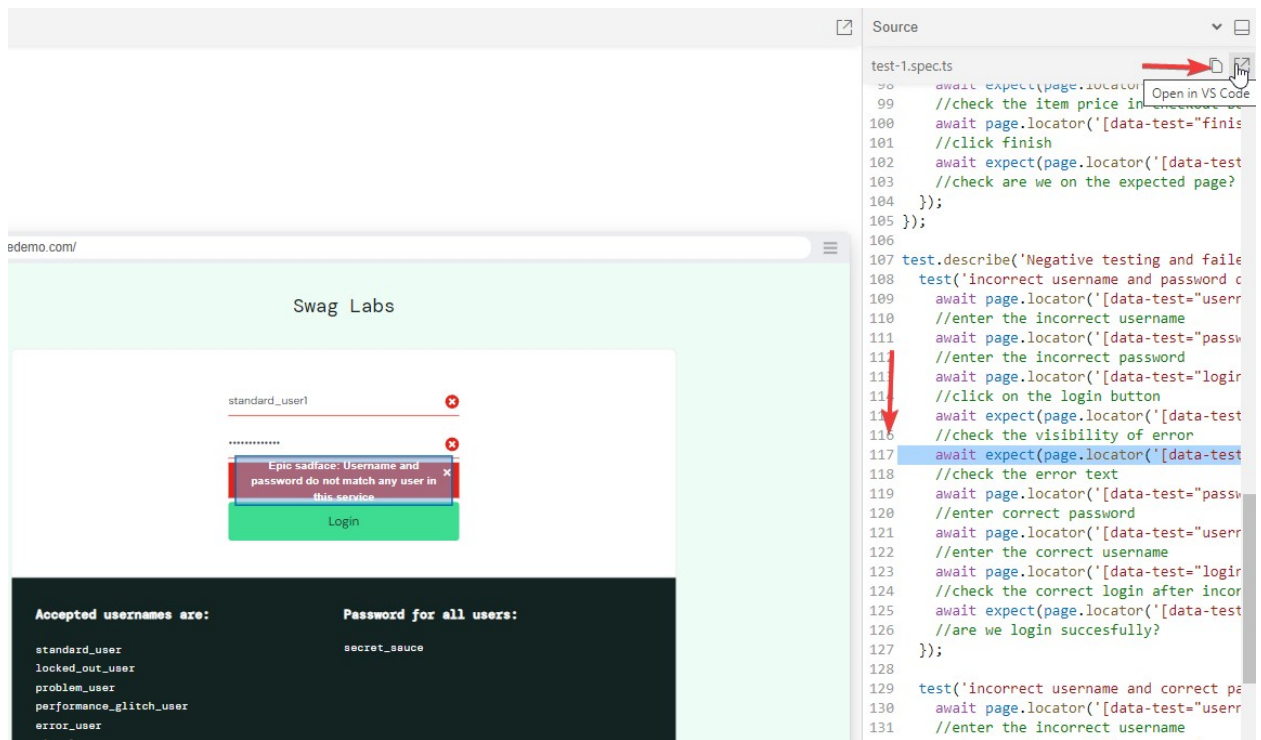


Рисунок 4.7 — Меню джерела виконання тесту

Наступним є меню посекундного виконання тесту, на якому можна приблизити при наведенні курсору на конкретний час екран. Це показано на рисунку 4.8.

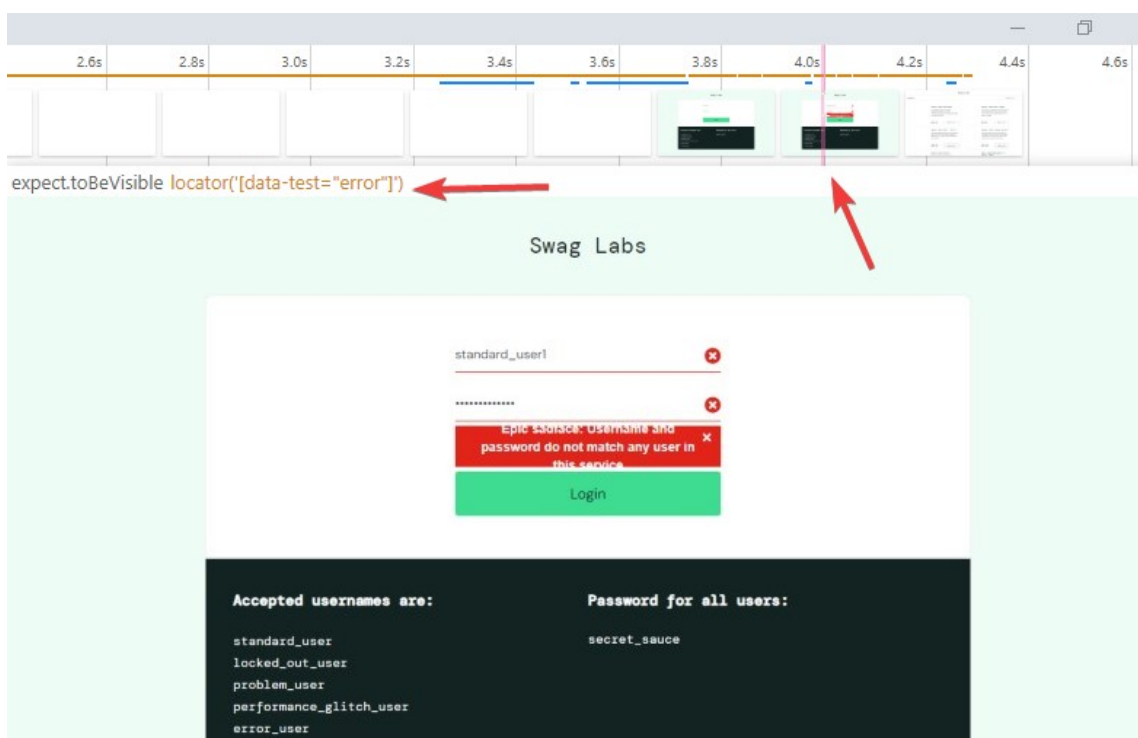


Рисунок 4.8 — Кадр перевірки чи видима для користувача помилка

Далі розглянемо утиліту для копіювання локатору з об'єкту відкритому в конкретний момент на екрані виконання.

На рисунку 4.9 показаний функціонал кнопки «Pick locator».

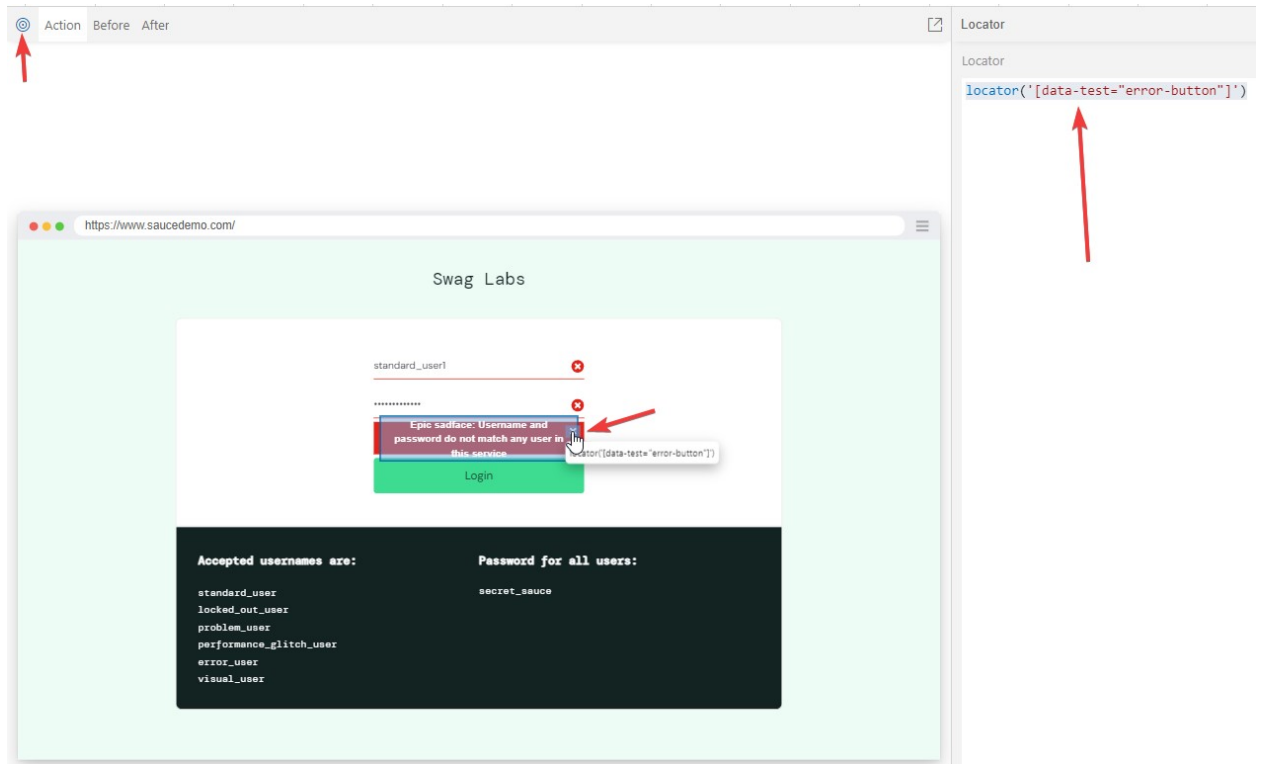


Рисунок 4.9 — Можливість скопіювати локатор кожного елементу на сайті

Також у меню «Call» можна подивитись деталі конкретно вибраної дії тесту, такі як:

- час на виконання;
- більш детальні параметри;
- значення що отримує програма після виконання команди.

Меню «Call» показано на рисунку 4.10.

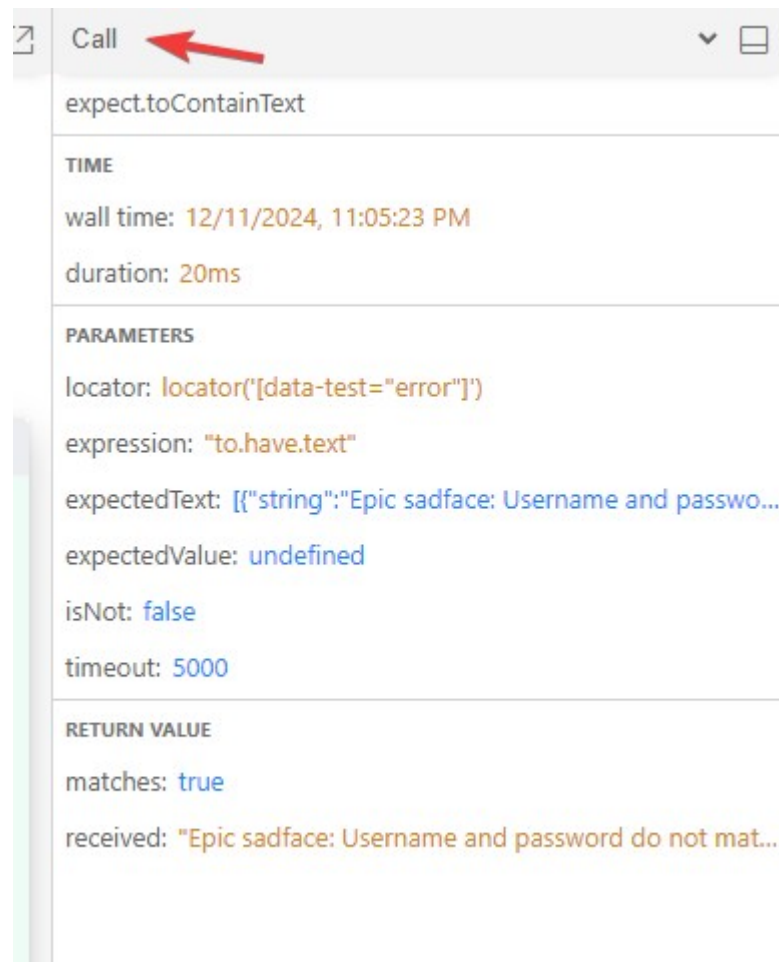


Рисунок 4.10 — Меню «Call» у інтерфейсі користувача

В даному випадку команда очікувала і отримала текст «Epic sadface: Username and password do not match...» за 20 мс у 11 годин вечора 12 числа.

4.5 Тестування автотестів у інтерфейсі користувача

У ході роботи автотестів було проведено спостереження та тестування правильності виконання і очікуваних вихідних даних.

Для початку давайте подивимось на результати виконання кожного тесту. Результати показані на рисунку 4.11.

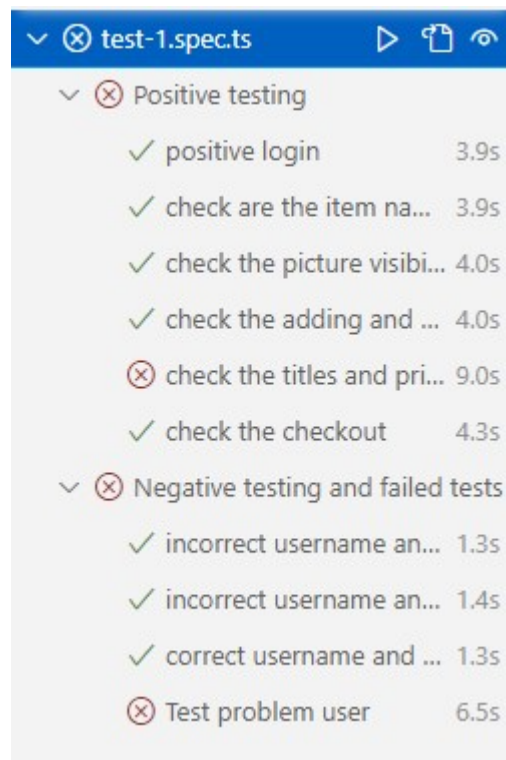


Рисунок 4.11 — Результати тестування сайту

У результаті проходження тестів були пройдені такі тести як:

- positive login;
- check are the item name clickable;
- check the picture visibility;
- check the adding and removing item to/from cart;
- check the checkout;
- incorrect username and password login;
- incorrect username and correct password login;
- correct username and incorrect password.

Результати тестів задовільні та повторюються неоднократно за адекватну кількість часу при стандартній швидкості інтернету.

Розглянемо тести, що повернули помилку.

Тест «check the titles and price of items» повертає помилку на 67 строці коду. Деталі показані на рисунку 4.12.

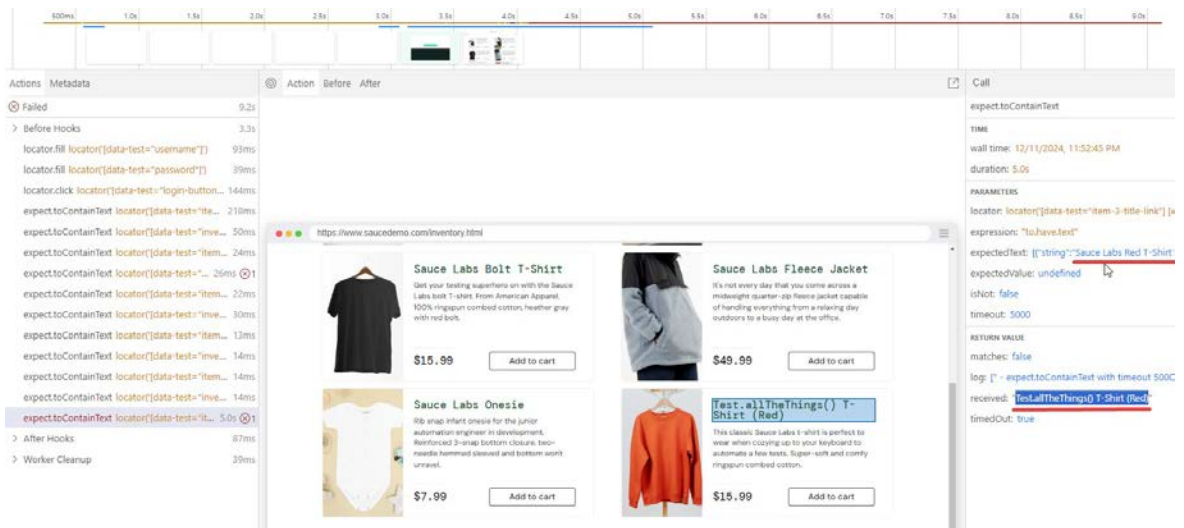


Рисунок 4.12 — Подробиці помилки тесту перевірки назв товарів у інтерфейсі користувача

В результаті ми дізнались подробиці що тест очікував знайти текст «Sauce Labs Red T-Shirt» у елементі останнього товару в списку а отримав «Test.allTheThings() T-Shirt (Red)» що являє собою проблему на стороні сайту а не тесту. Тест відпрацював стабільно і за сценарієм, виявив проблему.

Тест «Test problem user» повертає помилку на 204 строці під час виконання перевірки наявності заголовка «Thank you for your order!» на сторінці після успішної оплати.

Розглянемо результати тесту у інтерфейсі на рисунку 4.13.

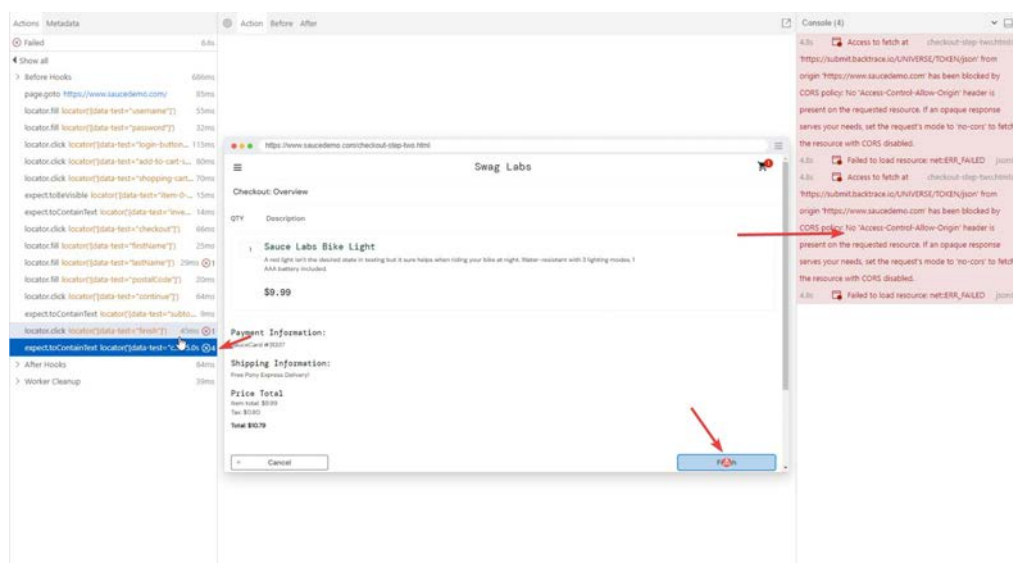


Рисунок 4.13 — Результат виконання тесту з проблемним користувачем

Можна побачити що помилка вийшла після натиснення кнопки закінчення покупки. Сайт не перейшов на наступну сторінку, відправивши у консоль помилку що можна пов'язати з тим що цей конкретний користувач не має доступу. Щоб перевірити це подивимось на дані для логіна цього користувача:

- ім'я користувача «error_user»;
- пароль «secret_sauce».

Тепер повторимо тест з даними звичайного користувача:

- ім'я користувача «standard_user»;
- пароль «secret_sauce».

Результат тесту з стандартним користувачем показаний на рисунку 4.14.

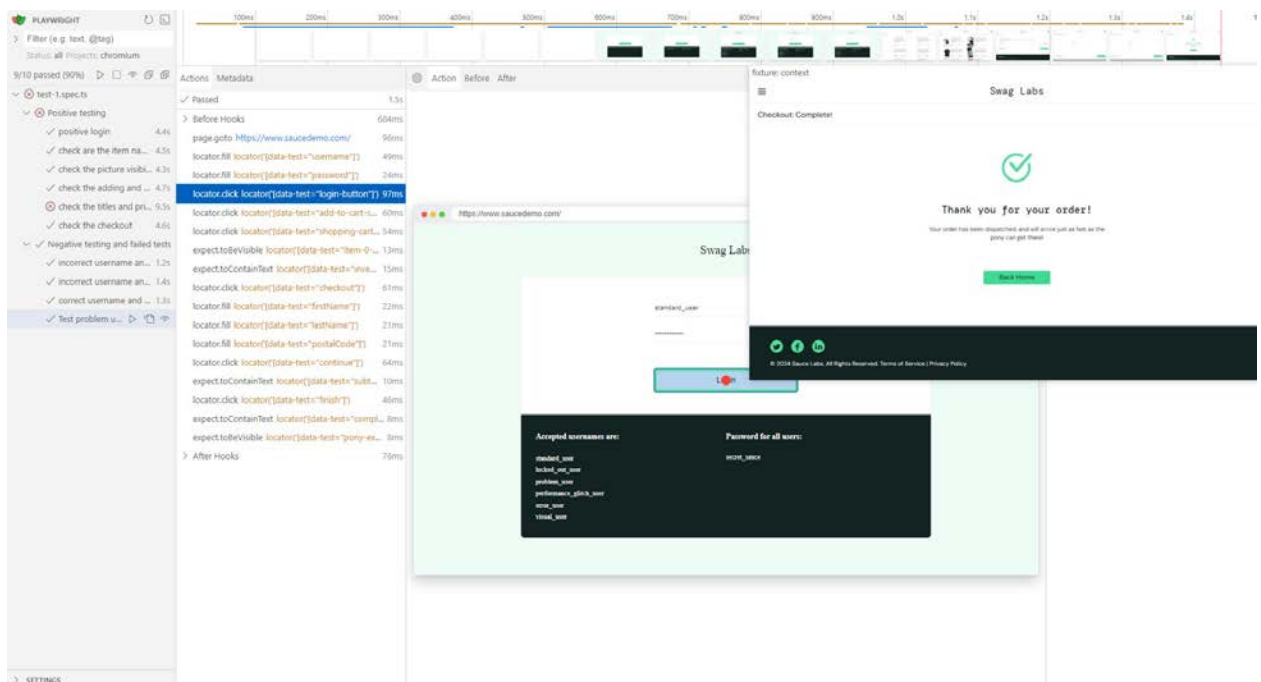


Рисунок 4.14 — Результат виконання тесту з стандартним користувачем

Як бачимо тест проходить успішно і за сценарієм, тож з цього робимо висновок що проблема на стороні серверу сайту а не тесту.

Крім цього було виконане навантажувальне тестування.

Навантажувальне тестування — це підмножина тестування продуктивності, яка використовується для програмного забезпечення, вебсайтів, програм і пов'язаних систем. Це нефункціональний тест, який імітує поведінку кількох користувачів, які одночасно отримують доступ до системи. Навантажувальне тестування, яке також називають «об'ємним тестуванням», відтворює продуктивність, стабільність і функціональність вебсистеми в реальних умовах, тому це один із останніх і найважливіших типів тестування, який виконується перед розгортанням [24].

Тестування навантаження визначає кілька критичних аспектів вебсистеми, зокрема:

- загальна робоча потужність програми, включаючи кількість одночасних користувачів, які можуть підтримуватися;
- здатність програми реагувати на пікове навантаження користувачів;
- стабільність інфраструктури програми;
- час відгуку програми, швидкість пропускання та потреби в ресурсах за різних рівнів навантаження користувачів [24].

Навантажувальне тестування є важливим процесом, який використовується перед запуском будь-якої клієнт/серверної інтернет та інтернетпрограми. Це стосується як зовнішнього програмного забезпечення, такого як вебсайт, так і внутрішніх систем, таких як сервери, на яких розміщено сайт [24].

Автотести були перевірені на роботу з повільним інтрнетом (2G) і вони проходять за 20 секунд кожен. Стабільність виконання пов'язана з тим що фреймворк Playwright має вбудовану затримку виконання сценарія у випадку повільної підгрузки сторінок.

Також було виконане тестування з надмірного заванжатення системи (99% процесору та оперативної пам'яті) і програма виконалась успішно за 23 секунди.

Було перевірене перенесення файлу тестів на інший комп'ютер з MacOS де задоволені всі умови по функціонуванню крім ОС. В результаті тести виконались з очікуваним результатом без аномальних затримок. Це пов'язано з тим що фреймворк повністю підтримує всі сучасні ОС.

4.6 Висновки за розділом 4

У цьому розділі було описано умови застосування, передачі тестів. Також було описано характеристики програмної частини тестів.

Було зазначене детальне керівництво розробника для використання графічного інтерфейсу користувача для тестування. Цим поприли майже весь функціонал інтерфейсу.

Було проведене тестування автотестів та перевірене функціонування кожного, чи відпрацьовують алгоритми по котрим написані тести, який результат вони показують та який очікується від них. Проведене навантажувальне тестування для перевірки стабільності виконання тестів у різних умовах.

В результаті отримані автотести оснофного (критичного) функціоналу тестового інтернет магазину. Котрі повністю задовольняють розробленим для них вимогам, та виконали основну задачу тестів: знайшли помилки у функціонуванні вебзастосунку.

ВИСНОВКИ

В процесі виконання дипломної кваліфікаційної роботи було проведено дослідження методів ручного і автоматизованого тестування вебзастосунків, дослідження актуальності автоматизованого тестування. Основною метою було розглянути і проаналізувати існуючі фреймворки для автоматизації тестування сайтів та реалізувати автотести вебзастосунку.

Були виконані перелічені кроки у процесі роботи:

- проведено аналіз предметної області;
- проведено аналіз актуальний фреймворків;
- визначено вимоги до автотестів;
- вибрано та обгрунтовано технології для реалізації автотестів;
- спроектовано тусту та структуру тест-кейсів для програми;
- розроблено програмним методом автоматизацію тестування вебзастосунку.

Тести написані на мові програмування TypeScript що є модифікацією мови JavaScript з використанням фреймворку Playwright версії v1.49.1 для запуску тестів і використання вбудованого інтерфейсу користувача. Проект створено у середовищі розробки Visual Studio Code версії 1.95.3.

В результаті виконання дипломної кваліфікаційної роботи було створено автотести для тестування основного функціоналу тестового інтернет магазину. Відтворюють запрограмований алгоритм і повністю виконують основну задачу тестів: знаходять недоліки у роботі вебзастосунку.

Тестування показало що автотести працюють стабільно, відтворюються без помилок та збоїв навіть при навантажувальному тестуванні.

Усі завдання дипломної кваліфікаційної роботи магістра виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Number of internet users worldwide from 2005 to 2024 [Electronic resource]. — Access mode: <https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/#:~:text=As%20of%202023%2C%20the%20estimated,67%20percent%20of%20global%20population/>.
2. Top Software Testing Trends for 2024 [Electronic resource]. — Access mode : <https://www.headspin.io/blog/top-testing-trends-2024>.
3. Uptime Institute's 2022 Outage Analysis Finds Downtime Costs and Consequences Worsening as Industry Efforts to Curb Outage Frequency Fall Short [Electronic resource]. — Access mode: <https://uptimeinstitute.com/about-ui/press-releases/2022-outage-analysis-finds-downtime-costs-and-consequences-worsening>.
4. Ручне тестування – що це таке, типи, процеси, підходи, інструменти та інше! [Electronic resource]. — Access mode : <https://www.zaptest.com/uk/ручне-тестування-що-це-таке-типи-проц>.
5. Автоматизоване тестування: працюють роботи — людина щаслива! [Electronic resource]. — Access mode : <https://campus.epam.ua/ua/blog/477>.
6. Ручне тестування vs Автоматизація [Electronic resource]. — Access mode: <https://training.qatestlab.com/blog/technical-articles/manual-testing-vs-automation-testing/>.
7. Best Test Automation Frameworks in 2024 [Electronic resource]. — Access mode: <https://www.browserstack.com/guide/best-test-automation-frameworks>.
8. Selenium framework [Electronic resource]. — Access mode: <https://github.com/SeleniumHQ/selenium>.
9. Cypress framework [Electronic resource]. — Access mode: <https://github.com/cypress-io/cypress>.
10. Playwright framework [Electronic resource]. — Access mode:

<https://github.com/microsoft/playwright>.

11. Nightwatch framework [Electronic resource]. — Access mode: <https://github.com/nightwatchjs/nightwatch>.

12. Appium framework [Electronic resource]. — Access mode: <https://github.com/appium/appium>.

13. Puppeteer framework [Electronic resource]. — Access mode: <https://github.com/puppeteer/puppeteer>.

14. JavaScript [Electronic resource]. — Access mode: <https://uk.wikipedia.org/wiki/JavaScript>.

15. Що таке мова програмування Python? [Electronic resource]. — Access mode: <https://freehost.com.ua/ukr/faq/wiki/chtotakoe-jazik-programmirovanija-python/>.

16. Playwright Supported languages [Electronic resource]. — Access mode: <https://playwright.dev/python/docs/languages>.

17. Visual Studio Code [Electronic resource]. — Access mode: <https://blog.skillfactory.ru/glossary/visual-studio-code/>.

18. Installation [Electronic resource]. — Access mode: <https://playwright.dev/docs/intro>.

19. Visual Studio Code [Electronic resource]. — Access mode: <https://code.visualstudio.com/>.

20. Node.js [Electronic resource]. — Access mode: <https://nodejs.org/uk>.

21. Тест-дизайн у тестуванні [Electronic resource]. — Access mode: <https://ru.hexlet.io/blog/posts/test-dizayn-v-testirovanii-chtoto-takoe>.

22. Playwright Test for VSCode [Electronic resource]. — Access mode: <https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright>.

23. Swag Labs [Electronic resource]. — Access mode: <https://www.saucedemo.com/>.

24. Що таке тестування навантаження? [Electronic resource]. — Access mode: <https://www.zaptest.com/uk/що-таке-тестування-навантаження-глиб>.

ДОДАТОК А
Технічне завдання

Вступ

Автоматизація тестування знаходить застосування в різних проєктах, де необхідно прискорити повторювані дії, підвищуючи ефективність і швидкість перевірки вже перевіреного спеціалістом функціоналу. Автоматизація доказала свою ефективність у розробці будьякого ПЗ, як то вебзастосунки так і серверні чи мобільні рішення.

Тема для проєктування системи — дослідження та програмна реалізація автотестів вебзастосунків.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу магістра на тему: «Дослідження засобів автоматизації тестування вебзастосунків та програмна реалізація тестів», затверджене наказом Національного університету «Запорізька політехніка» № 398 від 07 жовтня 2024 р.

A.2 Основні вимоги до тестів, що розробляються

Головним завданням дипломної кваліфікаційної роботи є дослідження сучасних фреймворків автоматизації тестування і використання відповідного ПЗ для реалізації автоматизованого тестування вебзастосунку.

A.2.1 Вимоги до функціональних характеристик

Функціональні вимоги:

- система повинна проходити тестування сайту покроково;
- система повинна відображати статус тестів (пройдені/провалені);
- система повинна дати змогу відстежити у якій частині вебзастосунку проблема;

- система повинна дати змогу відстежити особисту недосконалість що призвела до проваленого тесту;
- система повинна проводити тестування на основних сучасних рушіях браузерів (Firefox, Chromium та Webkit);
- система повинна давати змогу улюбий момент запустити себе і відтворити тестування вебзастосунку при виконанні вимог.

A.2.2 Експлуатаційні вимоги

Вимоги будуть сформовані здебільшого від вимог середовища у якому воно має бути запущено:

Мінімальні вимоги до програмно-технічних засобів:

- ОС Windows 10/11 (64-bit);
- процесор з тактовою частотою не менше за 2.5 ГГц;
- наявність оперативної пам'яті не менше 8 Гб;
- місце на диску не менше за 100 Мб;
- .NodeJS останньої версії;
- наявність екрану;
- наявність пристроїв типу клавіатура і миша.

A.3 Вхідні дані до роботи

Вхідними даними для роботи програми є дії користувача, зокрема запуск тестів, що виконуються інтерфейсом користувача. Крім того, користувач має багато даних про виконання тестів через графічний інтерфейс, наприклад, помилки що виводяться у консоль чи на якому моментів тесту стався збій. Усі ці дані забезпечують виконання основних операцій тестів.

ДОДАТОК Б
Текст програми

```

Test.spec.ts:
import { test, expect } from '@playwright/test';

test.beforeEach(async ({ page }) => {
  await page.goto('https://www.saucedemo.com/');
});

test.describe('Positive testing', () => {
  test('positive login', async ({ page }) => {
    await page.locator('[data-test="username"]').fill('standard_user');
    //enter the correct username
    await page.locator('[data-test="password"]').fill('secret_sauce');
    //enter correct password
    await page.locator('[data-test="login-button"]').click();
    //click on the login button
  });

  test('check are the item name clickable', async ({ page }) => {
    await PositiveLoginStandard(page);
    await page.locator('[data-test="item-0-title-link"]').click();
    await page.locator('[data-test="back-to-products"]').click();
    await expect(page.locator('[data-test="inventory-list"]')).toBeVisible();
  });

  test('check the picture visibility', async ({ page }) => {
    await PositiveLoginStandard(page);
    await expect(page.locator('[data-test="item-0-img-link"]')).toBeVisible();
  });

  test('check the adding and removing item to/from cart', async ({ page })
=> {
    await PositiveLoginStandard(page); //login
    await page.locator('[data-test="add-to-cart-sauce-labs-bike-
light"]').click(); //add to cart
    await expect(page.locator('[data-test="shopping-cart-
badge"]')).toContainText('1');
    //are cart display 1 item icon?
    await page.locator('[data-test="shopping-cart-link"]').click();
    //go to cart
    await page.locator('[data-test="remove-sauce-labs-bike-light"]').click();
    //remove
    await page.locator('[data-test="continue-shopping"]').click();
    //go back to the products tab
    await expect(page.locator('[data-test="inventory-list"]')).toBeVisible();
  }
});

```

```

    //are we in corr tab?
  });

  test('check the titles and price of items', async ({ page }) => {
    await PositiveLoginStandard(page);
    //login
    await expect(page.locator('[data-test="item-4-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Backpack');
    //are the Sauce Labs Backpack item has correct title?
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$29.99');
    //and correct price?
    await expect(page.locator('[data-test="item-0-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Bike Light');
    //are the Sauce Labs Bike Light item has correct title?
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$9.99');
    //and correct price?
    await expect(page.locator('[data-test="item-1-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Bolt T-Shirt');
    //are the Sauce Labs Bolt T-Shirt has correct title?
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$15.99');
    //and correct price?
    await expect(page.locator('[data-test="item-5-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Fleece Jacket');
    //are the Sauce Labs Fleece Jacket item has correct title??
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$49.99');
    //and correct price?
    await expect(page.locator('[data-test="item-2-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Onesie');
    //are the Sauce Labs Onesie item has correct title?
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$7.99');
    //and correct price?
    await expect(page.locator('[data-test="item-3-title-link"] [data-
test="inventory-item-name"]')).toContainText('Sauce Labs Red T-Shirt');
    //are the Sauce Labs Red T-Shirt item has correct title?
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$15.99');
    //and correct price?
  });

  test('check the checkout', async ({ page }) => {

```

```

    await PositiveLoginStandard(page);
    // login
    await expect(page.locator('[data-test="inventory-
list"]')).toContainText('$15.99');
    //check the price of item in catalog
    await page.locator('[data-test="add-to-cart-test\\allthethings\\(\\)-t-shirt-
\\(red\\)"]').click();
    //add item to cart
    await page.locator('[data-test="shopping-cart-link"]').click();
    //go to cart
    await expect(page.locator('[data-test="inventory-item-
price"]')).toContainText('$15.99');
    //check the price in the cart
    await page.locator('[data-test="checkout"]').click();
    //go to the checkout page
    await page.locator('[data-test="firstName"]').fill('testfirst1');
    //enter user`s first name
    await page.locator('[data-test="lastName"]').fill('testlast1');
    //enter user`s last name
    await page.locator('[data-test="postalCode"]').fill('69069');
    //enter user`s postcode
    await page.locator('[data-test="continue"]').click();
    //go to the payment
    await expect(page.locator('[data-test="title"]')).toContainText('Checkout:
Overview');
    //check are we on the expected page?
    await expect(page.locator('[data-test="inventory-item-
price"]')).toContainText('$15.99');
    //check the item price
    await expect(page.locator('[data-test="subtotal-
label"]')).toContainText('Item total: $15.99');
    //check the item price in checkout before taxes
    await page.locator('[data-test="finish"]').click();
    //click finish
    await expect(page.locator('[data-test="complete-
header"]')).toContainText('Thank you for your order!');
    //check are we on the expected page?
  });
});

test.describe('Negative testing', () => {
  test('incorrect username and password data login', async ({ page }) => {
    await page.locator('[data-test="username"]').fill('standard_user1');
    //enter the incorrect username
    await page.locator('[data-test="password"]').fill('secret_sauce1');
  });
});

```

```

    //enter the incorrect password
    await page.locator('[data-test="login-button"]').click();
    //click on the login button
    await expect(page.locator('[data-test="error"]')).toBeVisible();
    //check the visibility of error
    await expect(page.locator('[data-test="error"]')).toContainText('Epic
sadface: Username and password do not match any user in this service');
    //check the error text
    await page.locator('[data-test="password"]').fill('secret_sauce');
    //enter correct password
    await page.locator('[data-test="username"]').fill('standard_user');
    //enter the correct username
    await page.locator('[data-test="login-button"]').click();
    //check the correct login after incorrect
    await expect(page.locator('[data-
test="title"]')).toContainText('Products');
    //are we login succesfully?
  });

  test('incorrect username and correct password data login', async ({ page })
=> {
    await page.locator('[data-test="username"]').fill('standard_user1');
    //enter the incorrect username
    await page.locator('[data-test="password"]').fill('secret_sauce');
    //enter the correct password
    await page.locator('[data-test="login-button"]').click();
    //click on the login button
    await expect(page.locator('[data-test="error"]')).toBeVisible();
    //check the visibility of error
    await expect(page.locator('[data-test="error"]')).toContainText('Epic
sadface: Username and password do not match any user in this service');
    //check the error text
    await page.locator('[data-test="error-button"]').click();
    //close the error
    await page.locator('[data-test="password"]').fill('secret_sauce');
    //enter correct password
    await page.locator('[data-test="username"]').fill('standard_user');
    //enter the correct username
    await page.locator('[data-test="login-button"]').click();
    //check the correct login after incorrect
    await expect(page.locator('[data-
test="title"]')).toContainText('Products');
    //are we login succesfully?
  });

```

```

test('correct username and incorrect password data login', async ({ page })
=> {
  await page.locator('[data-test="username"]').fill('standard_user');
  //enter the incorrect username
  await page.locator('[data-test="password"]').fill('secret_sauce1');
  //enter the correct password
  await page.locator('[data-test="login-button"]').click();
  //click on the login button
  await expect(page.locator('[data-test="error"]')).toBeVisible();
  //check the visibility of error
  await expect(page.locator('[data-test="error"]')).toContainText('Epic
sadface: Username and password do not match any user in this service');
  //check the error text
  await page.locator('[data-test="error-button"]').click();
  //close the error
  await page.locator('[data-test="password"]').fill('secret_sauce');
  //enter correct password
  await page.locator('[data-test="username"]').fill('standard_user');
  //enter the correct username
  await page.locator('[data-test="login-button"]').click();
  //check the correct login after incorrect
  await expect(page.locator('[data-
test="title"]')).toContainText('Products');
  //are we login succesfully?
});

test('Test problem user', async ({ page }) => {
  await page.goto('https://www.saucedemo.com/');
  await page.locator('[data-test="username"]').fill('error_user');
  //enter the problem user
  await page.locator('[data-test="password"]').fill('secret_sauce');
  //enter correct password
  await page.locator('[data-test="login-button"]').click();
  await page.locator('[data-test="add-to-cart-sauce-labs-bike-
light"]').click();
  //add item to cart
  await page.locator('[data-test="shopping-cart-link"]').click();
  //go to cart
  await expect(page.locator('[data-test="item-0-title-
link"]')).toBeVisible();
  //check the item visibility
  await expect(page.locator('[data-test="inventory-item-
price"]')).toContainText('$9.99');
  //check the price
  await page.locator('[data-test="checkout"]').click();

```

```

    //go to the checkout
    await page.locator('[data-test="firstName"]').fill('testfirst1');
    //enter the first name
    await page.locator('[data-test="lastName"]').fill('testlast1');
    //enter the last name
    await page.locator('[data-test="postalCode"]').fill('testcode');
    //enter test code
    await page.locator('[data-test="continue"]').click();
    //go to the checkout
    await expect(page.locator('[data-test="subtotal-
label"]')).toContainText('Item total: $9.99');
    //check the price
    await page.locator('[data-test="finish"]').click();
    //finish
    await expect(page.locator('[data-test="complete-
header"]')).toContainText('Thank you for your order!');
    //check if successfull
    await expect(page.locator('[data-test="pony-express"]')).toBeVisible();
  });
});

async function PositiveLoginStandard(page) {
  //enter the username
  await page.locator('[data-test="username"]').fill('standard_user');
  //enter the correct password
  await page.locator('[data-test="password"]').fill('secret_sauce');
  await page.locator('[data-test="login-button"]').click();

```

```

}
```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів
2024

**Дослідження засобів автоматизації тестування вебзастосунків
та програмна реалізація тестів
(Research of Automation Tools for Testing Web Applications and
Software Implementation of Tests)**

Виконав студент групи КНТ-223м
Керівник к.т.н.

ЯВОРСЬКИЙ БОГДАН
ШИТІКОВА ОЛЕНА

1

Рисунок В.1 — Слайд 1

Об'єкт, предмет та мета роботи

- Об'єктом роботи є процес автоматизації тестування вебзастосунків;
- Предметом роботи є засоби (фреймворки) автоматизації тестування вебзастосунків;
- Метою роботи є дослідження сучасних інструментів автоматизації тестування вебзастосунків та програмна реалізація тестів.

Для досягнення поставленої мети необхідно:

- Дослідити інструменти автоматизації тестування;
- Обґрунтувати вибір фреймворку;
- Виконати застосування обраної технології для програмної реалізації автоматичних тестів вебзастосунків.

2

Рисунок В.2 — Слайд 2

Порівняння фреймворків автоматизації

Фреймворк	Selenium	Cypress	Playwright	NightwatchJS
Підтримка мов програмування	Java, Python, C#, Ruby	JavaScript	JavaScript, Java, Python, C#	JavaScript
Підтримка браузерів	Chrome, Firefox, Safari	Chrome, Firefox	Chrome, Firefox, Safari	Chrome, Firefox, Safari
Тестування API	-	+	+	+
Кількість матеріалу для навчання	Велика	Середня	Середня	Мала
Простота використання	Складно	Просто	Дуже просто	Середньо
Швидкість і стабільність	Повільно, нестабільно	Швидко, нестабільно	Швидко, стабільно	Середньо, стабільно

3

Рисунок В.3 — Слайд 3

Вибір мови програмування

Мова	JavaScript/ TypeScript	Python	Java	C#
Підтримка мови у фреймворку	Рідна мова	Хороша	Обмежена	Обмежена
Легкість в навчанні	Легка	Легка	Важка	Важка
Швидкість розробки	Висока	Висока	Середня	Середня
Швидкість виконання тестів	Велика	Середня	Середня	Середня
Інтеграція з методом CI/CD	Широка	Середня	Середня	Низька

4

Рисунок В.4 — Слайд 4

Порівняння середовищ розробки

Середовище	Visual Studio Code	WebStorm	Atom	Notepad++
Підтримка мови програмування	Відмінна	Відмінна	Середня	Середня
Інтеграція з фреймворком	Спеціалізована	Можливість налаштування	Тільки код	Тільки код
Налагодження	+	+	-	-
Автозаповнення	+	+	-	-
Простота використання	Легко	Середньо	Легко	Легко
Інтеграція з системою Git	+	+	/+	Повільно, стабільно

5

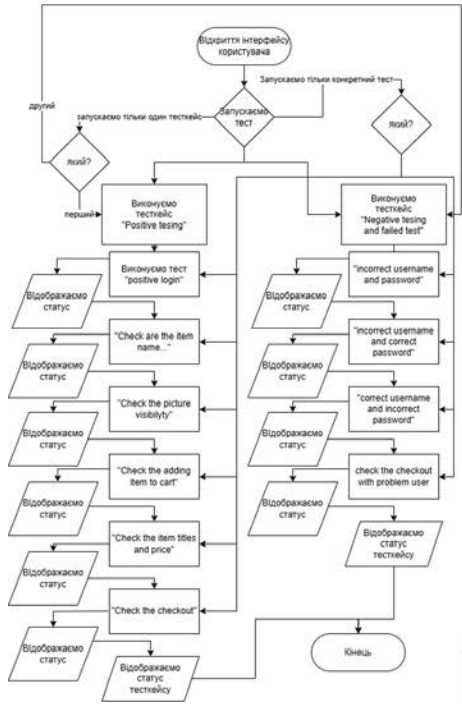
Рисунок В.5 — Слайд 5

Спроектовані тест-кейси

Позитивне тестування:	Негативне тестування:
— тестування логіну;	— тестування логіну з невалідними даними;
— тестування відображення картинок товарів;	— тестування відображення помилок;
— тестування клікабельності елементів;	— тестування покупки товару з невалідними даними.
— тест видалення товару з кошика;	
— тест відображення кількості товарів у іконці кошика;	
— тест покупки товару.	

6

Рисунок В.6 — Слайд 6



Алгоритми роботи програми (виконання тест-кейсів)



Рисунок В.7 — Слайд 7

Алгоритми роботи програми (виконання тест-кейсів)

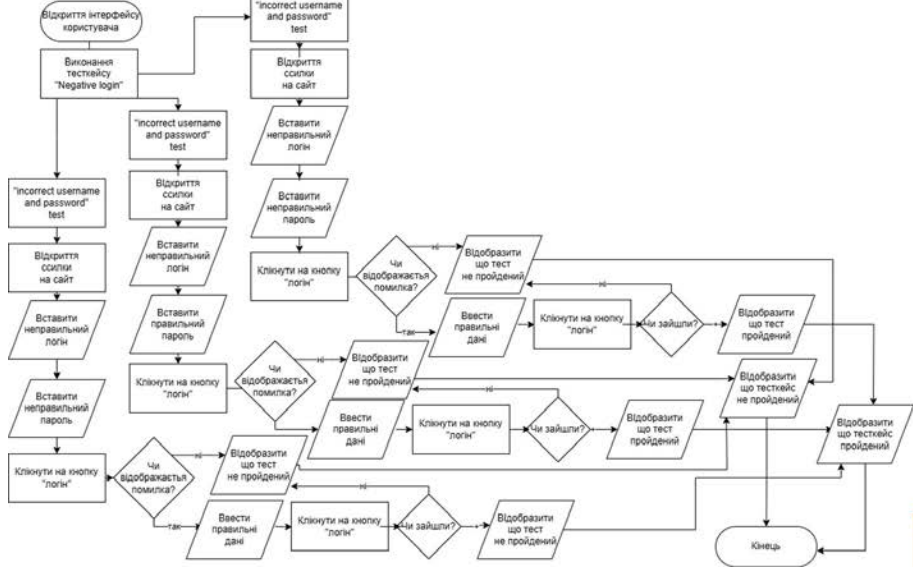


Рисунок В.8 — Слайд 8

Результат виконання тестів

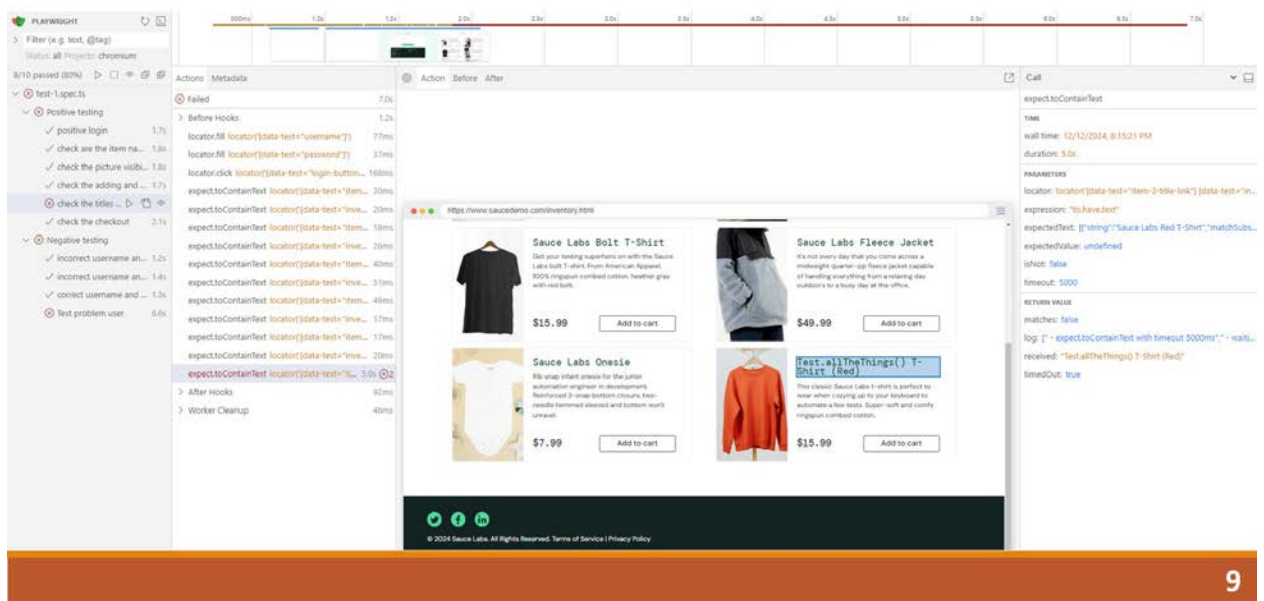


Рисунок В.9 — Слайд 9

Тестовий «проблемний» користувач

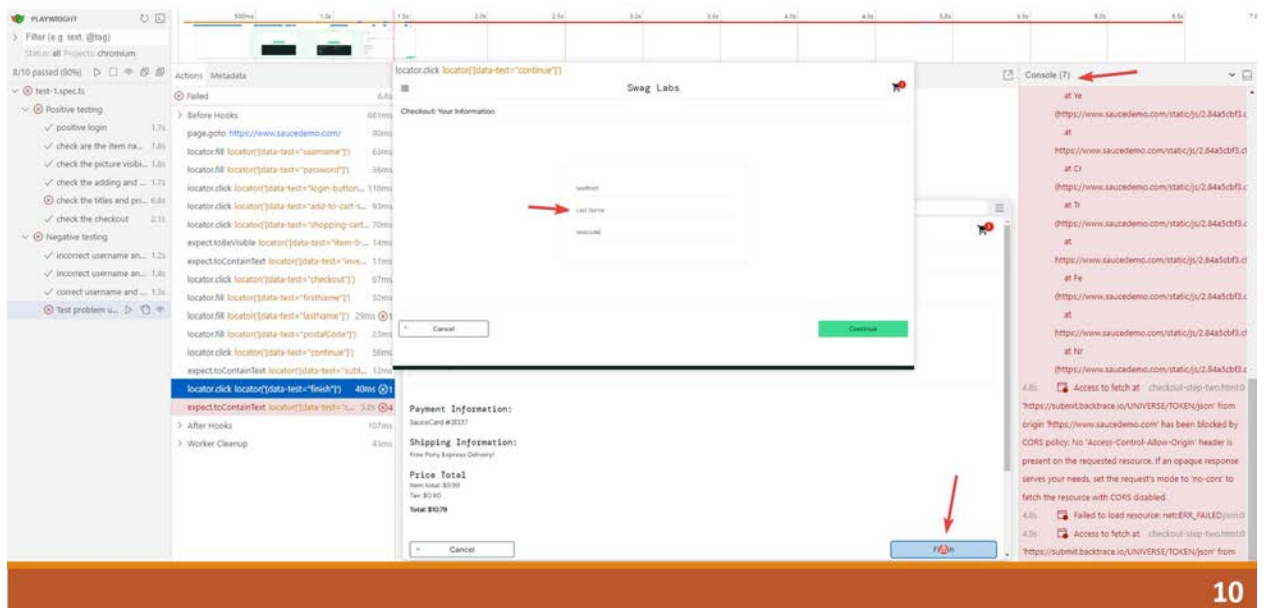


Рисунок В.10 — Слайд 10

Останній тест з стандартним користувачем

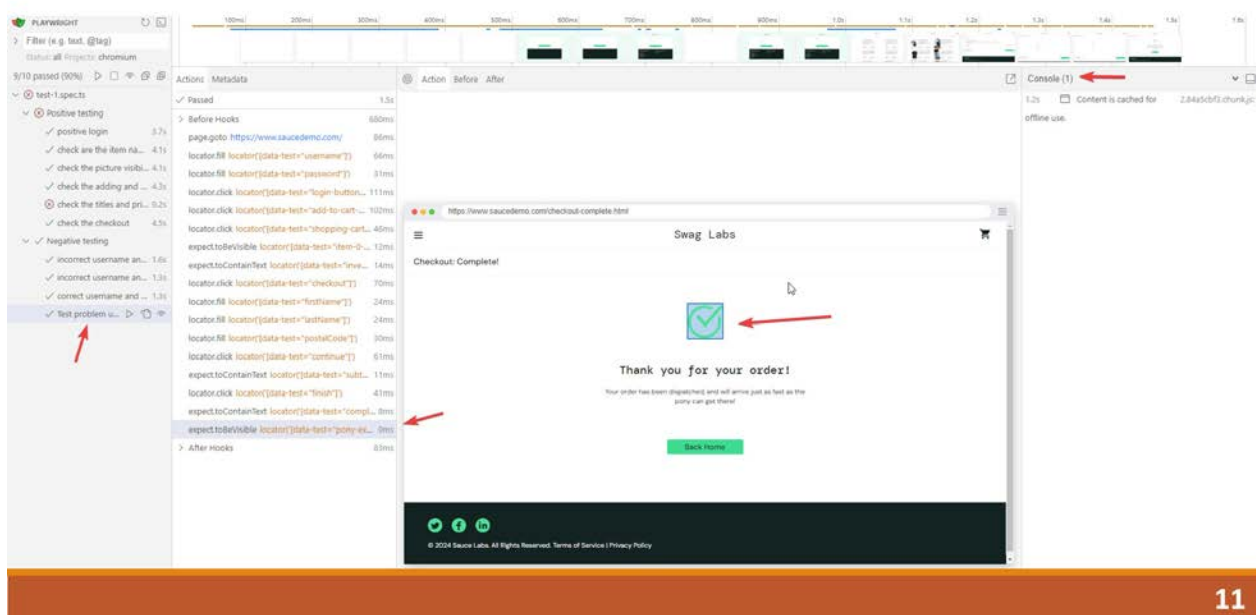


Рисунок В.11 — Слайд 11

Висновки

- В процесі виконання дипломної роботи було проведено дослідження актуальних інструментів з автоматизації тестування вебзастосунків. Основною метою було проаналізувати існуючі фреймворки та програмно реалізувати автотести інтернет магазину.
- Були створені автотести для тестування основних функцій тестового інтернет магазину. Інтерфейс користувача зручний у використанні а також тести повністю задовольняють вимогам технічного завдання.
- Тестування роботи автотестів показало високу стабільність і швидкість виконання при навантажувальному тестуванні.
- Усі завдання дипломної кваліфікаційної роботи магістра виконано.

Рисунок В.12 — Слайд 12

Дякую за увагу!
