ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система автоматизації тестування веб-сайтів на основі Selenium »

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ВОЛОДЬКО
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Олександр ВОЛОДЬКО
група КНДМ-62

Керівник: Олена ШИКУЛА
науковий ступінь, вчене звання д.ф-м.н., професор

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Володьку Олександру Борисовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система автоматизації тестування веб-сайтів на основі Selenium»

керівник кваліфікаційної роботи Олена ШИКУЛА д.ф-м.н., професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, види тестування, технічна документація Selenium.

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

4.1. Поняття тестування та його види.

4.2. Проаналізувати інструменти для автоматизації тестування .

4.3. Визначення вимог тестування.

4.4 Автоматичне тестування з використанням засобів із набору інструментів Selenium.

5. Перелік графічного матеріалу: *презентація*

5.1 Основні види тестування.

5.2 Структура набору інструментів Selenium.

5.3 Використання Selenium IDE та Selenium WebDriver.

5.4 Переваги Selenium.

6. Дата видачі завдання«16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-28.09.24	Виконано
2	Аналіз поняття тестування та його видів	31.09-10.10.24	Виконано
3	Аналіз інструментів для автоматичного тестування	12.10-29.10.24	Виконано
4	Вивчення набору інструментів Selenium	30.10-10.11.24	Виконано
5	Визначення вимог тестування	11.11-16.11.24	Виконано
6	Розгортання автоматичного тестування	17.11-26.11.24	Виконано
7	Оформлення роботи: вступ, висновки, реферат	27.11-06.12.24	Виконано
8	Розробка демонстраційних матеріалів	09.12-14.12.24	Виконано

Здобувач вищої освіти _____ Олександр ВОЛОДЬКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Олена ШИКУЛА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 71 стор., 39 рис., 26 джерел.

Мета роботи – вивчити можливості набору інструментів Selenium для автоматизації тестування вебсайтів та визначення їх переваг і недоліків.

Об'єкт дослідження – автоматизація тестування вебсайтів.

Предмет дослідження – інструменти для автоматизації тестування вебзастосунків.

Методи дослідження – класифікація, порівняльний аналіз, оцінка ефективності, експериментальний метод.

Короткий зміст роботи: Проведено дослідження видів тестування. Проаналізовано переваги та недоліки мануального та автоматичного видів тестування. Проведено аналіз інструментів для автоматичного тестування. Розглянуто набір інструментів Selenium .

Визначено вимоги для тестування та розроблено тест-кейси для впровадження автоматичного тестування. У результаті роботи розроблені тестові сценарії з використанням Selenium IDE, Selenium WebDriver, мови програмування Phyton та хмарної платформи Sauce Lab.

КЛЮЧОВІ СЛОВА: ТЕСТУВАННЯ, ВЕБСАЙТ, ІНСТРУМЕНТИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ, SELENIUM WEBDRIVER, SELENIUM IDE

ABSTRACT

Text part of the master's qualification work: 71 pages, 39 pictures, 26 sources.

Goal of the work is to study the capabilities of the Selenium toolset for automating website testing and to determine its advantages and disadvantages.Object of research – website testing automation

Subject of research – tools for automating web application testing.

Research methods – classification, comparative analysis, performance evaluation, experimental method.

Summary of the work: The research included an analysis of testing types, focusing on the advantages and disadvantages of manual and automated testing.

A review of tools for automated testing was conducted, with special attention given to the Selenium toolset. Testing requirements were defined, and test cases for implementing automated testing were developed.

As a result of the work, test scenarios were created using Selenium IDE, Selenium WebDriver, the Python programming language, and the Sauce Labs cloud platform.

KEYWORDS: TESTING, WEBSITE, TEST AUTOMATION TOOLS, SELENIUM WEBDRIVER, SELENIUM IDE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система.

CERN (European Organization for Nuclear Research) – Європейська організація з ядерних досліджень, яка відома своєю участю в розробці Інтернету.

SLAC (Stanford Linear Accelerator Center) – науково-дослідний центр, що займається фізикою елементарних частинок та технологіями, пов'язаними з інформаційними технологіями.

MIDAS (Mechanical Interface for Data Acquisition Systems) – система для збору та обробки даних, використовувана в наукових та інженерних проектах.

NCSA (National Center for Supercomputing Applications) – національний центр суперкомп'ютерних застосувань у США, де розробляли багато технологій для Інтернету.

ECMA (European Computer Manufacturers Association) – міжнародна організація, що встановлює стандарти для інформаційних технологій, включаючи мови програмування (наприклад, ECMAScript, який є основою для JavaScript).

CSS (Cascading Style Sheets) – каскадні таблиці стилів, технологія для стилізації вебсторінок.

HTML (HyperText Markup Language) – мова розмітки для створення вебсторінок.

W3C (World Wide Web Consortium) – міжнародна організація, яка розробляє стандарти для вебу.

AJAX (Asynchronous JavaScript and XML) – технологія для створення динамічних вебсторінок, яка дозволяє оновлювати частину вебсторінки без перезавантаження.

CMS (Content Management System) – система керування контентом, яка дозволяє створювати та редагувати контент на вебсайті без знань програмування (наприклад, WordPress, Joomla).

AI (Artificial Intelligence) – штучний інтелект, який використовує алгоритми для виконання завдань, що зазвичай потребують людського інтелекту

VR (Virtual Reality) – віртуальна реальність, технологія, що створює імітацію реального світу за допомогою комп'ютерних систем.

AR (Augmented Reality) – доповнена реальність, технологія, що накладає віртуальні елементи на реальний світ через камеру пристрою.

HTTPS (HyperText Transfer Protocol Secure) – протокол безпечної передачі гіпертексту, що забезпечує шифрування даних між браузером та сервером.

SSL (Secure Sockets Layer) – криптографічний протокол для захисту даних під час їх передачі через Інтернет (тепер замінено на TLS – Transport Layer Security).

UI (User Interface) – інтерфейс користувача, частина програми або вебсайту, з якою взаємодіє користувач.

UX (User Experience) – досвід користувача, загальне враження, яке користувач отримує від взаємодії з продуктом чи сервісом.

CI/CD (Continuous Integration/Continuous Deployment) – безперервна інтеграція та безперервне розгортання, методи розробки програмного забезпечення для автоматизації тестування та розгортання коду.

DevOps (Development and Operations) – культура та практики, що об'єднують розробку програмного забезпечення та операційні процеси для покращення ефективності.

IoT (Internet of Things) – Інтернет речей, концепція підключення фізичних об'єктів до Інтернету для обміну даними.

WORA (Write Once, Run Anywhere) – концепція, за якою програми можуть працювати на будь-якій платформі без необхідності змінювати код (особливо використовується для Java).

IDE (Integrated Development Environment) – інтегроване середовище розробки, програмне забезпечення, що об'єднує всі інструменти для написання, тестування та налагодження коду.

JSON (JavaScript Object Notation) – легкий формат для обміну даними, який часто використовується для передачі інформації між клієнтом і сервером у вебдодатках.

API (Application Programming Interface) – це набір правил і протоколів, які визначають, як різні компоненти програмного забезпечення можуть взаємодіяти між собою.

ВСТУП

Щодня з’являється 252 000 нових вебсайтів, а глобальна індустрія веброзробки на 2024 рік оцінюється у 2,1 мільярда доларів [8]. Зі зростанням кількості сайтів збільшуються і вимоги до них.

Найбільш імовірно, що, отримавши негативний досвід, споживач вже не повернеться. Непривабливий та застарілий дизайн, довге завантаження сторінок та їх вмісту, неякісне наповнення та присутність технічних помилок, особливо якщо це стосується важливого функціоналу, можуть стати причинами втрати клієнтів. Таким чином якість та доступність вебсайтів стають ключовими для відвідувачів, оскільки вони визначають яким саме буде їх досвід. Проте однаково важливими вони є й для власників сайтів, адже досвід користувачів прямо впливає на конкурентоспроможність бізнесу. Саме тестування й допомагає дійти до висновків про якість та доступність цього інтернет ресурсу.

Ряд досліджень продемонстрував, що виправлення недоліків на ранніх етапах обходиться значно дешевше. Тому доречно починати тестування ще на початкових стадіях розробки, включно з розробкою вимог та архітектури проєктів. Це дозволяє виявити потенційні проблеми раніше, знизити витрати на їх усунення та підвищити загальну якість кінцевого продукту.

Автоматичне тестування у свою чергу значно пришвидшує цей процес, а найпопулярнішим інструментом для автоматизації тестування є Selenium. Отже, тема автоматизації вебсайтів на основі Selenium є справді актуальною.

Метою дослідження є вивчення можливостей набору інструментів Selenium для автоматизації тестування вебсайтів, оцінка їх ефективності в різних сценаріях тестування та визначення їх переваг і недоліків у порівнянні з іншими інструментами.

Об’єкт дослідження – автоматизація тестування вебсайтів.

Предмет дослідження – інструменти для автоматизації тестування вебзастосунків.

Методи дослідження:

- класифікація;
- порівняльний аналіз - порівняння різних інструментів автоматизації тестування;
- експериментальний метод - проведення серії тестів на реальному вебсайті для оцінки практичної ефективності інструментів автоматизації тестування;
- оцінка ефективності

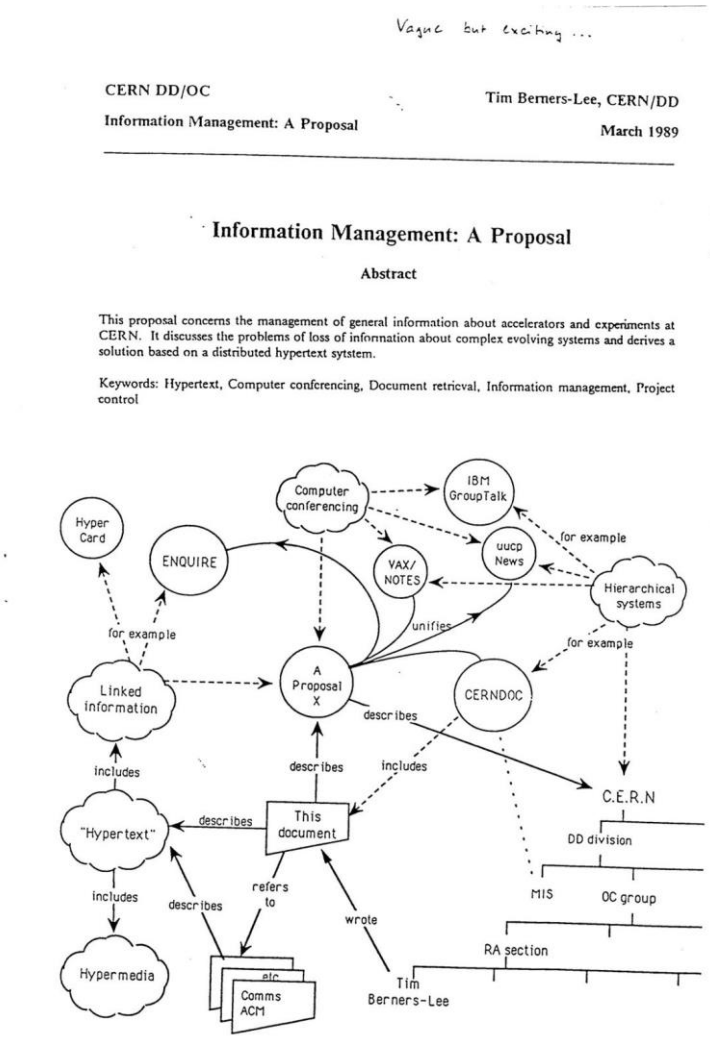
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Історія розвитку вебсайтів

За свою історію вебсайти пройшли великий шлях, зображений на рисунку 1.1. Їх розвиток продовжується і сьогодні. Своїм народженням вони завдячують Тіму Бернерсу-Лі, британському фахівцю з інформатики.

Рисунок 1.1 – Короткий таймлайн розвитку вебсайтів

У 1980-х роках, під час роботи в CERN, Тім Бернерс-Лі помітив, що складно відслідковувати проєкти та комп'ютерні системи тисяч дослідників, розкиданих по всьому світу. У той час інформація зберігалася на різних комп'ютерах, і для доступу до неї потрібно було входити в кожну систему окремо, а також вивчати різні програми. Тож у березні 1989 року Бернерс-Лі запропонував керівникам CERN систему управління інформацією, яка використовувала гіпертекст для зв'язку документів на різних комп'ютерах, підключених до Інтернету. Пропозиція, що зображена на рисунку 1.2, була спочатку відхилена, але, після його об'єднання з бельгійським інженером CERN Робертом Кайяо, він отримав час для роботи над проєктом 1990 року. Початковою назвою була Information Management, проте пізніше Бернерс-Лі змінив її на WorldWideWeb [14].



Vague but exciting ...

CERN DD/OC

Tim Berners-Lee, CERN/DD

Information Management: A Proposal

March 1989

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

Рисунок 1.2 − Реферат пропозиції Тіма Бернерса-Лі [16]

До кінця 1990 року, використовуючи комп'ютер NeXT, розроблений Стівом Джобсом, Бернерс-Лі створив основні технології, які стали основою Web, зокрема HTML, HTTP та URL. Він також розробив базовий браузер і програмне забезпечення для веб-серверів. 6 серпня 1991 року Бернерс-Лі опублікував перший вебсайт, присвячений проекту World Wide Web, який був розміщений на сервері CERN за адресою http://info.cern.ch [14].

Те, як виглядав цей сайт, можна побачити на рисунку 1.3 або ж перейти за посиланням: https://info.cern.ch/hypertext/WWW/TheProject.html .

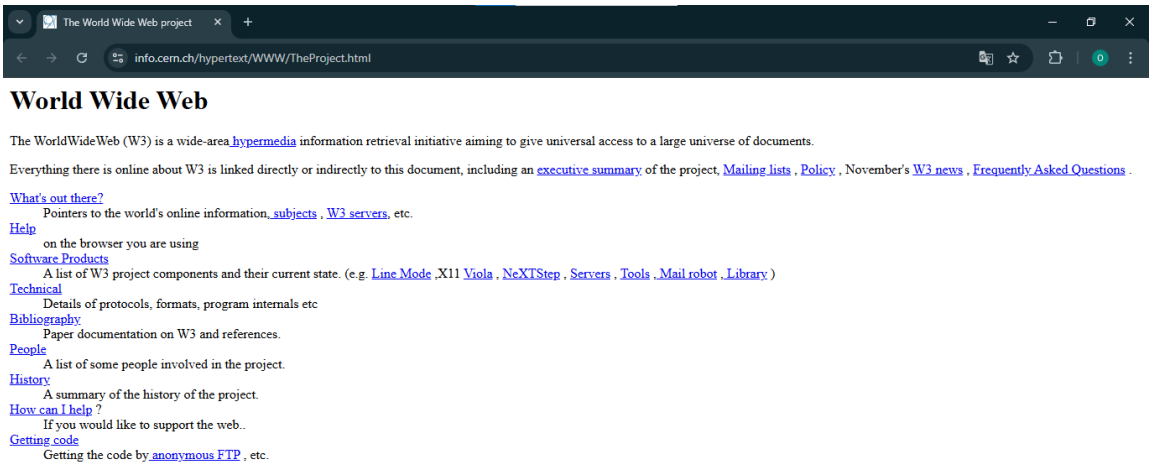


Рисунок 1.3 – Перший у світі вебсайт

Бернерс-Лі відмовився патентувати свою технологію, щоб забезпечити відкритість і швидкий розвиток Web.

У грудні 1991 року, завдяки Полу Кунцу та Луїзі Аддіс та їх роботі у Національній лабораторії прискорювачів, спочатку відомої як Стенфордський центр лінійних прискорювачів або ж SLAC, в Каліфорнії запрацював перший веб-сервер у США. Тоді існували лише браузери двох типів. Один був оригінальною версією, яка працювала, але була доступна лише на комп'ютерах NeXT. Іншим був браузер «лінійного режиму», який легко встановлювався і міг працювати на будь-якій платформі, але мав обмежену функціональність і зручність використання [1].

Невелика команда CERN не могла впоратися з усією роботою щодо подальшого розвитку системи, тож Тім Бернерс-Лі звернувся через Інтернет до інших розробників із проханням долучитися до проєкту. У результаті кілька ентузіастів створили браузери, здебільшого для X-Window System. Серед них виділялися MIDAS, розроблений Тоні Джонсоном із SLAC; Viola, створений Пеєм Веєм із технічного видавництва O'Reilly Books; і Erwise, розроблений фінськими студентами з Технологічного університету Гельсінкі [1].

У 1993 році команда в Університеті Іллінойсу випустила браузер Mosaic, який став першим популярним браузером серед широкої аудиторії. Після цього

з'явилися такі сайти, як Yahoo (1994), Amazon (1995), eBay (1995) та Google (1998). Як виглядали Yahoo, Amazon та Google зображено на рисунку 1.4.

Рисунок 1.4 − Вигляд Google, Yahoo та Amazon [2]

30 квітня 1993 року CERN зробив вихідний код WorldWideWeb доступним на безкоштовній основі, перетворивши його на програму з відкритим кодом. До кінця 1993 року було зареєстровано понад 500 вебсерверів, а WWW становила 1% інтернет-трафіку, що тоді вважалося значною часткою (основний трафік припадав на віддалений доступ, електронну пошту та передачу файлів).

1994 рік став «Роком вебу». У травні, з ініціативи Роберта Кайяо, у CERN відбулася Перша міжнародна конференція з World Wide Web. Захід відвідали 380 користувачів і розробників, а саму конференцію назвали «Woodstock of the Web» [1].

Разом з розвитком Вебу збільшувались і його згадки в медіа. Таким чином на другу конференцію, що проводилась в США новоствореним Міжнародним керівним комітетом конференцій всесвітнього павутиння разом із NCSA в жовтні 1994, уже прийшло 1300 учасників. До кінця 1994 року нараховувалося 10 000 вебсерверів, з яких 2000 були комерційними, а кількість користувачів всесвітньої павутини досягла 10 мільйонів [1].

У 1994 році Бернерс-Лі покинув CERN і заснував World Wide Web Consortium (W3C) в Массачусетському технологічному інституті для підтримки стандартів Web. Він став одним із 100 найважливіших людей століття за версією Time Magazine і був нагороджений орденом від королеви Єлизавети II у 2004 році. У 2009 році Бернерс-Лі заснував World Wide Web Foundation для забезпечення користі Web для людства. На відкритті Олімпійських ігор 2012 року в Лондоні його вшанували за його винахід.

З поширенням Інтернету веброзробка еволюціонувала, щоб задовольнити потреби користувачів. Бренданом Айком винайшов JavaScript у 1995 році. Спочатку його розробили для браузера Netscape 2, а вже від 1997 року він був стандартом ECMA-262 [10].

Після передачі JavaScript від Netscape до організації ECMA, розробку мови продовжив фонд Mozilla, який використовував її для браузера Firefox [10].

Починаючи з 1995 року, розробники почали експериментувати з JavaScript. Ця мова програмування дозволила їм спрощено додавати інтерактивний контент для створення створення динамічних та інтерактивних вебсайтів. Це значно покращило користувацький досвід. Таким чином, вперше в історії вебдизайну, статичні та динамічні елементи сайту об'єдналися. У цей час динамічний контент почав стрімко набирати популярності. Початок і середина 2000-х років стали періодом активного розвитку JavaScript у вебдизайні [2].

У 1996 році було представлено каскадні таблиці стилів (CSS), які дозволили створювати більш гнучкі та складні дизайни вебсайтів. Вони розділяли контент (який залишався в HTML) і його оформлення. Спочатку CSS зіткнулися з проблемами через відсутність підтримки у всіх браузерах, але ці труднощі поступово вирішили. Нині доступна третя версія CSS, яка широко використовується у веброзробці.

Андре Манро зазначав: « CSS вперше було запропоновано у 1994 році норвезьким технологом Гоконом Віумом Лі, який на той час працював у Європейській організації ядерних досліджень (CERN) разом із Бернерсом-Лі. Через два роки Консорціум Всесвітнього павутиння (W3C) ухвалив першу стандартизовану специфікацію для CSS, яка отримала назву CSS1. Ця версія була розроблена спільно Лі та нідерландським програмістом Бертом Босом. Microsoft Internet Explorer став першим комерційним браузером, який підтримував CSS. У 1998 році було випущено CSS2, який забезпечив покращений контроль макетів і можливість визначати, як контент буде відображатися на різних платформах, включаючи портативні пристрої, друк, екрани, телевізори та навіть шрифт Брайля. CSS3, випущений у 2011 році, додав нові функції, такі як адаптивний вебдизайн і

підтримка багатьох нових типів шрифтів. Також у CSS3 було введено модулі. Модулі CSS використовуються для уникнення конфліктів стилів, оскільки у модулях правила CSS за замовчуванням застосовуються локально, а не глобально. Крім того, модулі допомагають користувачам повторно використовувати компоненти коду, оскільки кожен компонент може мати власний набір стилістичних правил. Введення модулів підвищило безпеку CSS, забезпечивши локальне застосування правил. Окрім основного розвитку CSS, поява CSS-фреймворків, таких як Bootstrap, значно спростила адаптивний вебдизайн завдяки готовим для повторного використання компонентам, таким як панелі навігації та кнопки.» [12].

Зростання електронної комерції на початку 2000-х років призвело до розвитку веб-застосунків, які є сайтами з розширеними функціями та взаємодією з користувачем, такими як онлайн-шопінг, соціальні мережі та системи керування контентом [2].

З появою смартфонів мобільні пристрої стали викликом для веб розробки. Це призвело до появи Веб 2.0. Однією з проблем була висока форма екрану, а не широка. Таким чином маленькі посилання та кнопки важко натискати пальцем. Вебсайт, який не оптимізований для мобільних пристроїв, практично неможливо використовувати на смартфоні. Google навіть почав карати сайти, які не були оптимізованими для мобільних пристроїв, понижуючи їх позиції в результатах пошуку. Це призвело до початку ери адаптивного дизайну у 2008 році та створення Веб 2.0.

У своїй статті журналістка Катерина Романенко описує Web2 як другу версію Інтернету, що відкрила можливість створення інтерактивних вебсайтів із вдосконаленим дизайном, які легко знаходяться через пошукові системи; а також мобільних застосунків, здатних зберігати, оновлювати та відображати дані в режимі реального часу через інтерфейси. Цей етап розвитку розпочався у 1995 році завдяки виникненню технологій, таких як AJAX та Javascript. Web2 можна вважати сучасною версією Інтернету, до якої ми звикли та яка визначає наше сучасне

спілкування з будь-якою платформою як Amazon, Facebook, Spotify або Discord [23].

До початку 2000-х системи керування контентом (CMS) почали домінувати на веб-ринку, і з'явилися відкриті системи керування контентом та фреймворки, такі як Drupal (2000), WordPress (2003), SquareSpace (2003), а пізніше Weebly (2006) та Wix (2006). Їх продовжують активно використовувати й надалі у Веб 2.0. За інформацією Themisle :

- 68,7% всіх вебсайтів використовують CMS.
- WordPress утримує величезну частину ринку CMS з часткою використання 62,8%, за ним слідують Shopify (6,2%) і Wix (3,8%).
- Wix є найшвидше зростаючою CMS, з темпом зростання частки ринку на 800% між 2016 і 2023 роками.
- Серед 1 000 найпопулярніших сайтів за трафіком найпоширенішими CMS є WordPress (47,3%) та Drupal (4,7%).

Станом на серпень 2024 року в Інтернеті налічується понад 78 мільйонів активних вебсайтів (BuildWith, 2024), більшість з яких працюють на системах керування контентом. WordPress домінує на ринку CMS з часткою 62,7%, що робить його значно популярнішим за будь-яку іншу платформу [7].

Єдиною іншою CMS з часткою ринку, що перевищує 5%, є Shopify. Після Shopify йдуть Wix, Squarespace та Joomla, які займають місця з частками ринку від 2,4% до 3,9% відповідно.

Вищезазначені дані стосуються лише вебсайтів, на яких встановлена система керування контентом (CMS). Однак є мільйони інших вебсайтів, які не використовують CMS, а замість цього покладаються на жорстко закодований HTML або інші веб-технології.

Якщо враховувати і вебсайти без CMS, то WordPress займає 43,3% частки ринку (MobiLoud, 2024), порівняно з 31% вебсайтів, які не використовують систему керування контентом [6].

Та попри таку популярність ці сервіси мають достатньо значний недолік. Для їх використання усе ще потрібні знання коду та програмування. Хоча ця проблема не настільки суттєва для Wordpress завдяки плагінам типу Visual Composer чи Divi Builder.

«WordPress, Wix, Squarespace, Weebly та інші конструктори сайтів не покривають складних і зростаючих вимог бізнесу. Компанії використовують продукти, такі як Office 365, SharePoint, SAP, Oracle WebCenter, Episerver, Sitecore та інші корпоративні хмарні й локальні платформи. Вони публікують контент частіше, ніж будь-коли, і витрати на це стають дедалі більшими. Ці платформи не мають розкоші зручних у використанні конструкторів сторінок. Отже, наразі існує потреба в створенні сучасного веб-контенту. І не тільки для WordPress, але й для інших платформ. Це пояснюється тим, що бізнес зараз є більш гнучким і орієнтованим на цифрові рішення, тому існує важлива необхідність створювати сучасний веб-контент набагато швидше і на різних платформах. Іншими словами, їм потрібен конструктор сторінок для кожної платформи.», — зазначає Самі Аль Саєд у своїй статті [2].

Сучасна розробка вебсайтів впроваджує штучний інтелект (AI) та машинне навчання для створення персоналізованого користувацького досвіду. Віртуальна реальність (VR) і доповнена реальність (AR) почали впливати на веброзробку, відкриваючи нові можливості для створення занурювальних вражень. Прикладами цього у вебдизайні є [9]:

- IKEA Place(Рисунок 1.5). Застосунок доповненої реальності (AR), який дозволяє користувачам побачити, як меблі від IKEA виглядатимуть у їхньому домі. Користувачі можуть просто навести телефон на кімнату та побачити, як різні елементи меблів вписуються у простір.
- Google Expedition. VR-застосунок, який дозволяє користувачам досліджувати різні куточки світу. Користувачі можуть здійснювати віртуальні екскурсії до таких місць, як Великий Бар'єрний риф чи Тадж-Махал.

- Застосунок The New York Times VR дозволяє користувачам по-новому переживати новини. Користувачі можуть дивитися VR-відео, які занурюють їх у події, що відбуваються, або в історичні моменти.

Галузь продовжує швидко розвиватися, і розробники активно досліджують найновіші технології та стандарти.

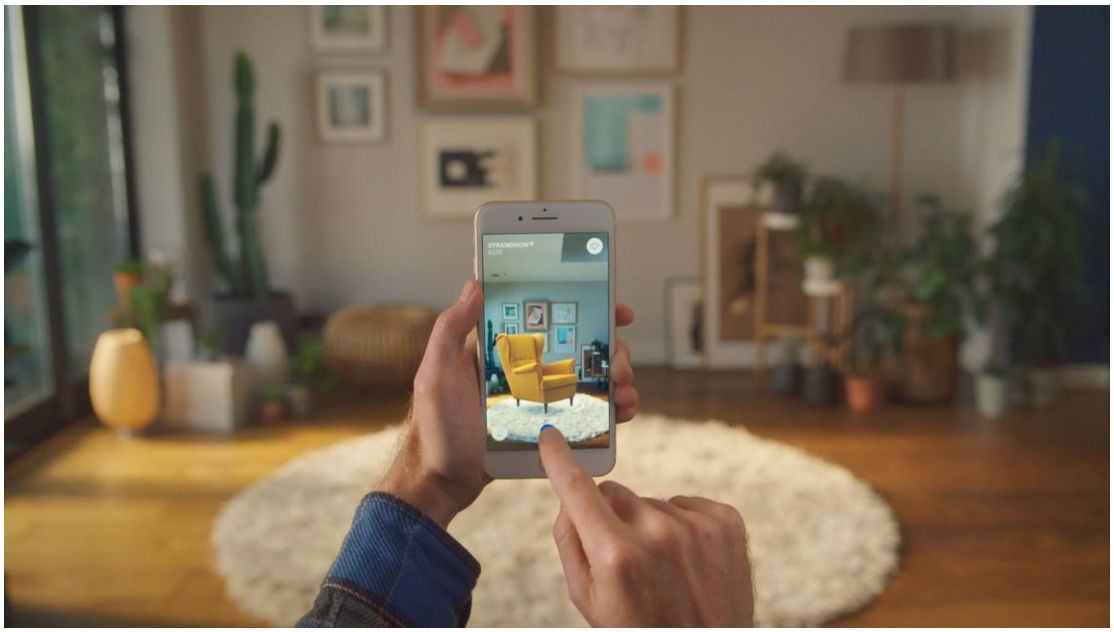


Рисунок 1.5 − IKEA Place [11]

1.2 Цикл розробки та роль тестування в ньому

Життєвий цикл вебсайту — це послідовність етапів і дій, які виконуються для розробки вебсайту.

Є сім етапів розробки вебсайту чи вебзастосунку(Рисунок 1.6) [15]:

- ретельне планування та стратегічний аналіз вимог;
- складання переліку вимог;
- дизайн і прототипування;
- повноцінна веброзробка;
- тестування;
- розгортання;
- обслуговування.

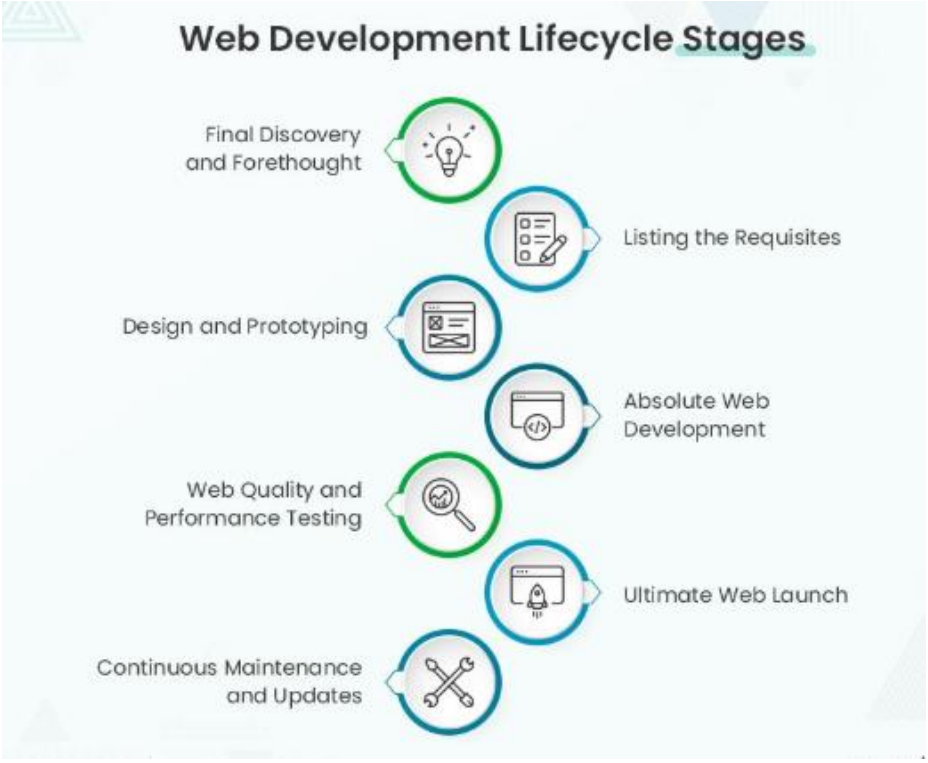


Рисунок 1.6 – Життєвий цикл web-розробки [15]

Визначення параметрів продукту, цілей та термінів є початковим етапом та чи не найважливішим, адже саме від цього й залежатимуть основні вимоги, портрет цільової аудиторії та подальша розробка та якість продукту. Чітке розуміння мети та кінцевого результату допоможе вибудувати структурований план проєкту та визначити необхідні етапи створення сайту для досягнення цієї мети [21].

Методи, такі як опитування, інтерв'ю та тести на зручність використання, допомагають виявити потреби користувачів. Інтерв'ю із замовниками надають інформацію про цілі бізнесу, цільовий ринок і очікування від продукту [15]. Обов’язково необхідно проаналізувати рішення, що вже існують. Це допоможе визначити що є важливим для користувачів та покращити конкурентоспроможність продукту.

Цей етап також включає оцінку ризиків і заходи з управління ними у разі їх виникнення. Результатом є повноцінний опис проєкту з визначеними цілями, необхідними функціями, технологіями, платформами та попереднім графіком робіт.

Під час цього етапу важливими аспектами є визначення цільової аудиторії, проведення аналізу конкурентів та формулювання графіків проєкту. Міцна основа, створена завдяки ретельному плануванню та збору вимог, є запорукою успішного початку веб-розробки.

На етапі складання переліку вимог або ж, як його ще називають, розробки технічного завдання, розробляється вичерпний та класифікований перелік вимог клієнта до конкретного продукту.

Технічне завдання – це офіційний документ та фундамент для подальшої роботи; в ньому прописуються всі деталі: структура або мапа сайту (кількість сторінок, розділів, категорій, блоків), вимоги стосовно дизайну, функціонального, візуального та текстового наповнення, а також технічні можливості [21].

Визначаються функціональні вимоги, які система має виконувати, наприклад, автентифікація користувачів, контроль доступу на основі ролей, функціонал електронної комерції, можливості управління контентом та інтеграція сторонніх систем.

Це допомагає визначити такі фактори, як час відгуку, заходи безпеки (HTTPS, SSL чи шифрування) та доступність. У цій категорії надаються рекомендації щодо типів контенту (текст, графіка, відео тощо) і способів їхнього управління та класифікації.

Також визначають технічні засоби, які будуть використані, наприклад, мови програмування, фреймворки (React, Angular), бібліотеки та бази даних (SQLite, MongoDB), а також інтеграція сторонніх API (платіжні шлюзи, авторизація через соцмережі). Документація проєкту дає змогу презентувати вимоги й потреби всім залученим сторонам для ефективної реалізації дизайну та розробки.

На стадії проектування та створення прототипу визначається фактичний зовнішній вигляд веб-рішення. Дизайнери UI/UX створюють макет, який перетворює ідеї на реальний об’єкт. Для цього вони розроблено декілька ескізів, обов’язково дотримуючись вимог, викладених в технічному завданні[21].

Інтерактивні прототипи створюються за допомогою інструментів, таких як Figma. Цей етап передбачає багаторазове узгодження дизайну із зацікавленими сторонами для його покращення з точки зору бізнесу та користувача..

Застосовуються принципи адаптивного дизайну, щоб забезпечити легкий доступ до сайту для різних користувачів і пристроїв. У результаті формується набір дизайн-ресурсів і специфікацій, які використовуються на етапі розробки.

Якщо дизайн уже є узгодженим починається наступний етап – розробка, або як ще це називають, верстка. Під час цього етапу фронтенд-розробники працюють над інтерфейсом користувача, а бекенд-розробники займаються функціональними завданнями та інтеграцією баз даних. Але, звісно, все це залежить від технічного завдання, адже можливе використання і CMS типу WordPress, а не тільки програмування з нуля. Результат верстки можна вважати інструментом, що працює. Але при цьому це буде інструмент з порожніми сторінками, тож їх необхідно наповнити вмістом(текстовими й графічними матеріалами). При цьому важливо дотримуватися стандартів пошукової оптимізації вебсайтів, адже це напряму впливатиме на трафік та просування продукту в пошукових системах, оскільки від цього залежить чи зможе користувач його знайти.

Тестування — це етап, що включає різноманітні перевірки на наявність помилок, некоректного функціонування та загальної працездатності ресурсу. Виявлені помилки виправляються спеціалістами доти, поки вони не будуть повністю усунуті [21].

Після ретельного перегляду та тестування вебзастосунку настає найзахопливіший момент – фактичний запуск. Для доступності вебдодатка для користувачів по всьому світу його необхідно завантажити на сервер. Також йому необхідно вибрати доменне ім’я. Підключення домену до хостингу – крок, який можна зробити в будь-який час [21]. Зазвичай це роблять на на початку. Перед запуском проводиться перевірка за чеклістом, щоб переконатися, що всі технічні деталі, дизайнерські елементи та контент відповідають задуму.

Після розміщення сайту в інтернеті обов'язково проводять заключне тестування для перевірки його працездатності.

Кожен процес розгортання проходить через CI/CD-пайплайн, щоб забезпечити плавний і ефективний реліз програмного забезпечення. Після розгортання впроваджуються інструменти моніторингу, такі як Google Analytics або New Relic, для оцінки ефективності сайту, активності користувачів і вирішення можливих проблем.

Життєвий цикл веброзробки не закінчується після запуску. Після розгортання вебсайту чи вебзастосунку важливо не зупинятися на досягнутому. Регулярні перевірки допомагають підтримувати високий рівень роботи системи.

На цьому етапі проводиться постійна оптимізація за допомогою таких сервісів, як Google Analytics, технічні аудити вебсайтів, що дозволяють виявляти слабкі місця.

Додатково потрібно виправляти наявні баги, покращувати функціональність та постійно оновлювати контент відповідно до нових потреб бізнесу чи користувачів. Не варто нехтувати регулярним резервним копіювання даних. Воно дозволяє уникнути проблем із відновленням у разі збою.

Таким чином якщо постійно оновлювати та підтримувати вебсайт він залишатиметься актуальним для користувачів.

1.3 Поняття та мета тестування

З однієї точки зору, тестування є процесом дослідження та випробування програмного продукту. Його призначення — демонстрація розробникам і замовникам того, що розроблена програма відповідає визначеним вимогам, і виявлення ситуацій, у яких поведінка програми є неправильною або не відповідає специфікації [22, с.8].

Водночас, тестування є перевіркою відповідності між реальним результатом та очікуваним. Воно здійснюється на кінцевому наборі тестів, обраному певним чином [22, с.8].

Тестування вебсайтів — це процес перевірки функціональності, безпеки та продуктивності сайту, щоб забезпечити його відповідність заданим вимогам [19].

Є різні способи класифікації тестування(Рисунок 1.7). Далі розглянемо ті, які найбільше використовуються.

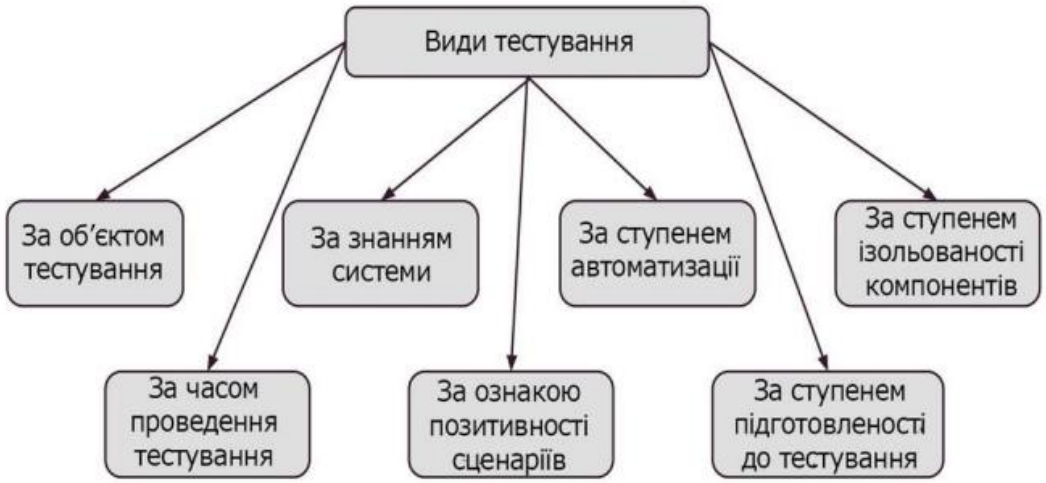


Рисунок 1.7 − Види тестування [22, с.9]

За об'єктом тестування виділяють наступні види тестування [22,с.9]:

- функціональне тестування;
- тестування продуктивності;
- тестування навантаження;
- стрес-тестування;
- тестування стабільності;
- конфігураційне тестування;
- юзабіліті-тестування;
- тестування інтерфейсу користувача;
- тестування безпеки;
- тестування локалізації;
- тестування сумісності.

Функціональне тестування є загальним терміном для багатьох типів тетування, які розглядають заздалегідь вказану поведінку і ґрунтуються на аналізі специфікацій функціональності компонентів або системи в цілому [22, с.9-10]

Тестування навантаження - це автоматизоване тестування, що дозволяє зрозуміти чи може продукт працювати стабільно при навантаженнях у допустимих межах і в межах, які трохи перевищують значення допустимих.

Стресове тестування – тестування, яке дозволяє перевірити надійність системи, її працездатність в умовах стресового навантаження, і зважити здатність системи повертатись до норми після його припинення . Стресовим навантаженням є екстремальні чи непропорційні навантаження, які перевищують розрахунковий рівень. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності [22, с.9-10].

Тестування стабільності (надійності) - це тестування, завданням якого є перевірка працездатності програми при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження [22, с.10].

Конфігураційне тестування (ConfigurationTesting) - спеціальний вид тестування, спрямований на перевірку роботи програмного забезпечення при різних конфігураціях системи (заявлених платформах, підтримуваних драйвери, при різних конфігураціях комп'ютерів і т.д.) [22, с.10].

Тестування зручності користування - це метод тестування, спрямований на визначення зручності та простоти використання для користувачів продукту. Метою юзабіліті тестування є забезпечення зручного та дружнього, тобто інтуїтивно зрозумілого, інтерфейсу для користувачів, щоб вони могли легко знайти необхідну їм інформацію та виконувати завдання. Юзабіліті тестування складається з тестування інтерфейсу користувача(UI Testing) та з тестування досвіду користувача User Experience (UX). Перше перевіряє коректність та загальну функціональність елементів інтерфейсу. Для вебсайтів це означає перевірити всі елементи інтерфейсу(текстові форми, навігаційні меню). А друге зосереджується на виявленні чи достатньо інтерфейс зрозумілий та зручний для кінцевого користувача та чи правильно функціонують його технічні елементи.

Тестування безпеки - це тестування, яке зосереджено на пошуці багів, що пов’язані із збереженням даних користувача. Воно використовується для перевірки

безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту програм, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних [22, с.11].

Тестування взаємодії - це один із типів функціонального тестування, який перевіряє здатність продукту взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності і інтеграційне тестування [22, с.11]

За рівнем знання про систему виділяють наступні види тестування: чорний ящик(BlackBox), білий ящик (WhiteBox) і сірий ящик (GreyBox).

Тестування чорного ящика (Black Box Testing) — це метод перевірки програмного забезпечення, який передбачає оцінку функціональних можливостей програми без необхідності знання її внутрішньої структури, коду або технічних деталей реалізації. Цей підхід зосереджується виключно на аналізі вхідних і вихідних даних, базуючись на вимогах і специфікаціях програмного забезпечення. Тестування чорного ящика також називають зовнішнім, закритим або поведінковим тестуванням [25]. Така назва через те, що тестувальник не знає нічого про систему, отже вона для нього є дійсно «чорним непрозорим ящиком»

Найбільш відомими видами тестування чорного ящику є функціональне тестування та нефункціональне тестування, а також регресійне тестування.

Переваги:

- може допомогти виявити неточності і протиріччя в специфікації, оскільки тестування проводиться з позиції кінцевого користувача [20];
- знання мов програмування не є обов’язковими [20] ;
- можливо проводити тестування і фахівцями, що не залежать від відділу розробки, чим допомагає уникнути упередженого ставлення [20];
- як тільки готова специфікація уже можна починати писати тест-кейси [20];
- низька деталізація документації;
- витрачає менше ресурсів порівняно з тестуванням білого ящику [25];

- ефективніший спосіб тестування великих частин продукту завдяки достатньо низьким вимогам до технічного боку тестування [25].

Недоліки:

- перевірити можливо лише обмежену кількість вхідних даних [25];
- через відсутність чіткої специфікації достатньо складно скласти ефективні тест-кейси [20];
- деякі тести можуть виявитися надмірними, якщо вони вже були проведені розробником на рівні модульного тестування [20];
- масштаб тестування є обмеженим, що впливає на отримані результати [25];

Тестування білого ящика White Box Testing — це метод перевірки програмного забезпечення, який зосереджується на внутрішньому функціонуванні системи. У цьому підході тестувальники мають доступ до структури, коду і будови програмного забезпечення, що дозволяє їм аналізувати, як система працює зсередини. Тестування білого ящика також відоме як тестування чистого ящика, тестування відкритого ящика, тестування прозорого ящика, тестування на основі коду та тестування скляного ящика [20]. Причиною таких назв є наявність усіх відомостей про систему.

Найбільш популярниими видами тестування білого ящика є модульне, циклічне та тестування на витік пам’яті.

Переваги:

- можна проводити тестування на ранніх етапах: немає потреби чекати створення інтерфейсу, призначеного для користувача [20];
- можливо забезпечити більш ретельне тестування, яке покриє велику кількість шляхів виконання програми [20];

Недоліки:

- для використання цієї техніки тесстування потрібна велика кількість спеціальних знань [20];

- при використанні автоматизації тестування на цьому рівні, якщо програма часто змінюється, підтримка тестових скриптів може виявитися досить накладною [20];
- є досить ресурсозатратним;

Метод сірого ящика - це техніка тестування, яка по своїй суті є комбінацією WhiteBox і BlackBox підходів. Внутрішня структура програми частково відома. Наприклад, є доступ до внутрішньої структури та алгоритмів роботи програмного забезпечення для написання максимально ефективних тест-кейсів [20]. Проте відбуватиметься тестування вже з використанням техніки чорного ящика, а отже з позиції користувача. Ось чому цей метод і називають методом напівпрозорого ящика: щось видно, а щось - ні. Цю методику можливо застосовати на різних рівнях тестування - від модульного до системного, але головним чином він використовується на інтеграційному рівні для перевірки взаємодії різних модулів програми [20].

Тестування за ступенем автоматизації поділяється на ручне (manual testing), автоматизоване (automation testing) та напівавтоматизоване (semiautomated testing). Ручне тестування повністю виконується людиною, та автоматизоване передбачає використання спеціальних інструментів для виконання тестів із мінімальною участю людини.

1.4 Ручне тестування проти автоматизованого

Часто ручне та автоматизоване тестування протиставляють настільки, що виникає помилкове враження що є тільки два варіанти: або це повністю ручне тестування(виконується тільки людиною), або це повністю автоматизоване тестування, що виконується з допомогою спеціальних інструментів. Але це не так. Це окремі способи тестування, які мають свої переваги й недоліки. Більше того, немає сенсу використовувати лише автоматизоване тестування під час тестування, адже це лише ускладнить його і витрачатиметься зайвий час на непотрібні процеси.

Не варто витрачати час на автоматизацію:

- нестабільних програмних продуктів, що знаходяться на стадії розробки та піддаються змінам;
- сценаріїв, які будуть рідко виконуватися;
- аналізу програмного коду і документації. Автоматизація цих процесів може в подальшому тільки додати клопоту [24].

Перевагами автоматизованого тестування є:

- вища точність та більша швидкість виявлення дефектів;
- економія зусиль і часу;
- збільшує покриття тестами;
- можливість запланувати тестовий прогін [24];
- можливість повторного використання вже написаного скрипту;
- допомагає поліпшити ROI (коефіцієнт окупності інвестицій).

Хоча автоматичне тестування має і вищу швидкість, оскільки має мінімальне втручання людини, одночасно з цим відсутність людської точки зору є недоліком такого виду тестування. Це робить його неможливим для використання для деяких тестів. Тому автоматичне тестування і є неуніверсальним інструментом та ніколи не замінить повністю ручне тестування. Оскільки найефективніше використовувати автоматизацію для процесів, що повторюються, то знову ж таки перед початком автоматизації необхідне ручне тестування.

Процеси, які можуть бути автоматизованими [24]:

- регресійне тестування;
- димове тестування;
- тестування продуктивності;
- тестування навантаження;
- тестування методом білого ящика (за допомогою інструментів модульного тестування);
- аналіз покриття коду.

2 ІНСТРУМЕНТИ ДЛЯ АВТОМАТИЧНОГО ТЕСТУВАННЯ

2.1 Мови програмування

Java, Javascript та Phyton є найпопулярнішими мовами для автоматичного тестування. Але також часто використовується і C#.

C# — це мова програмування, створена компанією Microsoft, яка поступово набуває популярності в автоматизації тестування. Вона заснована на концепціях об'єктно-орієнтованого програмування та є однією з найпоширеніших мов у рамках платформи .NET.

Згідно з опитуванням розробників Stack Overflow у 2019 році, 67% респондентів вважають C# найкращою мовою для автоматизації тестування, веб-розробки та інших завдань. C# підходить для автоматизації додатків, які працюють на платформах Android, Windows та iOS. На сьогодні актуальною є версія C# 8.0. Завдяки своїм потужним можливостям і сумісності із Selenium WebDriver ця мова стає все більш популярною серед тестувальників для автоматизації та кросбраузерного тестування [5].

Найпоширенішими фреймворками для автоматизації тестування на C# є [5].:

- NUnit
- MSTest
- xUnit.Net

Python — це мова програмування з відкритим кодом, яка чудово підходить як для новачків, так і для досвідчених розробників. Вона проста у вивченні, а її синтаксис легкий для розуміння, що дозволяє зосередитися на вирішенні задач, а не на технічних деталях.

Python використовується для автоматизації тестування як фронтенду, так і бекенду програмного забезпечення. Оскільки код легко читати, то це сприяє швидкому пошуку рішень. Бібліотеки, які пропонує Python, забезпечують величезні можливості для автоматизації, зокрема для тестування.

Ця мова знаходить застосування у розробці додатків для вбудованих систем та IoT-пристроїв. Python також широко використовується в автоматизації завдань системного адміністрування (DevOps). Ця мова програмування користується неабиякою популярністю у Big Data, машинному навчанні та штучному інтелекті.

Java — це універсальна мова програмування для автоматизації, яка належить корпорації Oracle. Вона базується на принципах об'єктно-орієнтованого програмування та слідує концепції WORA (Write Once, Run Anywhere), що забезпечує значні переваги для кросплатформенних рішень.

Багато великих компаній використовують Java для підтримки своїх бекенд-систем. Зараз існує понад 3 мільярди пристроїв, на яких працюють програми, створені за допомогою Java.

Хоча JUnit є популярним фреймворком для модульного тестування, на базі Java було створено безліч інших відкритих фреймворків для автоматизації тестування. Зокрема, автоматизоване тестування веб-продуктів (сайтів чи веб-додатків) можна виконувати, використовуючи JUnit разом із Selenium WebDriver.

Вибір мови програмування є важливим питанням, але немає єдиної правильної відповіді. При цьому не варто на цьому зациклюватись, адже мова програмування це всього лиш інструмент в руках тестувальника та багато чого залежить саме від його знань й умінь, а також від вимог проєкту. Та й автоматизація тестування це не завжди про написання автоматизованих тестів, адже технології розвиваються та зараз є багато інструментів, які взагалі не потребують знання програмування та написання коду.

2.2 Інструменти для автоматизації тестування

Надважливо обрати правильний інструмент для автоматизації тестування, але ця задача не є простою. Для ухвалення рішення потрібно враховувати вимоги до проєкту та такі фактори як:

- підтримка платформ та мов програмування;

- вартість та ліцензування;
- чи можливо використовувати інструмент для усіх необхідних видів тестування;
- простота використання;
- можливість інтеграції з різними системами(Jenkins, TestRail).

Наприклад, компанія EPAM у своїй роботі використовує такі інструменти:

- Selenium;
- VIVIDUS;
- Cypress;
- KatalonStudio;
- Playwright;
- WebdriverIO;
- Appium;
- інструмент їх власної розробки ReportPortal.

Katalon Studio — це простий інструмент для автоматизації тестування з підтримкою тестування на основі ключових слів (KDD). Він використовує визначену структуру артефактів, таких як тестові приклади, об'єкти та звіти, що полегшує створення тест-кейсів. Також є можливість додавати спеціальні ключові слова, запускати тести паралельно чи послідовно для прискорення процесу.

Appium – це фреймворк з відкритим вихідним кодом. Його використовують для автоматизованого тестування мобільних застосунків як для платформи Android, так і для iOs. Підтримка усіх цих платформ є його перевагою. Він дозволяє тестувати не тільки нативні застосунки, а й гібридні та веб. Для цього він використовує WebDriver API. За допомогою нього він отримує змогу взаємодіяти з мобільними застосунками аналогічно до взаємодії вебдрайвера зі сторінками вебсайту.

Cypress — це сучасний фреймворк для автоматизованого тестування вебзастосунків, що здобув свою популярність через свою орієнтованність на простоту та швидкість. Усі інструменти він об'єднує в одному вікні, що забезпечує легке налаштування, а його інтуїтивно-зрозумілий інтерфейс дозволяє початківцям

безперешкодно використовувати його для своєї роботи. Цей фреймворк надає детальні звіти для швидкого пошуку та виправлення помилок.

Фреймворк WebdriverIO побудовано на основі Selenium. Він є open source проєктом, тобто має відкритий код, та є власністю некомерційної організації OpenJS Foundation. Призначенням фреймворку є автоматизація тестування веб та мобільних застосунків. Серед переваг фреймворку можна зазначити його підтримку і інших сучасних фреймворків, як React і Angular. А наявність величезної різноманітності плагінів, створених спільнотою, надає можливість інтегрувати й розширювати налаштування системи відповідно до потреб проєкту. Ще однією його перевагою є те, що він не тільки запускає автоматизацію на основі WebDriver, але й додаткового використовує вбудовані API браузера, що дозволяє інтегрувати Chrome DevTools.

Playwright — це новий фреймворк, який є розробкою компанії Microsoft. Він набуває популярності зараз. Фреймворк створений для тестування вебзастосунків, при чому підтримує він як однопотокові(сценарії, що виконуються послідовно), так і багатопотокові(сценарії, що виконуються одночасно) сценарії. Він є кросбраузерним та підтримує Chromium, WebKit та Firefox. Підтримує і такі платформи як Windows, Linux та macOS. Його кросбраузерність та кросплатформеність можна вважати однією з його переваг. Іншою його перевагою є підтримка асинхронних операцій, що забезпечує ефективну роботу з багатопотоковими сценаріями тестування [26]. Проте, цей фреймворк є молодшим за Selenium та WebdriverIO, тому він має меншу кількість плагінів та меншу спільноту.

Серед серед застосунків-акселераторів EPAM інструмент власної розробки компанії, ReportPortal, має найбільший відсоток впровадження, що зображено на рис 2.1. Цей інструмент є платформою для аналітики та звітності результатів автоматизованого тестування, яка поширюється на умовах open source [26].

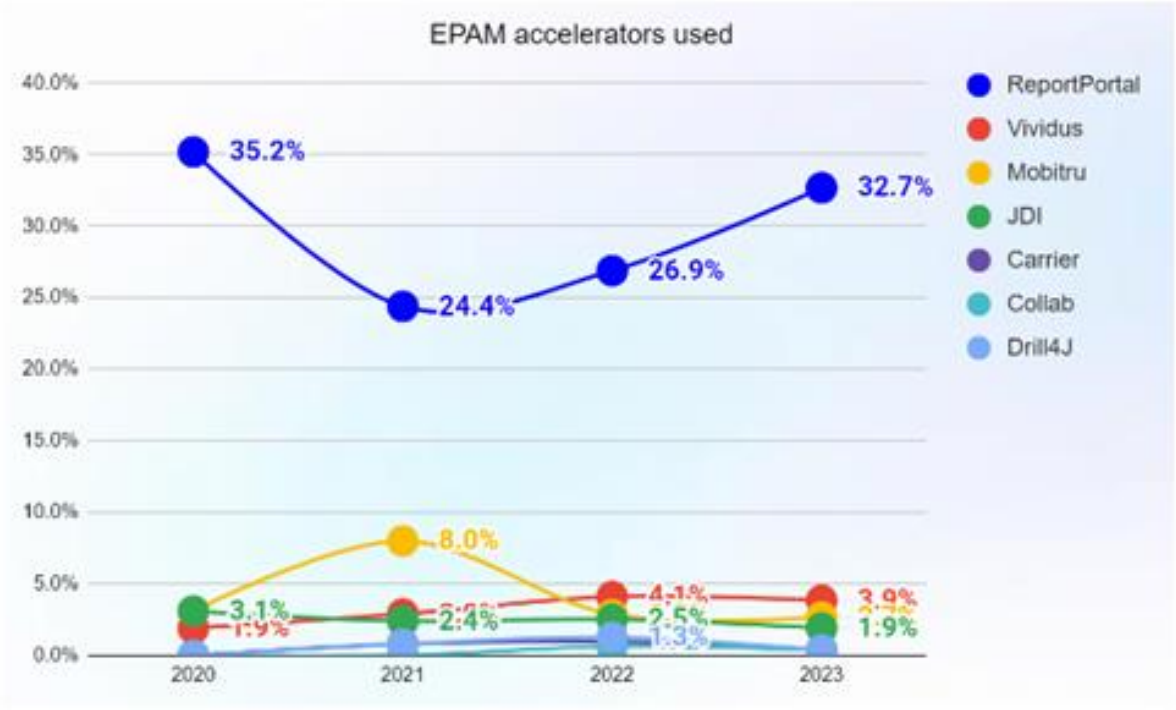


Рисунок 2.1 − Акселератори в компанії EPAM [26]

Можливостями платформи є збирання, зберігання, аналізування та звітування про результати автоматизованих тестів. Він забезпечує централізований доступ до всієї інформації про тестування, що полегшує виявлення проблем і прийняття рішень [26].

ReportReport забеспечує [26]:

- збір даних з різних джерел, включаючи фреймворки автоматизованого тестування, системи управління тестами та інструменти для моніторингу;
- зберігання даних в централізованій базі даних, що забезпечує легкий доступ до них;
- надання різних інструментів для аналізу даних, включаючи звіти, графіки та діаграми;
- створення звітів про дані з автоматизованого тестування.

Його перевагами є [26]:

- централізований доступ до даних: ReportPortal забезпечує централізований доступ до всієї інформації про тестування, що полегшує виявлення проблем і прийняття рішень;
- відстеження ефективності тестування: ReportPortal дозволяє відстежувати ефективність тестування, що допомагає підвищити якість програмного забезпечення;
- автоматизація звітності: ReportPortal дозволяє автоматизувати звітність про дані з автоматизованого тестування, що економить час і зусилля.

І нарешті Selenium. Це однин із найпопулярніших інструментів для автоматичного тестування, який є проєктом з відкритим кодом, сформованим спільнотою користувачів та Open Source контриб'юторів, які розробляють, використовують і просувають різні проєкти Selenium (IDE, Grid, WebDriver) та мають мету принести користь спільноті в цілому [17].

Його історія почалася ще в 2004 році в ThoughtWorks у Чикаго, де Джейсон Хаггінс створював Core режим під назвою JavaScriptTestRunner для тестування внутрішнього додатку Time and Expenses (Python, Plone). Автоматизоване тестування будь-яких додатків є основою стилю роботи ThoughtWorks, враховуючи їхню орієнтацію на Agile. Йому допомагаюли Пол Гросс і Джі Тіна Ван.

Джейсон почав демонструвати інструмент тестування своїм колегам. Багато з них були вражені його негайним і інтуїтивно зрозумілим візуальним зворотним зв'язком, а також його потенціалом стати багаторазовою тестовою платформою для інших вебзастосунків.

Невдовзі, у 2004 році, колега з ThoughtWorks Пол Хемант побачив демонстрацію і почав обговорювати ідею відкриття Selenium з вихідним кодом, а також визначення driven режиму Selenium, що дозволяв використовувати цей інструмент через мережу з будь-якої мови, яка вибиралася, оминаючи політику same origin policy. Інші (тоді) колеги, Аслак Хеллесой і Майк Мелія, експериментували з різними ідеями для частини сервера, включаючи переписування сторінок для обходу цієї політики. Пол написав початкову серверну

частину на Java, а Аслак і Обі Фернандез перенесли клієнтську частину на Ruby, заклавши основу для драйверів і на інших мовах.

ThoughtWorkers в різних офісах по всьому світу почали використовувати Selenium для комерційних проєктів і ділитися досвідом, здобутим у цих проєктах, що сприяло розвитку Selenium. Майк Вільямс, Даррелл Дебоер і Даррен Коттерілл допомогли покращити його можливості та надійність.

Хоча цей інструмент є достатньо старим, він все ще тримає планку та є вагомим конкурентом для нових інструментів. Він постійно оновлюється та пристосовується до нових вимог світу технологій, а тому його все ще активно використовують і такі технічні гіганти як Amazon та Netflix.

На рисунках 2.2 та 2.3 зображено порівняння Selenium із фреймворком Playwright, який зараз набуває популярності серед тестувальників.

Критерії	Selenium	Playwright
Мови програмування	Java, Python, C#, Ruby, Perl, PHP і JavaScript	Java, Python, .NET C#, TypeScript і JavaScript.
Браузери	Chrome, Safari, Firefox, Opera, Edge	Chromium, Firefox і WebKit
Підтримка спільноти	Велика кількість документації та велика спільнота	Менша спільнота та кількість документації, але вони постійно збільшуються
Необхідність вагомих знань	Велика спільнота полегшує автоматизацію, але все одно потрібні знання з програмування	Створений таким чином, що зручний навіть для новачків
Швидкодія	Середня	Швидкий
Асинхронний підхід	Немає	Є

Рисунок 2.2 − порівняння Selenium із Playwright частина 1

Перевагами Selenium є підтримка більшої кількості браузерів, більша спільнота та кількість документації. При цьому кількість останніх у Playwright збільшується.

Оскільки Playwright новіший фреймворк, він має і новішу архітектуру, побудовану із врахуванням сучасних технологій. Через це він швидший. Також використання цього фреймворку надає можливість використання асинхронного підходу.

Playwright створений таким чином, щоб новачки безперешкодно могли ним користуватись при роботі. Для використання Selenium необхідний більший багаж знань, але наявність величезної кількості документації та активної спільноти полегшує його використання.

Selenium також підтримує більше операційних систем, ніж Playwright. У свою чергу останній має позитивну особливість, яка відсутня у Selenium. Мова йде про автоматичне очікування завантаження елементів.

При використанні Selenium час очікування потрібно прописувати в скриптах вручну, але не завжди його можна підібрати одразу вдало. Це називається явне очікування або неявне очікування. Неявне очікування є глобальним очікуванням, яке WebDriver застосовує до всіх дій пошуку елементів. Якщо елемент недоступний, WebDriver чекатиме заданий час, перш ніж кидати виняток. Явне очікування дозволяє вам очікувати певної умови для конкретного елемента. Обидва підходи дозволяють уникнути проблем із тим, що елементи можуть ще не бути доступними на сторінці. Для динамічних сторінок краще використовувати явне очікування, а використання обох видів у одному тесті краще уникати, бо це може стати причиною конфлікту. А в Playwright очікування завантаження елементів є автоматичним.

можливість додавати плагіни	Є та не обмежують у цьому користувачів	Є
Автоматичне очікування елементів	Немає, потрібно вклазувати при написанні тестів	Автоматично очікує завантаження елементів
Підтримка реальних пристроїв	Доступна черз хмару та віддалені сервіси	Немає, але є підтримка емуляції
ОС	Windows, Linux, macOS, and Solaris	Windows, Linux, and macOS

Рисунок 2.3 − порівняння Selenium із Playwright частина 2

2.3 Огляд набору інструментів Selenium

До складу набору інструментів Selenium входить:

- SeleniumIDE;
- Selenium Grid;
- Selenium WebDriver.

До складу раніше також входив Selenium RC, але цей інструмент закрили у версії Selenium 3. Його використовували до появи WebDriver як інструмент для віддаленого керування браузером. По своїй суті він просто ін’єктував у браузер шматки Javascript коду, тобто в основу були закладені абсолютно інші підходи і функціонально цей інструмент також поступався WebDriver.

WebDriver розроблений як простий та більш лаконічний інтерфейс програмування та є компактним об'єктно-орієнтованим API. Він ефективно керує браузером.

Selenium Web Driver у Selenium 3 містить 4 основні компоненти(Рисунок 2.4) [13]:

- Selenium Client Library
- JSON WIRE PROTOCOL Over HTTP
- ClientBrowser Drivers
- Browsers

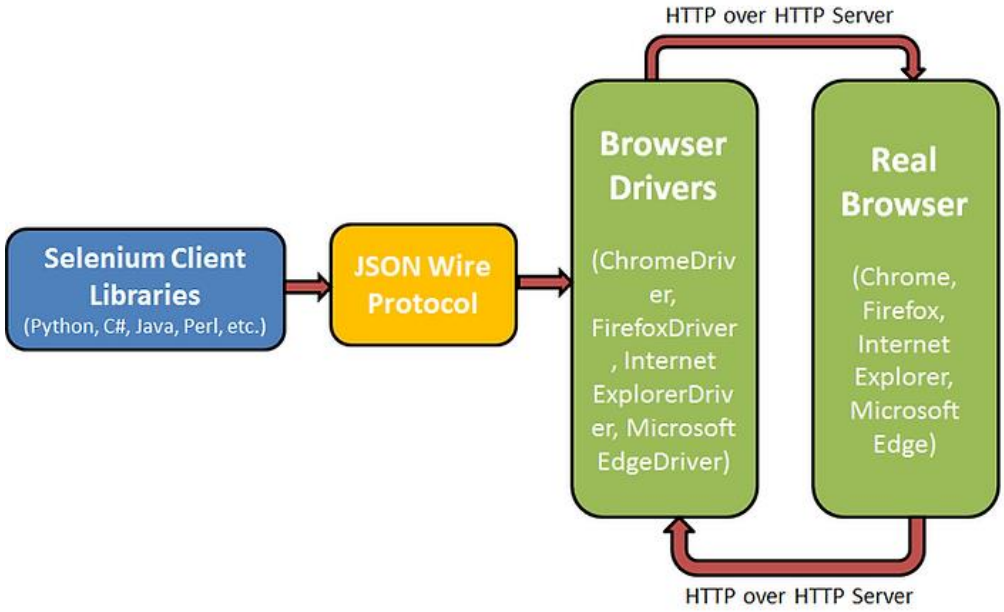


Рисунок 2.4 − Складові Selenium Web Driver Selenium 3 [13]

Є сервер web driver, специфічний для кожного браузера, який насправді відповідає за перетворення команд тестових скриптів в http команди та обмін повідомленнями через json wire protocol.

Для успішної роботи з Selenium Web Driver спочатку необхідно встановити Selenium bindings для тої мови програмування, яку планується використовувати при автоматизації тестування. Розробники інструменту розробили їх спеціально для кожної мови програмування, яку він підтримує. Це є Selenium Client Libraries/Language Bindings. Детальніше про це є в документації(Рисунок 2.5).

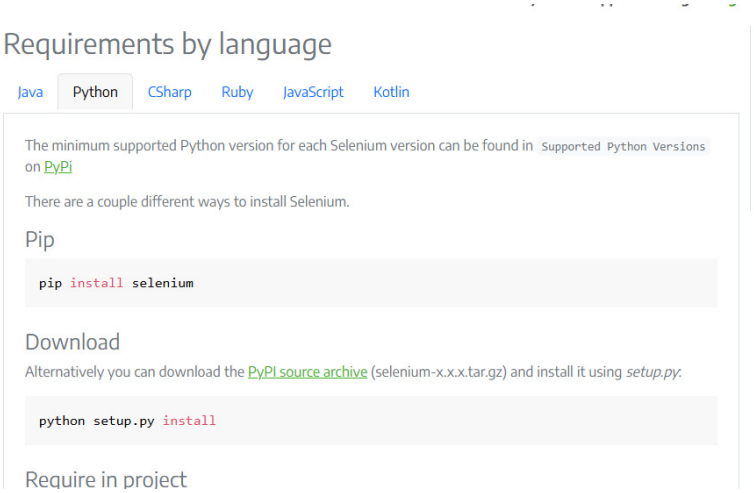


Рисунок 2.5 − Інструкції з установки Language Bindings в документації [18]

JSON означає JavaScript Object Notation. Він використовується для передачі даних між сервером і клієнтом в Інтернеті. JSON Wire Protocol — це REST API, який передає інформацію між HTTP сервером. Кожен драйвер браузера (наприклад такий як FirefoxDriver) має власний HTTP сервер.

Кожен браузер містить окремий драйвер браузера(Рисунок 2.6). Драйвери браузерів взаємодіють з відповідним браузером, не розкриваючи внутрішню логіку функціонування браузера. Коли драйвер браузера отримує будь-яку команду, то ця команда буде виконана у відповідному браузері, і відповідь повернеться у вигляді HTTP відповіді. Драйвер браузера також необхідно скачати й інсталювати перед початком роботи.

У Selenium 3 немає прямого зв'язку між клієнтськими бібліотеками (Java, JavaScript тощо) і драйверами браузерів. Сервер (драйвери браузерів) не розуміє мови, а лише протоколи, а клієнтські бібліотеки, в свою чергу, не розуміють протоколи, які використовують драйвери браузерів.

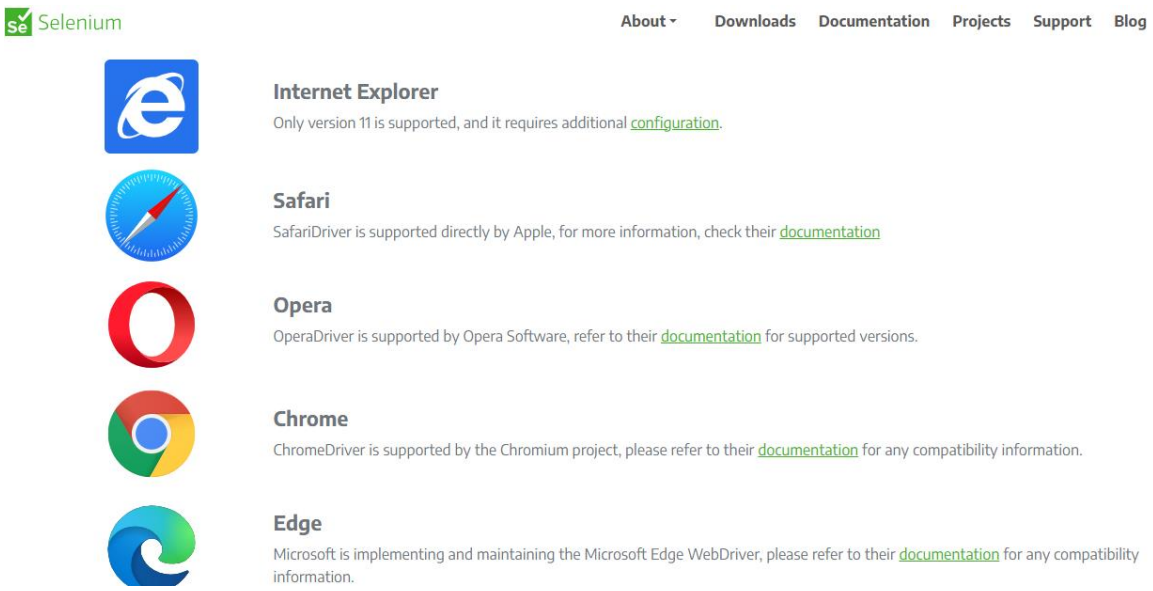


Рисунок 2.6 − Доступні драйвери браузерів [18]

Використання JSON Wire protocol як посередника між клієнтом і сервером для кодування та декодування запитів і відповідей, зроблених відповідно клієнтом і сервером призводило до обмеженої взаємодії з браузером, неефективного зв'язку та відсутності стандартизації, що в кінцевому підсумку призводило до ненадійних тестів і їх повільнішого виконання.

Надалі у Selenium 4 почав використовуватись W3C протокол(рисунок 2.7), а не JSON для зв’язку між клієнтськими бібліотеками та драйверами браузера.

W3C розшифровується як World Wide Web Consortium. Це міжнародна спільнота, що розробляє та підтримує стандарти та рекомендації для всесвітньої павутини. Головною метою спільноти є забезпечення довгострокового зростання та сумісності Інтернету.

Усі браузери та драйвери браузерів, окрім Selenium 3 WebDriver, в архітектурі Selenium відповідають W3C. Отже, для кодування та декодування запитів і відповідей використовується протокол JSON Wire. Selenium 4 WebDriver став сумісним з W3C, щоб зробити зв’язок між клієнтськими бібліотеками та драйверами браузера простим і прямим. Поліпшення комунікації призвело до більшої стабільності.

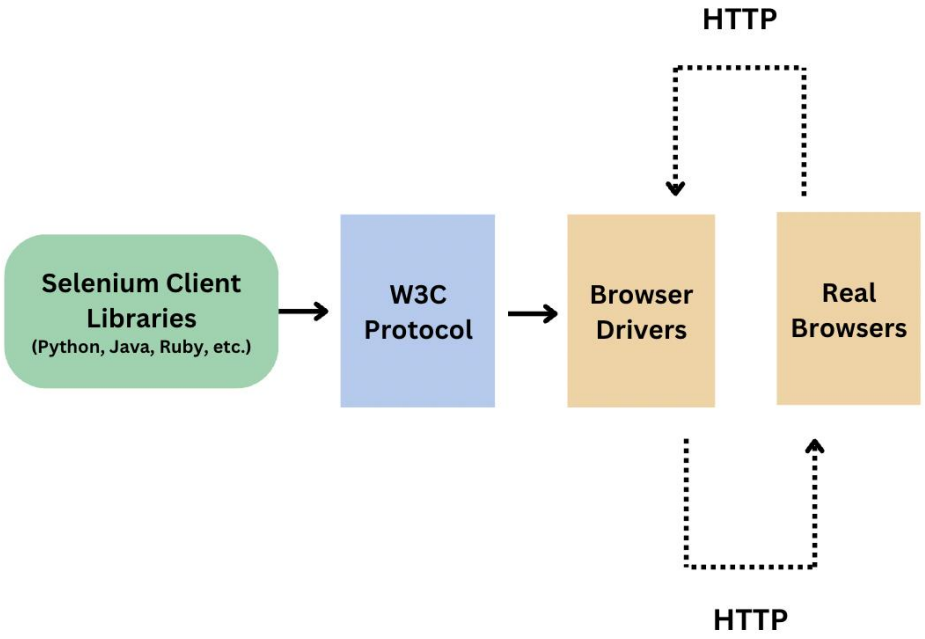


Рисунок 2.7 − Архітектура Selenium 4 [4]

Зв’язок між клієнтом та браузером можна описати так [4]:

- Запит на виконання команди надсилається клієнтом Selenium (тестовий сценарій, написаний на обраній мові програмування) для виконання різних дій у веб-переглядачі.
- Клієнт WebDriver серіалізує запит у стандартизований формат, визначений протоколом WebDriver. Цей формат може бути JSON або подібним, залежно від конкретної реалізації.
- Серіалізований запит передається драйверу браузера, який діє як міст між клієнтом веб драйвера і браузером.
- Драйвер браузера обробляє серіалізований запит, а потім виконує необхідні дії з браузером.
- Драйвер браузера генерує відповідь на виконання команди, яка включає відповідні дані або інформацію(наприклад статус і статус успіху чи невдачі).
- Драйвер браузера серіалізує відповідь у стандартизований формат за допомогою протоколу WebDriver і передає її назад клієнту.

Клієнт отримує відповідь від драйвера браузера та десеріалізує відповідь. Він витягує відповідну інформацію, і клієнт може використовувати цю інформацію для перевірки успіху / невдачі виконання команди.

Selenium IDE можна назвати версією фреймворку Selenium, що має графічний інтерфейс. У Selenium 3 цей інструмент було можливо використовувати лише з Firefox. Зараз він доступний як розширення і для інших браузерів(включно з Chrome).

Для того щоб використовувати його не потрібно знати жодну мову програмування. Він працює подібно до бота. Користувач виконує кроки тестування, а Selenium IDE записує їх. Пізніше він їх відтворює. При цьому інтерфейс є зручним та дружнім до користувача, тож прекрасно підійде для новачків.

Зазвичай, якщо використовують цей інструмент, то тільки для найпростіших сценаріїв. Більш складні тести не є надійними у ньому.

Selenium IDE дозволяє запускати всі ваші тести на будь-якому браузері, паралельно з допомогою на Grid. Для цього потрібно встановити Selenium IDE Command-Line Runner, інсталювати драйвери для браузера(якщо тести виконуватимуться локально, а не на серверах Selenium) та запустити runner через командний рядок із потрібними параметрами. Також можна генерувати код для Selenium WebDriver і використовувати його для тестування.

Selenium Server дозволяє керувати браузером з віддаленої машини через мережу. Можливо запустити тести на браузерах, недоступних на вашому локальному девайсі. Спочатку на пристрої, де повинен працювати браузер, встановлюється і запускається сервер. Потім на іншій машині запускається програма, що використовує спеціальний драйвер RemoteWebDriver, з’єднується з сервером і надсилає йому команди. Сервер, у свою чергу, запускає браузер і виконує команди, використовуючи відповідний драйвер браузера.

Selenium Grid — це компонент Selenium, який дозволяє виконувати автоматичне тестування на кількох машинах та в різних браузерах паралельно. Він використовує архітектуру Hub-and-Node, де Hub — це центральний сервер, який

керує виконанням тестів і приймає запити, а Node — це клієнтська машина, яка виконує тести у вказаному браузері та середовищі.

Одне з завдань Selenium Grid - «підбирати» відповідний вузол, коли під час запуску браузера вказуються його вимоги: тип браузера, версія, операційна система, архітектура процесора і інші атрибути.

Цей інструмент підтримує паралельне тестування, тому його чато використовують для кросбраузерного тестування. Також він має можливість централізованого виконання тестів(з одного хабу кількома вузлами), що буде особливо корисним для великих команд.

3 РОЗРОБКА ТЕСТОВИХ СЦЕНАРІЇВ

3.1 Визначення вимог тестування

Тестуватиметься сайт(Рисунок 3.1) провідного дитячого видавництва А-БА-БА-ГА-ЛА-МА-ГА https://store.ababahalamaha.com.ua/.

Сайт має інтуїтивно зрозумілий інтерфейс, приємний дизайн. Наявні функції реєстрації профілю та відповідно входу в нього. Його основним функціоналом є перегляд каталогу товарів, можливість додати товар у кошик, переглянути товар у кошику, керувати товарами у кошику, можливість оформити замовлення та одразу сплатити. Ці функції працюють як для користувачів, що зайшли у свій профіль, так і для гостей сайту.

Для того щоб додавати товари у список бажань спочатку необхідно пройти реєстрацію, про що користувач дізнається, якщо ще не входив у свій профіль чи його не має. Є верхнє навігаційне меню, зверху якого є Header. З самого низу сторінки розміщено статичний Footer.

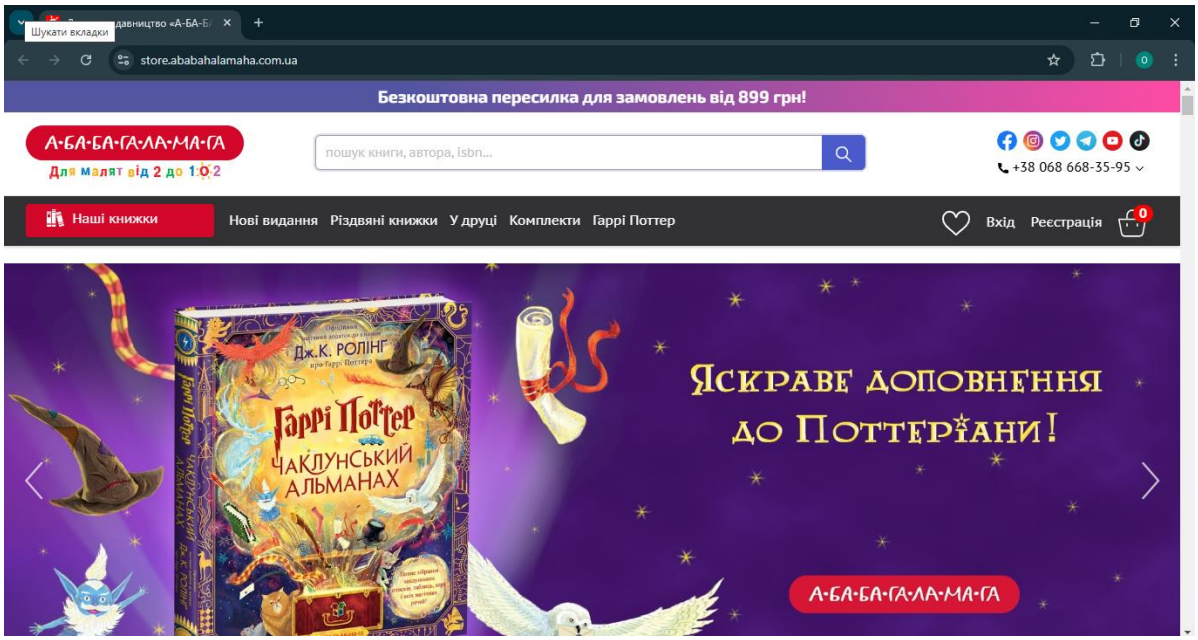


Рисунок 3.1 – Вигляд сайту

У хедері є поле для пошуку з пропозиціями за ключовими словами, контактна інформація, логотип, посилання на соціальні мережі. У футері знову ж таки є таки

є контактна інформація, гіперпосилання на найнеобхідніші сторінки для користувачів та поле для введення електронної пошти, якщо користувач бажає підписатись на розсилку.

Перед автоматизацією тестування завжди слід проводити спочатку ручне тестування, щоб знати як поводить себе середовище, що тестується. Проводилось функціональне тестування, що охоплювало основні функції вебсайту. Тест-кейси додавання товару в кошик, додавання товару в кошик із сторінки товару зображені на рис 3.2.

Test Case ID	Опис	Кроки	Очікуваний результат	Фактичний результат	Статус
1	Додавання товару в кошик	1. Відкрити сайт https://store.ababahalamahaa.com.ua/ 2. Ввести в пошук "локвуд і ко" 3. Натиснути на кнопку пошуку 4. Натиснути на кнопку купити	Товар додано в кошик	Як і очікуваний	Passed
2	Додавання товару в кошик із сторінки товару	1. Відкрити сайт https://store.ababahalamahaa.com.ua/ 2. Ввести в пошук "локвуд і ко" 3. Натиснути на кнопку пошуку 4. Натиснути на зображення обкладинки або на назву книги 5. Натиснути на кнопку купити	Товар додано в кошик	Як і очікуваний	Passed

Рисунок 3.2 – Тест-кейси 1 та 2

Негативне тестування підписки на розсилку(тестування неправильними даними) зображено на рисунку 3.3.

3	Негативне тестування підписки на розсилку	1. Відкрити сайт https://store.ababahalamahaa.com.ua/ 2. Проскролити до кінця сторінки 3. У текстове пола "Ваша електронна адреса вписати: " dfrejgej@owpfn" 4. Натиснути кнопку підписатися	Користувач отримує інформацію про неправильний формат електронної адреси	Як і очікуваний	Passed

Рисунок 3.3 – Тест-кейс 3

Тест-кейси для тестування входу в акаунт та додавання товару у список побажань зображено на рисунку 3.4.

4	Вхід у акаунт	Передумова: є зареєстрований профіль 1. Відкрити сайт https://store.ababahalamaha.com.ua/ 2. Натиснути кнопку Вхід 3. Ввести значення в поля логін та пароль, які відповідають значенням попередньо зареєстрованого акаунта 4. Натиснути Увійти	Користувач має змогу увійти у свій акаунт	Як і очікуваний	Passed
5	Додавання товарів у список побажань	Передумова: є зареєстрований профіль, вхід у профіль здійснений 1. Відкрити сайт https://store.ababahalamaha.com.ua/ 2. Натиснути у друці 3. Навести на бажаний товар(Child Roland and Other Knightly Tales (англійською) 4. Натиснути на кнопку із зображенням сердечка	Товар додано у список бажань	Як і очікуваний	Passed

Рисунок 3.4 − Тест-кейси 4 та 5

На рисунку 3.5 зображено тест-кейс для додавання фільтрів та перевірки правильності відображення інформації.

6	Тестування фільтрів	1. Відкрити сайт https://store.ababahalamaha.com.ua/ 2. Натиснути кнопку **Гаррі Поттер** 3. Навести на блок фільтрів 4. Натиснути на **текстове поле “від”** **5.** ввести в текстове поле значення **500** **6.** Натиснути **кнопку Ок**	Товари відсортувалися у порядку ціни від 500. Усі товари що відповідають цим умовам присутні у списку	Товари відсортувалися, але зникли **гоґвортські видання “Гаррі Поттер і Філософський камінь**” ціна яких **500** грн.	Failed

Рисунок 3.5 − Тест-кейс 6

На рисунку 3.6 зображено тест-кейс для збільшення кількості товарів у кошику за допомогою кнопки із зображенням стрілки вгору. У випадку успішного виконання користувач зможе бачити у текстовому полі що кількість збільшилась(+1). Так можна додати кількість, використавши це текстове поле.

7	Збільшення кількості товарів у кошику	1. Відкрити сайт https://store.ababahalamaha.com.ua/ 2. Ввести в пошук "локвуд і ко" 3. Натиснути на кнопку пошуку 4. Натиснути на кнопку купити 5. Навести на кнопку із зображенням стрілки вгору 6. Натиснути на кнопку із зображенням стрілки вгору	Кількість товару у кошику збільшилась та користувач може спостерігати це в текстовому полі	Як і очікуваний	Passed

Рисунок 3.6 − Тест-кейс 7

На рисунку 3.7 зображено тестування видалення доданого товару з кошика. Товар має видалитися та не відображатись у кошику. У випадку порожнього кошика користувач має бачити повідомлення про це.

8	Видалення товарів із кошику	1. Відкрити сайт https://store.ababahalamaha.com.ua/ 2. Ввести в пошук “локвуд і ко” 3. Натиснути на кнопку пошуку 4. Натиснути на кнопку купити 5. Навести на кнопку із зображенням хрестика 6. Натиснути на кнопку із зображення хрестика	Товар успішно видалився з кошику та не відображається в ньому	Як і очікуваний	Passed

Рисунок 3.7 − Тест-кейс 8

3.2 Розгортання автоматичного тестування

Розпочнемо із налаштування Selenium WebDriver. Я обирав між мовами програмування Phyton та Java. Перевагою використання Java є те, що налаштувати проєкт для роботи можна було б просто у файлі pom.xml, якщо використовувати Maven, та лише просто вказавши залежності. Проте краще вона підходить для масштабних та складних проєктів завдяки особливостям об’єктно-орієнтованого програмування, які полегшують управління та підтримку складних тест-кейсів. Ц моєму ж випадку це прості тест-кейси. Також Java має набагато більшу спільноту підтримки, тож було б легше шукати рішення для проблем. Але код на Python є зрозумілим навіть новачкам, тож це не буде проблемою. Python має більший вибір бібліотек та модулів, а також його часто використовують у сферах машинного навчання та аналізу даних, тому при інтеграції з такими технологіями це також кращий вибір .

Java більше підходить для проєктів корпоративного рівня, які потребують високої масштабованості та інтеграції з іншими інструментами.

Java забезпечує кращу продуктивність і надійність у певних сценаріях, особливо при роботі з великими наборами даних і складними алгоритмами.

Оскільки це прості тест-кейси, то для виконання кваліфікаційної роботи використовуватиметься мова Python. На рисунку 3.8 зображені специфікації пристрою, що використовуватиметься для автоматизації тестування. Як браузер використовуватиметься найпопулярніший – GoogleChrome.

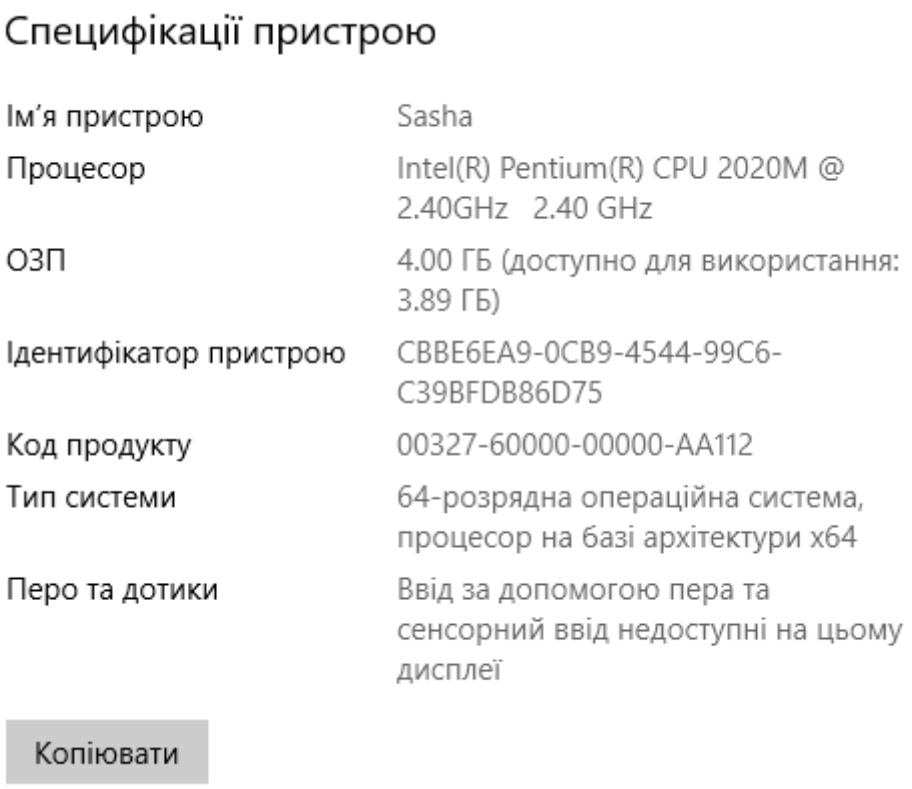


Рисунок 3.8 − Специфікації пристрою

Спочатку потрібно встановити Python. Для цього переходимо на сайт(Рисунок 3.9) https://www.python.org/downloads/ завантажуємо та інсталюємо його.

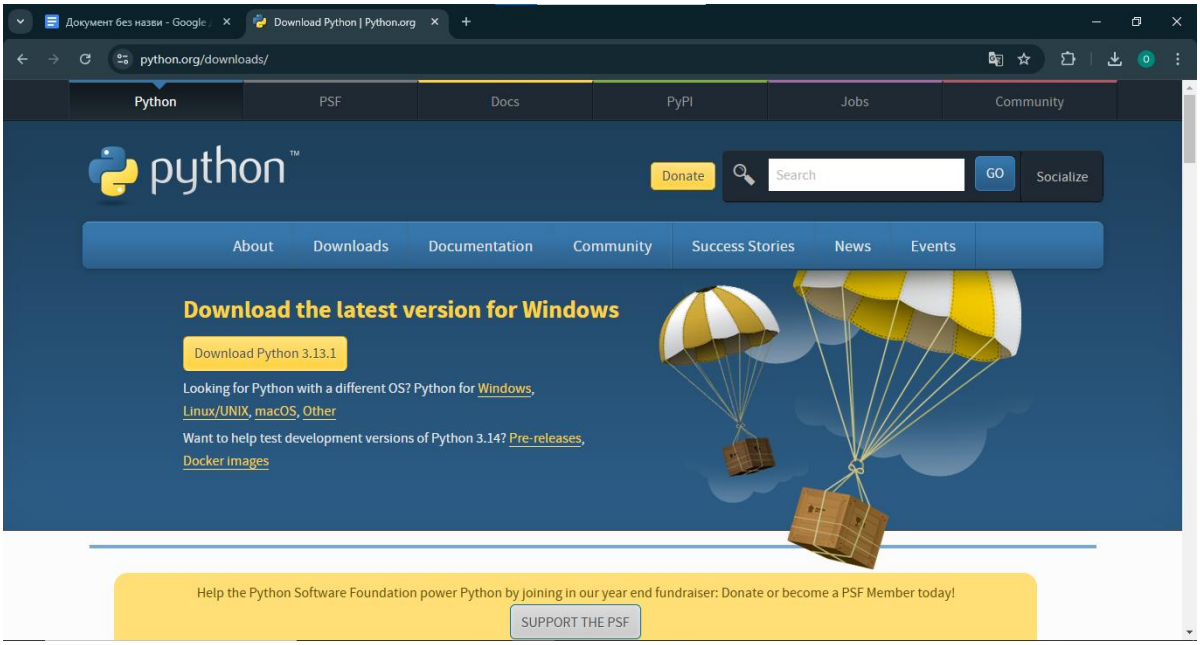


Рисунок 3.9 – Python.org

Рекомендується кастомізувати інсталяцію та одразу інсталювати pip, бо він буде потрібен для налаштування бібліотек.

Після встановлення відкриваємо командний рядок та вводимо команду py –version.

Microsoft Windows [Version 10.0.19045.5131]

(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\comp>py --version

Python 3.13.1

Далі перевіряємо версію pip та чи він взагалі встановлений. Це робимо за допомогою команди py -m pip –version.

```
C:\Users\comp>py -m pip --version
pip 24.3.1 from C:\Users\comp\AppData\Roaming\Python\Python313\site-packages\pip (python 3.13)
```

Далі за допомогою команди py -m pip install selenium інсталюємо бібліотеку Selenium.

```
C:\Users\comp>py -m pip install selenium
Defaulting to user installation because normal site-packages is not writeable
Collecting selenium
```

```
      Downloading selenium-4.27.1-py3-none-
any.whl.metadata (7.1 kB)
    Collecting urllib3<3,>=1.26 (from urllib3
[socks]<3,>=1.26->selenium)
      Downloading urllib3-2.2.3-py3-none-any.whl.metadata
(6.5 kB)
    Collecting trio~=0.17 (from selenium)
      Downloading trio-0.27.0-py3-none-any.whl.metadata
(8.6 kB)
    Collecting trio-websocket~=0.9 (from selenium)
      Downloading trio_websocket-0.11.1-py3-none-
any.whl.metadata (4.7 kB)
    Collecting certifi>=2021.10.8 (from selenium)
      Downloading certifi-2024.8.30-py3-none-
any.whl.metadata (2.2 kB)
    Collecting typing_extensions~=4.9 (from selenium)
      Downloading typing_extensions-4.12.2-py3-none-
any.whl.metadata (3.0 kB)
    Collecting websocket-client~=1.8 (from selenium)
      Downloading websocket_client-1.8.0-py3-none-
any.whl.metadata (8.0 kB)
    Collecting attrs>=23.2.0 (from trio~=0.17->selenium)
      Downloading attrs-24.2.0-py3-none-any.whl.metadata
(11 kB)
    Collecting sortedcontainers (from trio~=0.17-
>selenium)
      Downloading sortedcontainers-2.4.0-py2.py3-none-
any.whl.metadata (10 kB)
    Collecting idna (from trio~=0.17->selenium)
```

```
      Downloading idna-3.10-py3-none-any.whl.metadata (10
kB)
    Collecting outcome (from trio~=0.17->selenium)
      Downloading outcome-1.3.0.post0-py2.py3-none-
any.whl.metadata (2.6 kB)
    Collecting sniffio>=1.3.0 (from trio~=0.17->selenium)
      Downloading sniffio-1.3.1-py3-none-any.whl.metadata
(3.9 kB)
    Collecting cffi>=1.14 (from trio~=0.17->selenium)
      Downloading cffi-1.17.1-cp313-cp313-
win_amd64.whl.metadata (1.6 kB)
    Collecting wsproto>=0.14 (from trio-websocket~=0.9-
>selenium)
      Downloading wsproto-1.2.0-py3-none-any.whl.metadata
(5.6 kB)
    Collecting pysocks!=1.5.7,<2.0,>=1.5.6 (from urllib3
[socks]<3,>=1.26->selenium)
      Downloading PySocks-1.7.1-py3-none-any.whl.metadata
(13 kB)
    Collecting pycparser (from cffi>=1.14->trio~=0.17-
>selenium)
      Downloading pycparser-2.22-py3-none-any.whl.metadata
(943 bytes)
    Collecting h11<1,>=0.9.0 (from wsproto>=0.14->trio-
websocket~=0.9->selenium)
      Downloading h11-0.14.0-py3-none-any.whl.metadata
(8.2 kB)
    Downloading selenium-4.27.1-py3-none-any.whl (9.7 MB)
       ---------------------------------------- 9.7/9.7 MB
6.9 MB/s eta 0:00:00
```

```
    Downloading certifi-2024.8.30-py3-none-any.whl (167
kB)
    Downloading trio-0.27.0-py3-none-any.whl (481 kB)
    Downloading trio_websocket-0.11.1-py3-none-any.whl (17
kB)
    Downloading typing_extensions-4.12.2-py3-none-any.whl
(37 kB)
    Downloading urllib3-2.2.3-py3-none-any.whl (126 kB)
    Downloading websocket_client-1.8.0-py3-none-any.whl
(58 kB)
    Downloading attrs-24.2.0-py3-none-any.whl (63 kB)
    Downloading cffi-1.17.1-cp313-cp313-win_amd64.whl (182
kB)
    Downloading PySocks-1.7.1-py3-none-any.whl (16 kB)
    Downloading sniffio-1.3.1-py3-none-any.whl (10 kB)
    Downloading wsproto-1.2.0-py3-none-any.whl (24 kB)
    Downloading idna-3.10-py3-none-any.whl (70 kB)
    Downloading outcome-1.3.0.post0-py2.py3-none-any.whl
(10 kB)
    Downloading sortedcontainers-2.4.0-py2.py3-none-
any.whl (29 kB)
    Downloading h11-0.14.0-py3-none-any.whl (58 kB)
    Downloading pycparser-2.22-py3-none-any.whl (117 kB)
    Installing collected packages: sortedcontainers,
websocket-client, urllib3, typing_extensions, sniffio,
pysocks, pycparser, idna, h11, certifi, attrs, wsproto,
outcome, cffi, trio, trio-websocket, selenium
      WARNING: The script wsdump.exe is installed in
'C:\Users\comp\AppData\Roaming\Python\Python313\Scripts'
which is not on PATH.
```

```
    Consider adding this directory to PATH or, if you
prefer to suppress this warning, use --no-warn-script-
location.
  Successfully installed attrs-24.2.0 certifi-2024.8.30
cffi-1.17.1 h11-0.14.0 idna-3.10 outcome-1.3.0.post0
pycparser-2.22 pysocks-1.7.1 selenium-4.27.1 sniffio-1.3.1
sortedcontainers-2.4.0 trio-0.27.0 trio-websocket-0.11.1
typing_extensions-4.12.2 urllib3-2.2.3 websocket-client-
1.8.0 wsproto-1.2.0
```

Попередження «WARNING: The script wsdump.exe is installed in 'C:\Users\comp\AppData\Roaming\Python\Python313\Scripts' which is not on PATH» можна ігнорувати, якщо для виклику бібліотеки використовуватиметься IDE. Якщо використовуватиметься командний рядок, то потрібно додати шлях до PATH.

Аналогічно встановлюємо pytest. C:\Users\comp>py -m pip install pytest.

```
  C:\Users\comp>py -m pip install pytest
  Defaulting to user installation because normal site-
packages is not writeable
  Collecting pytest
    Downloading   pytest-8.3.4-py3-none-any.whl.metadata
(7.5 kB)
  Collecting colorama (from pytest)
    Downloading             colorama-0.4.6-py2.py3-none-
any.whl.metadata (17 kB)
  Collecting iniconfig (from pytest)
    Downloading iniconfig-2.0.0-py3-none-any.whl.metadata
(2.6 kB)
  Collecting packaging (from pytest)
    Downloading  packaging-24.2-py3-none-any.whl.metadata
(3.2 kB)
  Collecting pluggy<2,>=1.5 (from pytest)
```

```
      Downloading    pluggy-1.5.0-py3-none-any.whl.metadata
(4.8 kB)
    Downloading pytest-8.3.4-py3-none-any.whl (343 kB)
    Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
    Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
    Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
    Downloading packaging-24.2-py3-none-any.whl (65 kB)
    Installing collected packages: pluggy, packaging,
iniconfig, colorama, pytest
      WARNING: The scripts py.test.exe and pytest.exe are
installed                                              in
'C:\Users\comp\AppData\Roaming\Python\Python313\Scripts'
which is not on PATH.
      Consider adding this directory to PATH or, if you
prefer to suppress this warning, use --no-warn-script-
location.
    Successfully installed colorama-0.4.6 iniconfig-2.0.0
packaging-24.2 pluggy-1.5.0 pytest-8.3.4
```

Після чого виконуємо команду C:\Users\comp>py -m pip list.

```
C:\Users\comp>py -m pip list
Package            Version
------------------ -----------
attrs              24.2.0
certifi            2024.8.30
cffi               1.17.1
colorama           0.4.6
h11                0.14.0
idna               3.10
iniconfig          2.0.0
outcome            1.3.0.post0
packaging          24.2
pip                24.3.1
pluggy             1.5.0
pycparser          2.22
```

```
PySocks            1.7.1
pytest             8.3.4
selenium           4.27.1
sniffio            1.3.1
sortedcontainers   2.4.0
trio               0.27.0
trio-websocket     0.11.1
typing_extensions  4.12.2
urllib3            2.2.3
websocket-client   1.8.0
wsproto            1.2.0
```

Тепер скачуємо архів ChromeDriver та розпаковуємо, де забажаємо. Перевіряємо чи все встановилось(Рисунок 3.10) та можемо переходити то тестування.

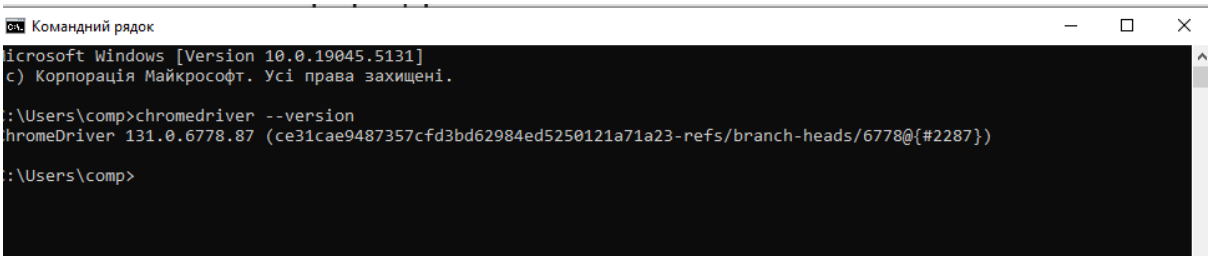


Рисунок 3.10 – Перевірка версії драйвера браузера

3.3 Автоматичне тестування

Для написання скриптів використовувались розроблені тест-кейси. Середовищем розробки було обрано PyCharm Community Edition 2023.2.8, оскільки він є зручним, автоматично підсвічує помилки синтаксису та дає підказки. Це допомагає уникнути простих помилок під час написання тестів і прискорює роботу. У PyCharm можна запускати окремі тести або весь набір тестів за допомогою кількох кліків. Також це середовище розробки є повністю безкоштовним та може інтегрватись із Git. Мовою програмування було обрано Python 3.10. Також використовувались фреймворки Selenium 4.27.1 та pytest 8.3.4. На рисунку 3.11 зображено налаштування проєкту в середовищі розробки PyCharm Community Edition 2023.2.8.

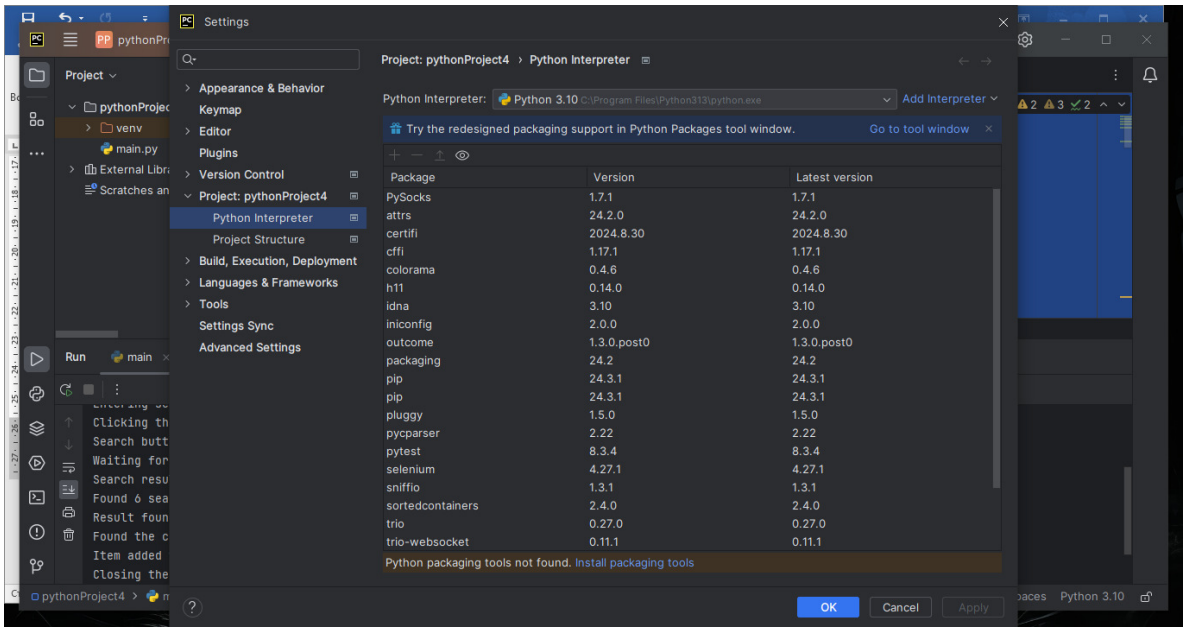


Рисунок 3.11 – Налаштування проєкту

Спочатку імпортуємо необхідні біблотеки.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
as EC
from time import sleep
```

Налаштовуємо веб драйвер та додаємо необхідне посилання.

```
try:
    # Setup WebDriver
    driver = webdriver.Chrome()
    print("Driver initialized.")

    # Navigate to the website
    print("Navigating to the website...")
    driver.get("https://store.ababahalamaha.com.ua/")
    print("Website loaded successfully.")
```

Вказуємо на розташування поля для пошуку.

```
    # Locate the search field
    print("Locating the search field...")
    search_field = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.CLASS_NAME,
"ant-input"))
    )
    print("Search field located successfully.")
```

Вводимо текст пошукового запиту та налаштовуємо клік кнопки пошуку.

```
    # Enter search query
    search_query = "локвуд і ко"
    print(f"Entering search query: {search_query}")
    search_field.send_keys(search_query)

    # Click the search button using the provided XPath
    print("Clicking the search button...")
    search_button = driver.find_element(By.XPATH,
                                        '//*
[@id="__layout"]/div/div [1]/header/section [1]/div [2]/div
[1]/div/div/div/div/ul/li/div/span [1]/span/button')
    search_button.click()
    print("Search button clicked.")
```

Очікуємо результатів пошуку.

```
    # Wait for the search results to load
    print("Waiting for search results...")
    WebDriverWait(driver, 15).until(
```

```
EC.presence_of_all_elements_located((By.CSS_SELECTOR,
".col-9_xs-12 a strong"))
    )
    print("Search results loaded successfully.")

    # Debugging: Let's print all search results to see if
we found the right elements
    search_results = driver.find_elements(By.CSS_SELECTOR,
".col-9_xs-12 a strong")
    print(f"Found {len(search_results)} search results.")
```

Додавання товару в кошик.

```
    for result in search_results:
        print(f"Result found: {result.text}")
        if "локвуд і ко" in result.text.lower():
            print(f"Found the correct book: {result.text}")
            buy_button = result.find_element(By.XPATH,

"./ancestor::div [contains(@class, 'block')]//button
[contains(@class, 'button-buy')]")
            buy_button.click()
            sleep(4)
            print("Item added to the cart successfully!")
            break
    else:
        print("No matching book found.")
Закриття браузера та драйвера.
except Exception as e:
    print(f"An error occurred: {e}")
finally:
```

```
print("Closing the browser.")
driver.quit()
```

Як видно на рисунку 3.12 тест скрипт відкрив браузер та автоматично ним керував, про що є повідомлення. Необхідні кроки тест-кейсу були успішно пройдені, а фактичний результат відповідав очікуваному.

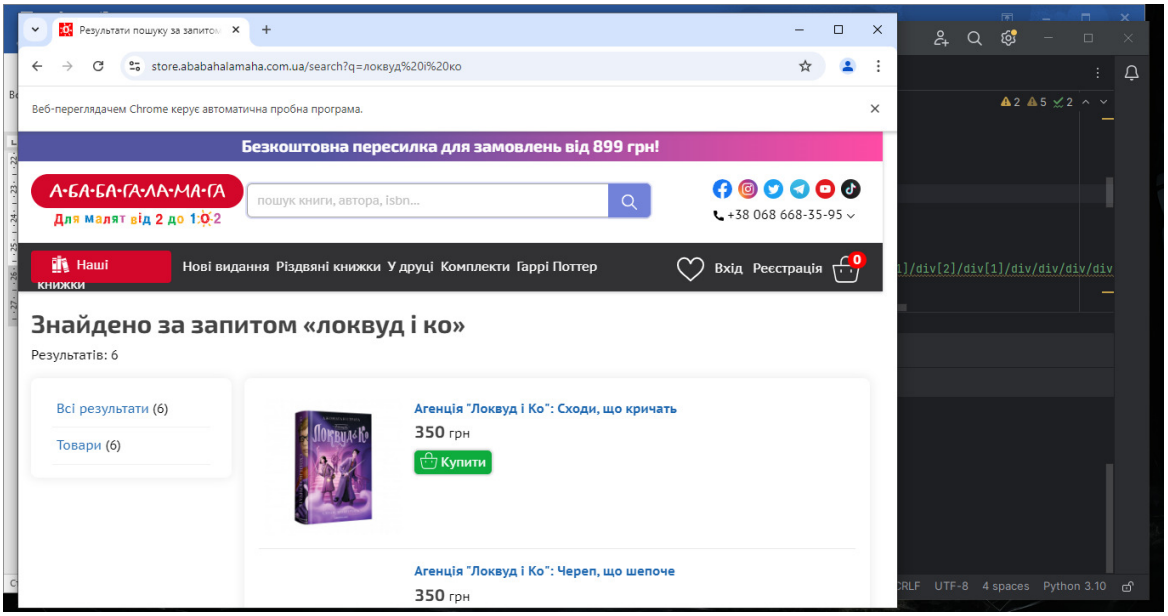


Рисунок 3.12 – Виконання скрипту для тест-кейсу 1

Ось що отрималось у терміналі:

- Driver initialized.
- Navigating to the website...
- Website loaded successfully.
- Locating the search field...
- Search field located successfully.
- Entering search query: локвуд і ко
- Clicking the search button...
- Search button clicked.
- Waiting for search results...
- Search results loaded successfully.
- Found 6 search results.
- Result found: Агенція "Локвуд і Ко": Сходи, що кричать
- Found the correct book: Агенція "Локвуд і Ко": Сходи, що кричать

- Item added to the cart successfully!
- Closing the browser.

Переходимо до автоматизованого тестування підписки на розсилку неправильними даними електронної пошти(тест-кейс номер 3).

```
# Прокрутка до футера
driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
time.sleep(5)

footer_textfield = driver.find_element(By.CSS_SELECTOR,
"input.ant-input.ant-input-lg")


# Ввести текст у поле
email = "dfrejgej@owpfn"
footer_textfield.send_keys(email)
print(f"Текст '{email}' успішно введено в поле.")

subscribe_button = driver.find_element(By.CSS_SELECTOR,
"button.ant-input-search-button")
subscribe_button.click()
time.sleep(5)


except Exception as e:
    print(f"Виникла помилка: {e}")
finally:
    # Закриття браузера
    driver.quit()
```

Як результат тест скрипт запустився. Усі кроки було пройдено, а очікуваний результат відповідав фактичному. Користувач отримав повідомлення про неправильний формат пошти.

Повторимо тестування, але тепер у поле електронної пошти введемо інші дані.

```
# Ввести текст у поле
email = "fkwln@exarthrh.crthm"
footer_textfield.send_keys(email)
print(f"Текст '{email}' успішно введено в поле.")
```

Результат можна спостерігати на рисунку 3.13. Попри те, що введений формат був неправильним та такої пошти й домену не існує, користувач отримав повідомлення що лист відправлено на пошту.

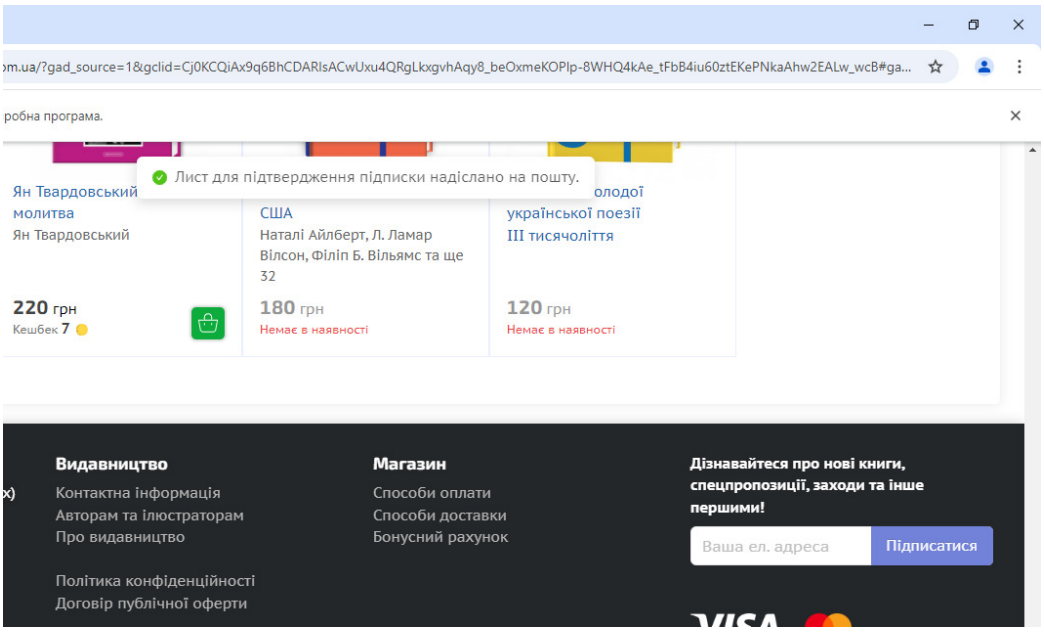


Рисунок 3.13 – Результат виконання скрипту для тест-кейсу 3

Перевіривши пошту за допомогою сервісу Zero Bounce можна переконатися що пошта має неправильний формат і не є дійсною(рис 3.14).Отже за допомогою цього тест скрипту було знайдено баг.

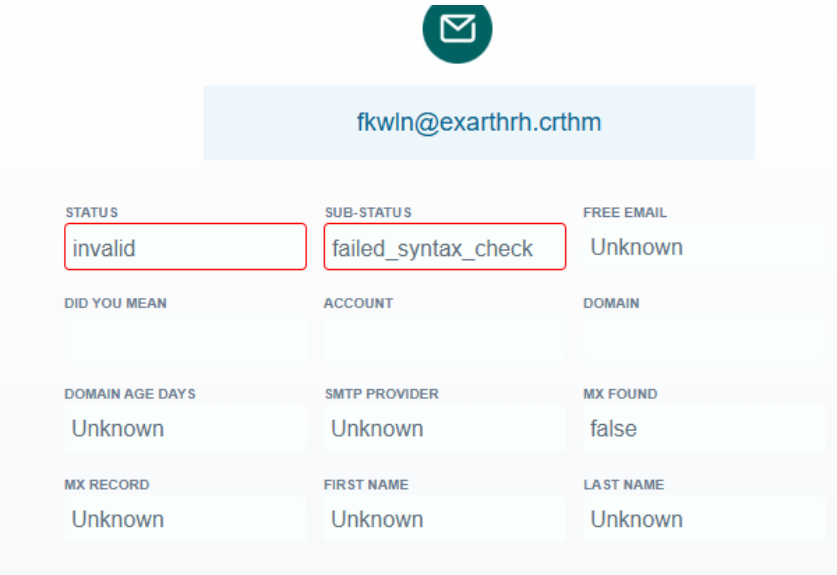


Рисунок 3.14 – Перевірка пошти

Автоматизуємо тест-кейс номер 4. На сайті при вході є перевірка з використанням інструменту reCAPTCHA, яку автоматична система керування не може пройти, через що тест-кейс буде неможливо виконати до кінця. Але вона не така жорстка та її можливо обійти, хоч і не завжди.

Для того щоб дії більше виглядали як ті, що виконує людина, було додано трохи часу очікування після вводу логіна та після вводу паролю.

```
# Клік по кнопці "Логін"
login_button = wait.until(
    EC.element_to_be_clickable((By.XPATH, '//*
[@id="__layout"]/div/div [1]/header/section [2]/div
[2]/ul/li [2]/a'))
)
login_button.click()

# Чекаємо, поки з'являться поля для логіну та пароля
username_field = wait.until(
    EC.visibility_of_element_located((By.XPATH, "//div
[2]/div/span/input"))
)
```

```
# Введення логіну
username_field.send_keys("your login")
time.sleep(5)

# Введення пароля
password_field = wait.until(
    EC.element_to_be_clickable((By.XPATH, '//div [2]/div
[2]/div/span/input'))
)
password_field.send_keys("password")
time.sleep(5)
```

У полях your login та password використовувались справжні дані. Результат зображено на рисунку 3.15.

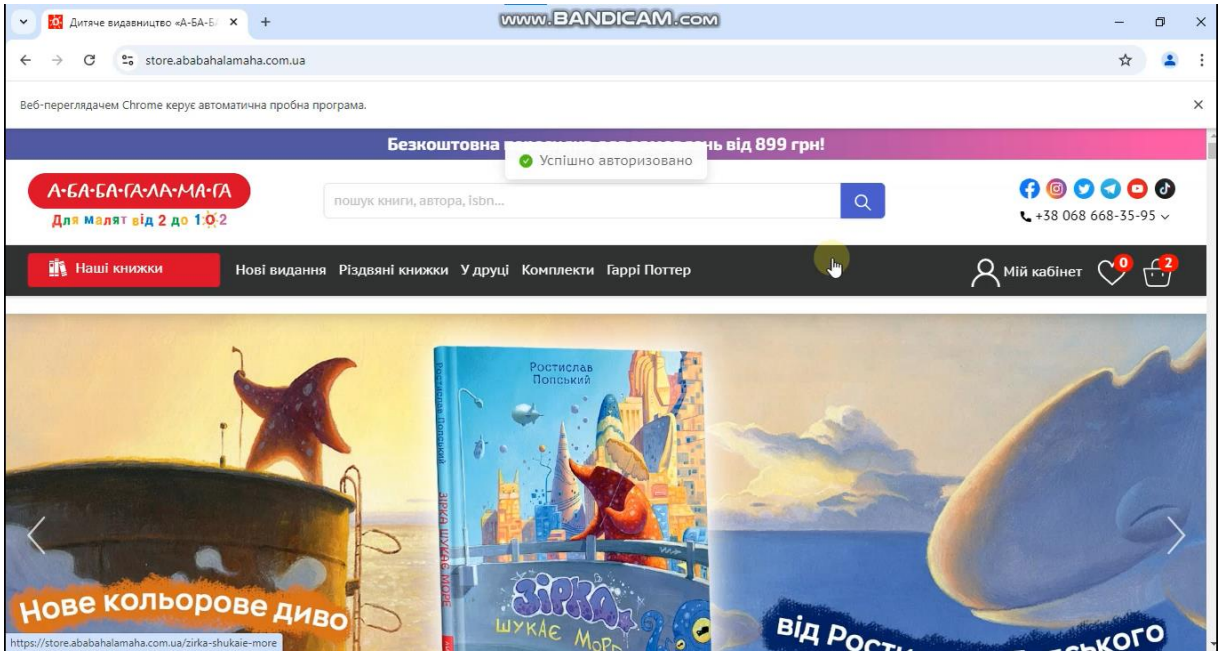


Рисунок 3.15 – Результат автоматизації тест-кейсу 4

Тепер автоматизуємо тест-кейс 5 та додамо обрану книгу в список бажань. Для цього спочатку налаштовуємо клік на кнопку «У друці».

```
in_print_button = wait.until(
    EC.element_to_be_clickable((By.XPATH, '//*
```

```
[contains(text(), "У друці")]'))
)
in_print_button.click()
```

Вказуємо XPath до необхідної книги.

```
# Знайти картку книги "Child Roland and Other Knightly
Tales (англійською)"
book_link = wait.until(
    EC.presence_of_element_located(
        (By.XPATH,

'//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1
]/div[13]//a[contains(text(), "Child Roland and Other
Knightly Tales (англійською)")]')
    )
)
```

Прокручуємо до необхідної книги. Цей крок зображено на рисунку 2.23.

```
driver.execute_script("arguments [0].scrollIntoView();",
book_link)
time.sleep(2)
```

Налаштовуємо клік на кнопку, що має вигляд іконки у формі сердечка.

```
      # Знайти кнопку "Додати в бажане" для цієї книги
favorite_button = wait.until(
    EC.element_to_be_clickable(
        (By.XPATH,

'//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1
]/div[13]//img[@src="/_nuxt/img/heart.665ede1.svg"]')
    )
)

# Клікаємо на кнопку "Додати в бажане"
favorite_button.click()
print("Кнопка 'Додати в бажане' натиснута.")
```

Відкриваємо список бажань.

```
wishlist_button = wait.until(
    EC.element_to_be_clickable((By.XPATH, '//img
[@alt="Скринька бажань"]'))
```

```
)
wishlist_button.click()
time.sleep(4)
```

Перевіряємо чи книга додалась у список бажань та чи це саме потрібна нам.

```
wishlist_book = wait.until(
    EC.presence_of_element_located(
        (By.XPATH, '//* [contains(text(), "Child Roland and
Other Knightly Tales (англійською)")]')
    )
)
time.sleep(4)
if wishlist_book:
    print("Книга 'Child Roland and Other Knightly Tales
(англійською)' успішно додана до списку бажань.")
else:
    print("Книга 'Child Roland and Other Knightly Tales
(англійською)' не знайдена у списку бажань.")
```

Результат цього тесту зображено на рисунку 3.16.

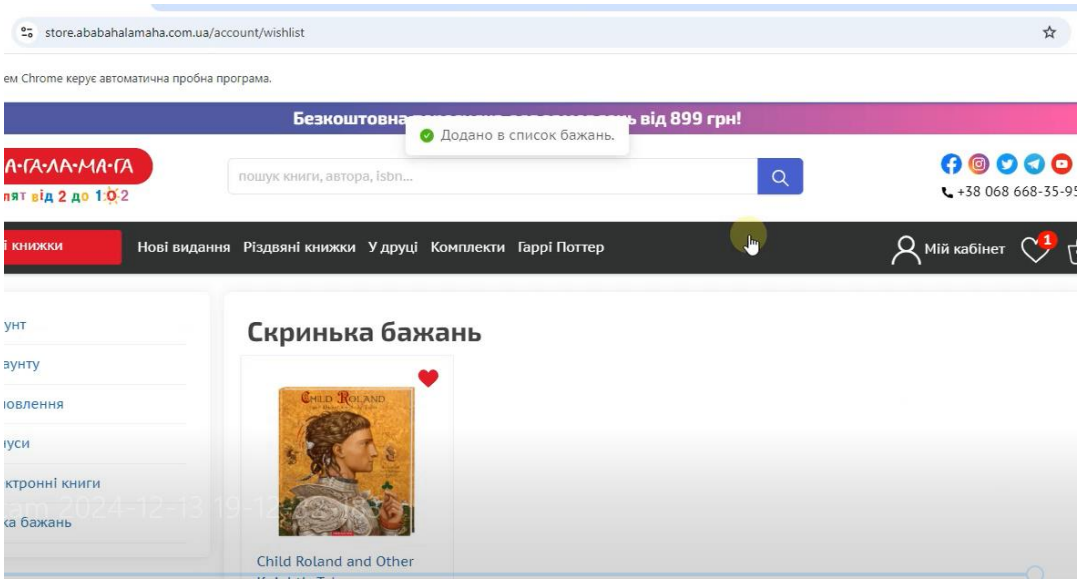


Рисунок 3.16 – Результат автоматизації тест-кейсу 5

Для автоматизації інших тест-кейсів випробуємо Selenium IDE. Для початку потрібно встановити плагін із магазину Chrome чи іншого браузера, який буде використовуватись. Після чого створюмо новий проєкт. Як виглядатиме проєкт після запуску показано на рисунку 3.17. Основним функціоналом є:

- зміна швидкості проходження тест-кейсу;
- запуск обраного тест-кейсу;
- відстеження кроків;
- пауза/продовження проходження тест-кейсу;
- запуск усіх тест-кейсів.

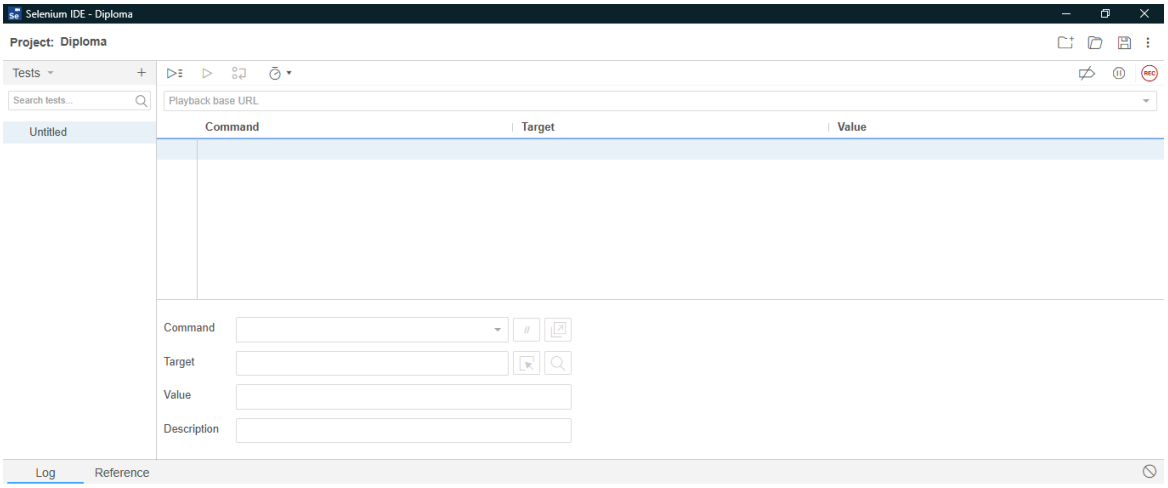


Рисунок 3.17 – Selenium IDE після створення нового проєкту

Обмеження цього інструменту:

- не підходить для тестування великих обсягів даних;
- не можна тестувати підключення до бази даних;
- не може обробляти динамічні частини вебзастосунків;
- відсутня можливість генерації звітів про результати тестування.

Використовувати пошук книг та авторів при записі у цьому інструменті було неможливо ні в браузері Chrome, ні в Edge. Сторінка просто переставала відповідати. Втім з іншими текстовими блоками проблем не виникало. Тобто цей інструмент не надійним на 100% і його потрібно використовувати суто для найпростіших тестів. Проте, змінивши вручну ціль команди, вказавши XPath

елемента, виконання тест-кейсу вдалося продовжити. Так само довелося робити і з кнопкою пошуку, бо результат був таким самим. Далі вже починався запис із цієї команди(Рисунок 3.18).

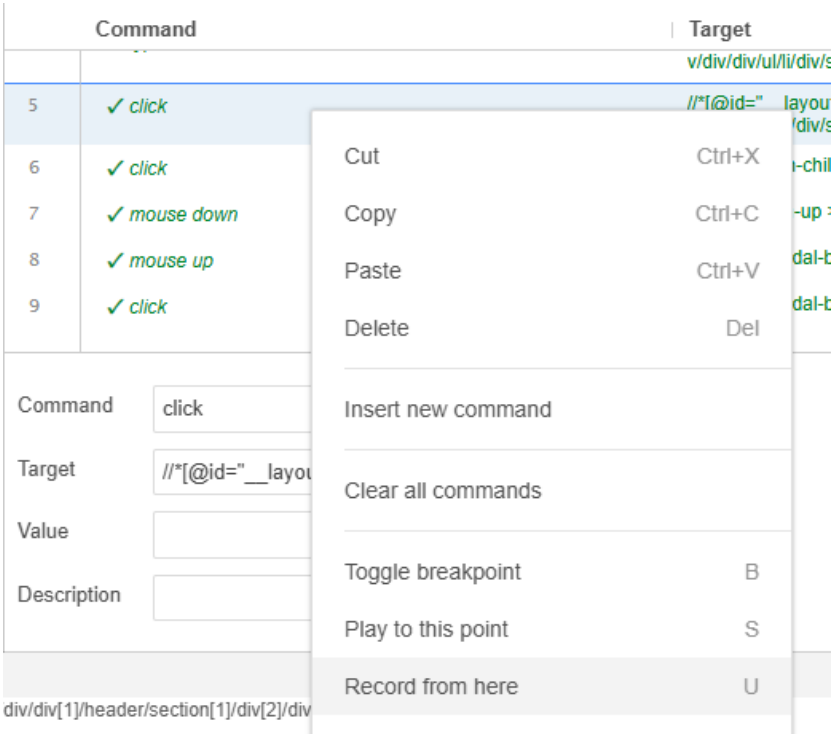


Рисунок 3.18 − Record from here

Коли всі кроки записані можна починати програвати тест. Після того команди, що пройшли успішно тестування будуть підсвічені зеленим кольором та позначені галочкою. Які ж ні, то будуть позначені червоним кольором і хрестиком(Рисунок 3.19).

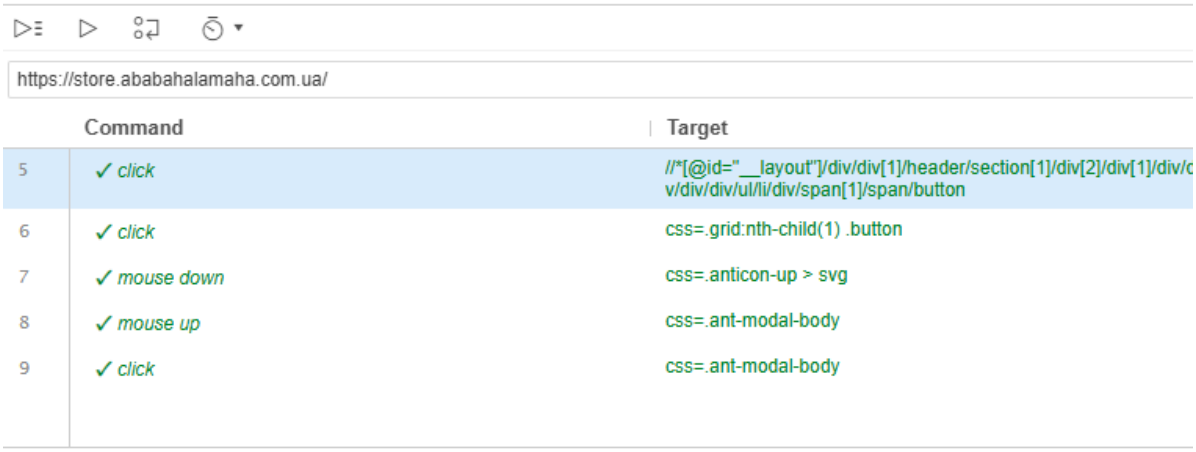


Рисунок 3.19 − Результат тест-кейсу 7

Аналогічно таким чином було автоматизовано тест-кейс 2(Рисунок 3.20).

	Command	Target	Value
		v/div/div/ul/li/div/span[1]/input	
4	✓ type	//*[@id="__layout"]/div/div[1]/header/section[1]/div[2]/div[1]/div/div/div/div/ul/li/div/span[1]/input	локвуд і ко
5	✓ click	//*[@id="__layout"]/div/div[1]/header/section[1]/div[2]/div[1]/div/div/div/div/ul/li/div/span[1]/span/button	
6	✓ click	css=.grid:nth-child(1) strong	
7	✓ click	css=.block > div > .grid > .col-9 > .button	

Рисунок 3.20 – Результат тест-кейсу 2

Selenium IDE надає можливість експортувати код, тож випробуймо її. Записуємо команди для тест-кейсу 8, перевіряємо та генеруємо код. Можна обрати необхідну мову програмування та налаштувати генерацію(Рисунок 3.21).

Експортований код є базою, яку можна використати і запустити і в інших IDE. Тож випробовуємо його в PyCharm.

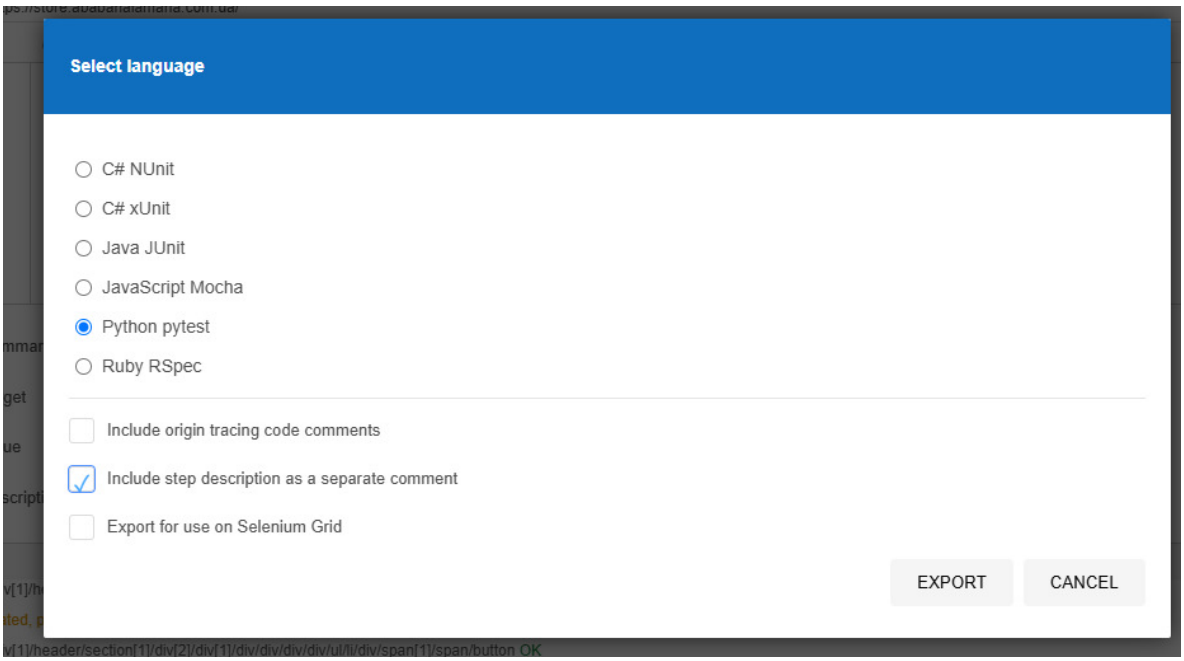


Рисунок 3.21 – Експорт коду

На рисунку 3.22 показано експортований код в PyCharm. Його можна одразу запукати, але експортований код без модифікації часто є неоптимізованим чи містить неефективні конструкції. Тому його все одно краще перед запуском модифікувати. Наприклад, додати час очікування, бо якщо елемент не встигне

завантажитись, то тест-кейс завершиться та отримає статус провалено, оскільки елемент не буде знайдено.

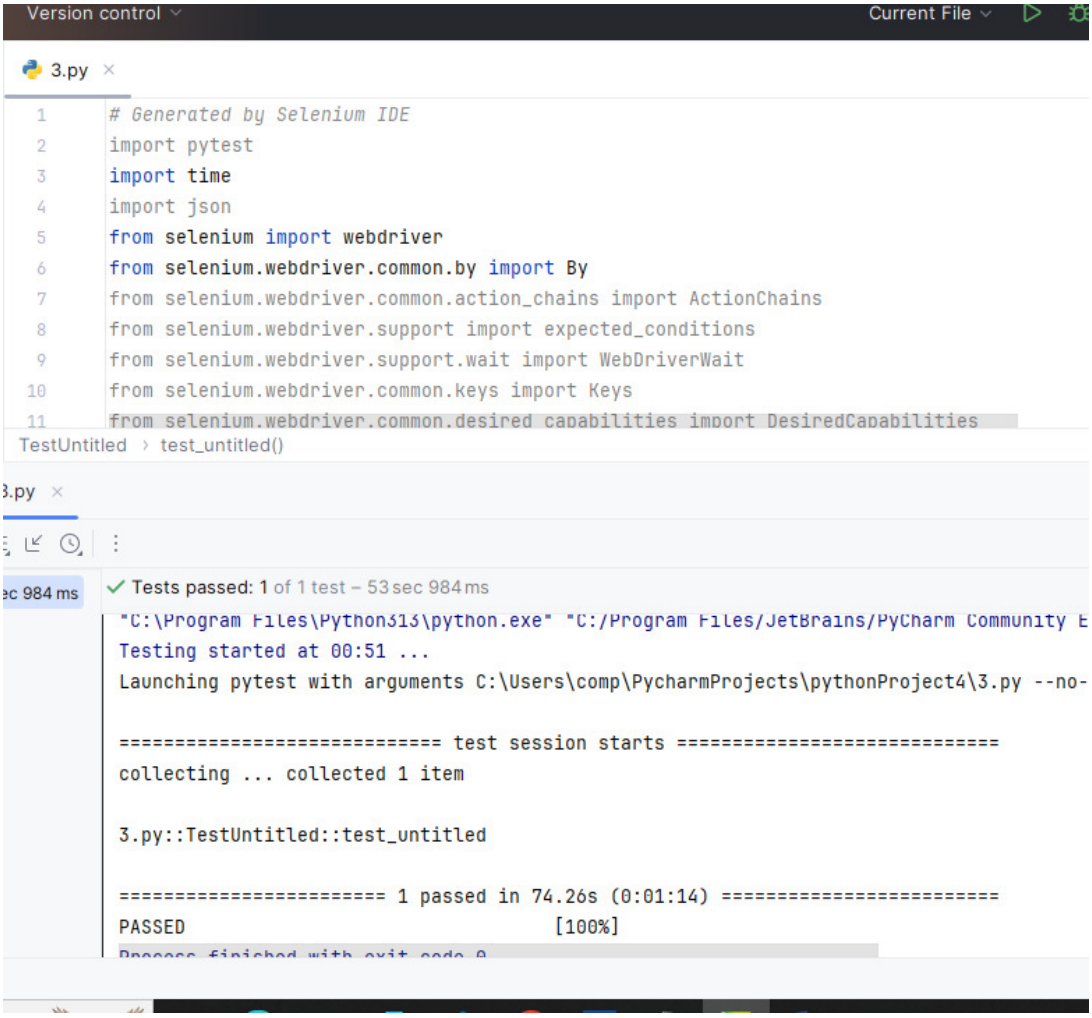


Рисунок 3.22 – Результат тест-кейсу 8

Перевагою є те, що завжди отримуєш інформацію про те, коли тест почався та скільки він тримав, а також в кінці відобразиться інформація про статус виконаного тест-кейсу(Passed чи Failed). При цьому можна так само спостерігати за вебпереглядачем, яким керуватиме автоматична програма.

Але для великих і складних тестових проєктів експортовані тести можуть стати важкими для управління, оскільки вони не завжди відповідають принципам модульності та повторного використання коду.

Для автоматизації тест-кейсу 6 використовуватимемо хмарну платформу Sauce Labs. За допомогою неї можна проводити мануальне та автоматизоване тестування на реальних пристроях та емуляторах.

Налаштовуємо дані свого профілю(замість user_cred та ur_access_key були використані справжні дані).

```
Sauce Labs credentials
SAUCE_USERNAME = "user_cred"
SAUCE_ACCESS_KEY = "ur_access_key"

# Sauce Labs URL
REMOTE_URL = "https://ondemand.eu-central-
1.saucelabs.com:443/wd/hub"
```

Для того щоб звіт про помилку одразу з'являвся інтегруємо Backtrace. Спочатку цю бібліотеку потрібно встановити аналогічним чином, як і selenium та pytest. Додатково потрібно її імпортувати.

```
import backtracepython as bt

# Ініціалізація Backtrace
bt.initialize(
    endpoint="*** ",
    token="*** "
)
```

На місці пропусків так само потрібно вказати значення. Токен необхідно створити для проєкту. Обираємо версії браузерів для паралельного тестування.

```
# Версії браузерів для тестування
browser_versions = {
    "chrome": ["131"],
    "firefox": ["133"]
}
```

Ініціалізуємо їх.

```
def initialize_browser(browser_name, browser_version):
    """Initializes the browser based on the selected
options."""
    options = ChromeOptions() if browser_name == "chrome"
else FirefoxOptions()
    options.set_capability("platformName", "Windows 10")
    options.set_capability("browserVersion",
browser_version)
    sauce_options = {
        "username": SAUCE_USERNAME,
```

```
        "accessKey": SAUCE_ACCESS_KEY,
        "build": "filter-test",
        "name": f"Test on {browser_name.capitalize()}
{browser_version}",
        "screenResolution": "1280x768"
    }
    options.set_capability("sauce:options", sauce_options)
    return webdriver.Remote(command_executor=REMOTE_URL,
options=options)
```

Прописуємо кроки тестування.

```
def perform_test_actions(driver):
    """Performs the actions in the test scenario."""
    driver.get("https://store.abababahalamaha.com.ua/")
    WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, '//*
[@id="__layout"]/div/div [1]/header/section [2]/ul/li
[5]/a'))
    ).click()
    filter_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.XPATH, '//*
[@id="__layout"]/div/div [2]/section/div [2]/div
[4]/div/input [1]'))
    )
    driver.execute_script("arguments [0].value = '';",
filter_input)
    filter_input.send_keys("500")
    driver.find_element(By.XPATH, '//*
[@id="__layout"]/div/div [2]/section/div [2]/div
[4]/div/button').click()
    time.sleep(5)
```

Налаштовуємо автоматичне надсилання звіту разом з атрибутами.

```
def report_issue_to_backtrace(driver, browser_name,
browser_version):
    try:
        raise Exception(f"Гаррі Поттер і філософський
камінь disappeared from list of items after filtering with
price from 500 however its price is 500 in
{browser_name.capitalize()} version {browser_version}.")
    except Exception as e:
```

```
        print(traceback.format_exc())
        bt.send_last_exception(
            attributes={
                "browser": browser_name,
                "browser_version": browser_version,
                "filter_value_from": 500,
                "missing_item": "Гаррі Поттер і
філософський камінь",
                "page_url": driver.current_url
            },
            annotations={
                "custom_message": str(e)
            }
        )
```

Запускаємо тести. Оскільки використовується безкоштовна версія сервісу, то запускаємо їх послідовно через обмеження. Звіт про помилку зображено на рисунку 3.23.

```
def run_tests():
    """Runs tests for all specified browser versions."""
    for browser_name, versions in browser_versions.items():
        for version in versions:
            print(f"Running test for
{browser_name.capitalize()} version {version}...")
            run_test_on_browser(browser_name, version)
            print(f"Test for {browser_name.capitalize()}
version {version} completed.")
            time.sleep(30)  # Delay between tests for
different versions

# Run tests
run_tests()
```

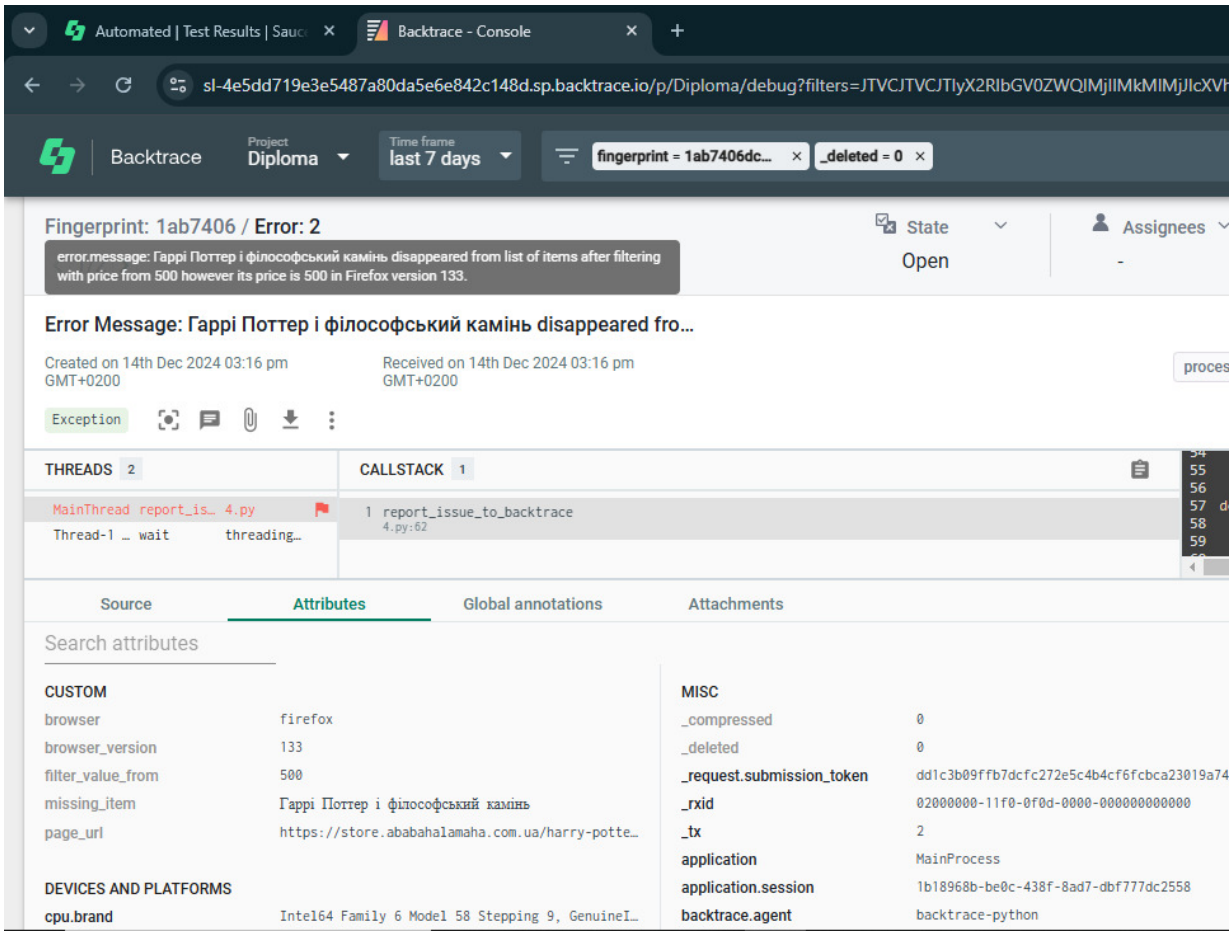


Рисунок 3.23 – Звіт про помилку в Backtrace

Таким чином було проведено кросбраузерне тестування. При цьому на рисунку 3.24 зображено його результати. Час проходження тест-кейсів 32 та 21 секунди.

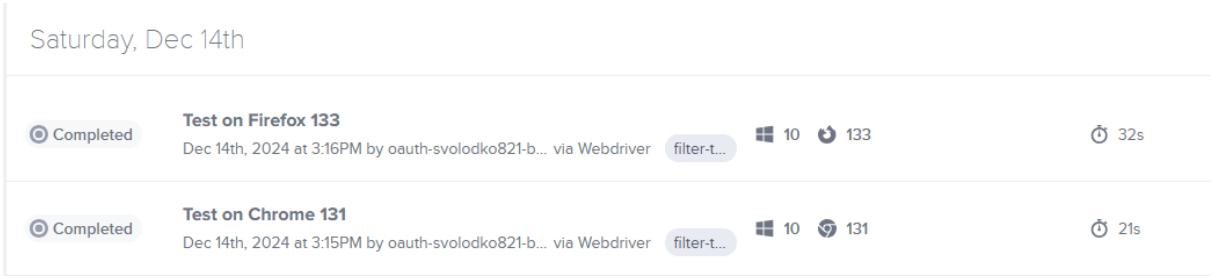


Рисунок 3.24 – Результат кросбраузерного тестування

Сервіс Sauce Labs одразу робить і знімки екрану, і запис відео навіть у безкоштовній версії. У ній також надається 60 хвилин автоматизованого та ручного тестування. Обмеження є і по кількості машин для паралельного використання. Відео можна скачати та додати до звіту про помилку в Backtrace.

3.4 Аналіз результатів тестування

Перед початком автоматичного тестування проводилось мануальне тестування та написання тест-кейсів. Це було важливо, оскільки без цього автоматизація не має сенсу

Мануальне тестування – на початковому етапі тестувалися визначені функції онлайн книгарні. Це дозволило глибше зрозуміти, як працює система.

Написання тест-кейсів – для визначених вимогами тестування функцій створювались тест-кейси, які описували умови тестування, кроки виконання тесту, очікувані результати, фактичні. Також вказувалась передумова та статус тест-кейсу(Passed або Failed). Це забезпечило базу для подальшої автоматизації тестування. Завдяки тест-кейсам було легше писати тест скрипти, бо послідовність дій була в результаті більш зрозумілою.

Після налаштування тестового середовища та всіх необхідних інструментів почалось автоматизоване тестування. При цьому використовувались різні інструменти.

Автоматизація це не завжди про написання коду, тому для тест-кейсів 7 та 2 використовувався Selenium IDE. Він працює як бот: спочатку записує команди та дії користувача. Пізніше їх можна підправити(прибрати зайве) та запускати програвання. Інструмент не є надійним на 100%, бо навіть при виконанні найпростіших тест-кейсів можуть трапитись неправдиві результати, але він покращується. Так при записі не вдалось скористатись пошуком на сайті, але вдалось налаштувати команди для цього вручну.

Selenium IDE загалом використовують тільки для найпростіших дій, бо більш складні тести будуть ненадійними. Але загалом це хороший та зручний інструмент та чудово підходить для ознайомлення з автоматизацією та для автоматизації найпростіших випадків. Також було випробувано генерацію коду після виконання тест-кейсу 2. Код експортувався для використання з Web Driver та запускався в PyChart. Експортований код мав суттєвий недолік: не було визначено часу

очікування. Це довелось додавати вручну, бо деякі елементи не встигали завантажитись і це завершувало виконання тесту як невдалу спробу. Значною перевагою цього інструмента є те, що він повністю безкоштовний і при цьому має хорошу якість і є хорошим конкурентом Katalon Studio, ліцензія якого коштує достатньо дорого. Також у терміналі можна спостерігати інформацію про те, скільки часу зайняло виконання тесту та який статус отримано.

Тест-кейси 5,4,3,1 було автоматизовано скриптами з використанням SeleniumWeb Driver. Хоч при вході була перевірка reCAPTCHA, було вирішено все одно автоматизовувати і вхід у профіль. Капча була не жорстка і було достатньо робити скрипти більш схожими на дії людини. Значною перевагою була можливість використовувати повторно частини скриптів. Це значно пришвидшило процес та зберегло час.

Тест-кейс 6 використовувався для кросбраузерного тестування(тестувались Firefox 133 версії та Chrome 131). Для цього так само використовувався Selenium Web Driver, PyCharm але додатково використовувались такі інструменти як хмарна платформа Sauce Labs та Backtrace. Таким чином не тільки вдалося провести кросбраузерне тестування, а й випробувати використання хмарних технологій у автоматизації тестування, а також звіт про результат проходження тесту автоматично надсилався в систему. Значною перевагою є те, що записується відео під час тестування і не потрібно використовувати для цього додаткові інструменти. Його можна скачувати та додавати у звіт. На рисунку 3.25 зображено порівняння ручного тестування з автоматичним кросбраузерним. Ручне виконувалось при специфікації:

- Intel(R) Pentium(R) CPU 2020M @ 2.40GHz 2.40 GHz;
- Windows 10 ;
- 19045.5247;
- Chrome 131.0.6778.140.

А автоматичне кросбраузерне при специфікації:

- Intel64 Family 6 Model 58 Stepping 9, GenuineIntel;
- Windows;
- 10.0.19045;
- Chrome 131.

Test Case ID	**Вид тестування**	с	с	с	с
6	**Ручне**	15.72	15	14.29	20
6	**Автоматичне Sauce Labs(Firefox)**	32	30	29	36
6	**Автоматичне Sauce Labs(Chrome)**	27	20	22	21

Рисунок 3.25 − Ручне та автоматичне кросбраузерне тестування

На перший погляд ручне тестування швидше, а отже і краще. Але не варто забувати, що при ручному тестуванні ще додатково піде час на оформлення звіту, а при автоматичному його можливо одразу надіслати .

Отже, після розробки тестових сценаріїв за допомогою набору інструментів Selenium підіб'ємо підсумки.

Selenium є найпопулярнішим інструментом для автоматичного тестування, отже має велику спільноту та кількість документації, що полегшує вирішення проблем та отримання підтримки.

Це повністю безкоштовні продукти, а отже не буде проблем з ліцензією та її вартістю.

Для використання Selenium IDE не потрібно володіти навичками програмування та він чудово підходить для ознайомлення із автоматизацією тестування. Є можливість експорту коду, проте його необхідно модифікувати. Він чудово підходить для найпростіших сценаріїв, а от використання цього

інструменту для складніших тестів не гарантує достовірність результатів тестування. Таким чином Selenium IDE підходить для тестування простих випадків, для початкового тестування або для швидкого створення прототипів тестів, але для глибокого та надійного автоматизованого тестування краще використовувати інші інструменти з пакету Selenium, такі як Selenium WebDriver.

Selenium WebDriver потребує навичок програмування, особливо для масштабування тестування. Користувачів не обмежують у використанні плагінів та дозволяють інтегрувати його з іншими фреймворками, платформами та інструментами(SauceLabs, Selenium-Grid, Extent, JUnit тощо). Таким чином можна збільшити покриття тестування. Наприклад, інтеграція з фреймоврком Appium дозволить проводити тестування як веб версій сайтів для смартфонів, так і нативні й гібридні застосунки.

Selenium підтримує широкий спектр мов програмування, тож можна обрати ту, з якою найкомфортніше працювати, а його основною перевагою є підтримка багатьох платформ та браузерів, через що його часто використовують для кросбраузерного тестування. Ще однією перевагою є підтримка паралельного тестування. Для цього можна використовувати і платформи, як Sauce Labs, які надають вже готову інфраструктуру, але можливості платформи залежать від ліцензії. Selenium Grid та Selenium IDE також надають можливість паралельного тестування, але є безкоштовнми, проте у їх випадку потрібно вручну налаштовувати середовище тестування та розгортати його у хмарі або локально суто з використанням власних ресурсів. Вибір між цими інструментами залежить від вимог проєкту та бюджету.

ВИСНОВКИ

У ході виконання магістерської роботи було:

- Розглянуто види тестування, проаналізовано ручне та автоматичне тестування, їхні переваги й недоліки. При цьому було визначено, що автоматизоване тестування не є заміною ручному, адже використання лише цього виду тестування не має сенсу та створює більше проблем. Найкращим рішенням для забезпечення якості та надійності продукту є баланс між ручним та автоматичним тестуванням;
- Проведено дослідження інструментів автоматизації тестування. Розглянуто набір інструментів Selenium та вивчено його можливості;
- Визначено вимоги тестування. Відповідно до них було проведено ручне тестування, за результатами якого було створено тест-кейси, які в подальшому використовувались для автоматизації тестування;
- Розроблено тестові сценарії на основі тест-кейсів з використанням інструментів з набору Selenium(Selenium IDE, Selenium WebDriver), мови програмування Python та платформи Sauce Labs;
- Оцінено ефективність та слушність використання інструментів Selenium за результатами виконання тестових сценаріїв. Визначено переваги й недоліки їх використання в порівнянні з іншими інструментами.

Хоч виходять нові фреймворки, як Playwright, Selenium залишається потужним та гнучким інструментом для автоматичного тестування. Він постійно розвивається та адаптується до нових технологій і потреб користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. A short history of the Web [Електронний ресурс] – Режим доступу до ресурсу: https://home.cern/science/computing/birth-web/short-history-web.
2. AlSayyed S. The History of Websites 101. URL: https://www.linkedin.com/pulse/history-websites-101-sami-alsayyed/ (date of access: 09.12.2024).
3. Architecture of Selenium WebDriver | BrowserStack. BrowserStack. URL: https://www.browserstack.com/guide/architecture-of-selenium-webdriver
4. Architecture of Selenium WebDriver | BrowserStack. BrowserStack. URL: https://www.browserstack.com/guide/architecture-of-selenium-webdriver
5. Best Programming Language For Automation Testing. URL: https://www.appsierra.com/blog/7-best-programming-language-for-automation-testing
6. CMS Market Share: The Most Popular Website Platforms in 2024. MobiLoud - Convert Your Website to Native Mobile Apps. URL: https://www.mobiloud.com/blog/cms-market-share
7. CMS Usage Distribution on the Entire Internet. BuiltWith. URL: https://trends.builtwith.com/cms/traffic/Entire-Internet
8. How Many Websites Are There?. Digital Silk. URL: https://digitalsilk.com/digital-trends/how-many-websites-are-there/
9. Jamtion. How AR and VR are Changing the Way We Interact with Websites?. LinkedIn: Log In or Sign Up. URL: https://www.linkedin.com/pulse/how-ar-vr-changing-way-we-interact-websites-jamtion/
10. JavaScript Історія [Електронний ресурс] // Refsnes Data – Режим доступу до ресурсу: https://www.w3schools.com/js/js_history.asp.

11. Launch of new IKEA Place app – IKEA Global. IKEA. URL: https://www.ikea.com/global/en/newsroom/innovation/ikea-launches-ikea-place-a-new-app-that-allows-people-to-virtually-place-furniture-in-their-home-170912/
12. Munro A. CSS | Definition, History, & Facts | Britannica. *Encyclopedia Britannica*. URL: https://www.britannica.com/technology/CSS-programming-language
13. NextGenerationAutomation. Learn Selenium Web Driver Architecture. NGAutomation. URL: https://www.nextgenerationautomation.com/post/learn-selenium-web-driver-architecture
14. Nix E. The World's First Web Site [Електронний ресурс] / Elizabeth Nix // A&E Television Networks. – 2016. – Режим доступу до ресурсу: https://www.history.com/news/the-worlds-first-web-site.
15. Panchasara M. 7 Steps of Web Development Process. Radixweb. URL: https://radixweb.com/blog/web-development-process
16. Skillings J. Images: Berners-Lee and the dawn of the Web [Електронний ресурс] / Jon Skillings. – 2009. – Режим доступу до ресурсу: https://www.cnet.com/pictures/images-berners-lee-and-the-dawn-of-the-web/
17. Structure and Governance. Selenium. [Електронний ресурс] - режим доступу до ресурсу: https://www.selenium.dev/project/
18. The Selenium Browser Automation Project. Selenium. URL: https://www.selenium.dev/documentation/
19. What is Web Testing? Definition, Tools, Best Practice. katalon.com. [Електронний ресурс] - режим доступу до ресурсу: https://katalon.com/resources-center/blog/what-is-web-testing
20. White/black/grey box-тестування - QALight. QALight. URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/

21. Етапи створення веб сайтів: які є основні кроки розробки - WebTune. Webtune. URL: https://webtune.com.ua/statti/web-rozrobka/etapy-stvorennya-veb-sajtiv/
22. Золотухіна О.А., Негоденко О.В., Резник С.Ю., Разіна С.Я.. «Якість та тестування інформаційних систем». - 2020
23. Романенко К. WEB 1.0, WEB 2.0 І WEB 3.0 – В ЧОМУ ВІДМІННОСТІ ТА ІСТОРІЯ СТВОРЕННЯ. Друкарня. URL: https://drukarnia.com.ua/articles/web-1-0-web-2-0-i-web-3-0-v-chomu-vidminnosti-ta-istoriya-stvorennya-MgWuT#heading-3-239
24. Ручне тестування і автоматизація. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/manual-testing-vs-automation-testing/
25. Яка різниця між Black Box & White Box Testing?. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/whats-the-difference-between-black-box-white-box-testing/
26. Які інструменти використовують в EPAM для автоматизації тестування. URL: https://careers.epam.ua/blog/test-automation-tool

ДОДАТОК А. Коди тестових сценаріїв

Тест-кейс 1:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from time import sleep

try:
    # Setup WebDriver
    driver = webdriver.Chrome()
    print("Driver initialized.")

    # Navigate to the website
    print("Navigating to the website...")
    driver.get("https://store.ababahalamaha.com.ua/")
    print("Website loaded successfully.")

    # Locate the search field
    print("Locating the search field...")
    search_field = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.CLASS_NAME, "ant-
input"))
    )
    print("Search field located successfully.")

    # Enter search query
    search_query = "локвуд і ко"
    print(f"Entering search query: {search_query}")
    search_field.send_keys(search_query)

    # Click the search button using the provided XPath
    print("Clicking the search button...")
    search_button = driver.find_element(By.XPATH,

'//*[@id="__layout"]/div/div[1]/header/section[1]/div[2]/div[1]/div/
div/div/div/ul/li/div/span[1]/span/button')
    search_button.click()
    print("Search button clicked.")
```

```
    # Wait for the search results to load
    print("Waiting for search results...")
    WebDriverWait(driver, 15).until(
        EC.presence_of_all_elements_located((By.CSS_SELECTOR, ".col-
9_xs-12 a strong"))
    )

    print("Search results loaded successfully.")

    # Debugging: Let's print all search results to see if we found
the right elements
    search_results = driver.find_elements(By.CSS_SELECTOR, ".col-
9_xs-12 a strong")
    print(f"Found {len(search_results)} search results.")

    for result in search_results:
        print(f"Result found: {result.text}")
        if "локвуд і ко" in result.text.lower():
            print(f"Found the correct book: {result.text}")
            buy_button = result.find_element(By.XPATH,

"./ancestor::div[contains(@class, 'block')]//button[contains(@class,
'button-buy')]")
            buy_button.click()
            sleep(4)
            print("Item added to the cart successfully!")
            break
    else:
        print("No matching book found.")

except Exception as e:
    print(f"An error occurred: {e}")
finally:
    print("Closing the browser.")
    driver.quit()
```

Тест-кейс 3:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

try:
    # Setup WebDriver
    driver = webdriver.Chrome()
    print("Driver initialized.")

    # Navigate to the website
    print("Navigating to the website...")
    driver.get("https://store.abababahalamaha.com.ua/")

    # Scroll to the footer
    driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
    time.sleep(5)
    footer_textfield = driver.find_element(By.CSS_SELECTOR,
"input.ant-input.ant-input-lg")

    # Enter text into the field
    email = "dfrejgej@owpfn"
    footer_textfield.send_keys(email)
    print(f"Text '{email}' successfully entered into the field.")
    subscribe_button = driver.find_element(By.CSS_SELECTOR,
"button.ant-input-search-button")
    subscribe_button.click()
    time.sleep(8)

except Exception as e:
    print(f"An error occurred: {e}")
finally:
    # Close the browser
    driver.quit()
```

Тест-кейс 6 :

```
import time
import traceback
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options as
ChromeOptions
from selenium.webdriver.firefox.options import Options as
FirefoxOptions
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import backtracepython as bt

# Ініціалізація Backtrace
bt.initialize(
    endpoint="***",
    token="***"
)

# Sauce Labs credentials and URL
SAUCE_USERNAME = "***"
SAUCE_ACCESS_KEY = "***"
REMOTE_URL = "https://ondemand.eu-central-
1.saucelabs.com:443/wd/hub"

# Версії браузерів для тестування
browser_versions = {
    "chrome": ["131"],
    "firefox": ["133"]
}

def initialize_browser(browser_name, browser_version):
    """Initializes the browser based on the selected options."""
    options = ChromeOptions() if browser_name == "chrome" else
FirefoxOptions()
    options.set_capability("platformName", "Windows 10")
    options.set_capability("browserVersion", browser_version)
    sauce_options = {
        "username": SAUCE_USERNAME,
        "accessKey": SAUCE_ACCESS_KEY,
        "build": "filter-test",
        "name": f"Test on {browser_name.capitalize()}
{browser_version}",
        "screenResolution": "1280x768"
    }
    options.set_capability("sauce:options", sauce_options)
```

```
    return webdriver.Remote(command_executor=REMOTE_URL,
options=options)

def perform_test_actions(driver):
    """Performs the actions in the test scenario."""
    driver.get("https://store.ababahalamaha.com.ua/")
    WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH,
'//*[@id="__layout"]/div/div[1]/header/section[2]/ul/li[5]/a'))
    ).click()
    filter_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.XPATH,
'//*[@id="__layout"]/div/div[2]/section/div[2]/div[4]/div/input[1]')
)
    )
    driver.execute_script("arguments[0].value = '';", filter_input)
    filter_input.send_keys("500")
    driver.find_element(By.XPATH,
'//*[@id="__layout"]/div/div[2]/section/div[2]/div[4]/div/button').c
lick()
    time.sleep(5)

def report_issue_to_backtrace(driver, browser_name,
browser_version):
    try:
        raise Exception(f"Гаррі Поттер і філософський камінь
disappeared from list of items after filtering with price from 500
however its price is 500 in {browser_name.capitalize()} version
{browser_version}.")
    except Exception as e:
        print(traceback.format_exc())
        bt.send_last_exception(
            attributes={
                "browser": browser_name,
                "browser_version": browser_version,
                "filter_value_from": 500,
                "missing_item": "Гаррі Поттер і філософський
камінь",
                "page_url": driver.current_url
            },
            annotations={
                "custom_message": str(e)
            }
        )

def run_test_on_browser(browser_name, browser_version):
    """Runs the full test scenario for the selected browser and
version."""
    driver = initialize_browser(browser_name, browser_version)
    try:
        perform_test_actions(driver)
        try:
```

```
            driver.find_element(By.XPATH, '//a[@href="/harri-potter-
i-filosofskyi-kamin-gryfindor"]')
            print(f"Item found in {browser_name.capitalize()}
version {browser_version}.")
        except Exception as e:
            print(f"Item disappeared in {browser_name.capitalize()}
version {browser_version}.")
            report_issue_to_backtrace(driver, browser_name,
browser_version)
    finally:
        driver.quit()

def run_tests():
    """Runs tests for all specified browser versions."""
    for browser_name, versions in browser_versions.items():
        for version in versions:
            print(f"Running test for {browser_name.capitalize()}
version {version}...")
            run_test_on_browser(browser_name, version)
            print(f"Test for {browser_name.capitalize()} version
{version} completed.")
            time.sleep(30)  # Delay between tests for different
versions

# Run tests
run_tests()
```

Тест-кейс 5:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

# Ініціалізація WebDriver
driver = webdriver.Chrome()

try:
    # Перехід на сайт
    driver.get("https://store.ababahalamaha.com.ua/")
    driver.maximize_window()

    # Використання WebDriverWait для очікування елемента
    wait = WebDriverWait(driver, 10)
```

```
    # Клік по кнопці "Логін"
    login_button = wait.until(
        EC.element_to_be_clickable((By.XPATH,
'//*[@id="__layout"]/div/div[1]/header/section[2]/div[2]/ul/li[2]/a'
))
    )
    login_button.click()
    print("Кнопка 'Логін' натиснута.")

    # Зачекати, поки з'являться поля для логіну та пароля
    username_field = wait.until(
        EC.visibility_of_element_located((By.XPATH,
"//div[2]/div/span/input"))
    )
    print("Поле логіну завантажене.")

    # Введення логіну
    username_field.send_keys("***")
    time.sleep(8)
    print("Логін введено.")

    # Введення пароля
    password_field = wait.until(
        EC.element_to_be_clickable((By.XPATH,
'//div[2]/div[2]/div/span/input'))
    )
    password_field.send_keys("***")
    time.sleep(17)
    print("Пароль введено.")

    # Натискання кнопки авторизації
    submit_button = wait.until(
        EC.element_to_be_clickable((By.XPATH,
'//button[@type="submit" and contains(text(), "Увійти")]'))
    )
    submit_button.click()
    time.sleep(9)


    wait.until(EC.visibility_of_element_located((By.XPATH,
'//*[contains(text(), "У друці")]')))


    # Клік по кнопці "У друці"
    in_print_button = wait.until(
        EC.element_to_be_clickable((By.XPATH, '//*[contains(text(),
"У друці")]'))
    )
    in_print_button.click()
    print("Кнопка 'У друці' натиснута.")
    time.sleep(4)
```

```
    # Знайти картку книги "Child Roland and Other Knightly Tales
(англійською)"
    book_link = wait.until(
        EC.presence_of_element_located(
            (By.XPATH,

'//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1]/div[13]
//a[contains(text(), "Child Roland and Other Knightly Tales
(англійською)")]')
        )
    )

    # Прокрутити до цієї картки
    driver.execute_script("arguments[0].scrollIntoView();",
book_link)
    time.sleep(2)

    # Знайти кнопку "Додати в бажане" для цієї книги
    favorite_button = wait.until(
        EC.element_to_be_clickable(
            (By.XPATH,

'//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1]/div[13]
//img[@src="/_nuxt/img/heart.665ede1.svg"]')
        )
    )

    # Клікаємо на кнопку "Додати в бажане"
    favorite_button.click()
    print("Кнопка 'Додати в бажане' натиснута.")

    # Перевірка: відкриваємо "Скриньку бажань"
    wishlist_button = wait.until(
        EC.element_to_be_clickable((By.XPATH, '//img[@alt="Скринька
бажань"]'))
    )
    wishlist_button.click()
    time.sleep(4)
    print("Скринька бажань відкрилась.")

    # Перевірка наявності книги у списку бажаного
    wishlist_book = wait.until(
        EC.presence_of_element_located(
            (By.XPATH, '//*[contains(text(), "Child Roland and Other
Knightly Tales (англійською)")]')
        )
    )
    time.sleep(4)
    if wishlist_book:
        print("Книга 'Child Roland and Other Knightly Tales
(англійською)' успішно додана до списку бажань.")
```

```
    else:
        print("Книга 'Child Roland and Other Knightly Tales
(англійською)' не знайдена у списку бажань.")

except Exception as e:
    print(f"Виникла помилка: {e}")
finally:
    # Закриття браузера
    time.sleep(5)
    driver.quit()
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Дипломна робота на ступінь вищої освіти магістр із
спеціальності 122 Комп'ютерні науки

Система автоматизації тестування веб-сайтів на основі Selenium

Виконав: студент 6 курсу, групи КНДМ-62
Володько Олександр Борисович

Керівник: д.ф-м.н, професор
Шикула Олена Миколаївна

Київ - 2024

ЗАГАЛЬНА ХАРАКТЕРИСТИКА РОБОТИ 2

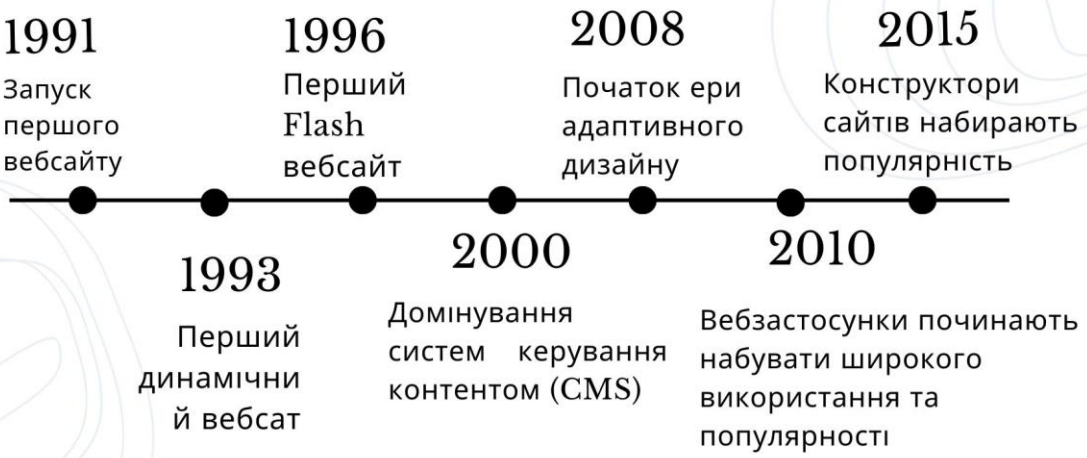
Мета дослідження: вивчення можливостей набору інструментів Selenium для автоматизації тестування вебсайтів

Об'єкт дослідження: автоматизація тестування вебсайтів

Предмет дослідження: інструменти для автоматизації тестування вебсайтів

Етапи розвитку вебсайтів

3



Ручне VS автоматизоване тестування

4

Ручне	Автоматичне
наявна людська точка зору	відсутня людська точка зору
може бути повторюваним та нудним	можливо повторно використовувати скрипти
можна протестувати практично будь-який продукт	можливо запускати велику кількість тестів одночасно.
присутній людський фактор	вища точність та більша швидкість виявлення дефектів
підходить для проєктів із частими змінами	не ефективне при нестабільній системі

Ручне тестування проти автоматизованого

5

Обидва види тестування мають свої переваги і недоліки та не є заміною один одному. Який вид тестування обрати залежить від проєкту та його особливостей.

Найкращий спосіб забезпечити якість та надійність продукту – баланс між ручним та автоматичним тестуванням, тому на реальних проєктах їх часто комбінують.

Набір інструментів Selenium

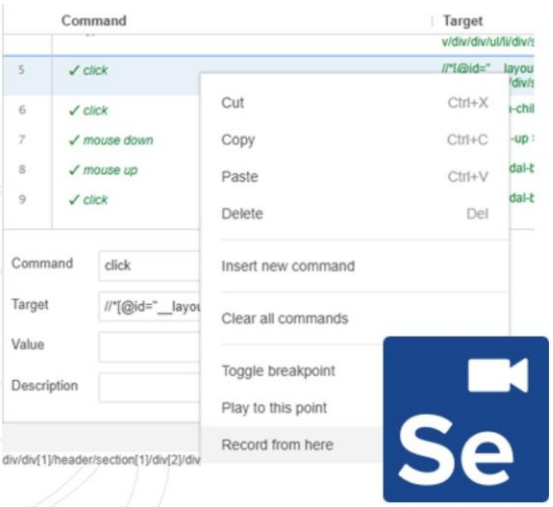
6

Selenium Grid

Selenium WebDriver

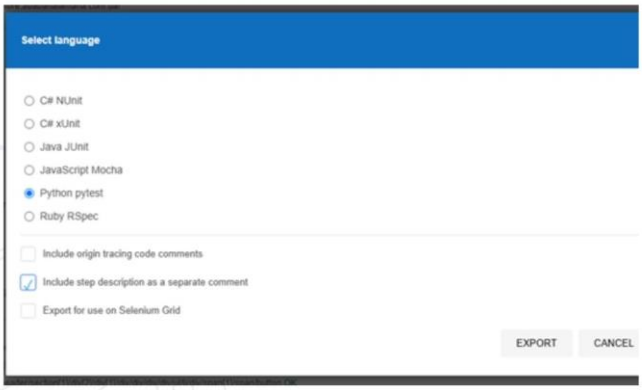
Selenium IDE

Selenium IDE

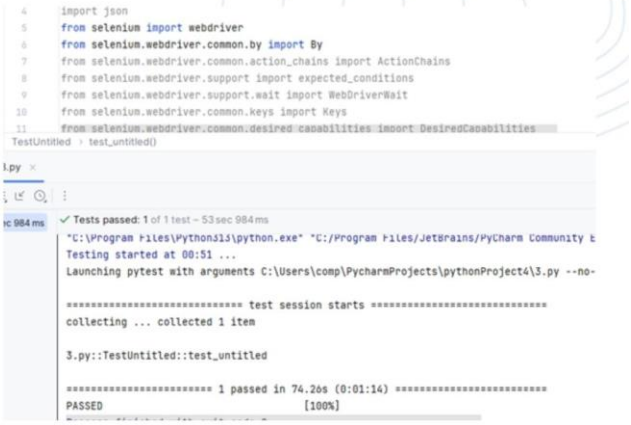


- Не потрібні знання мов програмування для використання
- Працює з Chrome, Firefox, та Edge
- Записує дії користувача, а тоді відтворює їх
- Є можливість експорту коду
- Не є надійним для використання із складнішими тестами

Selenium IDE

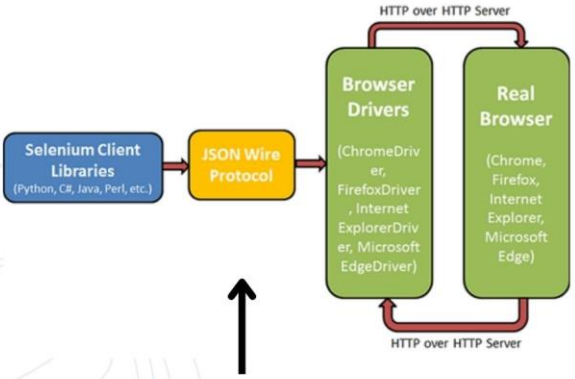


Експорт коду



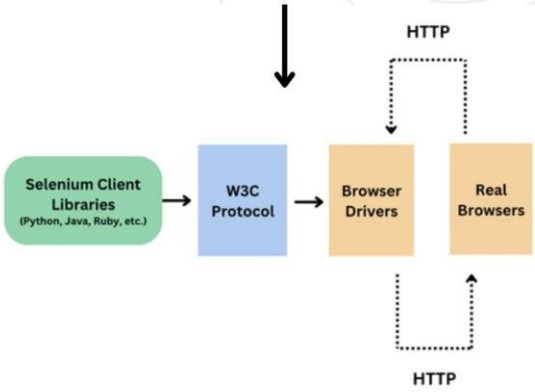
Використання експортованого коду разом із Selenium WebDriver

Selenium WebDriver



АРХІТЕКТУРА SELENIUM WEB DRIVER 3

АРХІТЕКТУРА SELENIUM WEB DRIVER 4



Selenium WebDriver


```
# Клік по кнопці "Логін"
login_button = wait.until(
    EC.element_to_be_clickable(
        (By.XPATH, '//*[@id="__layout"]/div/div[1]/header
            /section[2]/div[2]/ul/li[2]/a')
    )
)
login_button.click()
# Чекаємо, поки з'являться поля для логіну та пароля
username_field = wait.until(
    EC.visibility_of_element_located((By.XPATH, "//div[2]
        /div/span/input"))
)
# Введення логіну
username_field.send_keys("your login")
time.sleep(5)
# Введення пароля
password_field = wait.until(
    EC.element_to_be_clickable((By.XPATH, "//div[2]
        /div[2]/div/span/input"))
`
```

скрипт для авторизації

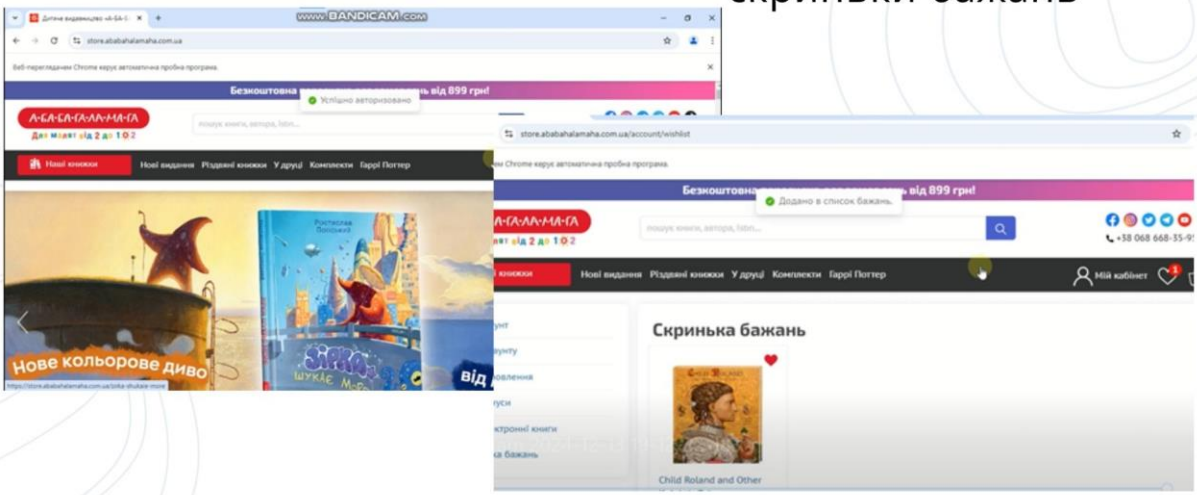
```
book_card = wait.until(
    EC.presence_of_element_located(
        (
            By.XPATH,
            '//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1]
                /div[11]/a',
        )
    )
)

driver.execute_script("arguments[0].scrollIntoView();", book_card)
time.sleep(2)

favorite_button = wait.until(
    EC.element_to_be_clickable(
        (
            By.XPATH,
            '//*[@id="__layout"]/div/div[2]/section/div[2]/div[5]/div[1]
                /div[11]/div[2]/img',
        )
    )
)
favorite_button.click()
```

скрипт додавання товару до скриньки бажань

Авторизація та додавання товару до скриньки бажань



Selenium Grid

Selenium Grid спеціалізується на одночасному виконанні кількох тестів у різних браузерах, операційних системах і на різних машинах. Інструмент допомагає розширити масштаби тестування підходить для великих команд

Найкраще використовувати:

- Коли потрібно виконати тести на різних браузерах (та їх версіях), пристроях і операційних системах, щоб перевірити сумісність і стабільну роботу застосунку

- Коли тестовий набір великий, і послідовне виконання тестів займає надто багато часу. Selenium Grid дозволяє виконувати тести паралельно, що значно прискорює процес.

Переваги Selenium Grid

- **Підтримує паралельне тестування.**

Дозволяє запускати кілька тестів одночасно, що значно скорочує час виконання. Це особливо корисно для великих тестових наборів, де послідовне виконання забирало б багато часу.

- **Дозволяє централізоване виконання тестів.**

Керує тестуванням з одного хабу, що спрощує управління кількома вузлами (машинами чи віртуальними машинами), які виконують тести. Це особливо корисно для великих команд.

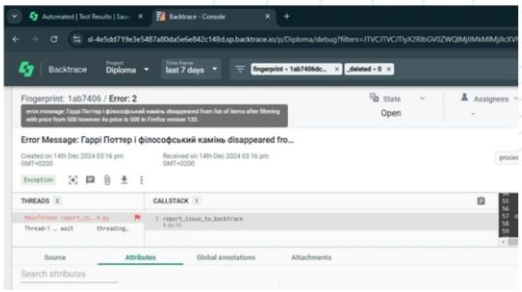
- **Кросбраузерне тестування.**

Забезпечує перевірку роботи застосунку на різних браузерах та їх версіях, гарантуючи сумісність і стабільну роботу.

- Легко розширює інфраструктуру тестування, додаючи нові вузли до мережі, що дозволяє запускати більше паралельних тестів і справлятися з великим навантаженням.

Sauce Labs

- Хмарна платформа, яка надає готову інфраструктуру.
- Не потрібно встановлювати чи налаштовувати локальні вузли.
- Підтримує широкий спектр браузерів, операційних систем і пристроїв.
- Необмежена масштабованість завдяки хмарній інфраструктурі.
- Ідеально підходить для великих проєктів і глобальних команд.
- Надає можливість масштабного паралельного тестування в хмарі без додаткового навантаження на локальні ресурси.



Висновки

Хоч виходять нові фреймворки, як Playwright, який зараз набирає популярності, Selenium все ще залишається потужним та гнучким інструментом для автоматичного тестування. Він постійно розвивається та адаптується до нових технологій і потреб користувачів.

Більшою мірою створений для тестування вебсайтів та вебзастосунків він продовжує надавати розробникам та тестувальникам потужний набір інструментів для автоматичного управління вебпереглядачами та моделювання вазаємодії користувачів із вебсайтами та вебзастосунками.