МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ТИМЧУК ОЛЕГ ГЕННАДІЙОВИЧ

Допускається до захисту:

в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____О. В. Зелінська

«_____» _____2024 р.

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ АНАЛІЗУ ДАНИХ ОЦІНКИ ПРОЦЕСІВ ТЕСТУВАННЯ ВЕБДОДАТКІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:

Н. А. Потапова, доцент кафедри
інформаційних технологій,
канд. екон. наук, доцент

_____
(Підпис)

Оцінка: _____/_____/_____
(бали/ за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(Підпис)

Вінниця – 2024

АНОТАЦІЯ

Тимчук О.Г. Інформаційна система для аналізу даних оцінки процесів тестування вебдодатків. Спеціальність 122 «Комп'ютерні науки». Освітня програма Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Магістерська робота присвячена розробці інформаційної системи для аналізу даних процесів тестування вебдодатків. Основна мета системи – отримання та обробка результатів тестування з використанням спеціально розроблених тест-кейсів. Система виконує тестування вебдодатків і здійснює порівняння отриманих результатів для оцінки якості процесів тестування. Для розробки використовувалися мова програмування Python, середовище розробки PyCharm, а також інструменти автоматизованого тестування Selenium і Playwright.

Магістерська робота складається зі вступу, трьох розділів, висновків і додатків. У вступі обґрунтовується актуальність теми, визначається мета дослідження та завдання, які необхідно вирішити. Перший розділ охоплює теоретичні засади створення інформаційних систем та особливості тестування веб-додатків. У другому розділі розглянуто технології та методології розробки веб-додатків, класифікацію методів тестування, а також інструменти та математичні метрики для оцінки результатів. У третьому розділі описані практичні результати роботи інформаційної системи аналізу даних, отримані на основі проведеного тестування.

Загальний обсяг роботи – 110 сторінок, вона містить 41 рисунок, 10 таблиць, а також список літератури з 40 джерел.

ABSTRACT

Tymchuk O.H. Information System for Data Analysis of Web Application Testing Processes. Specialty 122 "Computer Science". Educational Program "Data Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

The master's thesis is devoted to the development of an information system for analyzing data from web application testing processes. The primary purpose of the system is to collect and process the results of testing using specially developed test cases. The system performs testing of web applications and compares the obtained results to evaluate the quality of the testing processes. The development was carried out using the Python programming language, the PyCharm development environment, and automated testing tools such as Selenium and Playwright.

The thesis consists of an introduction, three chapters, conclusions, and appendices. The introduction substantiates the relevance of the topic, defines the research objectives, and sets the tasks to be addressed. The first chapter covers the theoretical foundations of creating information systems and the specific features of web application testing. The second chapter examines the technologies and methodologies for web application development, classification of testing methods, as well as tools and mathematical metrics for evaluating testing results. The third chapter presents the practical results of the data analysis system for evaluating testing processes based on the conducted tests.

The master's thesis includes an introduction, 3 chapters, conclusions, a bibliography with 40 sources, 41 pictures, 10 tables and applications. The total volume of the work is 110 pages.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTML – (HyperText Markup Language)

APP – додаток (Application)

API – інтерфейс прикладного програмування (Application Programming Interface)

REST – (Representational State Transfer)

SOAP – (Simple Object Access Protocol)

CI/CD – (Continuous Integration/Continuous Delivery)

XML – розширювана мова розмітки (Extensible Markup Language)

XPATH – (XML Path Language)

PIP – (Python Package Installer)

NP – (NumPy)

JSON – нотація об’єктів JavaScript (JavaScript Object Notation)

URL – (Uniform Resource Locator)

IDE – (Integrated Development Environment)

KPI – (Key Performance Indicators)

ВСТУП

Актуальність теми. У сучасному цифровому середовищі вебдодатки займають провідне місце, обслуговуючи різні категорії користувачів та забезпечуючи важливі функції для бізнесу і особистого користування. Однак зі зростанням складності цих додатків виникає потреба у більш ефективному тестуванні, щоб гарантувати їх високу якість і надійність. Автоматизація процесу тестування веб-додатків дозволяє значно підвищити ефективність та зменшити кількість людських помилок. Використання інформаційних систем для аналізу даних процесів тестування стає ключовим підходом до оптимізації тестування, дозволяючи скоротити час виявлення дефектів і покращити загальну якість програмного забезпечення.

Мета магістерської роботи полягає у розробці інформаційної системи для аналізу даних процесів тестування вебдодатків з метою вдосконалення цих процесів.

Завдання магістерської роботи. Для реалізації поставленої мети було поставлено ряд завдань:

- Дослідити літературу та вивчити основні методи тестування вебдодатків, зокрема автоматизоване та ручне тестування.
- Проаналізувати потреби та вимоги до інформаційних систем, що використовуються для аналізу тестових даних, включаючи функціональні та нефункціональні вимоги.
- Розробити концепцію та архітектуру інформаційної системи для збору, зберігання та аналізу даних тестування вебдодатків.
- Розробити прототип інформаційної системи з функціональністю для аналізу тестових даних.
- Провести тестування системи, включаючи функціональні та нефункціональні тести, для оцінки ефективності та точності аналізу даних.

- Проаналізувати отримані результати, визначити переваги та обмеження розробленого прототипу і надати рекомендації щодо подальшого вдосконалення системи.

Об'єкт дослідження – інформаційна система для аналізу даних процесів тестування вебдодатків.

Предмет дослідження – методи, алгоритми та моделі аналізу даних, що використовуються для тестування вебдодатків.

В магістерській роботі використовувались **методи:** аналізу літературних джерел, класифікації, декомпозиції систем, проєктного аналізу, порівняння та оцінювання метрик, планування експерименту, тестування,

Наукова новизна дослідження полягає в тому, що:

Вперше розроблено: інформаційну систему для автоматизованого аналізу даних процесів тестування веб-додатків, що дозволяє вдосконалити процес тестування за допомогою порівняння тестових результатів.

Вдосконалено: підхід до аналізу даних тестування вебдодатків через використання автоматизованих інструментів, що дозволяє підвищити точність та швидкість виявлення дефектів.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків.

У першому розділі розглянуто теоретичні аспекти створення інформаційних систем та процесів тестування вебдодатків.

У другому розділі проведено аналіз технологій розробки вебдодатків, класифікацію методів тестування та огляд інструментів для їх реалізації.

У третьому розділі представлено результати практичного застосування розробленої системи та оцінка її ефективності.

Практичне значення роботи полягає в:

- можливості використання системи для автоматизації процесу аналізу результатів тестування вебдодатків;

- застосуванні розробленого прототипу для підвищення ефективності та точності тестування програмного забезпечення, а також скорочення часу на виявлення дефектів.

Апробація результатів. Результати даного дослідження було апробовано в доповіді «Ризики в управлінні ІТ-проєктами» [1] на ІІ Міжнародній науково-практичній конференції «Прикладні аспекти сучасних міждисциплінарних досліджень» (м. Вінниця, 24 листопада 2023 року), в доповіді «Моделі тестування програмних продуктів» [2] на V Всеукраїнській науково-практичній конференції «Прикладні інформаційні технології» (м. Вінниця, 24 травня 2024 року) та в доповіді «Метрики вимірювання ефективності процесів тестування програмних продуктів» [3] на ІІІ Міжнародній науково-практичній конференції «Прикладні аспекти сучасних міждисциплінарних досліджень» (м. Вінниця, 1 листопада 2024 року).

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ФУНКЦІОНУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМИ В ПРОЦЕСАХ ТЕСТУВАННЯ ВЕБ -ДОДАТКІВ

1.1 Теоретичні основи функціонування інформаційних систем

У широкому сенсі будь-яку систему, що обробляє інформацію, можна назвати інформаційною системою. Інформаційні системи дозволяють отримувати, перетворювати, обробляти, зберігати інформацію та передавати результати обробки споживачам (людям, машинам та іншим інформаційним системам). Прикладами сучасних інформаційних систем є редакції газет і журналів, оснащені комп'ютерною технікою.

Інформаційна система (в загальному розумінні) - це система, яка виконує або є об'єктом інформаційних процесів, таких як отримання, збирання, зберігання, передача та обробка інформації [4].

Інформаційні системи можуть виконувати один, два або більше процесів одночасно. Обробка інформації залежить від змісту вхідної інформації, але в самому процесі інформація не осмислюється, а лише перетворюється відповідно до раніше розробленого алгоритму.

Інформаційна система (у вузькому розумінні) - це сукупність інформації, технології, програмного забезпечення та організаційних засобів, необхідних для автоматизованої обробки інформації [5].

В інформаційних системах відбуваються такі процеси:

- Введення інформації, отриманої з інформаційних джерел;
- Обробка (перетворення) інформації
- Зберігання вхідної та обробленої інформації;
- Видача інформації користувачам;
- Передача та прийом інформації через мережі.

Розробка інформаційних систем пов’язана з двома проблемами

- Імпорт в систему даних з конкретних предметних областей;
- Створення інтерфейсу користувача для отримання необхідної інформації.

Дані в інформаційних системах можуть зберігатися в неструктурованих або структурованих форматах. Неструктуровані дані - це звичайні текстові документи, такі як статті, реферати, журнали та книги. Системи, що зберігають неструктуровані дані, не зобов’язані надавати конкретні відповіді на запитання користувача, але можуть надати текст документа або список документів для пошуку відповідей.

Рисунок 1.1. – Схема роботи інформаційної системи

Структурування даних вимагає створення правил, які визначають їхній формат, тип, розмір, значення тощо. Щоб підкреслити використання електронних комп’ютерів для автоматизації інформаційних процесів, сучасні інформаційні системи часто називають "автоматизованими інформаційними системами".

Багато речей можна вважати інформаційними системами, включаючи телебачення, мобільні мережі зв’язку, цифрові фотоапарати та відеокамери, а також людей. Інформаційні системи отримують дані від джерела інформації. Ці дані надсилаються в систему для зберігання або певної обробки, а потім передаються споживачеві).

Споживачем може бути людина, пристрій або інша інформаційна система. Зворотний зв’язок може бути встановлений між споживачем і самою інформаційною системою (від споживача до інформаційно-пошукової системи).

Принципи побудови інформаційних систем є відносно сталими. Однак, через різноманітність сфер застосування та форм сучасних інформаційних технологій, існує також велика різноманітність способів класифікації інформаційних систем.

Класифікація інформаційних систем може відбуватись за наступними ознаками:

1. За ступенем автоматизації:

- Ручні: обробка інформації здійснюється людиною;
- Автоматизовані: деякі функції (підсистеми) управління або обробки даних виконуються автоматично, а деякі - людиною
- Автоматизовані: всі функції управління та обробки даних виконуються технічними засобами без участі людини

2. За масштабом використання:

- Поодинокі, які зазвичай реалізуються на окремому персональному комп’ютері без необхідності підключення до комп’ютерної мережі і включають кілька простих компонентів зі спільною базою знань;
- Групові, які орієнтовані на колективне використання інформації, як правило, на основі локальної комп’ютерної мережі;
- Для великих підприємств, де підтримка географічно віддалених комп’ютерних інформаційних вузлів і мереж. Як правило, ієрархічна клієнт-серверна структура з особливим акцентом на серверах;
- Глобальні, які охоплюють територію однієї держави або континенту (наприклад, Інтернет).

3. За цільовим сектором (предметною областю):

- Економіка (управлінські функції на підприємствах);
- Медичні;
- Географічні;
- Менеджмент;
- Виробництво;
- Освіта;
- Навколишнє середовище;
- Судова медицина;
- Військова справа.

4. За місцем діяльності:

- Наукові, які призначені для автоматизації діяльності науковців, аналізу статистики та контролю експериментів.
- Організаційне управління, призначене для автоматизації функцій адміністративного (управлінського) персоналу та окремих офісів (філій) промислових компаній і непромислових організацій (банків, бірж, страхових компаній, готелів тощо).
- Комп'ютерне проектування, які призначені для автоматизації роботи інженерів-конструкторів та розробників нових пристроїв і технологій. Вони допомагають здійснювати:

- розробку нових виробів і технологій їхнього виробництва;
- різноманітні інженерні розрахунки:
- визначення технічних параметрів виробів;
- видаткових норм – трудових, матеріальних, фінансових;
- створення графічної документації (креслень, схем, планувань);
- моделювання проектованих об'єктів;
- створення програм для верстатів з числовим програмним керуванням.

- Організаційне управління, призначене для автоматизації функцій адміністративного (управлінського) персоналу та окремих офісів (філій) промислових компаній і непромислових організацій (банків, бірж, страхових компаній, готелів тощо).
- Системи управління технологічними процесами, призначені для автоматизації різних технічних процесів (гнучкі виробничі процеси, металургія, енергетика тощо).
- Типи взаємодії інформаційних систем:
- Будь-яка взаємодія між двома комп'ютерами (наприклад, через модем). Вимагає участі операторів, які отримують та відправляють. Дозволяє обмінюватися даними у довільних, але заздалегідь визначених форматах;
- Двосторонній віддалений діалог між комп'ютерами та інформаційними системами, наприклад, за протоколом http. Оператор-відправник автоматично обробляє отриманий документ;
- Контрольована обробка потоку. Наприклад, файл, отриманий з електронної пошти, містить HTML-форму, активація якої запускає процес обробки документа; або оператор отримує документ у форматі, зазначеному в електронному листі, і активує програму обробки. Контроль з боку оператора на стороні отримання є обов'язковим;
- Повністю автоматизований процес, який приймає та обробляє електронні документи у визначеному форматі.

У сучасному інформаційному суспільстві створено багато інформаційних систем на різних рівнях автоматизації, з використанням різного обладнання та для різних цілей.

1.2. Застосування інформаційних систем в процесах тестування

Інформаційні системи відіграють важливу роль у тестуванні програмного забезпечення і допомагають забезпечити ефективність, точність і повноту процесу тестування. Сучасні підходи до тестування якості програмного забезпечення включають автоматизоване тестування програмного забезпечення. Це передбачає використання відповідних методів та інструментів і звільняє людей від конкретних функціональних завдань у процесі тестування. Інформаційні системи використовуються протягом усього процесу автоматизованого тестування. Їх можна розділити на наступні системи:

- Системи управління тестуванням – це такі системи дозволяють планувати, виконувати, контролювати та аналізувати весь процес тестування. Вони можуть створювати тестові кейси, керувати наборами тестів, відстежувати дефекти та генерувати звіти [6]. Найпоширенішою системою цього типу є TestRail – спеціалізована система управління тестуванням, яка дозволяє створювати, редагувати та запускати тестові кейси. Вона також надає можливість відстежувати результати тестування та аналізувати звіти.

- Системи відстеження помилок - це системи, які дозволяють реєструвати, відстежувати і управляти дефектами, виявленими під час тестування. Вони забезпечують механізм для спільної роботи команди тестувальників і розробників над виправленням помилок. Самою поширеною такою системою є Jira. Хоча вона в першу чегру відома як система управління проектами, Jira також має змогу відстежувати дефекти та дозволяє створювати завдання щодо відстеження помилок, призначати їх виконавців, відстежувати статуси та пріоритети дефектів, а також генерувати звіти [7].

- Системи контролю версій – такі системи дозволяють зберігати, керувати і відстежувати зміни вихідного коду програмного забезпечення. Вони грають важливу роль у відстеженні версій програми та управлінні кодом, що тестується. Самою поширеною системою виступає Git. Багато людей обирають цю систему

через їшвидкість простоту використання. GitHub, GitLab та Bitbucket - це платформи для спільної роботи над проектами, які підтримують Git [8].

- Системи генерації тестових даних – це системи для автоматичної генерації тестових даних для виконання різних тестових сценаріїв і покриття різних варіантів введення даних. Існує багато таких систем, але серед них можна виділити SQL Data Generator. Цей інструмент надає багато можливостей для налаштування для створення великих обсягів тестових даних.

Під час користування будь-якою програмою ми навіть не замислюємося про те, чого та чи інша програма працює без помилок, зависань тощо. Саме в цьому аспекті тестування відіграє далеко не останню скрипку.

Тестування веб-додатків подібне до перевірки контролю якості веб-сайту. Він сканує веб-сайт на наявність помилок, недоліків і помилок, які можуть погіршити взаємодію з користувачем. Це тестування також діє як ангел-охоронець для веб-сайту, перевіряючи, чи все працює, перш ніж оприлюднити його для громадськості. Це як злегка постукати по кавуну, збираючи його, щоб перевірити, чи він дозрів. Це те ж саме, тільки для вашого сайту [9].

Чому ж тестування в рамках даного процесу настільки важливе? Ось кілька причин:

- За допомогою тестування ви можете перевірити, чи всі вимоги програмного забезпечення, яке ви розробляєте, реалізовано правильно.
- Тестування допомагає виявити помилки та гарантує, що вони виявлені й усунені до етапу доставки програмного забезпечення.
- Тестування зменшує наслідки та ризик втрати, якщо програмний продукт випускається відповідно до неправильних вимог. У цьому випадку ми частково модифікуємо програму, переоцінимо її та спробуємо вдосконалити.
- Тестування допомагає перевірити належну інтеграцію та взаємодію програм і середовищ.

Тому основною метою тестування ПЗ є вимірювання рівня якості програмного продукту на основі його основних вимог. Людям властиво помилятися. Людська помилка може порушити нормальну роботу програмного забезпечення на будь-якому етапі розробки, і наслідки можуть варіюватися від незначних до катастрофічних.

Для складних веб-додатків неможливо усунути всі проблеми. Проте тестування може допомогти виявити більшість помилок, з якими можуть зіткнутися користувачі, потенційно зменшивши ризик майбутніх проблем із вашою програмою [10].

1.3 Класифікація тестування та програмне забезпечення для його реалізації

Тестування програмного забезпечення – процес аналізу програмного забезпечення та супутньої документації з метою виявлення дефектів і підвищення якості продукту.

Тестування може включати різні види тестів, такі як модульні тести, інтеграційні тести, системні тести, регресійні тести, функціональні тести, навантажувальні тести та інші. Кожен тип тесту має свої особливості та цілі, спрямовані на виявлення конкретних проблем у програмному забезпеченні.

Основні мети тестування включають виявлення дефектів, перевірку правильності функціонування програми, валідацію відповідності вимогам, покращення якості та надійності програмного забезпечення, забезпечення безпеки та впевненості в його працездатності. Тестування має багато класифікацій в залежності від задач, серед них можна виділити основну (спрощену) класифікацію тестування:

Рисунок 1.2 – Спрощена класифікація тестування

- По запуску коду на виконання:
  - Статичне тестування
  - Динамічне тестування

Статичне тестування – це тестування без виконання коду. В рамках цього підходу може тестуватись різні компоненти, такі як документація, графічні прототипи, програмний код, параметри (налаштування) середовища виконання програми та підготовлені тестові дані.

Динамічне тестування – тестування із запуском коду на виконання. Запускатися може як код всього додатку цілком, так і код кількох взаємозалежних частин, окремих частин і навіть окремі ділянки коду. Основна ідея цього виду тестування полягає в тому, що перевіряється реальна поведінка (частини) програми.

- За доступом до коду і архітектури програми:
  - Метод білої скриньки
  - Метод чорної скриньки
  - Метод сірої скриньки

Метод білої скриньки – у тестувальника є доступ до внутрішньої структури та коду програми, а також достатньо знань для розуміння побаченого.

Метод чорного ящика – у тестувальника або немає доступу до внутрішньої структури і коду програми, або недостатньо знань їхнього розуміння, або він свідомо не звертається до них у процесі тестування.

Метод сірої скриньки – комбінація методів білої скриньки та чорної скриньки, яка полягає в тому, що до частини коду та архітектури у тестувальника доступ є, а до частини – ні.

- За ступенем автоматизації:
  - Ручне тестування
  - Автоматизоване тестування

Ручне тестування – це тестування, при якому людина запускає тестовий

випадок вручну, не використовуючи інструменти автоматизації. Хоча це звучить дуже просто, такі якості, як терпіння, спостережливість, креативність, здатність проводити нестандартні експерименти та здатність бачити та розуміти, що відбувається в "системі", вимагаються від тестера в певний час.

Автоматизоване тестування – набір технік, підходів та інструментальних засобів, що дозволяє виключити людину із виконання деяких завдань у процесі тестування. Тест-кейси частково або повністю виконує спеціальний інструментальний засіб, проте розробка тест-кейсів, підготовка даних, оцінка результатів виконання, написання звітів про виявлені дефекти - все це і багато іншого робить людина [11].

- По рівню деталізації додатку:
  - Модульне (компонентне) тестування
  - Інтеграційне тестування
  - Системне тестування

Модульне (компонентне) тестування – це тестування спрямоване на перевірку окремих невеликих частин програми, які (зазвичай) можна досліджувати ізольовано з інших подібних частин. За виконання цього тестування можуть перевірятися окремі функції чи методи класів, самі класи, взаємодія класів, невеликі бібліотеки, окремі частини додатка.

Інтеграційне тестування - це тестування спрямоване на перевірку взаємодії між декількома частинами програми (кожна з яких, у свою чергу, перевірена окремо на стадії модульного тестування).

Системне тестування - це тестування, спрямоване на перевірку всього додатка як єдиного цілого, зібраного з частин, перевірених на двох попередніх стадіях. Тут виявляються дефекти «на стиках» компонентів і з'являється можливість повноцінно взаємодіяти з додатком з погляду кінцевого користувача, застосовуючи безліч інших видів тестування, перелічених у розділі.

- За степенем важливості тестованих функцій:
  - «Димове» тестування
  - Тестування критичного шляху
  - Розширене тестування

Димове тестування - це тестування спрямоване на перевірку найголовнішої, найважливішої функціональності, непрацездатність якої робить безглуздою саму ідею використання програми.

Тестування критичного шляху - це тестування спрямоване дослідження функціональності, використовуваної типовими користувачами у типової повсякденної діяльності.

Розширене тестування - це тестування, спрямоване на дослідження всієї заявленої у вимогах функціональності – навіть тієї, яка низько проранжована за ступенем важливості. При цьому тут також враховується, яка функціональність є більш важливою, а яка менш важливою.

- За принципом роботи з додатком:
  - Позитивне тестування
  - Негативне тестування

Позитивне тестування - це тестування спрямоване на дослідження програми в ситуації, коли всі дії виконуються суворо за інструкцією без будь-яких помилок, відхилень, введення невірних даних і т.д. Якщо позитивні тест-кейси завершуються помилками, це тривожна ознака – програма працює невірно навіть в ідеальних умовах (і можна припустити, що в неідеальних умовах вона працює ще гірше).

Негативне тестування – це тестування, спрямоване на дослідження роботи програми в ситуаціях, коли з ним виконуються некоректні операції та/або використовуються дані, що потенційно призводять до помилок (як приклад – ділення на нуль) [12].

Для порівняння характеристик існуючих інструментів, за визначенням

переваг і недоліків кожного з них, буде доцільним розглянути деякі з них, такі як: Selenium, Postman, Allure Report, Cypress, Katalon Studio.

Selenium - це інструмент для автоматизованого тестування веб-додатків. Він надає розширений набір інструментів для автоматизації тестування веб-додатків у різних браузерах і мовах програмування [13].

Selenium підтримується Microsoft Internet Explorer, Google Chrome, Mozilla Suite і Mozilla Firefox для Microsoft Windows, Linux і Apple Macintosh.

Selenium підтримує різні мови програмування, такі як Java, Python, C#, Ruby, JavaScript (з використанням WebDriverIO) тощо. Клієнтські бібліотеки надають розширені можливості для автоматизації тестів у вибраній мові програмування [14].

Postman - це інструмент для розробки та тестування API (інтерфейсів програмування додатків). Він надає можливості для створення, надсилання та прийому HTTP-запитів і відповідей. Postman доступний як настільний додаток для Windows, macOS та Linux, а також як розширення для браузерів Google Chrome та Mozilla Firefox.

Postman дозволяє розробникам та тестувальникам взаємодіяти з різними видами API, включаючи REST, SOAP та GraphQL. З його допомогою можна виконувати різноманітні дії, такі як створення та збереження запитів, організація запитів у колекції, автоматизація тестування API, моніторинг API та багато іншого [15].

Allure Report – це популярний інструмент для генерації докладних і візуально привабливих звітів про результати автоматизованого тестування. Він дозволяє структурувати, аналізувати й візуалізувати тестові дані, надаючи користувачам можливість перегляду результатів тестів у зрозумілому форматі через веб-інтерфейс. Allure підтримує інтеграцію з багатьма фреймворками для тестування, такими як JUnit, TestNG, NUnit, Pytest, Selenium, Playwright та іншими [16].

Cypress – це інструмент для автоматизованого тестування веб-додатків, спеціалізований на тестуванні веб-інтерфейсів. Він призначений для забезпечення простого та ефективного способу створення, запуску та відлагодження тестів, що базуються на JavaScript. Cypress доступний для використання на різних операційних системах, таких як Windows, macOS та Linux [17].

Katalon Studio представляє інтегроване середовище для автоматизованого тестування, яке призначене для автоматизації тестів веб-додатків, мобільних додатків, веб-служб та API [18]. Воно надає рішення "все в одному", що включає в себе не лише інструмент для створення тестів, але й функціональності для їх управління, виконання та аналізу результатів. Katalon Studio має безкоштовну версію з базовим функціоналом, а також комерційні плани з розширеними можливостями для підприємств. Для користувачів, які володіють програмуванням, Katalon Studio дозволяє створювати скриптовані тести за допомогою мов програмування, таких як Groovy або Java [19].

ВИСНОВКИ З РОЗДІЛУ 1

У процесі дослідження було здійснено поглиблений аналіз ключових аспектів тестування вебдодатків – важливого етапу у розробці програмного забезпечення. Серед розглянутих тем були класифікація інформаційних систем, їх роль у розробці вебдодатків, типи тестування та інструменти, що застосовуються для проведення тестування.

Дослідження показало, що існують різні класифікації інформаційних систем, і вони використовуються в таких сферах, як бізнес, освіта та охорона здоров'я. Тестування вебдодатків виявилося вкрай важливим, оскільки воно дозволяє виявити помилки та недоліки у програмному забезпеченні.

Особливу увагу було приділено ролі тестування у розробці вебдодатків, а також обговорені різні види тестування, наведено спрощену класифікацію тестування.

Проведене дослідження надало глибше розуміння методів і інструментів, що використовуються для тестування вебдодатків, а також важливості цього процесу для забезпечення якості програмного забезпечення.

РОЗДІЛ 2
ПРОЦЕС ТЕСТУВАННЯ ВЕБДОДАТКІВ ТА ОЦІНКА ЙОГО ЕФЕКТИВНОСТІ

2.1 Підходи та інструменти для реалізації процесів тестування

Тестування веб-додатків тісно пов’язане з методологіями розробки програмного забезпечення. Кожна з методології впливає на процес тестування, його організацію та інтеграцію в цикл розробки. Існує три популярні підходи до розробки та тестування: Agile, Waterfall та DevOps.

Agile – це методологія розробки програмного забезпечення, яка характеризується ітеративним підходом до розробки, гнучкістю та швидкою реакцією на зміни. Основні принципи Agile включають постійне співробітництво між командами розробників та тестувальників, часті релізи невеликих частин продукту, а також безперервну адаптацію до зворотного зв’язку [20].

Тестування в Agile інтегроване у кожну ітерацію, що дозволяє швидко знаходити та виправляти помилки. Воно проводиться на всіх етапах розробки, починаючи з планування й закінчуючи виконанням автоматизованих і ручних тестів під час кожного спринту.

Рисунок 2.1 – Структура роботи моделі Agile

Одна річ, яка відрізняє Agile від інших підходів до розробки програмного забезпечення, це зосередженість на людях, які виконують роботу, і на тому, як вони працюють разом. Рішення розвиваються завдяки співпраці між само організованими між функціональними командами, які використовують відповідні практики для свого контексту [21].

Переваги Agile для тестування:

- Тестування є безперервним процесом.
- Висока швидкість виявлення та виправлення помилок.
- Можливість швидко реагувати на зміни вимог.

Waterfall – це традиційна методологія розробки, яка передбачає послідовне виконання фаз проекту. Процес починається з аналізу вимог, потім проходить через стадії проектування, розробки, тестування, впровадження та обслуговування [22].

Тестування у Waterfall відбувається на окремій фазі, яка починається після завершення етапу розробки. Це означає, що тестувальники отримують завершений продукт і перевіряють його на відповідність вимогам. Хоча такий підхід забезпечує чітку структуру та планування, його головний недолік полягає в тому, що помилки можуть бути виявлені пізно, коли їх виправлення стає складнішим і дорожчим.

На відміну від інших методів, таких як методологія Agile, Waterfall не допускає гнучкості. Ви повинні закінчити одну фазу, перш ніж почати наступну. Більше того, як зазначено в нашому вступному посібнику з керування проектами, ваша команда не може усунути помилки чи технічну заборгованість, якщо вона вже перейшла до наступної фази проекту.

Переваги Waterfall для тестування:

- Чітка послідовність фаз.
- Легко планувати тестування.

Недоліки:

- Пізнє виявлення помилок.
- Відсутність гнучкості при зміні вимог.

Рисунок 2.2 – Структура роботи моделі Waterfall

DevOps – це методологія, що об’єднує процеси розробки та операцій з акцентом на автоматизацію та безперервну інтеграцію/доставку (CI/CD). DevOps спрямований на прискорення випуску програмного забезпечення за рахунок автоматизації всіх стадій розробки, включаючи тестування [23].

У моделі DevOps команди розробників та операторів більше не є «ізольованими». Ці дві команди можуть бути об’єднані, і інженери працюють з низкою міждисциплінарних навичок протягом усього життєвого циклу програми, від розробки до тестування, розгортання та експлуатації.

Команди DevOps використовують інструменти для автоматизації та прискорення процесів, що сприяє підвищенню надійності. Ланцюжок інструментів DevOps допомагає командам вирішувати важливі основи DevOps, включаючи постійну інтеграцію, безперервну доставку, автоматизацію та співпрацю [24].

Рисунок 2.3 – Структура роботи моделі DevOps

У DevOps тестування є невід'ємною частиною CI/CD-процесу. Це означає, що кожна зміна коду автоматично тестується перед інтеграцією в основну гілку. Такий підхід дозволяє виявляти помилки на ранніх стадіях та швидко їх виправляти.

Переваги DevOps для тестування:

- Безперервне тестування та інтеграція.
- Висока швидкість виявлення помилок та виправлення.
- Автоматизація рутинних процесів.

Недоліки:

- Висока залежність від автоматизації.
- Потребує значних інвестицій у налаштування середовища CI/CD.

Автоматизація тестування дозволяє зменшити витрати часу на повторювані тести, підвищити точність тестування та забезпечити своєчасне виявлення дефектів. У контексті веб-додатків важливо використовувати інструменти, які дозволяють оцінювати як функціональність інтерфейсу, так і продуктивність системи.

Обґрунтування вибору інструментів:

- Allure Report вибрано за його здатність візуалізувати результати тестування та аналізувати продуктивність веб-додатків. Це особливо важливо для систем, що мають великий та тяжкий функціонал.

- Selenium є стандартом для автоматизації функціонального тестування UI. Він дозволяє забезпечити крос-браузерне тестування і тестувати інтерактивні елементи інтерфейсу веб-додатків, що необхідно для забезпечення високої якості роботи користувача.

- Playwright є потужним і сучасним інструментом для автоматизації браузерів, який добре підходить для тестування веб-додатків на різних платформах і забезпечує високу якість і швидкість тестування.

Allure Report – це популярний інструмент для генерації докладних і візуально привабливих звітів про результати автоматизованого тестування. Він дозволяє структурувати, аналізувати й візуалізувати тестові дані, надаючи користувачам можливість перегляду результатів тестів у зрозумілому форматі через веб-інтерфейс. Allure підтримує інтеграцію з багатьма фреймворками для тестування, такими як JUnit, TestNG, NUnit, Pytest, Selenium, Playwright та іншими. [25]. Основні його функції та переваги:

- Докладні звіти: Allure збирає всю інформацію про тести, включно з описом тестових випадків, результатами кожного тесту (успішні, невдалі, пропущені), часом виконання тестів і будь-якими додатковими даними, такими як скріншоти, логи, відео тощо.

- Візуалізація результатів: Allure надає зручний і привабливий веб-інтерфейс, який дозволяє переглядати результати тестування. В інтерфейсі можна побачити структуру тестів, загальні статистичні дані та деталі про кожен тестовий випадок.

- Інтерактивність: Allure дозволяє взаємодіяти зі звітом: користувачі можуть сортувати, фільтрувати й групувати дані тестування, швидко знаходити необхідну інформацію, що значно спрощує аналіз результатів тестів.

- Підтримка різних платформ: Allure підтримує велику кількість фреймворків і мов програмування, що робить його універсальним інструментом для різних команд та проектів.

Selenium – це лідер серед інструментів для автоматизації функціонального тестування веб-додатків. Він дозволяє створювати автоматизовані сценарії, які взаємодіють з інтерфейсом веб-додатка так, як це робив би реальний користувач [26]. Архітектура Selenium складається з 4 компонентів (рис. 2.4):

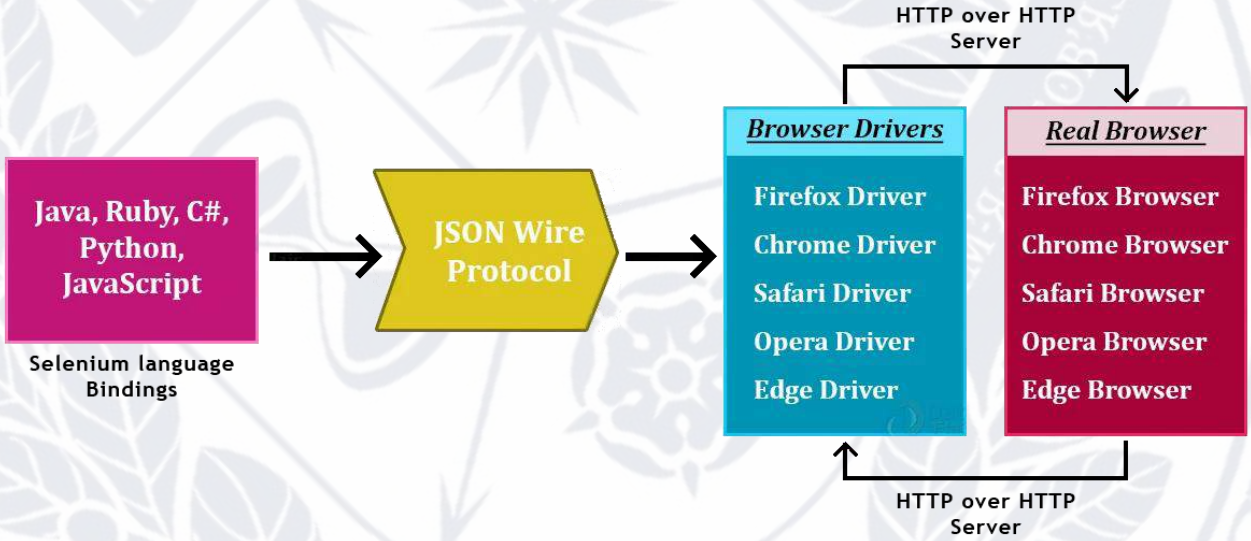

Рисунок 2.4 – Схема роботи компонентів архітектури Selenium

Основні функції та переваги:

- Крос-браузерне тестування: Selenium дозволяє тестувати додатки в різних браузерах (Chrome, Firefox, Safari, Edge), що забезпечує перевірку сумісності між платформами.

- Широка підтримка мов програмування: Інструмент підтримує Java, Python, C#, Ruby, що дозволяє інтегрувати тестування у будь-який стек технологій.

- Гнучкість та масштабованість: Selenium легко адаптується для проєктів різних масштабів, забезпечуючи функціональне тестування як невеликих модулів, так і складних інтерактивних веб-додатків.

- Інтеграція з іншими інструментами: Selenium може працювати в парі з TestNG або JUnit для покращення організації тестів, а також з системами безперервної інтеграції (CI).

Playwright – це інструмент для автоматизованого тестування веб-додатків, розроблений компанією Microsoft. Він забезпечує сучасні функціональні можливості для написання тестів, включаючи підтримку різних браузерів та платформ. Ось детальний опис Playwright і його основних переваг:

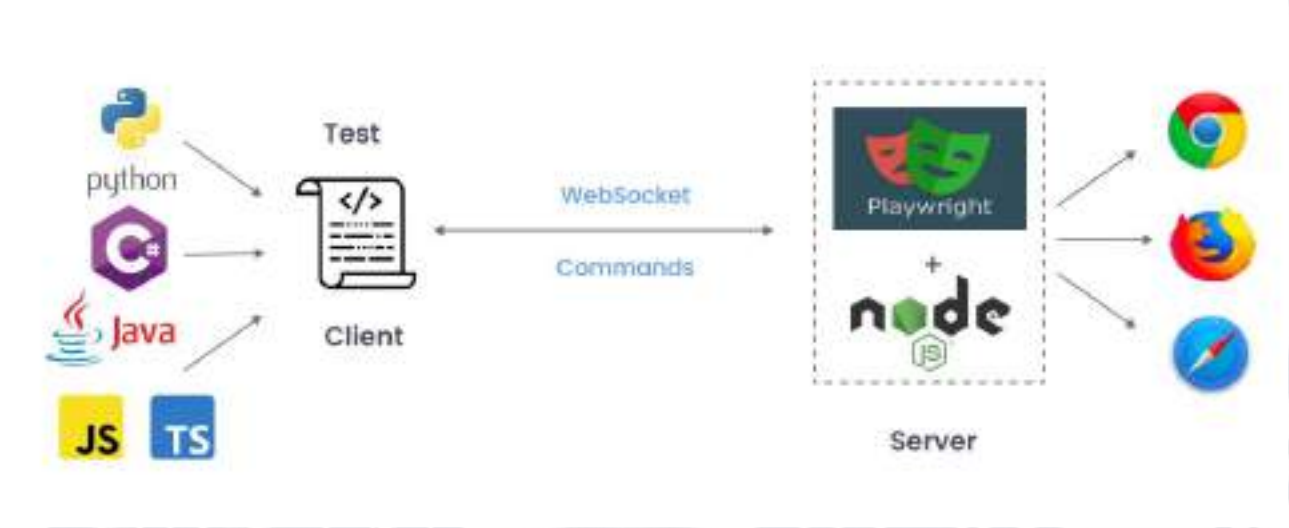

Рисунок 2.5 – Схема роботи компонентів архітектури Playwright

Playwright дозволяє тестувати вебдодатки в різних браузерах (Chromium, Firefox, WebKit) і на різних платформах (Windows, macOS, Linux). Він дозволяє автоматично взаємодіяти з елементами веб-сторінок, такими як заповнення форм, натискання кнопок та перевірка результатів [27]. Основні функції та переваги:

- Підтримка кількох браузерів: Playwright підтримує всі основні браузери (Chromium, Firefox, WebKit) з одного API. Це забезпечує єдину точку доступу для тестування в різних середовищах.
- Міцна підтримка JavaScript і TypeScript: Playwright має повну підтримку для написання тестів на JavaScript і TypeScript, що полегшує інтеграцію з сучасними технологіями фронтенду.
- Завдання паралельного тестування: Playwright дозволяє запускати тести паралельно, що значно зменшує час виконання тестів і підвищує ефективність.
- Контроль за браузерами на рівні процесів: Playwright управляє браузерами через окремі процеси, що знижує ризик впливу тестів один на одного та забезпечує більш стабільні результати.
- Інструменти для відладки: Playwright надає інструменти для відладки, такі як записування відео тестів, створення скріншотів та інтерактивний режим, що допомагає швидше виявляти і усувати помилки.
- Інтеграція з CI/CD системами: Playwright легко інтегрується з популярними системами безперервної інтеграції та доставки (CI/CD), такими як GitHub Actions, GitLab CI, Jenkins.
- Може виконувати тести в режимі headless: Playwright підтримує безголовий режим (headless), що дозволяє запускати тести без графічного інтерфейсу браузера, що корисно для автоматизованих тестових середовищ.
- Підтримка мультимовності та локалізації: Playwright дозволяє тестувати вебдодатки в різних мовах і регіонах.

- API для управління мережевими запитами: Playwright дозволяє перехоплювати, модифікувати і візуалізувати мережеві запити, що може бути корисним для тестування API та відлагодження.

Порівняльна таблиця інструментів тестування наведена в таблиці 2.1.

Таблиця 2.1 – Порівняльна таблиця інструментів тестування

	Selenium	Playwright
Тип тестування	Функціональне тестування UI	Функціональне тестування
Підтримка мов	Java, Python, C#, Ruby	TypeScript, Python, JavaScript
Основне призначення	Тестування інтерфейсу	Паралельне тестування
Інтеграція з CI/CD	Так	Так
Візуалізація результатів	Обмежено	Обмежено

У поєднанні ці інструменти покривають різні аспекти тестування веб-додатків – від оцінки продуктивності до генерації детальних звітів про функціональне тестування.

2.2 Практичне застосування інструментів для тестування

Для початку роботи з Allure Report потрібно встановити Java на робочий девайс, це можна зробити використовуючи офіційний сайт Java або посилання на сайт де зберігаються версії Java для Windows.

Рисунок 2.1 – Посилання на завантаження Java

Після установки, потрібно зайти в консоль та використовуючи команду java – version перевірити чи все установилось.

```
C:\Users\Oleh>java -version
java version "1.8.0_411"
Java(TM) SE Runtime Environment (build 1.8.0_411-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.411-b09, mixed mode)
```

Рисунок 2.2 – Перевірка встановленої версії Java

Наступним кроком буде встановлення самого Allure Report. Для цього потрібно перейти на сторіку Allure Report на GitHub та завантажити актуальну версію.

Рисунок 2.3 – Сторінка Allure Report на GitHub

Після завантаження потрібно відкрити архів та знайти папку bin та скопіювати шлях до цієї папки.

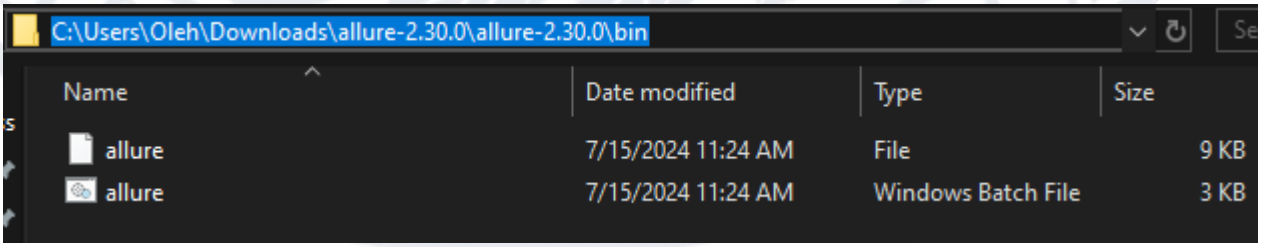

Рисунок 2.4 – Шлях до папки bin

Далі потрібно перейти за наступним шляхом: Control Panel > System and Security > System > Advanced system settings. Далі у вікні потрібно обрати Environment Variables.

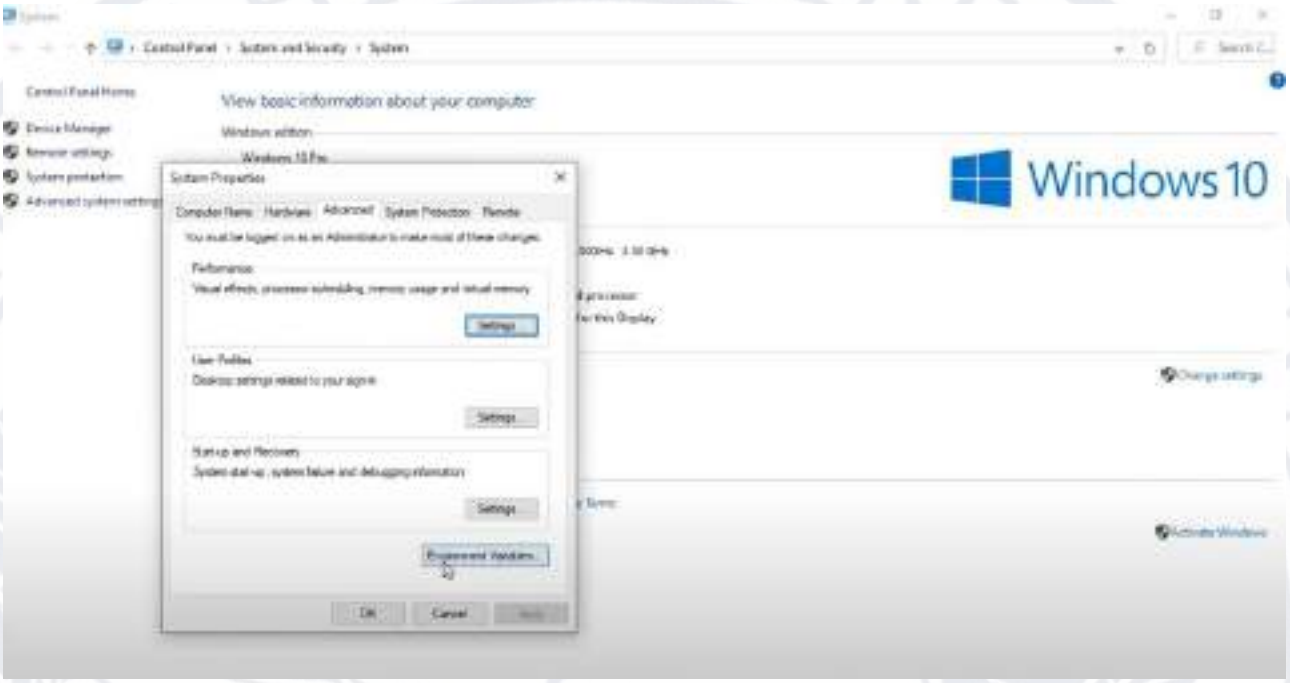

Рисунок 2.5 – Шлях до Environment Variables

Наступним кроком потрібно натиснути сексію Path, після чого відкриється вікно Edit enviroment variable. Натискаємо кнопку New та вставляємо раніше скопіювоний шлях до папки bin. Після цього все має працювати. Для перевірки потрібно використати консоль та команду allure --version.

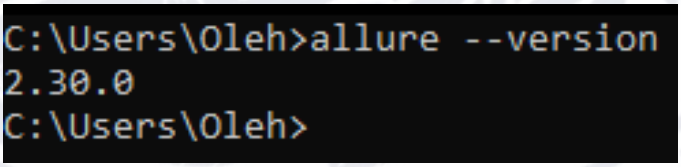

Рисунок 2.6 – Перевірка версії Allure

Щоб використовувати Selenium, потрібно встановити PyCharm - інтегроване середовище розробки, яке працює на мові Python. Python [28] є мовою високорівневого програмування, яка зручна для швидкого прототипування та розробки. Вона використовується для розробки веб-застосунків, науки про дані, штучного інтелекту та багатьох інших галузей. Python має простий та лаконічний синтаксис, що дозволяє зменшити кількість коду та підвищити продуктивність розробки. Вона також має велику кількість сторонніх бібліотек та інструментів, які дозволяють розширити її можливості.

PyCharm - це добре відоме і потужне інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains. Воно пропонує розширені можливості для розробки Python-проектів, включаючи підтримку автодоповнення коду, налагодження, тестування, управління залежностями тощо. [29].

PyCharm надає потужні інструменти для аналізу коду, які допомагають виявляти помилки, небажаний код, вразливості безпеки та інші проблеми. Він також надає підказки та документацію під час розробки, щоб допомогти покращити якість та продуктивність коду.

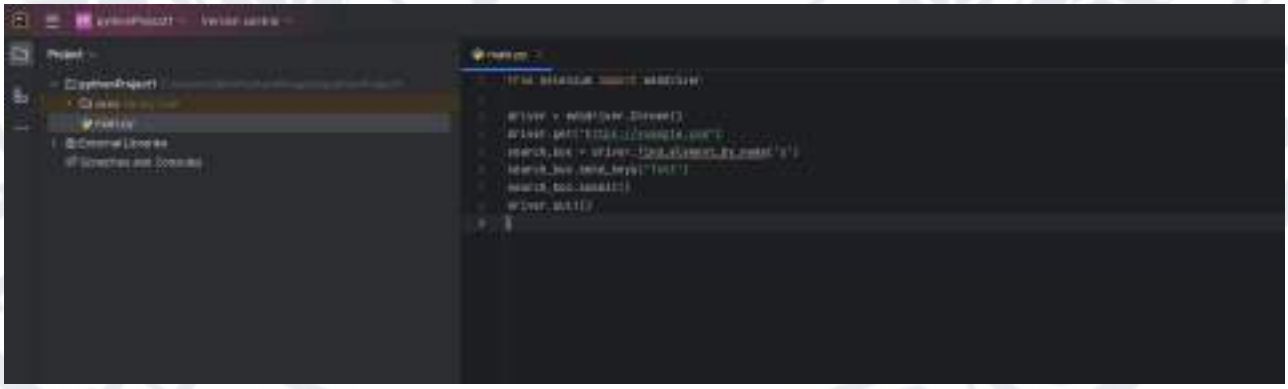

Рисунок 2.12 – Інтегроване середовище розробки PyCharm

Одним з основних недоліків PyCharm є те, що це комерційна ліцензія; існує безкоштовна версія PyCharm Community Edition, але є також платний план PyCharm Professional з додатковими функціями PyCharm – це потужне середовище розробки.

Як потужне середовище розробки, PyCharm вимагає значних комп'ютерних ресурсів. Великі проекти та тривалі робочі сесії можуть споживати багато оперативної пам'яті та процесора, що може вплинути на продуктивність.

При використанні PyCharm слід пам'ятати про необхідний обсяг дискового простору: Встановлення та використання PyCharm може вимагати значного обсягу вільного місця на жорсткому диску, особливо якщо врахувати резервні копії проектів та кешування даних.

PyCharm є хорошим кандидатом на роль інтегрованого середовища розробки завдяки своїй орієнтованості на підтримку мови програмування Python, вбудованій підтримці багатьох технологій і корисним механізмам налагодження, які значно прискорюють і спрощують процес розробки.

Для роботи з бібліотекою Selenium у Python, необхідно спочатку встановити її за допомогою менеджера пакетів pip. Pip – це інструмент для встановлення та управління пакетами Python. Pip є менеджером пакетів, що дозволяє встановлювати, оновлювати та видаляти сторонні бібліотеки та модулі з офіційного репозиторію Python. У середовищі PyCharm це можна зробити кількома способами: через вбудований термінал або графічний інтерфейс.

Для встановлення Selenium через термінал потрібно в нижній частині вікна PyCharm знайти вкладку Terminal та натиснути на неї, щоб відкрити вбудований термінал. Далі необхідно виконати команду pip install selenium після чого pip завантажить і встановить Seleniuum.

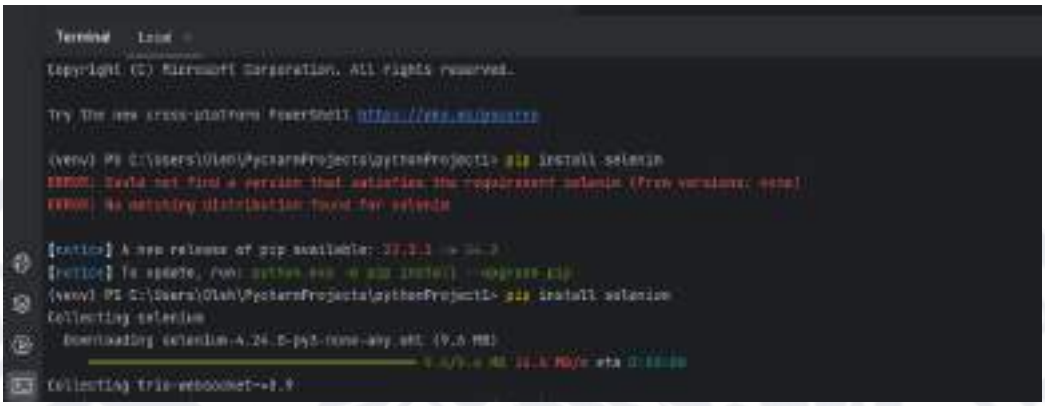

Рис 2.13 – Термінал інтегрованого середовища розробки PyCharm

У якості тестового прикладу буде створено автотест, який буде відкривати сайт Rozetka.com [30] та виконувати пошук використовуючи пошукову стрічку, пошук буде йти за запитом «Мобільні телефони», після чого буде виконуватись перевірка чи дійсно пошуковий запит був правильним.

Автотест – це автоматизований тест, який використовується для перевірки правильності роботи програмного забезпечення без участі людини. Основна мета автотестів – зменшити кількість рутинної роботи тестувальників та підвищити ефективність тестування.

Вони дозволяють швидко перевіряти програму після кожної зміни коду, виявляти дефекти, а також забезпечувати стабільність функцій програми.

Рисунок 2.14 – Реалізація автотесту

Зміна link відповідає за збереження адреси сайту, на якому буде виконуватись тестування. Зміна browser відповідає за зберігання інформації з вибору браузера для роботи, у цьому випадку це браузер Google Chrome [31]. Зміна search_string зберігає інформацію про місце розташування пошукової стрічки на сторінці сайту, для цього вона використовує XPATH [32].

XPATH – це мова запитів, для вибору вузлів з XML документів, або для обчислення величин (наприклад, рядкових, числових або булевих) на основі вмісту XML документа.

Використовуючи search_string.send_keys(«Мобільні телефони») вказуємо автотесту який саме текст потрібно вписати в пошуковий рядок.

За допомогою зміни button записуємо XPATH запит де розташована кнопка виконання пошуку на сторінці.

Викликаємо метод button.click() який натисне саму кнопку пошуку після чого відбудеться пошук по вказаному раніше тексту.

Зміна check служить для зберігання отриманого тексту після пошуку для перевірки чи сходяться отриманий результат пошуку з початковий запитом.
Використовуючи звичайну конструкцію if else перевіряємо чи сходяться отриманий результат пошуку з початковий запитом. Після чого запускаємо автотест та отримуємо результат (рис.2.10).

Рис. 2.15 – Результат виконання автотесту

Як видно, результат успішний і тест пройшов за планом.

Щоб використовувати Playwright з Python, потрібно встановити PyCharm – інтегроване середовище розробки, яке підтримує Python. PyCharm є популярним і функціонально багатим IDE для Python, розробленим компанією JetBrains. Воно пропонує розширені можливості для розробки Python-проектів, такі як автодоповнення коду, налагодження, тестування та управління залежностями. PyCharm доступний у безкоштовній Community Edition та комерційній версії з розширеними можливостями.

Після встановлення PyCharm слід налаштувати проект для роботи з Playwright. Відкрийте PyCharm і створіть новий проект або відкрийте існуючий. Для того щоб інтегрувати Playwright у ваш проект, відкрийте термінал у PyCharm і виконайте команду pip install pytest-playwright. Це дозволить встановити Playwright разом з усіма необхідними бібліотеками для Python.

Далі, для забезпечення коректної роботи Playwright, потрібно завантажити браузери, які використовуються для тестування. Для цього виконайте команду playwright install у терміналі PyCharm. Ця команда автоматично завантажить та встановить всі необхідні браузери, такі як Chromium, Firefox і WebKit.

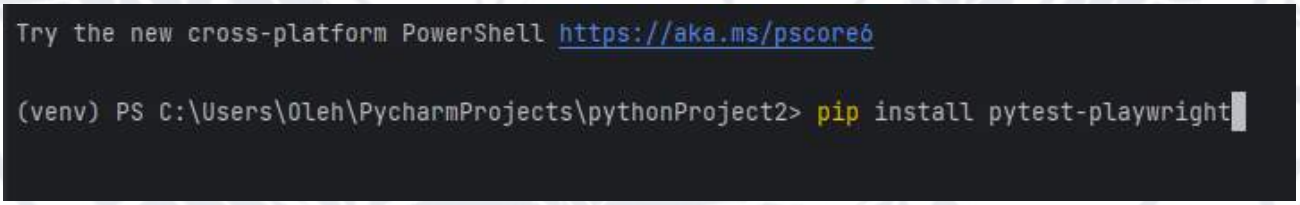

Рис. 2.16 – Встановлення Playwright

Після завершення установки Playwright та браузерів, ви можете створити нові файли тестів у вашому проекті та почати написання тестових скриптів. Для запуску тестів скористайтеся командою playwright test у терміналі або запустіть тести безпосередньо з PyCharm, натиснувши кнопку запуску поруч із вашим

тестовим файлом.

Рис. 2.17 – Встановлення бібліотек Playwright

Для перевірки роботи функціоналу Playwright було створено 4 теста, які перевіяють сайт Wikipedia.org на роботу його функціоналу.

- Перший тест виконує перевірку на перехід на англійську версію сайту, після цього перевіряє на ній наявність напису «Welcome to Wikipedia».

```
from playwright.sync_api import Page, expect

def test_wiki(page: Page):
    page.goto('https://www.wikipedia.org')
    page.get_by_role('link', name='English').click()
    expect(page.get_by_text('Welcome to Wikipedia')).to_be_visible()


def test_search_box(page: Page):
    page.goto('https://www.wikipedia.org')
    expect(page.get_by_role('searchbox')).to_be_visible()


def test_other_languages_links(page: Page):
    page.goto('https://www.wikipedia.org')
    expect(page.get_by_role('link', name='Deutsch')).to_be_visible()
    expect(page.get_by_role('link', name='Español')).to_be_visible()
    expect(page.get_by_role('link', name='Français')).to_be_visible()


def test_contributions_button(page: Page):
    page.goto('https://en.wikipedia.org/wiki/Main_Page')
    expect(page.get_by_role('link', name='Contributions')).to_be_visible()

```

Рисунок 2.18 – Лістинг створених тестів

- Другий тест виконує перевірку наявності елемента "Search" на головній сторінці.
- Третій тест виконує перевірку наявності посилання на інші мови.
- Четвертий тест виконує перевірку наявності логотипу Wikipedia на сторінці "English".

Щоб запустити в роботу тести потрібно виконати команду pytest –headed, яка запустить пробіг усіх тестів. Після виконання буде отримано результат у консоль.

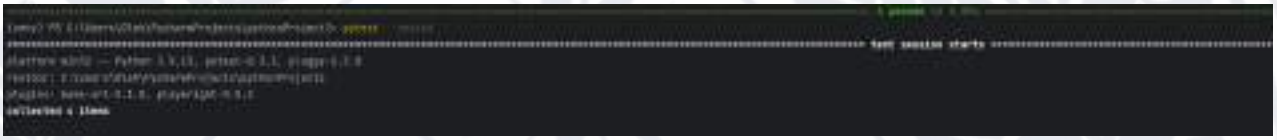

Рисунок 2.19 – Результат виконання тестів

Як видно, усі тести успішно спрацювали та видали очікуваний результат. Якщо будь-який з тестів провалиться то результат у консолі про це повідомить та покаже який саме тест провалився.

Рисунок 2.20 – Результат виконання тестів з помилкою

2.3 Вимірювання ефективності тестування: метрики та KPI

Метрики тестування використовуються для оцінки якості процесу

тестування, ефективності виконаних тестів та виявлення слабких місць у тестовому покритті. Оцінка цих показників дозволяє командам приймати обґрунтовані рішення щодо подальшої оптимізації процесу тестування та забезпечення якості програмного продукту. Нижче наведені ключові метрики, що широко застосовуються у тестуванні програмного забезпечення.

Метрика покриття тестами (Test Coverage). Ця метрика показує, яку частку коду або функціональності було охоплено тестами. Високий рівень покриття тестами свідчить про те, що більшість коду або функціоналу було перевірено на наявність помилок, що зменшує ризик випуску неперевіреного функціоналу [33]. Покриття тестами розраховується за такою формулою:

$$\text{ТС} = \left(\frac{\text{КПЕ}}{\text{ЗКЕ}}\right) \times 100\% \tag{2.1}$$

де КПЕ – загальна кількість функцій, модулів чи рядків коду, що були протестовані;

ЗКЕ – загальна кількість функцій, модулів чи рядків коду у проекті.

Час виконання тестів (Test Execution Time). Час виконання тестів є важливою метрикою, особливо у середовищах з короткими циклами розробки, таких як Agile або DevOps. Від ефективності виконання тестів залежить швидкість виходу продукту на ринок [34]. Час виконання тестів обчислюється так:

$$\text{ТЕТ} = \text{ЧЗТ} - \text{ЧПТ} \tag{2.2}$$

де ЧЗТ – це час, коли всі тести завершили своє виконання;

ЧПТ – це час, коли почалося виконання тестів.

Ця метрика оцінює, скільки часу потрібно для виконання всіх тестів.

Кількість знайдених дефектів (Defect Count (Defect Density)). Кількість знайдених дефектів показує, скільки проблем було виявлено під час тестування. Ця метрика дозволяє оцінити загальний стан продукту, розподіл і види помилок.

Вона може бути розбита за категоріями: критичні, функціональні, інтерфейсі тощо. Формула для розрахунку:

$$DC = \left(\frac{КВД}{ЗКЕ}\right) \quad (2.3)$$

де КВД – це загальна кількість помилок, знайдених у процесі тестування;
ЗКЕ – загальна кількість функцій, модулів чи рядків коду у проекті.

Чим більше дефектів було знайдено на етапі тестування, тим менше їх залишається для користувачів після релізу.

Коефіцієнт успішного виконання тестів (Pass Rate). Ця метрика визначає, який відсоток тестів було виконано успішно без виявлення помилок. Високий коефіцієнт проходження тестів може свідчити про стабільність продукту, однак він також може вказувати на недостатнє тестове покриття або надто прості тести [35]. Формула для розрахунку коефіцієнта успішного виконання тестів:

$$PR = \left(\frac{КУТ}{ЗКТ}\right) \times 100\% \quad (2.4)$$

де КУТ – це кількість тестів, які завершилися без виявлення дефектів;
ЗКТ – це загальна кількість тестових сценаріїв, що були виконані.

Швидкість тестування (Test Efficiency). Швидкість тестування визначає, наскільки ефективними є тестові сценарії щодо виявлення дефектів. Висока швидкість вказує на те, що тести знаходять велику кількість дефектів, використовуючи мінімум тестових сценаріїв [36].
Формула для розрахунку швидкості тестування:

$$TE = \left(\frac{КВД}{КПТ + КДПР}\right) \times 100\% \quad (2.5)$$

де КВД – це кількість дефектів, знайдених під час тестування;
КПТ – це загальна кількість тестових сценаріїв, що були виконані;

КДПР – це кількість дефектів після релізу продукту.

Регулярний аналіз метрик і KPI дозволяє команді підвищувати ефективність тестування. Використання автоматизованих інструментів, таких як Selenium або Playwright, може значно скоротити час виконання тестів та підвищити їх покриття. Важливо періодично переглядати тестові сценарії та оновлювати їх відповідно до змін у програмі для забезпечення актуальності тестування [37].

ВИСНОВКИ З РОЗДІЛУ 2

У другому розділі розглянуто ключові аспекти процесу тестування веб-додатків, які включають теоретичний аналіз методологій, огляд інструментів автоматизації, практичне застосування цих інструментів та вимірювання ефективності тестування.

Аналіз методологій тестування показав різноманітність підходів до забезпечення якості програмного забезпечення. Розглянуті методи допомагають ідентифікувати найважливіші аспекти тестування і зрозуміти, які з них є найбільш критичними для досягнення високої якості продукту.

Огляд інструментів для автоматизації та аналізу тестування дозволив вибрати найбільш ефективні рішення для автоматизованого тестування веб-додатків. Інструменти, такі як Allure Report, Selenium і Playwright.

Практичне застосування цих інструментів підтвердило їхню ефективність у реальних умовах. Налаштування середовища автоматизованого тестування, розробка і запуск тестових сценаріїв, а також інтеграція з системами управління проектами продемонстрували, як автоматизація може значно підвищити ефективність процесу тестування.

Вимірювання ефективності тестування через метрики і ключові показники ефективності (KPI) дало змогу оцінити результати тестування і визначити ключові аспекти, що впливають на якість тестування. Метрики покриття тестами, час виконання тестів, кількість знайдених дефектів, а також KPI, такі як коефіцієнт успішного виконання тестів і коефіцієнт дефектів, знайдених після релізу, є важливими інструментами для оцінки і покращення процесів тестування.

РОЗДІЛ 3
РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Обґрунтування вибору мови програмування та інтегрованого середовища розробки

Python - це мова програмування високого рівня, корисна для швидкого створення прототипів і розробки; її використовують у розробці веб-додатків, науці про дані, штучному інтелекті та багатьох інших галузях. Python має простий і лаконічний синтаксис, який зменшує кількість коду і підвищує продуктивність. Python також має ряд сторонніх бібліотек та інструментів, які можуть розширити її функціональність. Серед переваг мови Python виділяються:

- Велика кількість існуючих фреймворків та бібліотек.
- Висока швидкість розробки.
- Читабельність коду.

Але існують і мінуси:

- Низька швидкість виконання.
- Обмежена підтримка мобільної розробки.

Мова програмування Python - одна з найпопулярніших і найприємніших у використанні мов програмування, з багатьма перевагами і невеликою кількістю недоліків для прикладних галузей. Крім того, Python має активну спільноту розробників, яка активно співпрацює. Це дозволяє легко знайти безліч ресурсів, які допоможуть вам вирішити проблеми, навчитися та отримати підтримку від інших розробників; офіційна документація Python також містить багато корисної інформації.

З усього вищесказаного можна зробити висновок, що Python є найбільш підходящою мовою програмування для реалізації інформаційних систем.

PyCharm - добре відоме і потужне інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains (рис. 3.1). PyCharm пропонує

розширені можливості для розробки проектів на Python, включаючи підтримку завершення коду, налагодження, тестування та управління залежностями. PyCharm доступний у безкоштовній версії Community Edition та комерційній версії з додатковими можливостями.

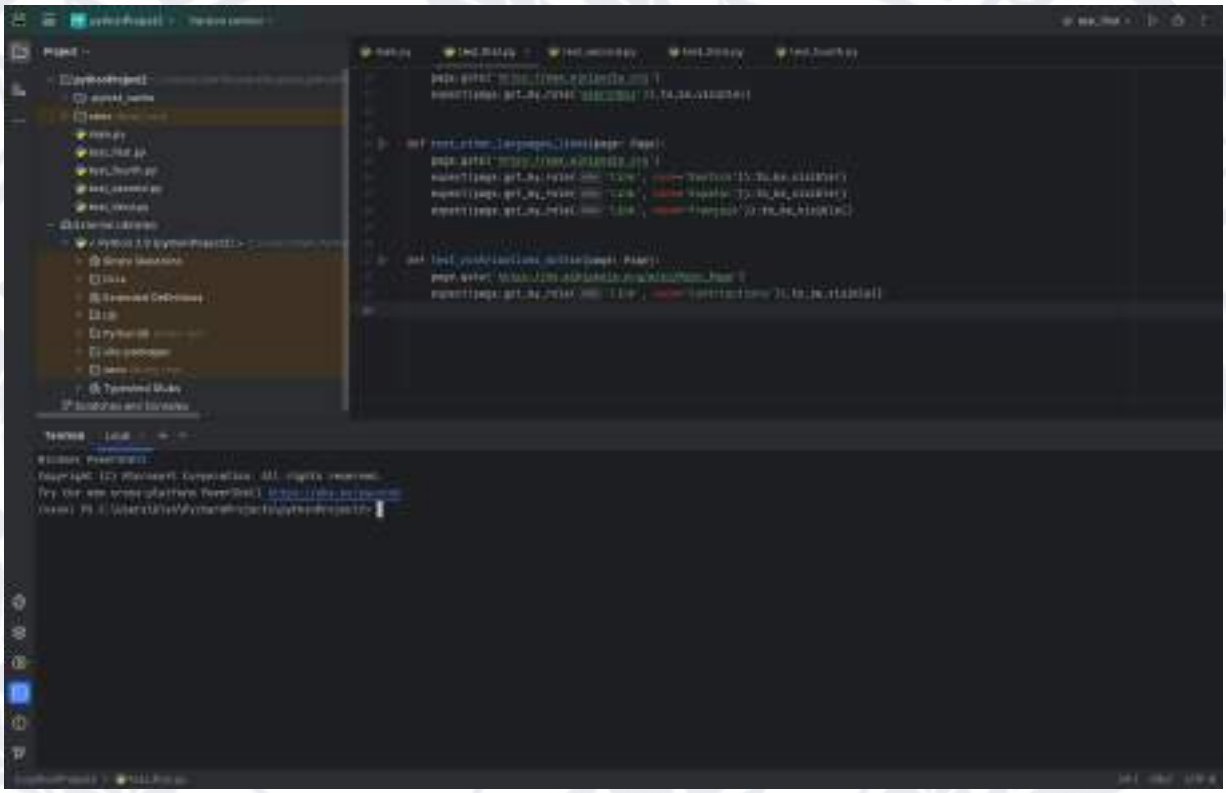

Рисунок 3.1 – Інтегроване середовище розробки PyCharm

PyCharm надає потужні інструменти для аналізу коду, які допомагають виявляти помилки, небажаний код, вразливості безпеки та інші проблеми. Він також надає підказки та документацію для покращення якості та продуктивності коду. Одним з основних недоліків PyCharm є те, що це комерційна ліцензія PyCharm Community Edition має безкоштовну версію, але є також платний план PyCharm Professional з додатковими можливостями PyCharm є потужним середовищем розробки.

Як потужне середовище розробки, PyCharm вимагає значних комп'ютерних ресурсів. Великі проекти та тривалі робочі сесії можуть споживати багато оперативної пам'яті та процесора, що може вплинути на продуктивність.

Використовуючи PyCharm, пам'ятайте про необхідний обсяг дискового простору: Встановлення та використання PyCharm може вимагати значного обсягу вільного місця на жорсткому диску, особливо для резервного копіювання проектів та кешування даних. PyCharm ідеально на роль інтегрованого середовища розробки, оскільки він зосереджений на підтримці мови програмування Python, має вбудовану підтримку багатьох технологій і корисний механізм налагодження, який значно прискорює і впорядковує процес розробки.

Структурний підхід до програмування є одним з найбільш фундаментальних методів розробки програмного забезпечення. В його основі лежить поділ програми на логічні блоки (функції), які легко зрозуміти, підтримувати та модифікувати (рис. 3.2).

Цей підхід виник як відповідь на труднощі, з якими стикалися розробники при використанні ранніх методів програмування, зокрема, так званого «спагетті-коду», коли логіка програми була заплутаною і складною для підтримки.

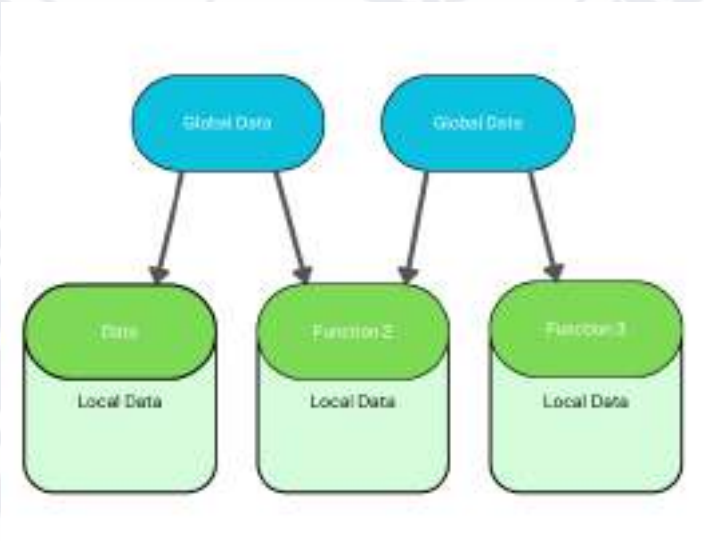

Рисунок 3.2 – Структурний підхід для розробки ПЗ

Застосування цього підходу дозволить спростити код, розділити вміст програми на блоки, які легко зрозуміти, підтримувати та модифікувати.

Чітке розділення логіки: Структурований підхід ділить програму на модулі, кожен з яких виконує певне завдання. Це робить код організованим і легшим для розуміння, а також сприяє повторному використанню функцій у різних проектах, зменшуючи повторне кодування.

Простіше тестування та налагодження: Розділяючи додаток на менші підзадачі, кожну можна тестувати окремо, що спрощує виявлення та виправлення помилок, а також аналіз продуктивності.

Краще розуміння коду та підтримка: Структуровані додатки легше підтримувати. Чітка структура дозволяє новим розробникам швидше включатися в проект, а модулі з визначеними обов'язками знижують ризик негативних впливів змін в інших частинах програми.

Зменшення складності програми: Структурований підхід допомагає управляти складністю великих систем, розбиваючи завдання на менші частини. Це важливо для командної роботи, де різні розробники відповідають за різні модулі.

Використання стандартних алгоритмів і структур даних: Структуроване програмування полегшує застосування готових рішень для типових завдань, таких як сортування та пошук, і сприяє інтеграції бібліотек, підвищуючи надійність програмного забезпечення.

Сприяння оптимізації: Чітке розділення модулів спрощує їх оптимізацію без впливу на інші частини системи, полегшуючи масштабування та адаптацію до нових вимог.

3.2 Розробка алгоритму та запровадження інформаційної системи

Алгоритм роботи програми (рис. 3.3) можна розділити на такі складові: Ініціалізація робочого вікна системи, проходження тестів на Selenium, проходження тестів на Playwright, аналіз результатів тестових випадків, отримання результатів тяестування.

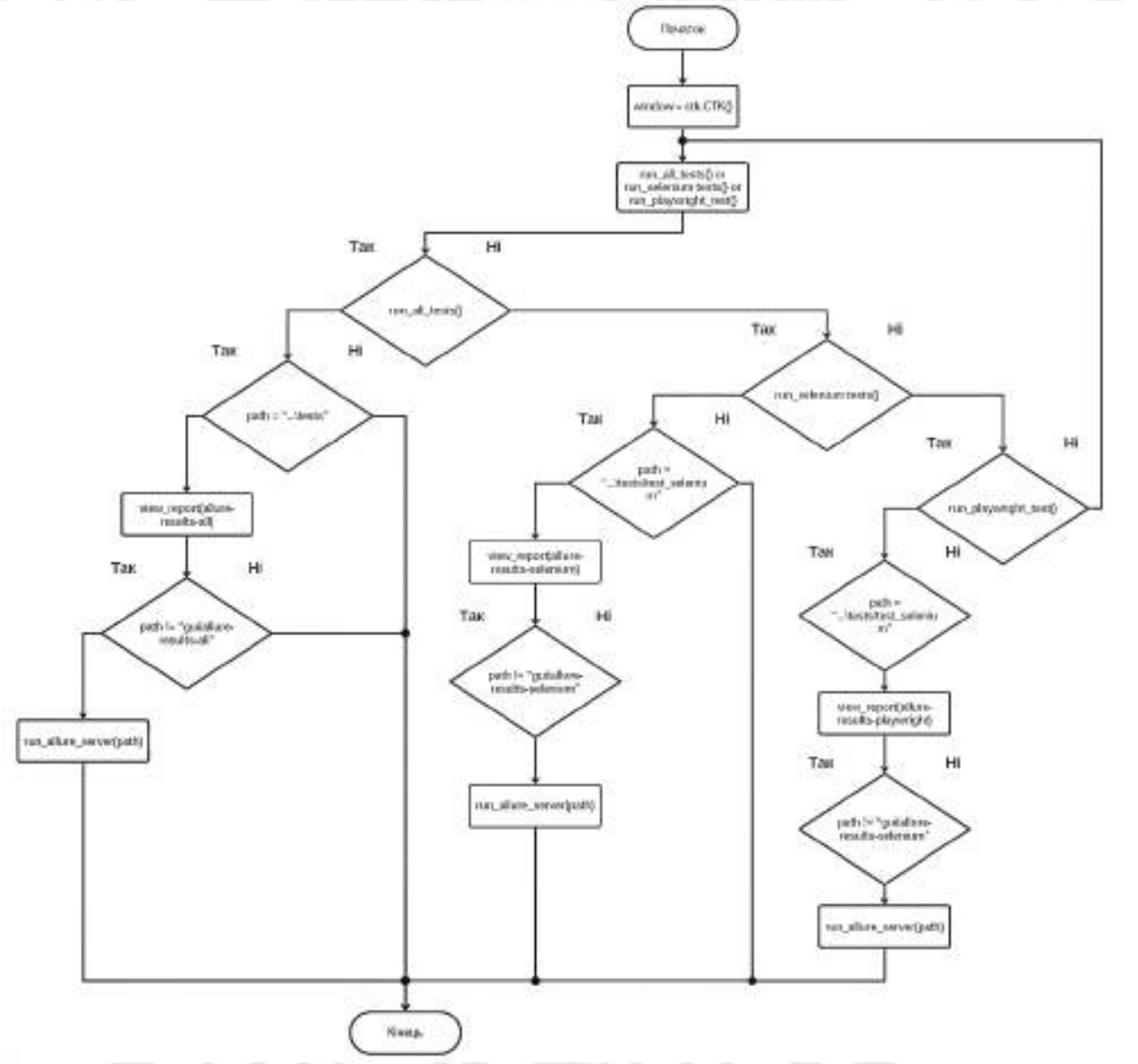

Рисунок 3.3 – Загальний алгоритм роботи програми

Тестування білого ящика і тестування чорного ящика - це два основні підходи до тестування програмного забезпечення. При тестуванні білого ящика тестувальник аналізує внутрішню структуру програми та її код, щоб знайти помилки та дефекти. При тестуванні «чорного ящика» тестувальник тестує зовнішню поведінку програми, не звертаючи уваги на її внутрішню структуру [38].

Оскільки тестування білого ящика вимагає достатніх знань з програмування та доступу до внутрішнього коду програми, цей метод вимагає від тестувальника більше досвіду та часу. Крім того, тестування білого ящика може пропустити деякі помилки, які не пов’язані з внутрішньою структурою програми.

У порівнянні з тестуванням білого ящика, тестування чорного ящика дозволяє проводити тестування швидше і ефективніше. Тестувальники можуть використовувати широкий спектр методів та інструментів для тестування додатку. Крім того, тестування білого ящика може виявити реальні проблеми, які можуть виникнути, коли додаток працює в реальному середовищі [39].

Тестовий кейс - це тестовий артефакт, який містить набір поведінки або умов, необхідних для перевірки конкретної функціональності програмної системи, що розробляється [40].

Обидва підходи до тестування мають свої переваги та недоліки, але тестування «чорного ящика» є більш ефективним підходом для тестування конкретної системи.

Перейдемо до розгляду реалізації коду в PyCharm. Цей код використовує бібліотеку customtkinter для налаштування зовнішнього вигляду інформаційної системи та взаємодії з її функціоналом. Таким чином, весь цей код встановлює зовнішній вигляд системи та елементи для взаємодії з нею. Всі параметри можна змінювати за відповідними потребами.

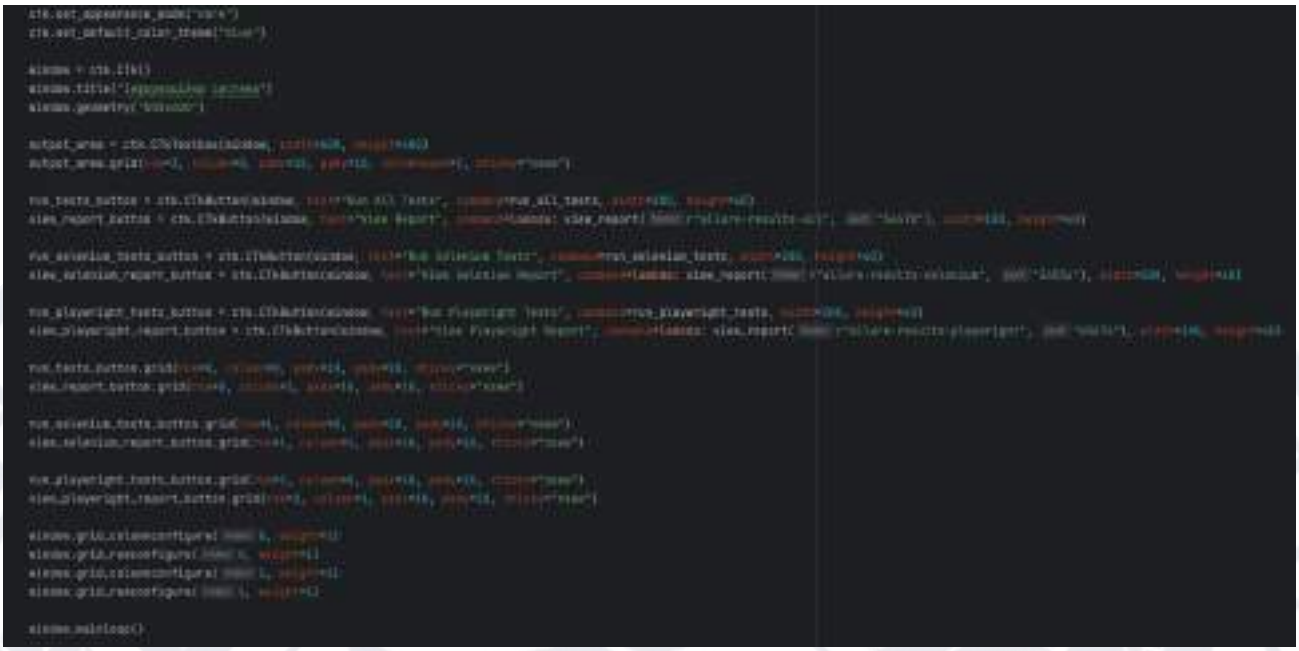

Рисунок 3.4 – Налаштування зовнішнього вигляду графічних візуалізацій

Далі для роботи імпортуються різні модулі та функції, які використовуються для роботи з тестами та інформаційною системою.

```
import json
import os
import shutil
import subprocess
import threading
import webbrowser
import socket
```

Рисунок 3.5 – Імпорт різних модулів та функцій

Описуємо функцію run_allure_server для запуску Allure серверу на якому будуть зберігатись результати тестування. Функція відповідає за автоматичний запуск Allure-сервера для перегляду звітів. Спочатку вона активує віртуальне середовище через виконання скрипту активації activate.bat, а потім, за допомогою

PowerShell, запускає команду allure serve, яка генерує і відкриває звіти з вказаного каталогу на вказаному порту.

```
def run_allure_server(folder, port):
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    full_path = os.path.join(os.getcwd(),folder)
    subprocess.run(["powershell.exe", "allure", "serve", full_path, "--port", port])
```

Рисунок 3.6 – Оголошення функції запуску серверу Allure

Далі описуємо функцію очищення тимчасових файлів, які створює Allure під час роботи серверу. Allure для збереження результатів тестування створює три окремі папки: allure-results-all, allure-results-playwright, allure-results-selenium. Для правильного обчислення метрик, потрібно щоб після кожного нового тестування ці файли видалялись, оскільки програма використовує парсинг з цих файлів для обчислення метрик.

```
def perform_cleanup():
    shutil.rmtree( path: "allure-results-all", ignore_errors=True)
    shutil.rmtree( path: "allure-results-playwright", ignore_errors=True)
    shutil.rmtree( path: "allure-results-selenium", ignore_errors=True)
```

Рисунок 3.7 – Очищення тимчасових файлів Allure

Наступний код є частиною функціоналу запуску роботи тестів. Функція run_all_tests автоматизує процес запуску тестів і обробки результатів. Спочатку вона активує віртуальне середовище за допомогою скрипту activate.bat, після чого запускає всі тести за допомогою команди pytest, зберігаючи результати тестування у визначену директорію allure-results-all для формування звітів Allure. Після завершення тестів функція викликає іншу функцію для отримання метрик

з цих результатів і виводить їх в інтерфейс програми через область для виводу (output_area), показуючи зібрані метрики для Selenium та Playwright.

```
def run_all_tests():
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r"..\tests", "--alluredir", "allure-results-all"])
    metrics = get_metrics('allure-results-all')
    output_area.insert( index: "end",  text: f"Selenium and Playwright combined metrics: \n")
    for metric, value in metrics.items():
        output_area.insert( index: "end",  text: f"{metric} = {value} \n")
```

Рисунок 3.8 – Функція запуску загального тестування

Функція run_selenium_tests активує віртуальне середовище, виконує тестування за допомогою фреймворку pytest для тестів Selenium, зберігаючи результати в директорії allure-results-selenium, а потім отримує метрики тестування за допомогою функції get_metrics.

Після цього функція вставляє в текстове поле output_area заголовок "Selenium metrics" та виводить кожну метрику з її значенням у форматі 'метрика = значення'.

```
def run_selenium_tests():
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r".\tests\test_selenium", "--alluredir", "allure-results-selenium"])
    metrics = get_metrics("allure-results-selenium")
    output_area.insert( index: "end",  text: f"Selenium metrics: \n")
    for metric, value in metrics.items():
        output_area.insert( index: "end",  text: f"{metric} = {value} \n")
```

Рисунок 3.9 – Функція запуску тестування на Selenium

Функція run_playwright_tests() виконує наступне: спочатку активує віртуальне середовище Python, використовуючи команду активації activate.bat. Потім вона запускає тести Playwright через фреймворк pytest, зберігаючи результати тестів у форматі Allure в директорії "allure-results-playwright".

Після цього викликається функція get_metrics(), яка отримує метрики тестування з цієї директорії. В кінці функція виводить отримані метрики у визначене місце інтерфейсу (змінну output_area), вставляючи кожну пару метрика-значення.

```
def run_playwright_tests():
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r"..\tests\test_playwright", "--alluredir", "allure-results-playwright"])
    metrics = get_metrics("allure-results-playwright")
    output_area.insert( index: "end",  text: f"Playwright metrics: \n")
    for metric, value in metrics.items():
        output_area.insert( index: "end",  text: f"{metric} = {value} \n")
```

Рисунок 3.10 – Функція запуску тестування на Playwright

Далі йде функція для обробки метрик. Функція get_metrics(folder) збирає і обробляє метрики тестування з JSON-файлів, що знаходяться в заданій директорії. Спочатку вона створює словник metrics. Потім вона проходить по файлах у вказаній директорії, шукаючи ті, що закінчуються на "result. json". Для кожного такого файлу відкривається і зчитується його вміст у форматі JSON, після чого витягуються час початку і завершення тестів (start, stop), а також їхні статуси (status). Використовуючи ці дані, функція обчислює і додає до метрик час виконання всіх тестів, коефіцієнт успішно пройдених тестів (прохідність) і коефіцієнт неефективних тестів (тести, що завершилися з помилкою). Результатом функції є словник з усіма метриками.

Функція view_report(folder, port) запускає новий потік, який відповідає за роботу Allure-сервера для перегляду тестових звітів. Вона створює окремий потік, де викликається функція run_allure_server з переданими аргументами: шляхом до папки з результатами тестів (folder) і портом (port), на якому буде запущений сервер. Потік оголошується як фоновий (daemon), що означає, що він

автоматично завершиться при завершенні головного процесу. Потім новий потік запускається для роботи у фоновому режимі.

Рисунок 3.11 – Функція обробки метрик тестування

```
def view_report(folder, port):
    t = threading.Thread(target=run_allure_server, args=(folder, port))
    t.daemon = True
    t.start()
```

Рисунок 3.12 – Функція обробки потоку для Allure-серверу

Окрім усього вищезгаданого, в проекті також є директорія **tests**, яка призначена для зберігання тестових сценаріїв, що будуть використовуватися під час розробки. Ця директорія структурована на дві окремі папки: **test_playwright** та **test_selenium**. Такий поділ забезпечує організованість тестів, полегшує їх супровід і дає змогу чітко розділяти тести, що використовують різні інструменти для автоматизації, зокрема Playwright та Selenium.

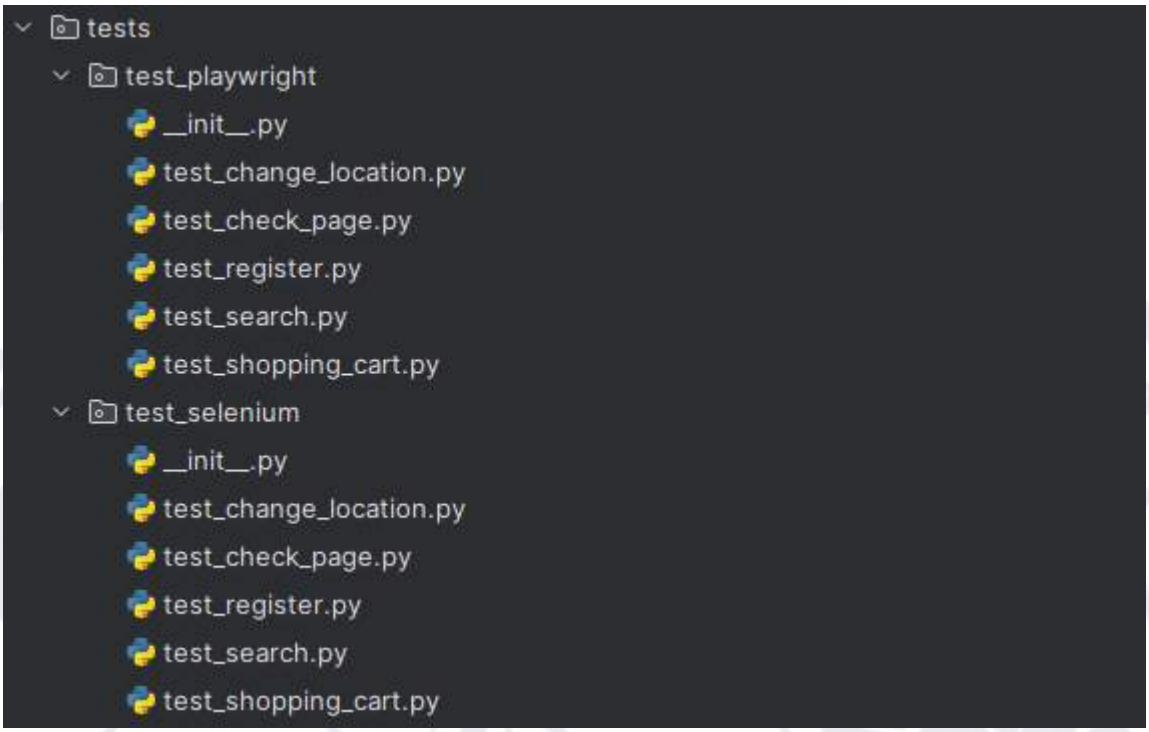

Рисунок 3.13 – Директорія для зберігання тестів

3.3 Тестування роботи інформаційної системи

Створення плану тестування програмного забезпечення є однією з основних концепцій у тестуванні програмного забезпечення. Він потрібен для перевірки роботи інформаційної системи, тест план буде включати в себе тестовий сценарій для перевірки модулів існуючого сайту.

Тест-план – це певний документ або дашборд, який описує стратегію тестування програмного забезпечення, визначає цілі, обсяги, підхід і загальну організацію процесу тестування. Зазвичай тест-план поділяється на менші набори тестів (test suite), які в свою чергу містять тест-кейси (test case) розбиті за категоріями чи функціоналом який тестують. Тест-набори (test suites), які входять до складу тест-плану, зазвичай поділяються на кілька категорій, залежно від типу тестування або функціональних аспектів, що перевіряються. Кожен тест-набір містить тест-кейси (test cases) – це конкретні сценарії тестування, що

описують умови, вхідні дані, дії, які потрібно виконати, очікуваний результат і фактичний результат тестування. Тест-кейси допомагають забезпечити послідовність підходу до тестування та полегшують документування результатів.

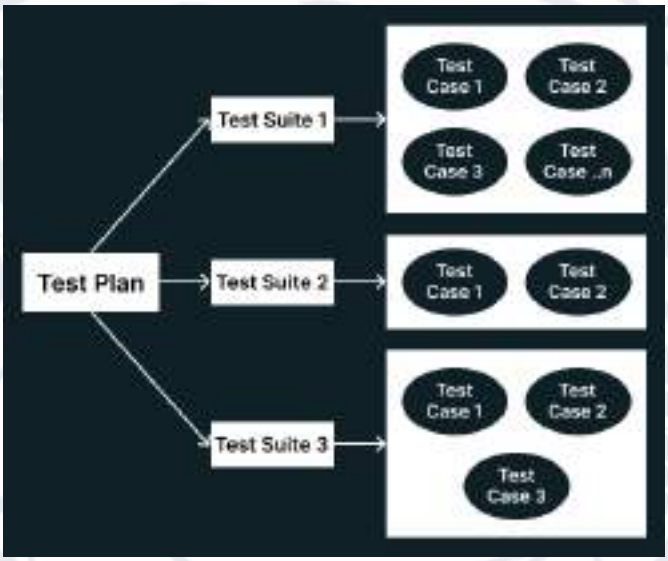

Рисунок 3.14 – Структура тест-плану з наборами тестів і тест-кейсами

Для перевірки функціональності та стабільності роботи розробленої інформаційної системи було створено детальний тест-план за зразком (рис. 3.14). Цей тест-план був розроблений для тестування популярного інтернет-магазину Rozetka. Основна мета плану полягала покритті деяких ключових функціональних аспектів веб-сайту, зокрема перевірки коректної роботи різних компонентів та взаємодії між ними.

Тест-план був структурований і організований за модульним принципом, що дозволило розділити тести на окремі групи відповідно до функціональних зон сайту. Такий підхід полегшив тестування та забезпечив більш точний аналіз результатів для кожного окремого модуля. Наприклад, були виділені окремі модулі для перевірки таких важливих функцій, як процес оформлення замовлення, пошук товарів, функціонування особистого кабінету користувача,

обробка кошика та багато інших. Це дало змогу краще керувати процесом тестування та фокусувати увагу на конкретних функціональних частинах сайту.

Рисунок 3.15 – Структура тест-плану для тестування модулів сайту Rozetka

Розроблена система має наступний вигляд файлів (рис. 3.16).

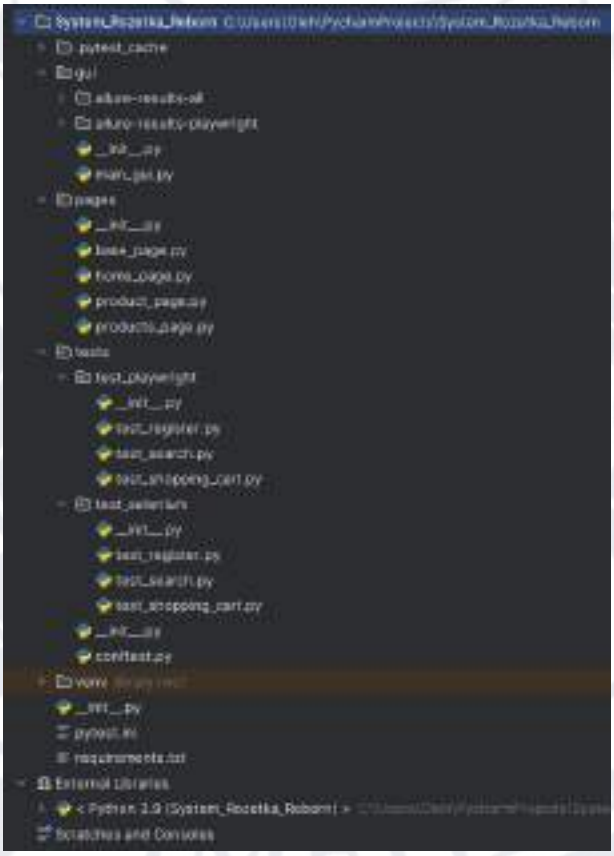

Рисунок 3.16 – Файли у папці з проектом

Тут .py файли розбиті за логічною структурою. Є папка tests, яка містить в собі папки test_playwright та test_selenium, які в свою чергу містять реалізовані тести на відповідних до назви інструментів тестування.

Для коректної роботи системи необхідно мати Python версії 3.10 і вище. Для запуску системи потрібно натиснути комбінацію клавіш Shift + F10. Після запуску відкривається основне вікно програми (рис. 3.17).

Рисунок 3.17 – Вигляд програми

Програма містить шість основних кнопок, кожна з яких виконує контроль та управління певними функціями інформаційної системи, надаючи користувачу інтуїтивно зрозумілий доступ до важливих інструментів (рисунок 3.7). Коротко розглянемо їх призначення та функціонал:

- Run All Tests – ця кнопка забезпечує запуск усіх наявних тестів, як на Selenium, так і на Playwright. Після завершення тестування всі результати зберігаються для подальшого аналізу. Це дозволяє автоматизовано перевіряти працездатність системи в цілому та зберігати результати для подальшого аналізу ефективності тестів.
- View Report – надає можливість переглянути детальний звіт про всі виконані тести, включаючи інформацію про успішність виконання кожного з

них, час виконання та інші ключові метрики. Це дозволяє швидко отримати загальне уявлення про стан тестованої системи.

- Run Selenium Tests – запускає виконання тестів виключно на основі Selenium, що дає можливість перевірити окремі аспекти системи, використовуючи цей інструмент для тестування веб-додатків.
- View Selenium Report – дозволяє переглянути звіт про результати тестування, виконаного за допомогою Selenium. Користувач може переглянути успішність виконаних тестів, виявлені помилки та час їх виконання.
- Run Playwright Tests – запускає виконання тестів, які використовують Playwright, ще один потужний інструмент для тестування веб-додатків. Це дозволяє зосередитися на тестуванні з використанням альтернативного підходу.
- View Playwright Report – ця кнопка надає доступ до звіту, що стосується результатів тестування, проведеного за допомогою Playwright. Користувач отримує можливість проаналізувати успіхи та невдачі тестів, виконаних з використанням цього інструмента.

Звіт відкривається у новій вкладці основного браузеру системи та працює на локальному сервері. Він містить детальну інформацію по виконаним тестам, їх результатам та часу виконання.

Рисунок 3.18 – Звіт по виконанню тестів

Після виконання будь-якого переліку тестів у головний рядок системи буде виводитись результат обробки метрик тестування, що дозволить побачити наглядно їх результат.

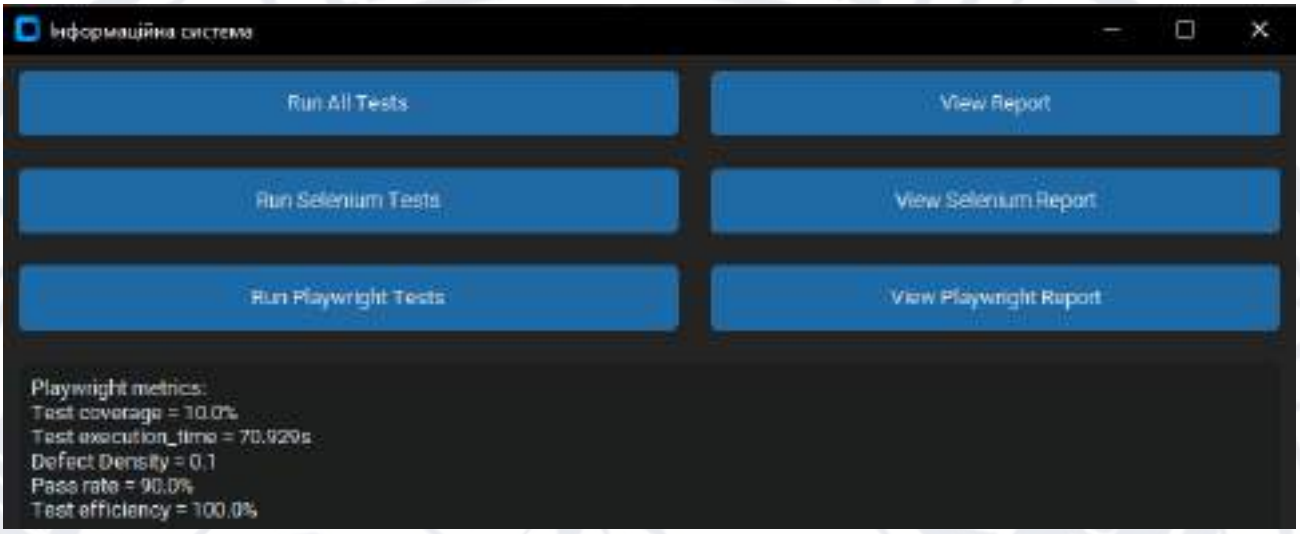

Рисунок 3.19 – Вивід результатів виконання програми

Для тестового сценарію було обрано сайт rozetka.com, який часто використовується українськими користувачами для купівлі товарів повсякденного вжитку та інших товарів. Це досить типовий приклад інтернет-магазину, але він слугує гарним прикладом для проведення тестового сценарію. Для тестування було розроблено п'ять тестових кейсів, реалізованих у двох тестових інструментах - Selenium та Playwright.

Перший тестовий випадок – перевірка правильного відображення інформації на сторінці магазину в конкретному місті. Тест починається з переходу на головну сторінку сайту Rozetka. Після цього користувач переходить до вкладки «Магазини Rozetka», де має відкритися відповідна сторінка. Далі, користувач вибирає місто Запоріжжя зі списку, після чого повинна відкритися карта міста із зазначеними магазинами. На завершення перевіряється правильність заголовка сторінки, який має бути «Магазини Rozetka у місті Запоріжжя» (Додаток А, таблиця А.1).

Другий тестовий випадок – перевірка зміни міста користувача на сайті Rozetka. Тест починається з переходу на головну сторінку сайту Rozetka. Далі користувач відкриває вкладку вибору міста, після чого має з'явитися відповідне вікно. У цьому вікні користувач вибирає місто Львів, при цьому кнопка вибору міста повинна бути виділена обводкою. Після натискання кнопки «Застосувати» вікно вибору закривається, а поточним містом магазину встановлюється «Львів» (Додаток А, таблиця А.2).

Третій тестовий випадок – перевірка роботи сторінки з акціями на сайті Rozetka. Користувач відкриває головну сторінку, після чого переходить до вкладки «Всі акції». Після цього повинна відкритися сторінка з акціями. На завершення перевіряється заголовок сторінки, який має бути «Акції» (Додаток А, таблиця А.3).

Четвертий тестовий випадок – перевірка функціоналу доступу до інформації про оплату на сайті Rozetka. Спочатку користувач відкриває головну сторінку, після чого натискає на вкладку «Довідковий центр», яка повинна відкрити відповідну сторінку. Потім користувач переходить на вкладку «Оплата», і відкривається сторінка з інформацією про оплату. Наприкінці перевіряється заголовок сторінки, який має бути «Оплата» (Додаток А, таблиця А.4).

П’ятий тестовий випадок – перевірка роботи функціоналу реєстрації через електронну пошту. Користувач спочатку відкриває головну сторінку, після чого натискає на кнопку реєстрації, що відкриває відповідне меню. Далі користувач обирає реєстрацію через електронну пошту, після чого з’являється поле для введення пошти. Користувач вводить свою електронну адресу та пароль, а після цього натискає кнопку для завершення реєстрації. Процес реєстрації має бути успішно розпочато (Додаток А, таблиця А.5).

Шостий тестовий випадок – перевірка функціоналу пошуку товарів на сайті Rozetka. Користувач відкриває головну сторінку, після чого знаходить пошукову

стрічку та вводить у неї запит «Мобільні телефони». Після натискання кнопки пошуку виконується процес пошуку. На завершення перевіряється, чи виконано пошук і чи заголовок результуючої сторінки має назву «Мобільні телефони» (Додаток А, таблиця А.6).

Сьомий тестовий випадок – перевірка доступу до категорії ноутбуків Apple на сайті Rozetka. Спочатку користувач відкриває головну сторінку, після чого натискає на кнопку «Каталог», яка відкриває меню з категоріями товарів. Далі користувач знаходить категорію ноутбуків та обирає фірму «Apple», що має відкрити відповідне меню. На завершення перевіряється, чи заголовок категорії дорівнює "Apple MacBook" (Додаток А, таблиця А.7).

Восьмий тестовий випадок – перевірка фільтрації ноутбуків за брендом на сайті Rozetka. Користувач відкриває головну сторінку, після чого натискає на кнопку «Каталог», що відкриває меню з категоріями товарів. Далі користувач знаходить категорію ноутбуків та натискає на неї, в результаті чого відкривається меню категорії "Ноутбуки". Потім користувач вибирає фільтр "HP", після чого товари мають бути відфільтровані за брендом "HP". На завершення перевіряється, чи заголовок результатів фільтрації дорівнює "Ноутбуки HP" (Додаток А, таблиця А.8).

Дев’ятий тестовий випадок – перевірка процесу додавання товару до кошика на сайті Rozetka. Користувач відкриває головну сторінку, після чого натискає на кнопку «Каталог», що відкриває меню з категоріями товарів. Далі він обирає категорію "Ноутбуки", яка відкриває відповідне меню. Потім користувач вибирає фільтр "HP", після чого товари фільтруються за брендом "HP". Користувач натискає на обраний товар, що відкриває сторінку товару. Після цього він натискає на кнопку для додавання товару до кошика, і товар має бути успішно додано до кошика. На завершення перевіряється, чи відкрите модальне вікно кошика (Додаток А, таблиця А.9).

Десятий тестовий випадок – перевірка функціоналу відстеження посилки на сайті Rozetka. Користувач відкриває головну сторінку, після чого переходить на вкладку «Відстежити посилку», яка відкриває відповідну сторінку. Далі він вводить номер замовлення у поле пошуку. Після цього натискає кнопку «Відстежити», що сигналізує про натискання кнопки. На завершення перевіряється результат відстеження, і має з'явитися повідомлення «За номером 123456789 посилку не знайдено…» (Додаток А, таблиця А.10).

Для оцінки ефективності автоматизованого тестування було проведено десять запусків тестів із використанням Playwright та Selenium. Основними показниками для аналізу стали час виконання тестів кожним інструментом та середній час їх виконання, що дозволяє зробити висновки щодо швидкості та стабільності цих рішень.

Для наочного порівняння було реалізовано тестові сценарії тричі, застосовуючи три різні методи: автоматизоване тестування з використанням Playwright (Додаток А, таблиця А.11), автоматизоване тестування з використанням Selenium (Додаток А, таблиця А.12) та ручне тестування (Додаток А, таблиця А.13).

Результати показали, що автоматизовані інструменти суттєво перевершують ручний метод за швидкістю виконання. Зокрема, автоматизація з використанням Playwright і Selenium забезпечує значно вищу швидкість, а також стабільніші результати завдяки зменшенню впливу людського фактора. Крім того, у випадку автоматизації спостерігається значно нижча варіативність часу виконання тестів, що вказує на меншу девіацію результатів порівняно з ручним тестуванням.

Це свідчить про високу надійність та ефективність автоматизованого тестування за допомогою сучасних інструментів, що дає змогу оптимізувати процес тестування та підвищити його якість, скорочуючи витрати часу і мінімізуючи помилки, пов’язані з людським фактором.

Для оцінки роботи програми було проведено 10 прогонів всіх тестів для перевірки успішності їх роботи.

Таблиця 3.1 – Виконання тестового сценарію за допомогою Selenium

П / Т	1	2	3	4	5	6	7	8	9	10
1	P	P	P	P	P	P	P	P	P	P
2	P	P	P	P	P	P	P	P	P	P
3	P	P	P	B	P	P	P	P	P	P
4	P	P	P	P	P	P	P	P	P	P
5	P	P	P	P	P	P	P	P	P	P
6	P	P	P	P	P	P	P	P	P	P
7	P	P	P	P	P	P	P	P	P	P
8	P	P	P	P	P	P	P	P	P	P
9	P	P	P	P	P	P	P	P	P	B
10	F	P	P	P	P	P	P	P	P	P

Після завершення виконання тестування можна зробити висновок, що з десяти запусків тестів лише в трьох виникли проблеми. У одному випадку один тест завершився невдачею, а в інших двох тестування було перервано через непередбачені обставини. Це свідчить про високу надійність та стабільність тестів, які здебільшого пройшли без збоїв і помилок.

Загалом із 100 виконаних тестів лише три не були завершені успішно, що демонструє ефективність на рівні 97%, з рівнем невдачі у 3%. Такий результат свідчить про стабільність тестів і високу якість автоматизованого тестування, що дає змогу довіряти інструменту Selenium для проведення регулярного тестування з мінімальною кількістю збоїв.

Таблиця 3.2 – Виконання тестового сценарію за допомогою Playwright

Т \ П	1	2	3	4	5	6	7	8	9	10
1	P	P	P	P	P	P	P	P	P	F
2	P	P	P	P	B	P	P	P	P	P
3	P	P	P	P	P	P	P	P	P	P
4	P	P	P	P	P	P	P	P	P	P
5	P	P	P	P	P	P	P	P	P	P
6	P	B	P	P	P	P	P	B	P	P
7	P	P	P	P	P	P	P	P	P	P
8	P	P	P	P	P	P	B	P	P	P
9	P	P	P	P	P	P	P	P	P	P
10	P	P	P	P	P	B	P	P	P	P

У випадку Playwright, після завершення тестування можна зробити висновок, що з десяти прогонів у п'яти з них були виявлені певні проблеми. В одному з прогонів один тест не зміг завершитися через неочікувану помилку, а в інших п’яти тестування було перерване. Така статистика свідчить про високу надійність тестових сценаріїв, адже більшість тестів пройшли успішно.

Загалом із 100 виконаних тестів лише шість завершилися збоєм, що забезпечує рівень ефективності на рівні 94% та відсоток невдач 6%. Цей результат підтверджує стабільність автоматизованого тестування, демонструючи якість тестів і їхню здатність виявляти незначну кількість збоїв.

Якщо порівняти результати тестування, проведеного з використанням двох різних інструментів, можна помітити, що їхні показники виявляються майже ідентичними (рисунок 3.10). Обидва інструменти демонструють вражаючий відсоток успішно виконаних тестів, що свідчить про їхню надійність і

ефективність у виконанні поставлених завдань. Водночас відсоток невдалих тестів залишається на надзвичайно низькому рівні.

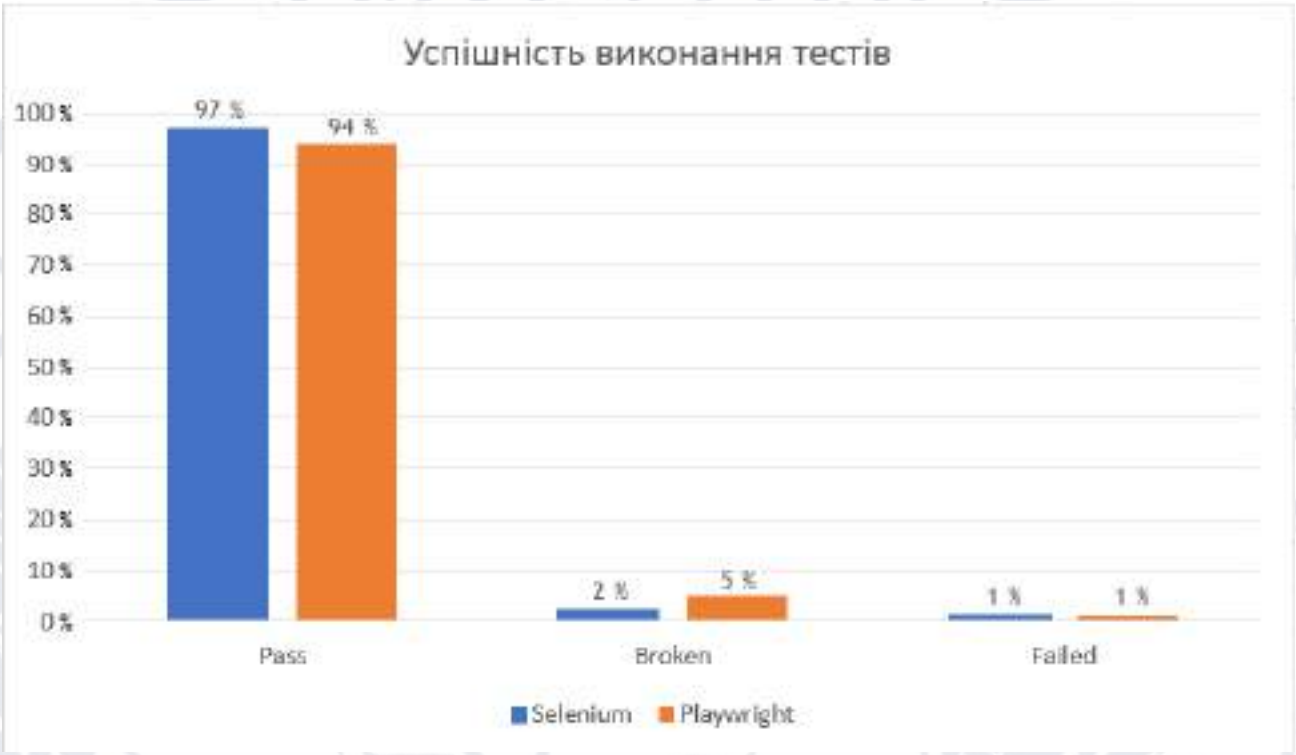

Рисунок 3.20 – Порівняння успішності виконання тестів

Ці результати дозволяють стверджувати, що обидва інструменти є ефективними у виконанні тестів і здатні забезпечити високоякісні результати. Однак важливо зазначити, що, хоча загальні показники успішності подібні, можуть існувати відмінності в специфічних аспектах виконання, таких як швидкість тестування, простота використання чи можливості налаштування. Подальше дослідження цих аспектів може дати більш глибоке розуміння сильних і слабких сторін кожного інструменту, що, у свою чергу, допоможе вибрати найбільш підходящий варіант для конкретних потреб проекту.

ВИСНОВКИ З РОЗДІЛУ 3

У цьому розділі обґрунтовано основні етапи, пов’язані з вибором технологій та підходів, що використовуються при розробці програмного забезпечення. Обрана мова програмування визначалася на основі її популярності, простоти використання, потужності та відповідності вимогам проекту, забезпечення гнучкості розробки та можливості інтеграції з іншими технологіями.

Вибір інтегрованого середовища розробки ґрунтувався на простоті використання, підтримці необхідних плагінів та ресурсів, а також інтуїтивно зрозумілому інтерфейсі, що допоможе підвищити продуктивність команди.

Структурований підхід був обраний для забезпечення чіткого поділу коду на модулі, що полегшує тестування та супровід. Це також зменшило ризик помилок при внесенні змін до додатку.

Алгоритми програмного забезпечення були детально розроблені з урахуванням усіх вимог до функціональності, оптимізації завдань та підвищення швидкості обробки даних.

Було розроблено інформаційну систему для оцінки процесів тестування веб додатків. Було проведено тестування, що сприяло стабільності та надійності системи, забезпечуючи виявлення та усунення можливих помилок.

Було проведено оцінку тестового сценарію, вдалось визначити ефективний час виконання сценарію, виконати порівняння ефективності роботи інструментів тестування, що в результаті дозволило зробити висновок про їх ефективність роботи.

ВИСНОВКИ

В результаті виконаного дослідження було практично розроблено інформаційну систему для аналізу даних оцінки процесів тестування веб-додатків, яка дозволяє збирати, зберігати та аналізувати результати тестування для підвищення ефективності цього процесу. На основі проведеного дослідження можна зробити наступні висновки:

1. На основі аналізу сучасних методів і підходів до тестування веб-додатків, було сформовано архітектурну концепцію інформаційної системи, яка забезпечує збір і аналіз даних про процес тестування.
2. Розроблено алгоритмічний підхід до автоматизації тестування веб-додатків, що дозволяє ефективно збирати тестові дані, а також проводити їх порівняльний аналіз для оцінки якості програмного забезпечення.
3. Обґрунтовано вибір інструментів для реалізації інформаційної системи, зокрема таких як Selenium та Playwright для автоматизації тестування, що дозволило забезпечити високу ефективність і точність збору результатів.
4. Проведено розробку прототипу інформаційної системи з функцією порівняльного аналізу результатів тестування, що дозволило оцінити його функціональність у реальних умовах тестування веб-додатків.
5. Проведене тестування системи продемонструвало її здатність скоротити час, необхідний для виявлення дефектів, а також підвищити загальну якість тестування.
6. Проаналізовано результати тестування і виділено переваги та обмеження розробленої системи, що створює підґрунтя для її подальшого вдосконалення та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Тимчук О. Г., Потапова Н. А. Ризики в управлінні ІТ-проєктами. Прикладні аспекти сучасних міждисциплінарних досліджень: ІІ Міжнародна науково-практична конференція (м. Вінниця, 24 листопада 2023 року). Вінниця: ДонНУ імені Василя Стуса, 2023. С. 185-187.

2. Тимчук О. Г., Потапова Н. А. Моделі тестування програмних продуктів. Прикладні інформаційні технології: матеріали V Всеукраїнської науково-практичної конференції здобувачів, аспірантів та молодих вчених (м. Вінниця, 24 травня 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024. С. 31 - 33.

3. Тимчук О. Г., Потапова Н. А. Метрики вимірювання ефективності процесів тестування програмних продуктів. Прикладні аспекти сучасних міждисциплінарних досліджень: ІІІ Міжнародна науково-практична конференція (м. Вінниця, 1 листопада 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024.

4. Інформаційні системи: їх види, апаратне та програмне забезпечення інформаційної системи. URL: https://spasscool.at.ua/index/informacijni_sistemi_jikh_vidi_aparatne_ta_programne_zabezpechennja_informacijnoji_sistemi/0-193.

5. Інформаційна система. URL: https://uk.wikipedia.org/wiki/Інформаційна_система.

6. Introduction to TestRail. URL: https://support.testrail.com/hc/en-us/articles/7076810203028-Introduction-to-TestRail#Thedashboard.

7. Веретенников В.І., Тарасенко Л.М., Гевлич Г.І. Управління проектами: навч. посібник. К.: Центр навчальної літератури, 2016. 280 с.

8. Сайт документації GitLab. URL: https://docs.gitlab.com.

9. Web Application Testing Guide. URL: https://testsigma.com/guides/web-application-testing/.

10. Kaner C., Falk J., Quoc Nguyen H. Testing computer software. Second edition. K: International Thomson Publishing Company, 2002. 538 p.

11. Svyatoslav Kulikov. Software Testing. Base Course. 3rd Edition. 2022. C. 60-75.

12. Anthony Adesanwo. The Modern Software Testing Handbook: A Quick Guide to Current Software Testing and Quality Assurance Strategies. 2022. 107 c.

13. Сайт Selenium. URL: https://selenium-python.readthedocs.io/getting-started.html.

14. Kalilur Rahman. Science of Selenium: Master Web UI Automation and Create Your Own Test Automation Framework, 1st edition. 2019. 406 p.

15. Сайт Postman. URL: https://www.postman.com/product/what-is-postman/.

16. Сайт Allure Report. URL: https://allurereport.org.

17. Сайт Cypress. URL: https://docs.cypress.io/guides/overview/why-cypress.

18. Сайт Katalon. Platform. URL: https://docs.katalon.com/katalon-platform/about-katalon-platform.

19. Сайт Java. URL: https://www.java.com/en/download/.

20. What is Agile?. URL: https://www.agilealliance.org/agile101/.

21. Dorothy Graham, Rex Black, Erik van Veenendaal. Foundations of Software Testing ISTQB Certification, 4th edition. 2019. 290 p.

22. Waterfall Methodology: A Comprehensive Guide. URL: https://www.atlassian.com/agile/project-management/waterfall-methodology.

23. What is DevOps? URL: https://www.atlassian.com/devops.

24. Mauricio Aniche. Effective Software Testing: A Developer's Guide. 2022. 75 p.

25. Allure Report Documentation. URL: https://allurereport.org/docs/.

26. Сайт документації Selenium. URL: https://www.selenium.dev/documentation/.

27. Сайт Playwright. URL: https://playwright.dev.

28. Сайт Python. URL: https://www.python.org/.

29. Сайт PyCharm. URL: https://www.jetbrains.com/pycharm/.

30. Сайт Rozetka. URL: https://rozetka.com.ua/ua/. (Дата звернення: 22.09.2024).

31. Сайт Google Chrome. URL: https://www.google.com/chrome/. (Дата звернення: 22.09.2024).

32. Сайт документації XPATH. URL: https://developer.mozilla.org/en-US/docs/Web/XPath.

33. Test Coverage. URL: https://en.training.qatestlab.com/blog/technical-articles/test-coverage/.

34. Measuring Testing Success. URL: https://www.qatouch.com/blog/measuring-testing-success-2024/.

35. QA Metrics. URL: https://www.testrail.com/qa-metrics/.

36. Agile Testing Metrics. URL: https://www.linkedin.com/pulse/agile-testing-metrics-software-ulf-eriksson/.

37. What is a KPI? Denition, Types, Examples and Best Practices URL: https://databox.com/what-is-a-kpi

38. Dorothy Graham, Rex Black, Erik van Veenendaal. Foundations of Software Testing ISTQB Certification, 4th edition. 2019. 290 с.

39. Kristin Jackvony. The Complete Software Tester: Concepts, Skills, and Strategies for High-Quality Testing. Independently published, 2022. 508 с.

40. База знань. Тестовий випадок. URL: http://qalight.ua/baza-znaniy/test-case-2/.

ДОДАТКИ

Додаток А

Таблиця А.1 – Перший тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Перейти по вкладці «Магазини Rozetka»	Вкладка «Магазини Rozetka» з’явилась
3	Вибрати серед списку міст Запоріжжя	Відкрилась карта міста Запоріжжя з магазинами
4	Перевірити заголовок сторінки	Заголовок сторінки «Магазини Rozetka у місті Запоріжжя»

Таблиця А.2 – Другий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Перейти по вкладці вибору міста користувача	Відкрилось вікно вибору міста
3	Вибрати місто Львів	Кнопка вибору міста Львів має обводку
4	Натиснути кнопку «Застосувати»	Вікно вибору закрилось. Поточне місто магазину «Львів»

Продовження додатку А

Таблиця А.3 – Третій тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Перейти по вкладці «Всі акції»	Сторінка «Акції» відкрилась
3	Перевірити заголовок сторінки	Заголовок сторінки має назву «Акції»

Таблиця А.4 – Четвертий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Натиснути на вкладку «Довідковий центр»	Сторінка довідкового центру відкрилась
3	Натиснути на вкладку «Оплата»	Сторінка оплати відкрилась
4	Перевірити заголовок сторінки	Заголовок сторінки має назву «Оплата»

Продовження додатку А

Таблиця А.5 - П’ятий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Натиснути на кнопку реєстрації	Відкрито меню реєстрації
3	Натиснути кнопку для реєстрації через електронну пошту	Відкрито поле для введення електронної пошти
4	Ввести електронну пошту	Текст електронної пошти введено
5	Ввести пароль	Пароль введено
6	Натиснути кнопку для завершення реєстрації	Реєстрація розпочалась

Таблиця А.6 – Шостий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перейти на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Знайти пошукову стрічку та ввести дані для пошуку «Мобільні телефони»	У пошуковій стрічці з’явились дані

Продовження додатку А

Продовження таблиці А.6

3	Натиснути кнопки пошуку	Виконується пошук
4	Перевірити чи виконано пошук	Заголовок результуючої сторінки «Мобільні телефони»

Таблиця А.7 – Сьомий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перехід на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Натиснути кнопки «Каталог»	Відкрито мені каталогу
3	Знайти категорію ноутбуків та натиснути на фірму «Apple»	Відкрито мені категорії «Apple»
4	Перевірити, чи заголовок категорії дорівнює "Apple MacBook"	Перевірити, чи заголовок категорії дорівнює "Apple MacBook"

Продовження додатку А

Таблиця А.8 – Восьмий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перехід на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Натиснути кнопки «Каталог»	Відкрито мені каталогу
3	Знайти категорію ноутбуків та натиснути на неї	Відкрито меню категорії "Ноутбуки"
4	Натиснути на фільтр "HP"	Товари фільтруються за брендом "HP"
5	Перевірити, чи заголовок результатів фільтрації дорівнює "Ноутбуки HP"	Заголовок результатів дорівнює "Ноутбуки HP"

Таблиця А.9 – Дев’ятий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перехід на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Натиснути кнопки «Каталог»	Відкрито мені каталогу
3	Натиснути на категорію "Ноутбуки"	Відкрито меню категорії "Ноутбуки"

Продовження додатку А

Продовження таблиці А.9

4	Натиснути на фільтр "HP"	Товари фільтруються за брендом "HP"
5	Натиснути на товар	Відкрито сторінку товару
6	Натиснути на кнопку для додавання товару до кошика	Товар додано до кошика
7	Перевірити, чи відкрито модальне вікно кошика	Відкрито модальне вікно кошика

Таблиця А.10 – Десятий тестовий випадок

Крок №	Опис кроку	Очікуваний результат
1	Перехід на сайт https://rozetka.com.ua/	Відкрито головне вікно сайту
2	Перейти по вкладці «Відстежити посилку»	Відкрито сторінку відстеження посилки
3	Ввести у поле пошуку номер замовлення	Введено номер замовлення
4	Натиснути кнопку «Відстежити»	Кнопка «Відстежити натиснута»
5	Перевірити результат відстеження	З’явився надпис «За номером 123456789 посилку не знайдено…»

Продовження додатку А

Таблиця А.11 – Результат виконання тестових випадків за допомогою Playwright

Playwright												
Тестовий пробіг												
Тестовий випадок	**№**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Сер.**
	1	5.9	5.9	5.9	5.9	6	5.9	6	5.9	6	5.9	5.9
	2	5.1	5	5	4.9	5	5	5.1	5	5	5.1	5
	3	5.1	5	4.9	5	5	5.1	4.9	5	5	5.1	5
	4	4.6	4.6	4.3	4.4	4.4	4.3	4.6	4.9	4.4	4.6	4.5
	5	7.9	8.1	8	8	7.9	7.9	8.1	8	7.9	7.9	8
	6	7.9	8.2	8.3	7.8	7.9	7.8	8.2	8	7.9	7.9	8
	7	6.9	6.8	6.7	6.8	6.7	6.5	6.8	6.8	6.7	6.9	6.8
	8	8.7	6.7	7.1	6.7	7	7.2	6.7	6.9	7	8.7	7.2
	9	10	10.2	9.9	10	10.1	9.8	9.9	10	10.1	10	10
	10	5.9	5.8	5.9	5.9	6.2	6	5.9	5.8	6.2	5.9	5.9
Заг.		68	66.3	66	65.4	66.2	65.5	66.2	66.3	66.2	68	66.3

Продовження додатку А

Таблиця А.12 – Результат виконання тестових випадків за допомогою Selenium

Selenium												
Тестовий пробіг												
Тестовий випадок	**№**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Сер.**
	1	7.1	7.2	7.1	7.3	7.1	7.2	7.3	7.2	7	7.1	7.1
	2	6.7	6.8	6.8	6.7	7	6.7	6.8	6.8	6.6	7	6.7
	3	5.1	5.2	5.2	5.7	5.4	5.3	5.3	5.1	5	5.4	5.3
	4	6.3	6.3	6.4	6.4	6.5	6.7	6.8	6.2	6.2	6.5	6.5
	5	4.1	4.2	4.2	4.3	4.2	4.9	4.1	4.1	4	4.2	4.3
	6	4.4	4.5	4.4	4.5	4.2	4.8	4.3	4.4	4.3	4.1	4.2
	7	5.6	5.5	5.5	5.6	5.7	6.1	5.6	5.4	5.5	5.6	5.6
	8	7.9	6.4	6.4	6.9	6.3	6.7	6.3	6.3	7.8	6.3	6.5
	9	7.7	7.5	7.5	7.7	7.8	7.7	7.3	7.4	7.6	7.8	7.5
	10	6.8	6.8	6.8	6.9	6.8	7.3	7.5	6.7	6.7	6.8	6.8
Заг.		61.7	60.4	60.3	62	61	63.4	61.3	59.6	60.7	60.8	60.5

Продовження додатку А

Таблиця А.13 – Результат виконання тестових випадків вручну

Вручну												
Тестовий пробіг												
Тестовий випадок	**№**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Сер.**
	1	8	8.3	8.4	8.8	8.9	9	9.1	8.5	8.8	8.2	8.5
	2	9.6	9.1	8.4	8.1	7.9	7.7	7.8	7.9	7.5	9	8.9
	3	6.7	6.8	6.9	6.5	6.7	6.8	6.4	6.4	6.1	6.7	6.5
	4	7.5	8	7.9	7.6	7.7	7.8	7.7	7.5	7.9	7.8	7.9
	5	9.9	10	9.8	9.5	9.4	9.3	9.5	9.1	9	9.8	9.2
	6	6.6	6.5	6.1	6	6.5	6.3	6.8	6.1	6.8	6.2	6.4
	7	7.7	7.9	8.5	9	8.5	7.8	8	9	9.2	8.1	8
	8	12.1	12.5	13	11.5	13.1	12.5	12.4	12.6	12.3	12	12.2
	9	20	21.1	20.8	21.5	20.5	21.9	20.1	20.6	20.1	20.9	20.2
	10	9.1	9.5	9.3	9.1	9.7	9.3	9.2	9.7	10	9	9.3
Заг.		97.2	99.7	99.1	97.6	98.9	98.4	97	97.4	97.7	97.7	97.1

Додаток Б

main_gui.py

```
import json
import os
import shutil
import subprocess
import threading
import webbrowser
import socket
import random

import customtkinter as ctk


def run_allure_server(folder, port):
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    full_path = os.path.join(os.getcwd(),folder)
    subprocess.run(["powershell.exe", "allure", "serve", full_path, "--port", port])


def perform_cleanup():
    shutil.rmtree("allure-results-all", ignore_errors=True)
    shutil.rmtree("allure-results-playwright", ignore_errors=True)
    shutil.rmtree("allure-results-selenium", ignore_errors=True)

def run_all_tests():
    perform_cleanup()
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r"..\tests", "--alluredir", "allure-results-all"])
    metrics = get_metrics("allure-results-all")
    output_area.insert("end", f"Selenium and Playwright combined metrics: \n")
    for metric, value in metrics.items():
        output_area.insert("end", f"{metric} = {value} \n")

def run_selenium_tests():
    perform_cleanup()
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r"..\tests\test_selenium", "--alluredir", "allure-results-
selenium"])
    metrics = get_metrics("allure-results-selenium")
```

Продовження додатку Б

```
    output_area.insert("end", f"Selenium metrics: \n")
    for metric, value in metrics.items():
        output_area.insert("end", f"{metric} = {value} \n")

def run_playwright_tests():
    perform_cleanup()
    subprocess.run([r"..\venv\Scripts\activate.bat"])
    subprocess.run(["pytest", r"..\tests\test_playwright", "--alluredir", "allure-results-
playwright"])
    metrics = get_metrics("allure-results-playwright")
    output_area.insert("end", f"Playwright metrics: \n")
    for metric, value in metrics.items():
        output_area.insert("end", f"{metric} = {value} \n")

def view_report(folder, port):
    t = threading.Thread(target=run_allure_server, args=(folder, port))
    t.daemon = True
    t.start()

def get_metrics(folder):
    metrics = dict()
    metrics['Test coverage'] = str(10 / 100 * 100) + "%"

    cases_starts = []
    cases_ends = []
    test_statuses = []
    for file in os.listdir(os.path.join(os.getcwd(), folder)):
        if file.endswith("result.json"):
            with open(os.path.join(os.getcwd(), folder, file), "r", encoding="utf-8") as f:
                contents = json.load(f)
                cases_starts.append(contents["start"])
                cases_ends.append(contents["stop"])
                test_statuses.append(contents["status"])

    metrics["Test execution_time"] = str((max(cases_ends) - min(cases_starts)) / 1000)
+ "s"

    metrics["Defect Density"] = str(len(tuple(filter(lambda x: x == "broken",
test_statuses))) / len(test_statuses))
```

Продовження додатку Б

```
    metrics["Pass rate"] = str(len(tuple(filter(lambda x: x == "passed", test_statuses))) /
len(test_statuses) * 100) + "%"

    print(str(len(tuple(filter(lambda x: x == "broken", test_statuses)))))

    broken_tests_count = len(tuple(filter(lambda x: x == "broken", test_statuses)))

    if broken_tests_count == 0:
        metrics["Test efficiency"] = "0%"
    else:
        metrics["Test efficiency"] = str(broken_tests_count / broken_tests_count * 100)
+ "%"

    return metrics

if __name__ == '__main__':
    ctk.set_appearance_mode("dark")
    ctk.set_default_color_theme("blue")

    window = ctk.CTk()
    window.title("Інформаційна система")
    window.geometry("800x600")

    output_area = ctk.CTkTextbox(window, width=600, height=400)
    output_area.grid(row=3, column=0, padx=10, pady=10, columnspan=2,
sticky="nsew")

    run_tests_button = ctk.CTkButton(window, text="Run All Tests",
command=run_all_tests, width=200, height=40)
    view_report_button = ctk.CTkButton(window, text="View Report",
command=lambda: view_report(r"allure-results-all",
f"{random.randint(40000,60000)}"), width=100, height=40)

    run_selenium_tests_button = ctk.CTkButton(window, text="Run Selenium Tests",
command=run_selenium_tests, width=200, height=40)
    view_selenium_report_button = ctk.CTkButton(window, text="View Selenium
Report", command=lambda: view_report(r"allure-results-selenium",
f"{random.randint(40000,60000)}"), width=100, height=40)
```

Продовження додатку Б

```
    run_playwright_tests_button = ctk.CTkButton(window, text="Run Playwright
Tests", command=run_playwright_tests, width=200, height=40)
    view_playwright_report_button = ctk.CTkButton(window, text="View Playwright
Report", command=lambda: view_report(r"allure-results-playwright",
f"{random.randint(40000,60000)}"), width=100, height=40)

    run_tests_button.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
    view_report_button.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

    run_selenium_tests_button.grid(row=1, column=0, padx=10, pady=10,
sticky="nsew")
    view_selenium_report_button.grid(row=1, column=1, padx=10, pady=10,
sticky="nsew")

    run_playwright_tests_button.grid(row=2, column=0, padx=10, pady=10,
sticky="nsew")
    view_playwright_report_button.grid(row=2, column=1, padx=10, pady=10,
sticky="nsew")

    window.grid_columnconfigure(0, weight=1)
    window.grid_rowconfigure(0, weight=1)
    window.grid_columnconfigure(1, weight=1)
    window.grid_rowconfigure(1, weight=1)

    window.mainloop()
```

base_page.py

```
import time

from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait


class BasePage:
    def __init__(self, browser):
```

Продовження додатку Б

```
        self.browser = browser
        self.base_url = r"https://rozetka.com.ua/"
        time.sleep(1)

    def open(self):
        self.browser.get(self.base_url)
        time.sleep(1)

    def search_for(self, prompt: str):
        search_field = WebDriverWait(self.browser, 10).until(
            EC.visibility_of_element_located((By.XPATH, "//input[@name='search']"))
        )

ActionChains(self.browser).move_to_element(search_field).click(search_field).send_
keys(prompt).perform()

        button = WebDriverWait(self.browser, 10).until(
            EC.element_to_be_clickable((By.XPATH, "//button[contains(@class, 'search-
form__submit')]"))
        )
        button.click()

    def check_header_is(self, text: str):
        assert WebDriverWait(self.browser, 10).until(
            EC.text_to_be_present_in_element((By.XPATH, "//h1[@class='catalog-
heading ng-star-inserted']"), text)
        )

    def open_catalog_window(self):
        katalog_button = WebDriverWait(self.browser, 10).until(
            EC.element_to_be_clickable(
                (By.XPATH,
                 "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/button")
            )
        )

ActionChains(self.browser).move_to_element(katalog_button).click(katalog_button).
perform()
```

Продовження додатку Б

```
        WebDriverWait(self.browser, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH,
                 "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/rz-fat-menu/div[2]/ul/li[1]/div")
            )
        )


    def open_apple_macbook_category(self):
        apple_category = WebDriverWait(self.browser, 10).until(
            EC.element_to_be_clickable(
                (By.XPATH,
                 "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/rz-fat-
menu/div[2]/ul/li[1]/div/div[2]/div[1]/div[1]/ul[1]/li/ul/li[6]/a")
            )
        )
        apple_category.click()

    def open_notebooks_category(self):
        notebook_category = WebDriverWait(self.browser, 10).until(
            EC.element_to_be_clickable(
                (By.XPATH,
                 "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/rz-fat-
menu/div[2]/ul/li[1]/div/div[2]/div[1]/div[1]/ul[1]/li/a"
                )
            )
        )
        notebook_category.click()

    def filter_by_hp(self):
        hp_filter = WebDriverWait(self.browser, 10).until(
            EC.element_to_be_clickable(
                (By.XPATH,
                 "/html/body/rz-app-root/div/div/rz-category/div/main/rz-
catalog/div/div/aside/div/rz-filter-stack/div[2]/div/rz-scrollbar/div[1]/div/div/rz-filter-
```

Продовження додатку Б

```
section-autocomplete/ul/li[6]/rz-indexed-link")
            )
        )
        hp_filter.click()

    def open_login_window(self):
        reg_button = self.browser.find_element(By.XPATH,
                                    "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/ul/li[3]/rz-auth-icon/button")
        reg_button.click()

    def click_login_with_email(self):
        google_by_email = self.browser.find_element(By.XPATH,
                                      "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-
modal-layout/div[2]/rz-auth-phone-enter/rz-link-button/button")
        time.sleep(1)
        google_by_email.click()

    def register_with_email(self, email, password):
        self.open_login_window()
        self.click_login_with_email()
        enter_email = self.browser.find_element(By.XPATH,
                                     "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-
modal-layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-email-
input/div/input")
        enter_email.send_keys(email)
        enter_password = self.browser.find_element(By.XPATH,
                                      "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-
modal-layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-
password-input/div/div/input")
        enter_password.send_keys(password)
        login = self.browser.find_element(By.XPATH,
                                "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-
layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-submit-
button/button")
        login.click()
```

Продовження додатку Б

product_page.py

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

from pages.base_page import BasePage


class ProductPage(BasePage):
    def add_product_to_shopping_cart(self):
        buy_notebook = self.browser.find_element(
            By.XPATH,
            "/html/body/rz-app-root/div/div/rz-product/div/rz-product-tab-
main/div[1]/div[1]/div[2]/div/rz-product-main-info/div/div[2]/div/div[3]/rz-product-
buy-btn/rz-buy-button/button"
        )
        buy_notebook.click()

    def check_product_added_to_cart(self):
        basket_text = WebDriverWait(self.browser, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH, "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-
layout/div[1]/h2"))
        ).text
        assert "Кошик" == basket_text
```

products_page.py

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

from pages.base_page import BasePage


class ProductsPage(BasePage):
    def open_first_product_page(self):
        select_to_buy = self.browser.find_element(By.XPATH,
                                                  "/html/body/rz-app-root/div/div/rz-category/div/main/rz-
catalog/div/div/section/rz-grid/ul/li[1]/rz-catalog-tile/app-goods-tile-
```

Продовження додатку Б

```
default/div/div[2]/div[1]/rz-button-product-page[2]/a/span")
        select_to_buy.click()
```

test_change_location.py (Playwright)

```
import time

import pytest
from playwright.sync_api import sync_playwright

class TestPlaywrightChangeLocation:
    def test_change_shop_location(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()

            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")

            time.sleep(1)

            change_location_button_xpath = "/html/body/rz-app-root/div/div/rz-main-
page/div/aside/rz-main-page-sidebar/rz-menu-shops/a"
            page.locator(f'xpath={change_location_button_xpath}').click()
            time.sleep(1)

            choose_location_button_xpath = "/html/body/rz-app-root/div/div/rz-basic-
pages-layout/div/div/main/rz-retail/div/rz-tag-list/div/div/ul/li[4]/a"
            page.locator(f'xpath={choose_location_button_xpath}').click()
            time.sleep(1)

            check_locator = page.locator('xpath=/html/body/rz-app-root/div/div/rz-basic-
pages-layout/div/div/main/rz-retail/div/h1')
            check = check_locator.text_content()
            assert check == "Магазини ROZETKA у місті Запоріжжя"

    def test_change_city(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
```

Продовження додатку Б

```
        page = browser.new_page()

        page.set_viewport_size({"width": 1920, "height": 1080})
        page.goto("https://rozetka.com.ua/")

        time.sleep(1)

        city_change_button_xpath = "/html/body/rz-app-root/div/div/rz-main-
page/div/aside/rz-main-page-sidebar/rz-menu-location/button"
        page.locator(f'xpath={city_change_button_xpath}').click()
        time.sleep(1)

        number_click_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-
modal-layout/div[2]/rz-location-popup/rz-location-popup-content/ul/li[6]/a"
        page.locator(f'xpath={number_click_xpath}').click()
        time.sleep(1)

        search_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-
layout/div[2]/rz-location-popup/rz-location-popup-content/fieldset/div/button"
        page.locator(f'xpath={search_xpath}').click()
        time.sleep(1)

        check_locator = page.locator(
            'xpath=/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-main-
page-sidebar/rz-menu-location/button/div/span[1]')
        check = check_locator.text_content().strip()
        assert check == "Львів"
```

test_check_page.py (Playwright)

```
import time

import pytest
from playwright.sync_api import sync_playwright


class TestPlaywrightCheckPage:
    def test_sales_page(self):
```

Продовження додатку Б

```
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()

            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")

            time.sleep(1)

            sales_button_xpath = "/html/body/rz-app-root/div/div/rz-main-
page/div/main/rz-main-page-content/rz-top-slider/div/a"
            page.locator(f'xpath={sales_button_xpath}').click()

            time.sleep(1)

            check_locator = page.locator(
                'xpath=/html/body/rz-app-root/div/div/rz-promotions/div/main/div/h1')
            check = check_locator.text_content()
            assert check == "Акції"


            browser.close()

    def test_centre_payment(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()

            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")

            time.sleep(1)

            centre_button_xpath = "/html/body/rz-app-root/div/div/rz-main-
page/div/aside/rz-main-page-sidebar/rz-menu-help-center/a"
            page.locator(f'xpath={centre_button_xpath}').click()
            time.sleep(1)

            payment_click_xpath =
```

Продовження додатку Б

```
"//*[@id='__nuxt']/div/div[1]/div/div[3]/div/div/div[1]/div/div[2]/a"
        page.locator(f'xpath={payment_click_xpath}').click()
        time.sleep(1)

        check =
page.text_content("//*[@id='__nuxt']/div/div[1]/div/div/div/div[1]/div/div[2]/h1")

        assert check == "Оплата"
```

test_register.py (Playwright)

```
import time

import pytest
from playwright.sync_api import sync_playwright


class TestPlaywrightRegister:
    def test_register(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()

            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")

            time.sleep(1)

            reg_button_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/ul/li[3]/rz-auth-icon/button"
            page.locator(f'xpath={reg_button_xpath}').click()
            time.sleep(1)

            google_by_email_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-
modal/rz-modal-layout/div[2]/rz-auth-phone-enter/rz-link-button/button"
            page.locator(f'xpath={google_by_email_xpath}').click()
            time.sleep(1)

            enter_email_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-
```

Продовження додатку Б

```
modal-layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-email-input/div/input"
        page.locator(f'xpath={enter_email_xpath}').fill("testdata@gmail.com")
        time.sleep(1)

        enter_password_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-password-input/div/div/input"
        page.locator(f'xpath={enter_password_xpath}').fill("testdata")
        time.sleep(1)

        login_xpath = "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-layout/div[2]/rz-auth-email-enter/rz-auth-wrapper/div/div[1]/form/rz-submit-button/button"
        page.locator(f'xpath={login_xpath}').click()
        time.sleep(1)
```

test_search.py (Playwright)

```
import time

import pytest
from playwright.sync_api import sync_playwright


class TestSearchPlaywright:
    def test_search(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()
            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")
            time.sleep(1)

            page.fill("input[name='search']", "Мобільні телефони")

            page.click("button.search-form__submit")

            page.wait_for_selector("h1.catalog-heading")
```

Продовження додатку Б

```
        check = page.locator("h1.catalog-heading").text_content()

        static_variable = "Мобільні телефони"
        assert check == static_variable

    def test_category_search(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()

            page.set_viewport_size({"width": 1920, "height": 1080})
            page.goto("https://rozetka.com.ua/")

            time.sleep(1)

            button0_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/button"
            page.locator(f'xpath={button0_xpath}').click()

            button_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/rz-fat-menu/div[2]/ul/li[1]/a"
            page.locator(f'xpath={button_xpath}').click()

            button2_xpath = "/html/body/rz-app-root/div/div/rz-super-
portal/div/main/section/div[2]/rz-dynamic-widgets/rz-widget-
list[1]/section/ul/li[1]/rz-list-tile/div/a[1]/img"
            page.locator(f'xpath={button2_xpath}').click()

            button3_xpath = "/html/body/rz-app-root/div/div/rz-category/div/main/rz-
catalog/div/div/aside/div/rz-filter-stack/div[2]/div/rz-scrollbar/div[1]/div/div/rz-filter-
section-autocomplete/ul/li[2]/rz-indexed-link/a"
            page.locator(f'xpath={button3_xpath}').click()

            element_text = page.text_content('h1')
            assert "Apple MacBook" == element_text

    def test_filter(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
```

Продовження додатку Б

```
        page = browser.new_page()

        page.set_viewport_size({"width": 1920, "height": 1080})
        page.goto("https://rozetka.com.ua/")

        time.sleep(1)

        katalog_button_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/button"
        page.locator(f'xpath={katalog_button_xpath}').click()

        notebook_category_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-
main-header/header/div/div/rz-fat-menu-header-btn/rz-fat-
menu/div[2]/ul/li[1]/div/div[2]/div[1]/div[1]/ul[1]/li/a"
        page.locator(f'xpath={notebook_category_xpath}').click()

        hp_filter_xpath = "/html/body/rz-app-root/div/div/rz-category/div/main/rz-
catalog/div/div/aside/div/rz-filter-stack/div[2]/div/rz-scrollbar/div[1]/div/div/rz-filter-
section-autocomplete/ul/li[6]/rz-indexed-link"
        page.locator(f'xpath={hp_filter_xpath}').click()

        time.sleep(1)
        check = page.text_content('h1')

        assert check == "Ноутбуки HP"
```

test_shopping_cart.py (Playwright)

```
import time

import pytest
from playwright.sync_api import sync_playwright


class TestShoppingCartPlaywright:
    def test_add_to_shopping_cart(self):
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=False)
            page = browser.new_page()
```

Продовження додатку Б

```
        page.set_viewport_size({"width": 1920, "height": 1080})
        page.goto("https://rozetka.com.ua/")

        time.sleep(1)

        katalog_button_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/button"
        page.locator(f'xpath={katalog_button_xpath}').click()
        time.sleep(1)

        notebook_click_xpath = "/html/body/rz-app-root/div/div/rz-header/rz-main-
header/header/div/div/rz-fat-menu-header-btn/rz-fat-
menu/div[2]/ul/li[1]/div/div[2]/div[1]/div[1]/ul[1]/li/a"
        page.locator(f'xpath={notebook_click_xpath}').click()
        time.sleep(1)

        select_HP_xpath = "/html/body/rz-app-root/div/div/rz-category/div/main/rz-
catalog/div/div/aside/div/rz-filter-stack/div[2]/div/rz-scrollbar/div[1]/div/div/rz-filter-
section-autocomplete/ul/li[6]/rz-indexed-link"
        page.locator(f'xpath={select_HP_xpath}').click()
        time.sleep(2)

        select_to_buy_xpath = "/html/body/rz-app-root/div/div/rz-
category/div/main/rz-catalog/div/div/section/rz-grid/ul/li[1]/rz-catalog-tile/app-goods-
tile-default/div/div[2]/div[1]/rz-button-product-page[2]/a/span"
        page.locator(f'xpath={select_to_buy_xpath}').click()
        time.sleep(1)

        buy_notebook_xpath = "/html/body/rz-app-root/div/div/rz-product/div/rz-
product-tab-main/div[1]/div[1]/div[2]/div/rz-product-main-
info/div/div[2]/div/div[3]/rz-product-buy-btn/rz-buy-button/button"
        page.locator(f'xpath={buy_notebook_xpath}').click()
        time.sleep(1)

        check_basket = page.text_content('h2')
        assert check_basket == "Кошик"

    def test_tracking(self):
        with sync_playwright() as p:
```

Продовження додатку Б

```
        browser = p.chromium.launch(headless=False)
        page = browser.new_page()

        page.set_viewport_size({"width": 1920, "height": 1080})
        page.goto("https://rozetka.com.ua/")

        time.sleep(1)

        tracking_button_xpath = "/html/body/rz-app-root/div/div/rz-main-
page/div/aside/rz-main-page-sidebar/rz-menu-tracking/a"
        page.locator(f'xpath={tracking_button_xpath}').click()
        time.sleep(1)

        number_click_xpath = "//*[@id='searchText']"
        page.locator(f'xpath={number_click_xpath}').fill("123456789")
        time.sleep(1)

        search_xpath = "/html/body/rz-app-root/div/div/rz-tracking-page/rz-
tracking/div/div/form/fieldset/div/div/button"
        page.locator(f'xpath={search_xpath}').click()
        time.sleep(1)

        check_locator = page.locator(
            "xpath=/html/body/rz-app-root/div/div/rz-tracking-page/rz-
tracking/div/div/rz-tracking-info/div")
        check = check_locator.text_content().strip()
        assert check == "За номером 123456789 посилку не знайдено. Можливо,
вона ще не передана для відправлення або номер некоректний."
```

test_change_location.py (Selenium)

```
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
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
import time
```

Продовження додатку Б

```
class TestSeleniumChangeCity:
    def test_change_shop_location(self):
        chrome_options = Options()
        chrome_options.add_argument("--start-maximized")
        browser = webdriver.Chrome(options=chrome_options)

        browser.set_window_size(1920, 1080)
        browser.get("https://rozetka.com.ua/")
        time.sleep(0.5)

        change_location_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,
                                        "/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-
main-page-sidebar/rz-menu-shops/a"))
        )
        change_location_button.click()
        time.sleep(0.5)

        choose_location_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,
                                        "/html/body/rz-app-root/div/div/rz-basic-pages-
layout/div/div/main/rz-retail/div/rz-tag-list/div/div/ul/li[4]/a"))
        )
        choose_location_button.click()
        time.sleep(0.5)

        check = WebDriverWait(browser, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH, '/html/body/rz-app-root/div/div/rz-basic-pages-
layout/div/div/main/rz-retail/div/h1'))
        ).text.strip()

        time.sleep(0.5)

        assert check == "Магазини ROZETKA у місті Запоріжжя"

        browser.close()

    def test_change_city(self):
```

Продовження додатку Б

```
    chrome_options = Options()
    chrome_options.add_argument("--start-maximized")
    browser = webdriver.Chrome(options=chrome_options)

    browser.set_window_size(1920, 1080)
    browser.get("https://rozetka.com.ua/")
    time.sleep(0.5)

    city_change_button = WebDriverWait(browser, 10).until(
        EC.element_to_be_clickable((By.XPATH,
                                    "/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-
main-page-sidebar/rz-menu-location/button"))
    )
    city_change_button.click()
    time.sleep(0.5)


    number_click = WebDriverWait(browser, 10).until(
        EC.element_to_be_clickable((By.XPATH,
                                    "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-
layout/div[2]/rz-location-popup/rz-location-popup-content/ul/li[6]/a"))
    )
    number_click.click()

    time.sleep(0.5)

    search_button = WebDriverWait(browser, 10).until(
        EC.element_to_be_clickable((By.XPATH,
                                    "/html/body/rz-app-root/rz-modal-outlet/rz-modal/rz-modal-
layout/div[2]/rz-location-popup/rz-location-popup-content/fieldset/div/button"))
    )
    search_button.click()

    time.sleep(0.5)

    check = WebDriverWait(browser, 10).until(
        EC.visibility_of_element_located(
            (By.XPATH, '/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-
main-page-sidebar/rz-menu-location/button/div/span[1]'))
```

Продовження додатку Б

```
    ).text.strip()
    time.sleep(0.5)

    assert check == "Львів"

    browser.close()
```

test_change_location.py (Selenium)

```
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
from selenium.webdriver.chrome.options import Options
import time


class TestSeleniumSalesPage:
  def test_sales_page(self):

    chrome_options = Options()
    chrome_options.add_argument("--start-maximized")
    browser = webdriver.Chrome(options=chrome_options)
    browser.set_window_size(1920, 1080)
    browser.get("https://rozetka.com.ua/")
    time.sleep(0.5)

    sales_button = WebDriverWait(browser, 10).until(
      EC.element_to_be_clickable((By.XPATH,
                    "/html/body/rz-app-root/div/div/rz-main-page/div/main/rz-
main-page-content/rz-top-slider/div/a"))
    )
    sales_button.click()
    time.sleep(0.5)

    check = WebDriverWait(browser, 10).until(
      EC.visibility_of_element_located(
        (By.XPATH, '/html/body/rz-app-root/div/div/rz-
promotions/div/main/div/h1'))
```

Продовження додатку Б

```
        ).text.strip()

        assert check == "Акції"

        browser.close()

    def test_centre_payment(self):

        chrome_options = Options()
        chrome_options.add_argument("--start-maximized")
        browser = webdriver.Chrome(options=chrome_options)

        browser.set_window_size(1920, 1080)
        browser.get("https://rozetka.com.ua/")

        centre_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,
                                        "/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-
main-page-sidebar/rz-menu-help-center/a"))
        )
        centre_button.click()


        payment_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,

"//*[@id='__nuxt']/div/div[1]/div/div[3]/div/div/div[1]/div/div[2]/a"))
        )
        payment_button.click()


        check = WebDriverWait(browser, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH,
"//*[@id='__nuxt']/div/div[1]/div/div/div/div[1]/div/div[2]/h1"))
        ).text

        assert check == "Оплата"
```

Продовження додатку Б

```
    browser.close()
```

test_register.py (Selenium)

```
from pages.home_page import HomePage


class TestRegister:
    def test_register(self, browser):
        home_page = HomePage(browser)
        home_page.open()
        home_page.register_with_email("testdata@gmail.com", "testdata")
```

test_search.py (Selenium)

```
import pytest

from pages.home_page import HomePage
from tests.conftest import browser


class TestSearchSelenium:
    def test_search(
        self,
        browser,
        tested_category="Мобільні телефони"
    ):
        home_page = HomePage(browser)
        home_page.open()
        home_page.search_for(tested_category)
        home_page.check_header_is(tested_category)

    def test_category_search(
        self,
        browser
    ):
        home_page = HomePage(browser)
        home_page.open()
        home_page.open_catalog_window()
```

Продовження додатку Б

```
    def test_filter(
        self,
        browser
    ):
        home_page = HomePage(browser)
        home_page.open()
        home_page.open_catalog_window()
        home_page.open_notebooks_category()
        home_page.filter_by_hp()
        home_page.check_header_is("Ноутбуки HP")
```

test_shopping_cart.py (Selenium)

```
import pytest
from pages.home_page import HomePage
from pages.product_page import ProductPage
from pages.products_page import ProductsPage
from tests.conftest import browser
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
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
import time


class TestShoppingCart:
    def test_add_to_shopping_cart(
        self,
        browser
    ):
        home_page = HomePage(browser)
        home_page.open()
        home_page.open_catalog_window()
        home_page.open_notebooks_category()
        home_page.filter_by_hp()
        products_page = ProductsPage(browser)
```

Продовження додатку Б

```
        products_page.open_first_product_page()
        product_page = ProductPage(browser)
        product_page.add_product_to_shopping_cart()
        product_page.check_product_added_to_cart()

    def test_tracking(self):
        chrome_options = Options()
        chrome_options.add_argument("--start-maximized")
        browser = webdriver.Chrome(options=chrome_options)

        browser.set_window_size(1920, 1080)
        browser.get("https://rozetka.com.ua/")
        time.sleep(1)
        tracking_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,
                                        "/html/body/rz-app-root/div/div/rz-main-page/div/aside/rz-
main-page-sidebar/rz-menu-tracking/a"))
        )
        tracking_button.click()
        time.sleep(0.5)

        number_input = WebDriverWait(browser, 10).until(
            EC.visibility_of_element_located((By.XPATH, "//*[@id='searchText']"))
        )
        number_input.send_keys("123456789")
        time.sleep(0.5)

        search_button = WebDriverWait(browser, 10).until(
            EC.element_to_be_clickable((By.XPATH,
                                        "/html/body/rz-app-root/div/div/rz-tracking-page/rz-
tracking/div/div/form/fieldset/div/div/button"))
        )
        search_button.click()
        time.sleep(0.5)

        check = WebDriverWait(browser, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH, "/html/body/rz-app-root/div/div/rz-tracking-page/rz-
tracking/div/div/rz-tracking-info/div"))
```

Продовження додатку Б

```
        ).text.strip()

        assert check == "За номером 123456789 посилку не знайдено. Можливо,
вона ще не передана для відправлення або номер некоректний."
        browser.close()
```

conftest.py

```
import pytest
from selenium import webdriver


@pytest.fixture(scope="function")
def browser():
    # Set preferences
    options = webdriver.ChromeOptions()
    options.add_experimental_option("excludeSwitches", ["enable-logging"])
    browser = webdriver.Chrome(options=options)
    browser.implicitly_wait(3)
    browser.set_window_size(1920, 1080)

    # Yield browser with set preferences
    yield browser

    # Close browser after the test
    browser.quit()

def pytest_sessionstart(session):
    session.results = dict()


@pytest.hookimpl(tryfirst=True, hookwrapper=True)
def pytest_runtest_makereport(item, call):
    outcome = yield
    result = outcome.get_result()

    if result.when == 'call':
        item.session.results[item] = result

def pytest_sessionfinish(session, exitstatus):
```

Продовження додатку Б

```
print()
print('run status code:', exitstatus)
print(session.results)
for el in session.results.values():
    print(el)
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)