Державний торговельно – економічний університет
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка фреймворку автоматизації тестування Web-сервісів на базі Python»

Студента 4 курсу, 9 групи
спеціальності
122 «Комп'ютерні науки»

Шапрана
Олексія
Юрійовича

_____
підпис студента

Науковий керівник
кандидат фізико -
математичних наук

_____
підпис керівника

Філімонова
Тетяна
Олегівна

Гарант освітньої програми
кандидат технічних наук, доцент

_____
підпис керівника

Демідов Павло
Георгійович

Київ 2022

Анотація

У кваліфікаційній роботі розглянута проблема автоматизації тестування web-додатків. Ця проблема є дуже актуальною через постійний розвиток web-технологій та збільшення складності й дороговизни розробки web-додатків та великої вартості часу спеціалістів. В кваліфікаційній роботі були теоретично проаналізовані основні підходи до тестування та розробки програмного забезпечення, а також можливості їх автоматизації. Були визначені поняття «web-додаток» та «web-сайт». Проаналізовані найбільш популярні фреймворки тестування. В результаті роботи був практично створений фреймворк автоматизації тестування web-додатків на базі мови програмування Python з використанням драйверу браузера Selemium. Внаслідок проведеного дослідження фреймворк був побудований з врахуванням особливостей web-розробки на принципах легкості розуміння, розширення, простоти, та сумісності.

Ключові слова: Фреймворк, автоматизація розробки, web-технології, програмування, розробка програмного забезпечення, web-додаток, тестування програмного забезпечення, Python, Selenium.

Abstract

The problem of automation of web-applications testing has been considered in the qualification work. This problem is very relevant due to the quick development of web-technologies and increasing complexity and cost of web-applications developing, as well the high cost of time of such specialists. The main approaches to software testing and development, as well as the possibilities of their automation were theoretically analyzed in the qualification work. The terms "web application" and "web site" were defined. The most popular testing frameworks have been analyzed. As a result of the theoretical work, a framework for testing of automatization of web-applications based on the Python programming language using the Selenium browser driver has been practically created. As a result of the study, the framework was built taking into account the peculiarities of web-development, using principles of ease of understanding, extension, simplicity, and compatibility.

Keywords: Framework, development automation, web-technologies, programming, software development, web-application, software testing, Python, Selenium.

ЗМІСТ

ВСТУП

При створенні сучасних додатків, програм або веб-сервісів перед розробниками стоїть проблема якості їх продукту. З метою перевірки кінцевого продукту на якість, що задовольняє замовників і користувачів, застосовується тестування. З розвитком технологій, ця задача стає все більш складною та рутинною, тому актуальності набуває автоматизоване тестування.

Як зазначає О.Бородіна, фреймворк автоматизованого тестування – набір припущень, концепцій та інструментів, які забезпечують підтримку для автоматизованого тестування програмного забезпечення. Основною перевагою такої структури, на думку автора, є низька вартість обслуговування. Якщо є зміни в будь-якому тест-кейсі, то необхідно оновити тільки тестовий файл тест-кейса, а сценарій драйверів й сценарій запуску залишиться у старому файлі. В ідеалі, немає необхідності оновлювати скрипти в разі внесення змін у додаток [2, с.345].

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність роботи сторінки. Воно тільки порівнює стан і поведінку продукту зі специфікацією. Зазвичай, поняття якості обмежується такими поняттями як коректність, надійність, практичність, безпечність, але може містити більше технічних вимог, котрі описані у стандарті ISO 9126. Склад та зміст супутньої документації процесу тестування визначається стандартом IEEE 829-1998 Standard for Software Test Documentation. [9].

Актуальність роботи. Розробка фреймворку автоматизації тестування допоможе підвищити якість та конкурентоспроможність web-сервісів.

Предмет дослідження: фреймворк автоматизації тестування web-сервісів.

Об’єкт дослідження: процес розробки фреймворку автоматизації тестування Web-сервісів.

Мета роботи: розробка і практична реалізація фреймворку автоматизації тестування web-сервісів на базі Python.

Для досягнення поставленої мети необхідно виконати наступні **завдання**.

1. Аналіз сучасних підходів до автоматизації тестування.
2. Порівняння існуючих фреймворків для тестування.
3. Розробка фреймворку для автоматизації тестування.

Для вирішення поставлених задач використовувалися такі **методи**:

- загальнонауковий аналітичний метод;
- метод порівняльного аналізу для порівняння фреймворків для тестування;
- методи об’єктно-орієнтованого програмування для розробки фреймворку автоматизації тестування.

Практичне значення. Результати, отримані роботі, а саме, розроблений фрейморк, може бути використаний для автоматизації тестування web-сервісів.

Публікації. Результати досліджень опубліковано на IX Міжнародній науково-технічній конференції «Інформатика, управління та штучний інтелект (ІУШІ-2022), Національний технічний університет «Харківський політехнічний інститут», 11-13.05.2022»; на науково-технічній конференції Вінницького національного технічного університету, НТКП ВНТУ, 31.05.2022 [15, 16].

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ WEB-СЕРВІСІВ НА БАЗІ PYTHON

1.1 Теоретичний аналіз сучасних підходів до автоматизації тестування

Проблема автоматизованого тестування програмного забезпечення (Software Automation Testing) як процесу його верифікації, при якому основні функції та кроки тесту виконуються автоматично за допомогою інструментів для автоматизованого тестування є актуальною у зв'язку з збільшенням комплексності сучасного програмного забезпечення.

Технічний фахівець з автоматизованого тестування програмного забезпечення (Software Automation Tester) забезпечує створення, налагодження і підтримку працездатного стану тест скриптів, тестових наборів та інструментів для автоматизованого тестування.

Існують два основні підходи до автоматизації тестування: на рівні коду – зокрема метод тестування програмного забезпечення, що полягає в окремому тестуванні кожного модуля коду програми; GUI-тестування – імітація дій користувача за допомогою спеціальних тестових фреймворків [2, с.345].

Найпоширенішою формою автоматизації є тестування додатків через графічний інтерфейс користувача. Популярність такого виду тестування пояснюється рядом факторів, а саме: додаток тестується тим же способом, що буде використаний майбутніми користувачами; не потрібен доступ до вихідного коду для здійснення тестування [3].

В цій класифікації можливо прослідкувати конкретизоване використання загально прийнятих підходів до автоматизації будь-яких додатків. Ці ж самі підходи чудово себе показують і в тестування веб

додатків. Наведемо перелік найбільш сучасних та відомих підходів до автоматизації тестування, а саме:

White-box testing (або тестування білого ящика) – це такий підхід до тестування програмного забезпечення (ПЗ), що передбачає знання тестувальником внутрішньої будови програми. Це дозволяє йому створювати тести, що спеціально випробовують програму в, на його думку, слабких місцях. Це саме такий підхід, що Бородіна назвала «На рівні коду».

Black-box testing (або тестування чорного ящика) – це такий підхід до тестування ПЗ що передбачає те, що тестувальник підходить до тестування ПЗ як звичайний користувач, що не має уяви про внутрішню її побудову. Такий спосіб є набагато швидшим, оскільки тестувальнику не потрібно витрачати час на те, що зрозуміти програму з середини або брати участь в її розробці. Це саме такий підхід, що О.Бородіна назвала «GUI тестування», адже при такому підходу тестувальник використовує користувацький інтерфейс в доволі креативному стилі, для того щоб виявити помилки або непередбачені ситуації. Так, як зауважують С.Крепич та І.Співак, при тестуванні чорної скриньки, тестувальник має доступ до програмного забезпечення тільки через ті ж інтерфейси, що і замовник або користувач (*Front-End, Browser, Desktop form*), або через зовнішні інтерфейси (*Web services, спеціалізовані служби*), що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування [4, с.209].

Grey-box testing (або тестування сірого ящика) – це такий підхід до тестування ПЗ, що комбінує підходи White-box testing та Black-box testing. Тестувальник лише частково знає внутрішню будову програми.

Unit-testing (або інтеграційне тестування) – це такий підхід до тестування ПЗ, сенс якого полягає в тому щоб розділити програму на маленькі неподільні компоненти, кожен с яких потрібно помістити в

середовище, яке буде симулювати потік даних, що програма могла би давати цьому компоненту. Після відбувається вивчення поведінки цього компоненту та зіставлення із вимогами до його роботи.

Отже найбільш поширеними підходами до тестування ПЗ, в тому числі автоматизованого, є: *white-box testing, black-box testing, grey-box testing та unit-testing.*

1.2 Проектування й розробка Web-сервісів

Для створення фреймворку автоматизації дуже важливо знати, як саме створюються web-додатки або web-сайти. Можна вважати, що сайт і web-додаток – це тотожні поняття, адже за внутрішньою будовою майже кожен web-додаток або web-сайт створено за допомогою web-технологій, що реалізують систему клієнт-сервер. Це працює так, що запит користувача web-додаток обробляє та відправляє через мережу до web-сервера, де власне знаходиться сайт або додаток. У відповідь web-сервер створює із форми відповідну веб-сторінку і відправляє її назад web-додатку. Web-додаток у свою чергу використовує браузер як інтерфейс користувача, щоб відобразити відповідну сторінку або інформацію.

Існують *статичні та динамічні web-додатки*. В тестуванні, а особливо його автоматизації більш потребують динамічні web-додатки, адже вони є інтерактивними та мають відповідним чином відповідати на запити користувача, коли головне для статичних web-додатків просто відображати інформацію в середовищі браузера.

Для розробки динамічних web-додатків можна використовувати *різні мови програмування*. Наприклад, особливо важливим для динамічних сторінок, є мова *JavaScript*, створена з єдиною метою – надати статичним

сторінкам інтерактивності. Варто відзначити, що завдяки проекту *Node.js*, існує і активно використовується можливість написання обох частин web-програми (і серверної, і клієнтської) з використанням однієї мови програмування – *JavaScript* [6]. Але зазвичай розробка поділяється на інформаційно-функціональну (*Back-end*), та дизайн (або інтерфейс) того, що буде бачити та з чим взаємодіяти користувач (*Front-end*).

Інтерфейс web-сайту має бути зрозумілим і легким для використання користувачем, а найголовніше, необхідно, щоб він відповідав завданню, що стоїть перед web-сайтом.

Погоджуємося з думкою А. Бублик, що «естетичні ефекти відіграють вирішальну роль у створенні профілю відправника і, таким чином, іміджу організації чи особи, яка стоїть за web-сайтом. Тому важливо, щоб навіть web-сайти, які переважно мають інформаційне призначення, також естетично відображали організацію, яка стоїть за цим сайтом. Для багатьох web-сайтів в Інтернеті функціональність має найвищий пріоритет, а естетика включається лише для її підтримки. Зазвичай це стосується жанру інформаційних сайтів, де головна мета – отримати необхідну інформацію якомога швидше та ефективно, як, наприклад, www.google.com» [7, с.9].

Отже, проектування сайту або web-додатку розпочинається із дизайну, лише потім йде мова про створення коду для web-сервера. Для створення макету дизайну додатку використовують таке ПЗ як *Photoshop*, або більш призначене саме для створення веб застосунків *Figma*.

Для написання *Back-end* частини, окрім *JavaScript*, найчастіше використовують такі мови програмування:

– Сценарні: *Python, Ruby, PHP, TypeScript.*
– Функціональні мови: *Elixir, Scala, Clojure, Haskell, Erlang*.

– Мультипарадигмні мови: *Goland, Rust*.

– Корпоративні рішення: *Java, .NET*.

Front-end же частина реалізується із створеного заздалегідь макету за допомогою:

– *HTML* (HyperText Markup Language) – мова для розмітки документів, створення структури сторінки: заголовки, абзаци, списки тощо.

– *CSS* (Cascading Style Sheets) – мова для опису і стилізації зовнішнього вигляду документа. Завдяки CSS-коду браузер розуміє, яким чином відображати елементи. CSS задає кольори і параметри шрифтів, визначає, як будуть розташовуватися різні блоки сайту.

– *JavaScript*, що був вже згаданий вище.

Також для роботи web-додатку має бути реалізована *база даних* (БД). *База даних* являє собою програмне забезпечення на сервері, завданням якого є збереження і швидка видача даних після запиту користувача. Необхідність використання БД зумовлена природою web-додатків, адже, будучи викладеними в всесвітню павутину ними можуть користуватися одразу декілька користувачів. Сенсу від web-додатку, що може обслуговувати лише одного користувача одночасно, – немає, отже потрібна система, що буде зберігати інформацію багатьох користувачів одночасно, при цьому вона повинна швидко знаходити ці самі данні.

Розглянувши всі складові сайту, можливо отримати уяву та послідовність його створення.

Основними кроками у розробці web-сайту Т.Котенко, В. Жирова, А.Чубаєвський, О. Десятко вважають:

1. Збір та підготовка інформації для розробки проекту; виділення та опис усіх необхідних функціональних компонентів.

2. Встановлення студії та необхідних плагінів для зручної розробки проєкту.

3. Розподіл проекту на необхідні компоненти для зручної покомпонентної розробки (наприклад, розподіл на шапку веб-сайту з підкомпонентами, основну частину та підвал).

4. Покомпонентна розробка та тестування web-сайту.

5. Збір усіх компонентів web-сайту та тестування.

6. Розміщення web-сайту в мережі [8, с.12].

Погоджуємося з думкою вищеозначених авторів щодо основних кроків для створення web-додатків. Спочатку потрібно визначитися з метою сайта, створити макет, потім підготувати робоче середовище розробки для кожного із розробників та розподілити завдання, потім приступити до втілення дизайну сайту, його логіки, реалізації бази даних, після чого потрібно перевірити всі компоненті та зібрати їх в єдине ціле. Нарешті, коли команда буде впевнена в якості продукту та добре проітерує кроки розробки, цей продукт викладуть на хостинг у всесвітню павутину.

1.3 Огляд існуючих фреймворків автоматизації тестування Web-сервісів

Розглянемо вже існуючі рішення до проблеми автоматизації тестування web-додатків через фреймворки. Дослідження ринку таких фреймворків показало, що існує декілька підходів та методологій до тестування, як і велика кількість інструментів, що займаются тестуванням конкретної частини або складової web-додатку. Однак найбільш відомим таким фреймворком є Selenium WebDriver. І.Богач та А. Кравець детально розглянули можливості цього фреймворку: «Selenium WebDriver (Selenium

2.0) – це програмна бібліотека для управління браузерами. Часто вживається також більш коротка назва WebDriver. Так як Selenium WebDriver драйвер браузера, то це програмна бібліотека, яка не має користувальницького інтерфейсу та дозволяє різним іншим програмам взаємодіяти з браузером, керувати його поведінкою, отримувати від браузера певні дані та змушувати його виконувати певні команди» [9, с. 17]. Отже, основною функцією WebDriver є надання автотестам доступу до браузера.

Ці ж автори акцентують увагу на особливостях роботи з цим фреймворком. «Запуск тестів за розкладом і публікацію звітів виконує сервер безперервної інтеграції – Jenkins, CruiseControl, Bamboo, TeamCity. Що обумовлює універсальність використання WebDriver для розробників та користувачів фреймворком. Можливості роботи WebDriver:

– пошук елементів в основному дереві сторінки, отримання їх атрибутів та стилів;

– взаємодія з елементами: фокусування, клік, натиснення на клавіші;

– використання різних типів очікування;

– виконування дій курсором миші (при цьому реальний курсор не використовується);

– перехід за посиланнями, перемикання між вікнами браузера, робота з cookies;

– можливість збереження скріншотів;

– завантажування та скачування файлів;

– виконання скриптів на сторінці (JavascriptExecutor);

– перемикання між вкладеними фреймами» [9, с. 18]

Для більш простого тестування PhP частини web-сайту дуже часто використовують *PHPUnit*. Цей фреймворк дозволяє створювати майже

необмежену кількість різноманітних тестів і надає можливість запускати перед внесенням змін до проекту будь-яку кількість тестів. Дуже корисною особливістю, що виділяє цей фреймворк, є його можливість проводити окремий аналіз покриття коду, який ще називають *code coverage або tests coverage*. Це така міра, що визначається відсотком тестованого вихідного коду програми. Вона дозволяє розуміти, яка частина (відсоток) web-додатку перевіряється – чим вищий відсоток покриття, тим більш можливо бути впевненим, що виникне менше помилок при внесенні змін, так як можливо дослідити та перевірити більш комплексні взаємодії, а, отже, виявити непередбачувані помилки і порушення логіки роботи частин проекту. Як заявляють Г. Кодола, Н. Волинець та І. Сербулова за допомогою цього фреймворку і набору бібліотек до нього можливо повноцінно провести функціональне, модульне тестування і тести бази даних [10].

Слід зазначити, що вищенаведені фреймворки базуються на *Test Driven Development* (TDD) підході, що розуміє під собою втілення тестів для щойно написаного коду в процесі розробки web-додатків. Однак, це не є єдиним підходом, тому популярності набуває інший – *Behaviour Driven Development* (BDD) підхід, сутність якого полягає в тестуванні всього продукту та клопітливій перевірці його на задоволення потреб замовника та його специфікаціям.

Хоча підхід TDD із попереднім тестуванням працює у багатьох розробників, він підходить не для усіх. На кожного розробника програм, що з успіхом застосовує TDD, як зазначають Т. Шатовська та І. Каменєва, знайдеться декілька розробників, які активно заперечують цей підхід. Незважаючи на численність інфраструктур тестування, таких як TestNG, Selenium і FEST, все одно знаходиться багато причин не виконувати

тестування коду. Дві звичайні причини відмови від TDD – «у нас недостатньо часу для тестування» і «код занадто складний і важко перевіряється» [11, с. 73-74].

Яскравим прикладом такого фрейморку, що концентрується на BDD є *Behat*. Цей фреймворк дозволяє писати дуже зрозумілий для людини код, що описує як саме має поводити web-додадок.

Таким чином, нами розглянуто найпопулярніші фреймворки автоматизованого тестування: багатоцільовий (Selenium WebDriver), для тестування PhP (PHPUnit) та базований на тестуванні поведінки сайту (Behat).

1.4. Особливості програмування на базі мови Python

Існує багато різних мов програмування, як і вузького напряму, так і широкого. Найбільш *відомими мовами програмування* широкого напряму є такі мови як: *Java, C++, C#, Go, Ruby, JavaScript*, та інші. Однак найбільш популярною серед них за індексом TIOBE, що є індикатором популярності та мови програмування є Python [13]. Індекс TIOBE хоч і не показує справжню долю використання мови, але він дозволяє побачити зацікавленість та популярність мови. Python набрав 15,33% від всіх пошукових запитів щодо мов програмування за результатами індексу в лютому 2021 року. Охарактеризуємо особливості, що призвели то такої шаленої відомості мови Python.

А. Костюченко в своєму навчальному посібнику визначає такі *особливості мови програмування Python* [12, с.180]:

–Проста і мінімалістична мова.

–Мова легка в освоєнні та простота для вивчення.

–Вільна і відкрита мова.

–Високорівнева мова.

–Мова загального призначення.

–Кросплатформенна мова.

–Інтерпретована мова.

–Мультипарадигменна мова.

–Мова з «широкою» стандартною бібліотекою.

–Вбудовувана мова.

–Розширювана мова.

Це є дуже вичерпним списком формулюванням особливостей мов програмування, що є загальноприйнятим в сфері програмування. Слід зазначити що багато з цих особливостей мови також присутні і в інших популярних мовах. Наддамо авторського означення найбільш важливих особливостей саме цієї мови.

Одним із обов'язкових означень будь якої мови програмування є її «Рівневість». Сенс мови програмування полягає в тому, щоб передавати команди людини комп'ютеру так, щоб це зміг зрозуміти комп'ютер. Однією з перших мов програмування є мова Асемблер, що складається із команд, що комп'ютер здатен зрозуміти напряму. Але є одна проблема – те, що зрозуміло комп'ютеру, часто є зовсім незрозумілим людині. Отже, й написання команд стає дуже важкою задачею.

Мови програмування дозволяють писати команди так, як це більш зрозуміло людині, при цьому переводити їх у зрозумілу для комп'ютера форму. Однак, програміст втрачає частину контролю над комп'ютером, адже зручна для людей форма – є більш абстрактною. Чим вища рівневість мови – тим більш зрозумілим є написаний код для людини, що дуже важливо та дозволяє швидше створювати додатки і краще працювати в

команді. Python – є *високорівневою* мовою. Вона – проста та не містить важких або заплутаних елементів. Можливо зрозуміти, що свою популярність Python набуває через простоту розуміння та гнучкість. Це підтверджується дуже широкою стандартною бібліотекою, що містить велику кількість вже написаних команд як загального, так і більш вузького напрямку, є оптимізованими та швидкими у виконанні. Замість того, щоб шукати чужі бібліотеки або написання вже давно розроблених та поширених функцій, програміст просто звертається до стандартної бібліотеки мови. Також мова є *мультипарадигмальною* – це означає, що вона може реалізовувати різні підходи до написання програм, як-от функціональне та об'єктно-орієнтоване програмування, без необхідності у вивченні нової мови або недостатніх можливостей для реалізації нового підходу.

Мова є *інтерпретованою*, це означає що код, написаний на Python, не переводиться одразу в машинний код, цим займається спеціальний інтерпретатор мови. Це дозволяє мові бути кроссплатформеною, адже для того, щоб адаптувати програму так, щоб вона працювала на іншій архітектурі, достатньо просто змінити інтерпретатор без необхідності модифікацій джерельного коду. Це дуже корисно для, наприклад, веб-розробки, адже сайти мають працювати на будь-якій популярній платформі, що підтримує браузери, а отже й перевірити веб застосунок потрібно на всіх платформах. Python є також *вільною* мовою – отже його використання не накладає обмежень на програміста, він має право користуватися мовою в будь-яких цілях.

Висновки

Автоматизація розробки є невід'ємною частиною розробки web-додатків та сайтів. Гнучкість технологій розробки дозволяє створити web-додаток для будь-якої сфери діяльності. Web-розробка стає все більш актуальною з розвитком інтернету речей, складність розробки web-сайтів зростає, як і необхідність в швидкій можливості їх тестування. Для реалізації такої цілі зручно використовувати фреймворки, що будуть містити вже написаний код для запуска тестів. Існує декілька підходів до тестування, як-от TDD та BDD підходи. Для реалізації такого фреймворку зручно використовувати мову програмування Python через її простоту, швидкість, та кросплатформеність.

РОЗДІЛ 2. РОЗРОБКА ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ WEB-СЕРВІСІВ НА БАЗІ PYTHON

2.1 Доцільність та актуальність розробки фреймворку автоматизації тестування

Як вже було визначено раніше, тестування грає дуже важливу роль в розробці web-сервісів. Автоматизація тестування може допомогти прискорити та здешевити розробку. Однак більшість веб сайтів мають свою власну, особливу структуру, що заважає створенню єдиного автоматизованого способу тестування. Через це доцільно розробляти тести за допомогою вже означених інструментів під конкретний сайт. На жаль такий підхід не є дуже швидким, оскільки кожному розробнику потрібно розробляти тести майже з нуля. Але можливо розділити тести на категорії за призначенням, і вже для кожної категорії створити вже готові тести що будуть задовольняти більшість класичних потреб. Якщо розробнику залишити можливість доповнювати вже заготовлені тести та дозволити їх конфігурацію то вийде скоротити час на розробку та надати розробнику підґрунтя для подальшого розвитку тестів. Для цього логічним було би створення фреймворку автоматизації тестування web-сервісів.

Вже існує велика кількість інструментів, що полегшує написання тестів, але майже відсутні фреймворки з наперед написаними тестами. Такого роду фреймворки могли б стати у нагоді розробникам що створюють сайт невеликою командою, або самотужки, або бути використаними для сайтів що були розробленими в конструкторі сайтів, що набирають шаленою популярності на сьогоднішній день.

2.2. Методи написання коду. Фреймворки

Якщо розробник бажає створити web-сайт або додаток то в нього існує 3 можливі шляхи реалізації цієї задачі в залежності від того наскільки сильно бажають використовувати вже завчасно створенні рішення в розробці:

1. Написання сайту з нуля. Такий підхід надає дуже велику гнучкість, оскільки розробник не має накладених на нього додаткових обмежень у зв'язку з використанням вже наперед написаної системи або основи, однак цей варіант є найбільш затратним по часу. Зазвичай такий підхід краще підходить для складних або незвичайних додатків, та часто використовується в великих компаніях.
2. Написання за допомогою фреймворку.

Фреймворк (Framework) – це інфраструктура програмних рішень, що полегшує розробку складних систем. Це можна уявити, як вже написаний каркас, на якому зручно будувати власні системи.

Фреймворк це не теж саме що й бібліотека коду. Бібліотека – це вже раніше написаний код що розробник може використовувати вільно і будь-яким чином. Фреймворк, в свою чергу, це набір щільно пов'язаних таких бібліотек та інструментів, він вимагає від розробника писати конкретним способом або стилем, що може створити проблеми або завадити реалізації деяких можливостей.

Використання фреймворків надає можливість сильно прискорити написання сайту. Фреймворк залишає можливість написання власних компонентів та систем, однак для його використання потрібно мати досвід або попередні знання.

3. Створення за допомогою готової CMS. CMS (Content Management System) – система керування вмістом, тобто програмне забезпечення для організації web-сайтів чи інших інформаційних ресурсів в Інтернеті. За допомогою такої системи можливо в режимі конструктору створити сайт або додаток без написання коду взагалі. Такий варіант є доволі популярним через свою простоту. Такі системи зазвичай створені таким чином що вони вже створюють надійні сайти, але тестування не буде зайвим, наприклад інтеграційне.

Створення свого фреймворку для автоматизації тестування було вибране через те, що цей підхід є доволі гнучким та дозволяє охопити велику кількість сайтів, в тому числі тих що буди створенні через CMS. Сам же фреймворк не буде писатися з чистого листа, адже браузери – це дуже складні та комплексні системи.

2.3 Піраміда тестування

На сьогоднішній день існує велика кількість видів тестування. Кожен із них має свої переваги та можливості застосування. Однак, які тести використовувати, чи можливо їх комбінувати, та в якій кількості конкретні тести мають бути присутніми? Відповідь буде залежать від суб'єкту тестування, але існує загальноприйнята піраміда тестування що допомагає в цьому питанні.

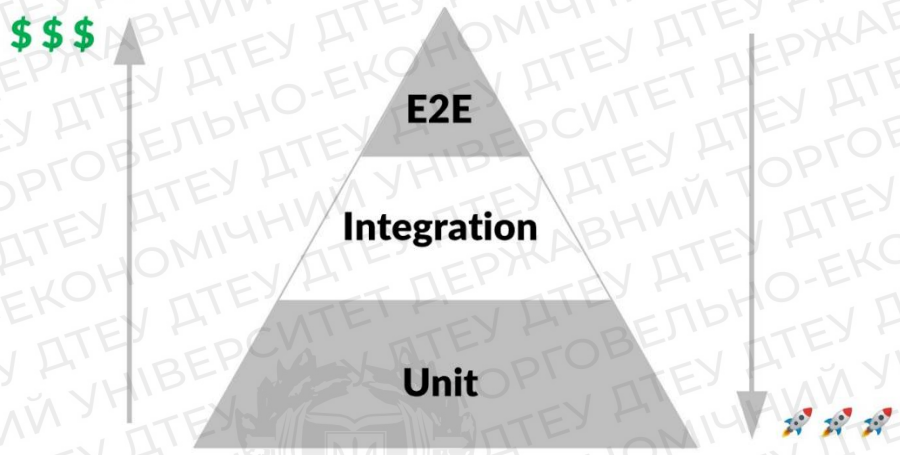

Рис. 2.1 Піраміда тестування

На рисунку 2.1 зображена модель піраміди автоматизації тестування.

Для початку слід розібрати, що таке автоматизація тестування та які тести вважаються автоматичними. Такими тестами є ті, що можуть проходити без участі людини. Існує велика кількість створення та запуску таких тестів, вони можуть бути запущені по команді, або ж зовсім автоматично, наприклад по розпорядку чи при збережені правок на дистанційному репозиторії.

На рисунку вісь X зображає потрібну кількість тестів, а вісь У – грошові затрати на написання та підтримку тестів. Також вісь У зворотньо-пропорційно відповідає за швидкість проведення тестів.

В основі піраміди лежить Unit-тести (Unit testing). Їх повинно бути найбільша кількість. Це автоматичні тести що перевіряють працездатність базових елементів, як-от наприклад натискання кнопки на сайті або роботу окремої функції. Для перевірки запускається робота такого елемента з раніше вирішеним набором даних на вході та правильними відповідями на виході. Перевіряється робота елементу окремо від всієї решти системи. Загальні ідея такого тестування це перевірка того що модифікації вже

написаного коду не змінили би його поведінку та не призвели до поломки вже працюючих частин сайту або програми. Такі тести є дуже простими, та легкі для автоматизації якщо вони вже були написані. Час витрачений на один такий тест є найменшим з-поміж всіх інших типів тестування.

Цей вид тестування є дуже корисним коли над проектом працює одночасно багато людей і становиться важко вслідкувати за усіма змінами та завданнями окремих елементів. Через те що в продукті існує велика кількість елементів unit-тестів також має бути багато - кожний для свого елемента.

Вище від unit-тестів в піраміді знаходяться інтеграційні тести (Integration testing). В таких тестах перевіряються не окремі елементи, а програмні модулі, а також їх сумісність та роботу разом, тобто інтеграцію. Потрібно впевнитися що навіть у випадку правильної роботи окремого модуля він буде адекватно взаємодіяти з іншими та не буде видавати помилок при переміщенні даних від одного модуля до іншого. Варто зазначити, що сучасних реаліях часто такі модулі створюються різними людьми або командами, що робить дуже важливою задачею перевірку продукту інтеграційними тестами.

Під час проведення таких тестів можуть використовувати «заглушки» - це спеціальні утиліти або модулі що можуть імітувати роботу або поведінку інших частин додатку.

Існує декілька підходів до інтеграційного тестування, як-от: Bottom Up Integration, де спочатку тестуються всі низькорівневі модулі, лише потім на них будуть тестувати високорівнені; Top Down Integration, де навпаки: тестують високорівневі модулі спочатку та потім підключаються модулі низького рівня, при необхідності їх функціонал можливо на деякий час замінити заглушкою; Big Bang Integration, де всі модулі тестуються

одночасно. Такий підхід може добре зекономити час, але це може бути доволі ризиковано, адже при великій кількості помилок, або при неправильному написанні тестів, оскільки буде важко зрозуміти де знаходяться помилки та як їх усунути.

На найвищому рівні піраміди знаходяться E2E тести. Це скорочення від End-to-end testing, тобто наскрізні тести та тести графічного інтерфейсу користувача. За їх допомогою перевіряється вже готовий продукт цілком, а також його користувацький інтерфейс. Такі тести займають величезну кількість часу, є доволі складними та коштовними, адже вони тестують одразу багато компонентів з різних сфер на взаємодію. Часто імітуються дії звичайного користувача, або виконується ручне тестування. Існує доволі багато інструментів для автоматизацію такого рівня тестів. Вони використовуються в останню чергу, після інтеграційних та модульних тестів, і можуть гарантувати гарну якість сайту або додатку. Часто такі тести створюються окремими спеціалістами через їх комплексність.

2.4 Концепція Фреймворку

Фреймворк автоматизації тестування web-сервісів створюється за допомогою популярного web-драйвера Selenium та комбінує в собі 2 підходи тестування: Integration тестування та Unit тестування, що є базою в піраміді тестування.

Фреймворк складає із себе інструмент автоматичного тестування сторінки авторизації користувача, та є розроблений таким чином щоб надати якнайбільше можливостей налаштування тестів та гнучкості використання.

Тестування може проходити як на сайті що вже розміщено в інтернеті, так і на локальній копії сайту, що може бути дуже доречно для тестування під час розробки, оскільки є набагато швидшим й простішим підняти локальну версію на локальному комп'ютері або сервері ніж на хостингу. Тестування відбувається наступним чином:

1. Користувач фреймворку скачує необхідні для роботи файли та утиліти, а саме Python версії 3.10 або вище, web-драйвер браузера Selenium та сам фреймворк й розпаковує його файли, що знаходяться в zip-архіві в будь-якому місці на своєму комп'ютері.
2. Користувач запускає файл init.py для того щоб згенерувати файл конфігурації.
3. При необхідності користувач модифікує файл конфігурації згідно документації.
4. Користувач запускає веб-сервер. Він може бути як локальним, так і онлайн в світовій павутині.
5. Користувач запускає файл Selenium_testing і через короткий проміжок часу отримує результати тестування.

Структура фреймворку складає з себе шаблон для написання тестів з вже написаними тестами що відповідають сучасним стандартам та методологіям тестування. При необхідності можливе конфігурування тестів під свій лад або написання своїх власних тестів як розширення шаблону. Сам же фреймворк складається з двох частин: Основного файлу з тестами; допоміжного файлу що створює конфігураційний файл з базовими налаштуваннями.

Висновки

Отже, розробка фреймворку тестування web-сервісів є дуже перспективним напрямком у зв’язку з розвитком та коштовністю часу розробників у web сфері. Використання фреймворку є дуже збалансованим способом розробки, що дозволяє зберегти час та гроші, але при цьому не лишаючи розробника гнучкості та свободи в плані реалізації продукту. Найбільш добре піддаються автоматизації unit-тести та integration-тести, оскільки вони є базовими та дуже однотипними та рутинними. Отже, раціональним є створення фреймворку тестування що буде працювати саме з цими типами тестування.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ WEB-СЕРВІСІВ НА БАЗІ PYTHON

3.1 Обґрунтування використаних модулів та фреймворків

Фреймворк був створений за допомогою мови програмування Python. Для розширення функціоналу були встановленні додаткові пакети за допомогою автоматичного встановлювача пакетів Pip (Package Installer for Python). Розглянемо використанні пакети:

Unittest

Unittest – це пакет для написання автоматизованих unit-тестів за допомогою Python. Необхідний для реалізації функціоналу фреймворку. Є одним із найбільш популярних пакетів тестування для цієї мови через свою зручність та простоту. Пакет включає в себе клас TestCase який може бути успадкованим для створення різних груп тестів, Test suite для групування та запуску великої кількості тестів в черзі, та середовище для запуску тестів.

Однак є декілька обмежень та стандартів. Якщо в Python методи можуть бути оголошені наступним чином:

```
class MyClass(unittest.TestCase):
    def myMethod(self, a,b):
        #Тут пишется код методу, наприклад додавання 2-х змінних
        return a + b
```

Рис. 2.2 Простий метод Python

То тести для цього методу мають мати приставку test_ в своїй назві.

Наприклад ось так:

```
def test_myMethod(self):
    self.assertEqual(myMethod(1,2),3)
```

Рис. 2.3 Приклад тестового метода

Це обов’язково для успішного знаходження тестів інтерпретатором, бо все що має назву, що не розпочинається на test_ буде проігноровано. Але така практика швидко стала стандартом серед розробників автоматизованих тестів тому це не є сильною проблемою в плані сумісності. Навіть при використанні фреймворків та бібліотек що не вимагають такого правила все рівно тести називають таким чином для кращого розуміння.

ConfigParser

Цей модуль для Python надає доступ до класу ConfigParser, що імплементує базову мову конфігурації для створення файлів з налаштуваннями. Мова схожа на формат .ini для Microsoft Windows. Це потрібно для того, щоб користувач фреймворку зміг швидко, просто, і зрозуміло надати потрібні дані для тестування, як-от адреса сайту, логін та пароль тестового користувача, і так далі. Використання такого підходу є значно простішим й зручнішим ніж запит даних перед тестами, оскільки це унеможливило би автоматизацію. Конфігураційний файл створюється окремим файлом init.py, що потрібно запустити перед використанням фреймворку. Також конфігураційний файл включає в себе дані за замовчуванням для наглядного показу роботи фреймворку.

Os

Цей модуль надає широкий функціонал для роботи з операційною системою (Operative system – операційна система, або скорочено os, звідки й походить назва модулю). Хоча спектр можливостей неймовірно широкий, в цьому випадку він потрібний для того щоб отримати доступ до розташування файлів фреймворку для коректної роботи з файлом конфігурації.

Selenium

Selenium WebDriver – головний модуль що відповідає за управління браузером та потрібен для виконання тестів. Модуль є дуже великим, тому потрібно підключати окремі підмодулі. Розглянемо модулі що були використані під час розробки фреймворку. Повний код знаходиться в Додатку А.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
```

Рис 3.1 Імпортування потрібних бібліотек

Підмодуль Keys містить словник з можливими варіантами вводу зі звичайної клавіатури, він потрібен щоб дати драйверу браузера зрозуміти натиснення яких клавіш потрібно емулювати.

Підмодуль Options дає можливість використовувати та налаштовувати параметри запуску web-драйвера. Наприклад запускати тестовий екземпляр браузера на повний екран. Реалізується це в методі

setUp классу TestCase, що виконується одноразово під час запуску пакета тестів.

```
class TestCase(unittest.TestCase):
    @classmethod
    def setUp(self):
        options = Options()
        options.add_argument("start-maximized")
        self.driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
        pass
```

Рис 3.2 Клас setUp фреймворку

Також на цьому коді можливо побачити роботу іншого підмодуля, а саме Service. Цей модуль необхідний для коректного запуску web-драйвера браузера з параметрами.

Наступний підмодуль – WebDriverWait. Він дозволяє призупиняти роботу web-драйвера для того щоб дати браузеру повністю завантажити сторінку, з якою потрібно проводити роботу. Це потрібно для тестування у випадку довгого завантаження сторінки через повільне з'єднання, нестабільності в роботі мережі, великою кількості складний та інтерактивних елементів що іноді при побудові деяких web-додатків може бути неможливо уникнути, або ж для задання своїх стандартів швидкості відповіді сторінки або завантаження конкретного елементу. Шлях до елементу, видимість якого перевіряється задається в конфігураційному файлі у вигляді адреси на мові XPath (XML Path Language – XML мова шляху). Це зручний спосіб конкретно вказати місцезнаходження потрібного елемента на сторінці.

```
try:
    wait = WebDriverWait(self.driver, timeout=5)
    submit_completed = wait.until(EC.visibility_of_element_located((By.XPATH, ReadConfig.element_pass_xpath)))
    if submit_completed:
        self.assertEqual(self.driver.current_url, "https://www.phptravels.net/account/dashboard", "Did not login")

except TimeoutException:
    self.driver.quit()
pass
```

Рис. 3.3 Реалізація перевірки завантаження сторінки користувача

Однак для такої реалізації необхідний ще один підмодуль: Expected Conditions що був скорочено добавлений як EC.

Наступним в імпорті йде модуль TimeoutException який потрібний для додання винятку коли сторінка не була завантажена вчасно. Дуже важливо в розробці правильно «Викидати» та обробляти різного типу винятки для забезпечення стабільності програми.

Наступним є критично важливий підмодуль Selenium: By. Він потрібний для забезпечення роботи локаторів, що є ключовою складовою роботи з тестуванням web-інтерфейсів користувача.

Для того щоб провести тестування потрібно здійснити деякі дії з елементами web-сторінки. Але для цього потрібно спочатку знайти ці елементи. Комп'ютер поки що не може здогадатися куди саме потрібно вводити дані на сайті, на відміну від людини. Тому знаходження елементів здійснюється за допомогою локаторів. Це спеціальний елемент Selenium у вигляді словника мови Python. Він дозволяє знайти необхідний HTML DOM елемент на сторінці. Існує багато способів знаходження елементів, наприклад:

-Через ID елемента, при наявності більшої кількості буде вибрано перший елемент;

-Через ім'я елемента;

-Через XPath, адже HTML сторінка може бути представлена як XML (XHTML) документ;

-Через ім'я класу елемента, при декількох буде повернуто лише перший;

-Через селектори CSS;

-Інші способи;

В якості ключа словника-локатора виступає імпортований елемент Selenium By., а в якості значення виступають власне дані пошуку елемента, найчастіше в форматі string.

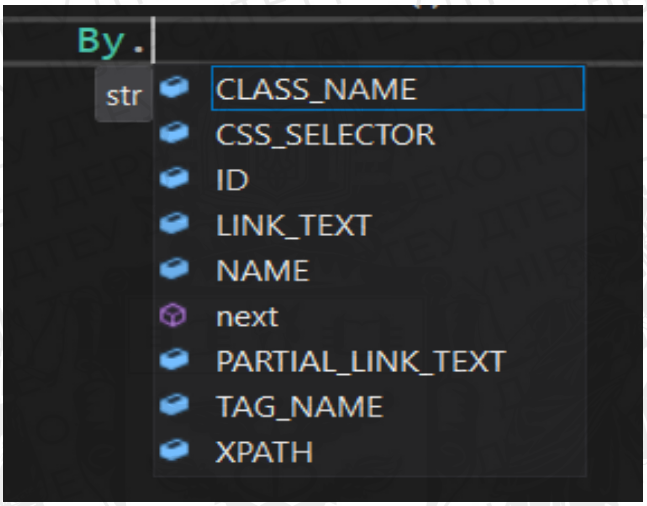

Рис 3.4 Можливі локатори By

Останнім підмодулем є ChromeDriverManager, що потрібний для того щоб запустити web-драйвер в вигляді браузера Google Chrome. Можливе використання будь-яких інших популярних браузерів, потрібно всього лише інсталювати необхідні пакети та імпортувати відповідний підмодуль.

Розроблений додатковий файл Python для ініціалізацію конфігураційного файлу для фреймворку. Він має іншу кількість імпортованих модулів, але використані все уже оглянути розширення.

3.2 Програмна реалізація фреймворку та його особливості

Фреймворк був створений за допомогою середовища розробки Visual Studio Community 2019. Хоча це середовище не підтримує роботу з Python це можливо легко виправити за допомогою установки офіційного пакету підтримки мови Python. Це можна зробити через офіційну утиліту Visual Studio Installer.

Також був установлений Python версії 3.10 з офіційного сайту www.Python.org.

Фреймворк складається з 2-х частин, Init.py для першого налаштування перед запуском шляхом створення конфігураційного файлу та Selenium_testing.py для тестування логін сторінок web-додатків.

init.py Selenium_testing.py

```
import configparser
import os

config = configparser.ConfigParser()
config.add_section('login_data')
config.set('login_data', 'user_email', 'user@phptravels.com')
config.set('login_data', 'user_password', 'demouser')
config.set('login_data', 'login_page_url', 'https://www.phptravels.net/login')
config.set('login_data', 'succesfull_login_page_url', 'https://www.phptravels.net/account/dashboard')

config.add_section('login_elements')
config.set('login_elements', 'email_field_name', 'email')
config.set('login_elements', 'password_field_name', 'password')
config.set('login_elements', 'button_login_name', 'submit')
config.set('login_elements', 'pass_element_xpath', "//*[@class='author-content']")

dirname = os.path.dirname(__file__)
filename = os.path.join(dirname, 'config')

# Записуємо стуктуру до нового файлу
with open(filename, 'w') as configfile:
    config.write(configfile)
```

Як можна бачити, створюються два сектори та ключі до них зі стандартними значеннями.

При запуску видно що був створений новий файл config.

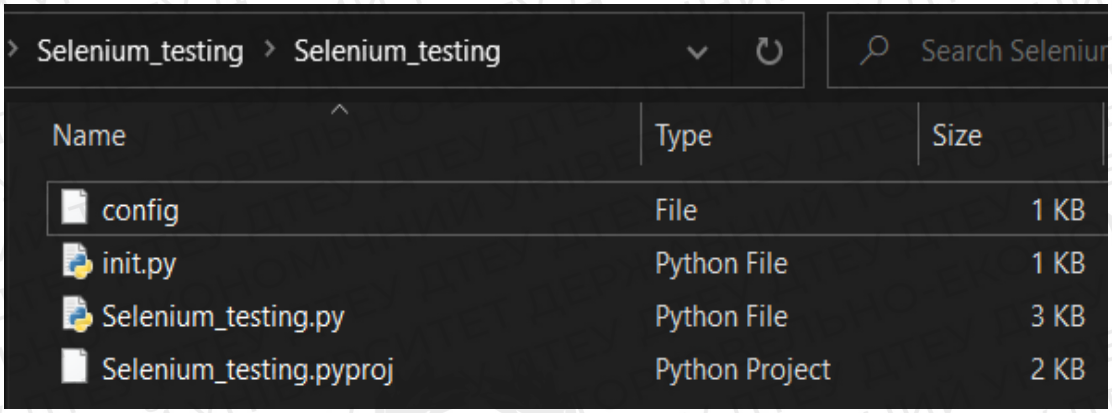

Рис. 3.4 Створений конфігураційний файл

Рис 3.5 Вміст конфігураційного файлу

В якості web-сайту для тестування було вибрано https://phptravels.com/. Це зручний макет сайту для бронювання квитків. Сайт був розроблений для того щоб розробники вчилися тестуванню. Також цей сайт доступний для будь-кого в мережі інтернет.

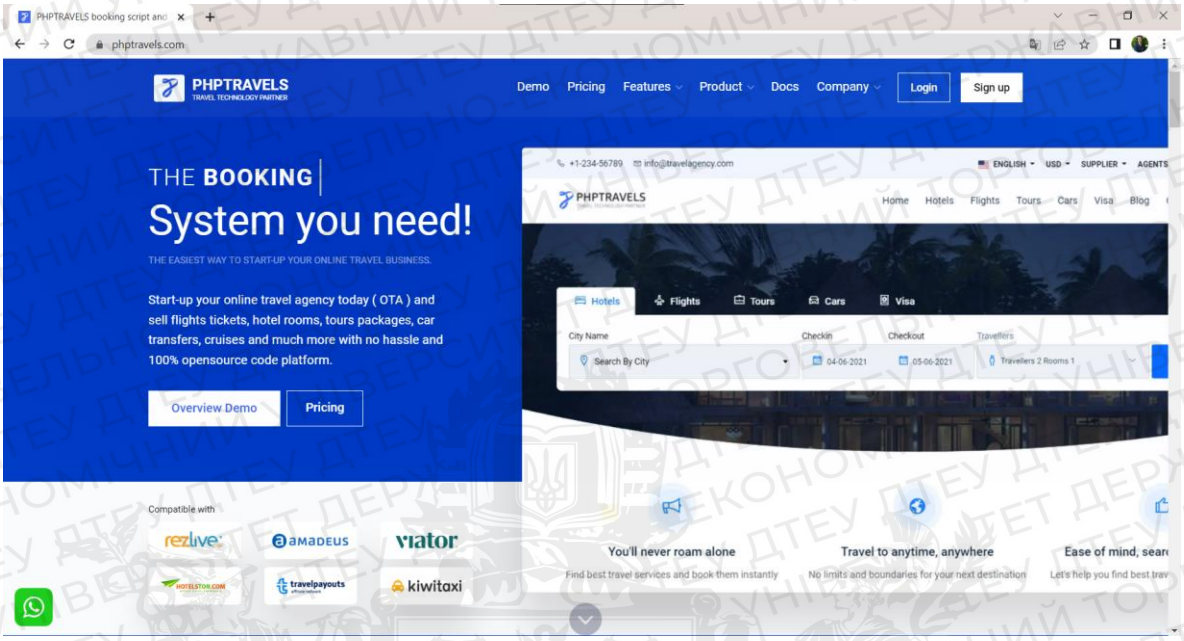

Рис 3.6 Web-сайт типу макета https://phptravels.com/

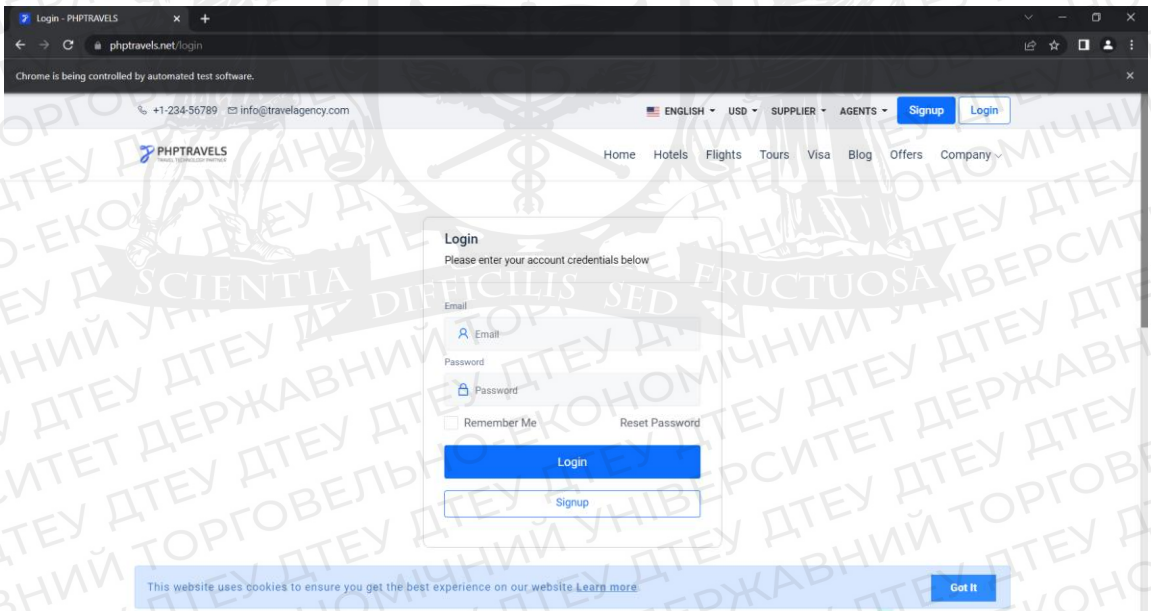

Рис 3.7 Форма логіну Web-сайту, відображена в тестовому екземплярі Chrome

Тестується форма авторизації користувача на сайті. Це дуже доречний вибір для початку розробки фреймворку автоматизації адже

такий шаблон сторінки дуже часто використовується в всесвітній павутині та потрібен для базової реалізації функцій web-додатків.

Логін та пароль тестового користувача можливо задати в конфігураційному файлі. Зазвичай адреси елементів для знаходження об'єктів логіну та пароля використовують однакові, тому можливо майже без змін тестувати будь-які форми авторизації. Але як и пароль та логін ці адреси можливо задати іншими, для покриття всіх випадків.

Після введення логіну та пароля web-драйвер натисне на кнопку авторизації, та буде чекати на завантаження сторінки. У випадку якщо сторінка не буде завантажена вчасно то буде викинуто Timeout Exception виняток. Якщо сторінка завантажиться вчасно то драйвер перевірить чи справді видно ключовий елемент (pass_element), та звіриться з правильною адресою. Якщо адрес не співпадає то буде викинутий виняток з причиною. Якщо все вірно й форма авторизації справді приймає дані користувача та вчасно перенаправляє на потрібну сторінку й на ній вдало завантажується ключовий елемент - то тест буде вважатися пройденим.

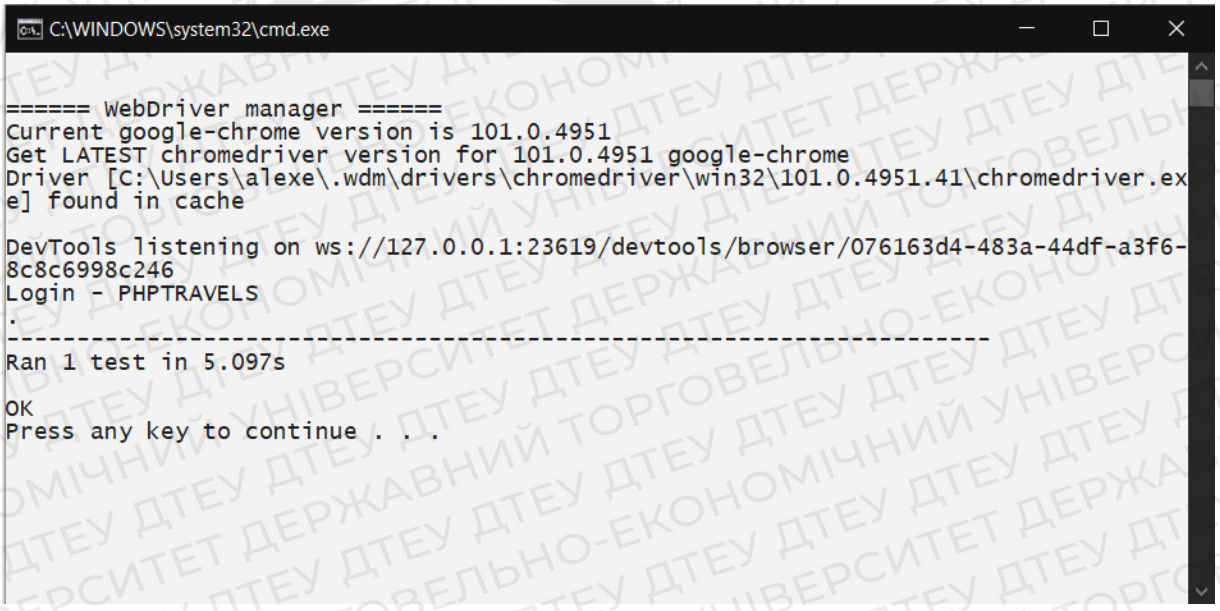

Рис. 3.8 Результат проходження тесту

Добре видно що тест зайняв лише 5 секунд на виконання, цей час включає в себе запуск web-драйвера, що є досить довгою задачею. Після виконання одного тесту може одразу бути запущений інший тест, тим самим можливо створити пакети тестів що будуть слідувати один за одним в одному запущеному екземплярі браузера, без необхідності постійно закривати та створювати нові екземпляри браузерів.

Висновки

Внаслідок проведеної роботи з аналізу особливостей тестування web-сервісів та web-додатків було обґрунтовано створення та практично реалізовано фреймворк тестування цих додатків та сервісів. Фреймворк був написаний на простій для розуміння мові програмування широкого спектру Python. Були використанні сучасні та популярні фреймворки розробки та пакети розширення, а саме Selenium WebDriver, Unittest, та ConfigParser. Використанні пакети розширення є легковісними та не мають великої кількості жорстких обмежень, а ті обмеження що є відповідають загальноприйнятим стандартам розробки, що дозволяє фреймворку бути надзвичайно гнучким та підходити під більшість потреб для швидкої та зрозумілої автоматизації тестування, що можливо буде при потребі адаптувати та доповнити під особисті потреби проекту.

ВИСНОВКИ

У випускній кваліфікаційній роботі представлено результати теоретичних і прикладних робіт, що полягають у розробці фреймворку автоматизації тестування web-додатків. В результаті проведених досліджень були отримані такі висновки:

1. Проаналізовано існуючі на ринку фреймворки автоматизації тестування.

2. Розглянутий процес створення web-додатків та потрібні для цього інструменти.

3. Розглянуті популярні підходи тестування програмного забезпечення.

4. Проаналізовано доцільність створення фреймворку автоматизації тестування.

4. Визначені та класифіковані сучасні підходи до тестування програмного забезпечення.

5. Розроблений фреймворк, що відрізняється високою гнучкістю та легкістю, а також простотою використання.

Отже, автоматизація тестування є дуже перспективним напрямком розробки у зв’язку з швидким розвитком web-технологій та дороговизни їх розробки. Для вирішення цієї проблеми був розроблений фреймворк тестування на базі Python з потребами ринку та урахуванням особливостей web-розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ільченко А. В. Дослідження методів автоматизації тестування програмних e-commerce systems : пояснювальна записка до атестаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 121 - Інженерія програмного забезпечення / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. Харків, 2021. 102 с.
2. Бородіна О.О. Автоматизація тестування програмного забезпечення засобами фреймворків. *Математичне та імітаційне моделювання систем*. МОДС 2018 : тези доп. Тринадцятої міжнар. наук.-практ. конф. (Чернігів, 25 – 29 черв. 2018 р.) / М-во освіти і науки України, Нац. Акад. наук України, Академія технологічних наук України, Інженерна академія України та ін. Чернігів : ЧНТУ, 2018. С. 343–346.
3. Selenium Documentation [Електронний ресурс]. – Режим доступу: https://www.seleniumhq.org/docs/
4. Якість програмного забезпечення та тестування: базовий курс. Навч. Посіб. / За ред. Крепич С.Я., Співак І.Я. / для бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення». Тернопіль: ФОП Паляниця В.А., 2020. 478с.
5. Зосімов В.В. Побудова загальної структури корпоративних веб-додатків. *Управляющие системы и машины*. 2019. № 1. С. 31–40.
6. Андріїв І.В. Аналіз сучасних засобів веб-розробки *Інформатика, комп'ютерна техніка та автоматизація. Сер: Технічні науки.* 2017 [Електронний ресурс]. – Режим доступу: https://www.sworld.com.ua/konfer49/44.pdf
7. Бублик А. Р. Дизайн веб-сторінки, як основа зручності читання, утримання та естетики сучасного веб-сайту. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*: *наук. журн.* 2021. Вип. № 45. С.5-11. [Електронний ресурс]. – Режим доступу: https://www.sworld.com.ua/konfer49/44.pdf
8. Котенко Т. О., Жирова В. І., Чубаєвський А. М., Десятко О. Н. Дослідження основних тенденцій сучасної розробки веб-сайтів.

Кібербезпека: освіта, наука, техніка: електронне фахове наукове видання. 2019. Вип. 5. Том 1. С. 6-15.

9. Богач І.В., Кравець А.Ю. Автоматизація тестування web-сторінок за допомогою драйвера браузера Selenium Webdriver. *«Молодий вчений»*. 2015. № 6 (21). Ч. 1. С. 16-19.
10. Кодола Г. М., Волинець Н. С., Сербулова І. В. Автоматизоване тестування веб-додатків з різнорівневою архітектурою. *Вісник НТУ «ХПІ». Сер.: Нові рішення в сучасних технологіях.* 2019. № 5 (1330). С.91-100.
11. Шатовська Т. Б., Каменєва І. В. Дослідження ефективності застосування BDD-фреймворків у тестуванні безпеки web-орієнтованого програмного забезпечення. *Вісник НТУ «ХПІ»*. 2015. №21(1130). С.69-75. [Електронний ресурс]. – Режим доступу: http://mtsc.khpi.edu.ua/article/view/43709
12. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Чернігів: ФОП Баликіна С.М., 2020. 180 с.
13. TIOBE Index for May 2022 TIOBE [Електронний ресурс]. – Режим доступу: https://www.tiobe.com/tiobe-index/
14. Воронін, М. Г. Автоматизація тестування вебдодатків : дипломний проєкт ... бакалавра : 123 Комп'ютерна інженерія / Воронін Микита Глібович. – Київ, 2020. – 77 с.
15. Шапран О.Ю., Філімонова Т.О. Особливості тестування WEB-технологій та інтернету речей. Збірник тез IX Міжнародної науково-технічної конференції. Інформатика, управління та штучний інтелект. – Харків: НТУ "ХПІ", 11-13.05.2022. – С. 139.
16. Філімонова Т.О., Шапран О.Ю. Розробка фреймворку тестування Web – сервісів. Науково-технічна конференція ВНТУ, Вінниця, 30-31.05.2022. [Електронний ресурс]. − Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/14829/12492

ДОДАТКИ

Додаток А

Selenium_testing.py:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
import unittest
import os
import configparser
import time

class ReadConfig():
    dirname = os.path.dirname(__file__)
    confname = os.path.join(dirname, 'config')
    config_obj = configparser.ConfigParser()
    config_obj.read(confname)
    config_key_user = config_obj["login_data"]
    user_email = config_key_user["user_email"]
    user_pass = config_key_user["user_password"]
    login_page = config_key_user["login_page_url"]
    user_page = config_key_user["succesfull_login_page_url"]

    config_key_elements = config_obj["login_elements"]
    element_name_email =  config_key_elements["email_field_name"]
    element_name_pass = config_key_elements["password_field_name"]
    element_name_button = config_key_elements["button_login_name"]
    element_pass_xpath = config_key_elements["pass_element_xpath"]

class TestCase(unittest.TestCase):
    @classmethod
    def setUp(self):
        options = Options()
        options.add_argument("start-maximized")
        self.driver =
webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
        pass

    def test_login(self):
        self.driver.get(ReadConfig.login_page)
        print(self.driver.title)
        email_field = self.driver.find_element(by=By.NAME,
value=ReadConfig.element_name_email)
        self.assertTrue(email_field, "No suitable email element found")
        email_field.clear()
        email_field.send_keys(ReadConfig.user_email)
```

```
        passw_field = self.driver.find_element(by=By.NAME,
value=ReadConfig.element_name_pass)
        self.assertTrue(passw_field, "No suitable password element found")
        passw_field.clear()
        passw_field.send_keys(ReadConfig.user_pass)
        button = self.driver.find_element(by=By.XPATH, value="//button[@type='" +
ReadConfig.element_name_button + "']")
        self.driver.execute_script("arguments[0].scrollIntoView();", button)
        self.driver.execute_script("arguments[0].click();", button)

        try:
            wait = WebDriverWait(self.driver, timeout=5)
            submit_completed = wait.until(EC.visibility_of_element_located((By.XPATH,
ReadConfig.element_pass_xpath)))
            if submit_completed:
                self.assertEqual(self.driver.current_url, ReadConfig.user_page, "Did
not login")

        except TimeoutException:
            self.driver.quit()
        pass

    @classmethod
    def tearDown(self):
      self.driver.close()
      pass


if __name__ == "__main__":
        unittest.main()
```

init.py:

```
import configparser
import os

config = configparser.ConfigParser()
config.add_section('login_data')
config.set('login_data', 'user_email', 'user1@phptravels.com')
config.set('login_data', 'user_password', 'demouser')
config.set('login_data', 'login_page_url', 'https://www.phptravels.net/login')
config.set('login_data', 'succesfull_login_page_url',
'https://www.phptravels.net/account/dashboard')

config.add_section('login_elements')
config.set('login_elements', 'email_field_name', 'email')
config.set('login_elements', 'password_field_name', 'password')
config.set('login_elements', 'button_login_name', 'submit')
config.set('login_elements', 'pass_element_xpath', "//*[@class='author-content']")

dirname = os.path.dirname(__file__)
filename = os.path.join(dirname, 'config')

# Записуємо стуктуру до нового файлу
with open(filename, 'w') as configfile:
        config.write(configfile)
```