МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Система автоматизованого тестування вебзастосунків з використанням Selenium та PyTest

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.22011006

Здобувач _____В. М. Жердецький

підпис

«__» _____ 2024 р.

Керівник канд. пед. наук, доцент _____К. О. Кірей

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувачу освіти групи 408 факультету комп'ютерних наук

_____Жердецькому Владиславу Миколайовичу _____

(прізвище, ім'я, по батькові здобувачки освіти)

1. Тема кваліфікаційної роботи

Система автоматизованого тестування вебзастосунків з використанням Selenium та PyTest_____

Затверджена наказом по ЧНУ від «22» грудня 2023 р. № __269__

2. Строк представлення кваліфікаційної роботи «____» _____ 2024 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є розробка системи автоматизованого тестування вебзастосунків з використанням Selenium та PyTest

4. Перелік питань, що підлягають розробці:

- дослідження предметної області;

- формування специфікації вимог до програмного забезпечення;

- визначення архітектури для проектування програмного забезпечення;

- моделювання та проектування програмного забезпечення;

- розробка програмного забезпечення;

- проведення аналізу результатів розробки;

5. Перелік графічних матеріалів:

Презентація

6. Завдання до спеціальної частини

Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А.О	Кафедра екології ЧНУ ім. Петра Могили	Спеціальна частина з охорони праці

Керівник роботи ___канд.пед.наук, доцент Кірей Катерина Олександрівна _____

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання ____Жердецький Владислав Миколайович ____

(прізвище, ім'я, по батькові здобувачки освіти)

_____

(підпис)

Дата видачі завдання «22» грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи бакалавра

Тема: «Система автоматизованого тестування вебзастосунків з використанням Selenium та PyTest»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	31.11.2023	06.12.2023	Виконано
2	Огляд літератури за темою роботи	12.12.2023		Виконано
3	Складання календарного плану КРБ	06.02.2024	07.03.2024	Виконано
4	Аналіз предметної галузі	07.03.2024	11.03.2024	Виконано
5	Розробка проєктних рішень	16.03.2024	22.03.2024	Виконано
6	Моделювання та конструювання ПЗ	23.03.2024	07.04.2024	Виконано
7	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	08.04.2024	10.05.2024	Виконано
8	Оформлення КРБ та презентації	15.05.2024	27.05.2024	Виконано
9	Розробка спеціальної частини з охорони праці	28.05.2024	02.06.2024	Виконано
10	Попередній захист	03.06.2024	04.06.2024	Виконано
11	Відгук керівника КРБ	10.06.2024	15.06.2024	Виконано
12	Рецензування	16.06.2024	16.06.2024	Виконано
13	Захист кваліфікаційної роботи	24.06.2024	27.06.2024	Виконано

Розробила здобувачка освіти ____Жердецький Владислав Миколайович_____
(прізвище, ім'я, по батькові) *(підпис)*
«22» __квітня__ 2024 р.

Керівник роботи _____ канд.пед.наук, доцент Кірей Катерина Олександрівна
(посада, прізвище, ім'я, по батькові) *(підпис)*

«22» __квітня_____ 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Система автоматизованого тестування вебзастосунків з використанням Selenium та PyTest»

Студента 408 групи: Жердецький Владислав Миколайович

Керівник: канд.пед.наук, доцент Кірей К. О.

Кваліфікаційна робота присвячена розробці системи автоматизованого тестування вебзастосунків з використанням Selenium та PyTest.

Об’єктом кваліфікаційної роботи є методи організації автоматизованого тестування вебзастосунків.

Предметом кваліфікаційної роботи є організація тестування вебзастосунків з використанням набору бібліотек Selenium та фреймворку PyTest.

Метою кваліфікаційної роботи є розробка програмного забезпечення тестування вебзастосунків для підвищення ефективності цього процесу шляхом пришвидшення роботи тестів.

Кваліфікаційна робота складається з вступу, 4 розділів, висновків та переліку джерел посилань.

У вступі визначається актуальність теми, мета, предмет та об’єкт дослідження.

В першому розділі аналізуються існуючі аналоги, детально пояснюються наявні плюси та мінуси аналогів в порівнянні з розроблюваною системою. За результатами аналізу проводиться аналіз власної системи, розроблюються функціональні вимоги, та способи використання системи.

Другий розділ містить можливі способи використання системи, для цього створюються тест-кейси трьох рівнів в яких детально описується кожен крок використання. Окрім цього створюються UML діаграми які демонструються способи використання системи з технічого боку.

В третьому розділі розробляється безпосередньо архітектура системи, з використанням діаграм різних типів, на кшталт діаграми класів, пакетів, компонентів. Також демонструється стек технологій, і описуються технології які

використані в розробці системи.

Четвертий розділ містить в собі розробку системи автоматизованого тестування вебзастосунків з використанням Selenium та PyTest. Демонструються різні технічні рішення які були задіяні під час розробки, надається повний об’єм інформації того що було розроблено, а також демонстрація роботи всіх задіяних та розроблених елементів.

У висновках проводиться аналіз виконаних робіт та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 62 сторінки, містить 4 розділи,
44 ілюстрації, 6 таблиці, 20 джерел в переліку посилань.

Ключові слова: розробка системи, UML діаграми, вимоги.

ABSTRACT

of the Bachelor's Thesis

«Development of an Automated Web Application Testing System using Selenium and PyTest»

Student: Vladyslav Zherdetskyi

Supervisor: Candidate of Pedagogical Sciences, Associate Professor Kirei Kateryna Oleksandrivna

This thesis is dedicated to the development of an automated testing system for web applications using Selenium and PyTest.

The object of the qualification work is methods of organizing automated testing of web applications.

The subject of the qualification work is the organization of testing web applications using a set of Selenium libraries and the PyTest framework.

The goal of the qualification work is to develop web application testing software to increase the efficiency of this process by speeding up the tests.

The qualification work consists of an introduction, 4 sections, conclusions and a list of reference sources.

The introduction defines the relevance of the topic, the purpose, subject and object of the research.

The first section analyzes the existing analogues, explains in detail the pros and cons of the analogues in comparison with the developed system. Based on the results of the analysis, an analysis of the own system is carried out, functional requirements and ways of using the system are developed.

The second section contains possible ways of using the system, for this, three-level test cases are created in which each step of use is described in detail. In addition, UML diagrams are created that demonstrate ways of using the system from the technical side.

In the third section, the system architecture is developed directly, using diagrams of various types, such as diagrams of classes, packages, and components. The technology stack is also demonstrated, and the technologies used in the development of the system are described.

The fourth section includes the development of a system for automated testing of web applications using Selenium and PyTest. Various technical solutions that were used during development are demonstrated, a full volume of information about what was developed is provided, as well as a demonstration of the work of all involved and developed elements.

The conclusions analyze the work performed and the results obtained.

The bachelor's qualification work is laid out on 62 pages, contains 4 chapters, 44 illustrations, 6 tables, 20 sources in the list of references.

Keywords: system development, UML diagrams, requirements.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

UML – Unified Modeling Language

ВСТУП

Кваліфікаційну роботу бакалавра присвячено розробці системі автоматизованого тестування вебзастосунків з використанням Selenium та PyTest.

Актуальність теми полягає у зростаючій потребі розробників програмного забезпечення в ефективних методах автоматизації тестування вебзастосунків. З поширенням вебтехнологій і зростанням кількості вебзастосунків збільшується необхідність у швидкому та надійному тестуванні для забезпечення якості програмного продукту. Використання Selenium та PyTest дозволяє автоматизувати процес тестування, зменшуючи витрати на ручне тестування і підвищуючи продуктивність розробки. Така система також забезпечує більшу точність і стабільність тестів, що дозволяє вчасно виявляти помилки та покращувати якість вебзастосунків.

Об’єктом кваліфікаційної роботи є методи організації автоматизованого тестування вебзастосунків.

Предметом кваліфікаційної роботи є організація тестування вебзастосунків з використанням набору бібліотек Selenium та фреймворку PyTest.

Метою кваліфікаційної роботи є розробка програмного забезпечення тестування вебзастосунків для підвищення ефективності цього процесу шляхом пришвидшення роботи тестів.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) дослідження предметної області;
2) формування специфікації вимог до програмного забезпечення;
3) визначення архітектури для проектування програмного забезпечення;
4) моделювання та проектування програмного забезпечення;

5) розробка програмного забезпечення;

6) проведення аналізу результатів розробки.

Кваліфікаційна робота бакалавра викладена на 62 сторінок, містить 4 розділи, 44 ілюстрацій, 6 таблиць, 20 джерел в переліку посилань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд та аналіз аналогів

На етапі розробки будь-якого продукту ключовою є аналітична робота, спрямована на вивчення конкурентного середовища та визначення місця, яке ваш продукт може зайняти на ринку. Аналіз конкурентів допоможе виділити основні недоліки існуючих пропозицій, що стане важливим кроком у розробці стратегії покращення вашого продукту. Аналіз включає вивчення потреб користувачів, аналіз аналогів, визначення функціональних та нефункціональних вимог, оцінку технологічних можливостей та розробку архітектури системи. Результатом цього аналізу може стати розробка нововведень, які раніше не були доступні в аналогах. Такий підхід не лише забезпечить залучення нових користувачів, але й забезпечить стабільність та конкурентоспроможність вашого продукту на ринку. Із цією метою було виділено прямих конкурентів, серед яких важливо відзначити:

1) Appium (див. табл. 1.1 та рис. 1.1);
2) Katalon Studio (див. табл. 1.2 та рис. 1.2);
3) BrowserStack (див. табл. 1.3 та рис. 1.3).

Appium [1]

Таблиця 1.1 – Аналіз системи Appium

Виробник	Appium
Архітектура	3-tier application
Мова реалізації	Java, JavaScript

Кінець таблиці 1.1

Основні функції	Автоматизоване тестування мобільних додатків, вебінтерфейсів, гібридних додатків, ігрових додатків,
Переваги	1) кросплатформенність, 2) розширені можливості
Недоліки	1) вимога до високої продуктивності обладнання (Appium може вимагати значних обчислювальних ресурсів, особливо при роботі з великою кількістю симуляцій або емуляцій)
Вебсайт	https://appium.io/docs/en/latest/

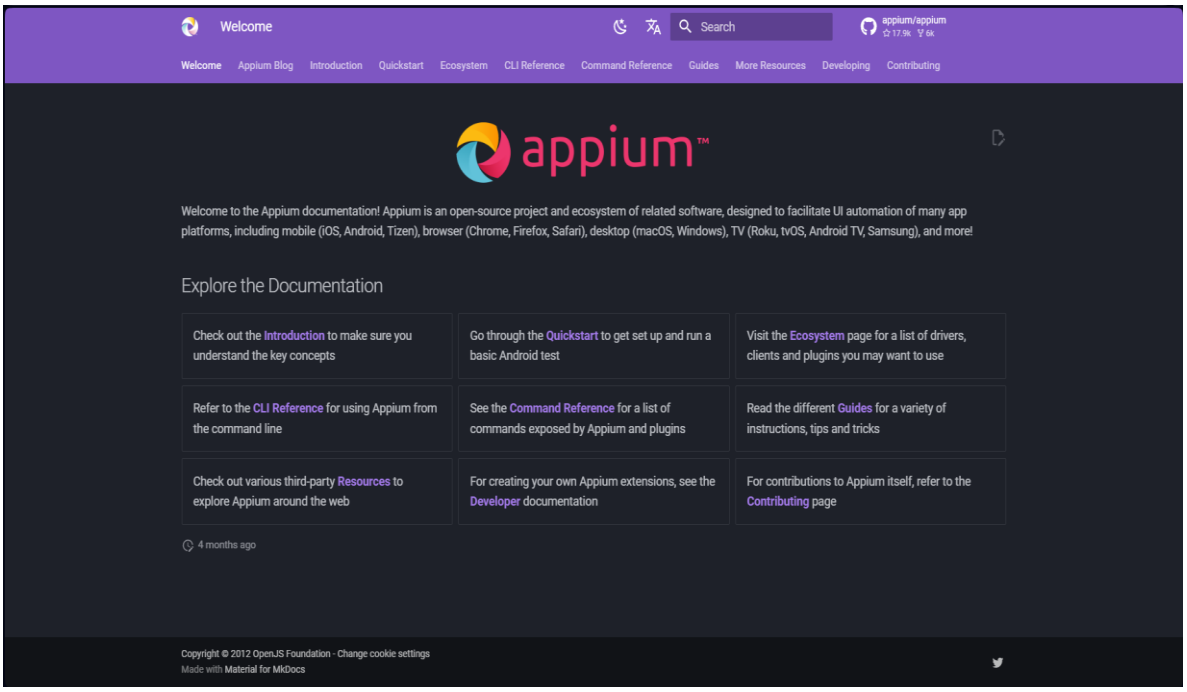


Рисунок 1.1 – Вигляд інтерфейсу застосунку Appium

Katalon Studio [2]

Таблиця 1.2 – Аналіз системи Katalon Studio

Виробник	Katalon
Архітектура	3-tier application
Мова реалізації	Java
Основні функції	Автоматизоване тестування мобільних додатків, вебінтерфейсів,
Переваги	1) простий у використанні 2) зручний інтерфейс
Недоліки	3) продуктивність у великих проектах (Певні дослідження зазначають, що Katalon може сповільнюватися або ставати менш стабільним у дуже великих проектах тестування)
Вебсайт	https://katalon.com

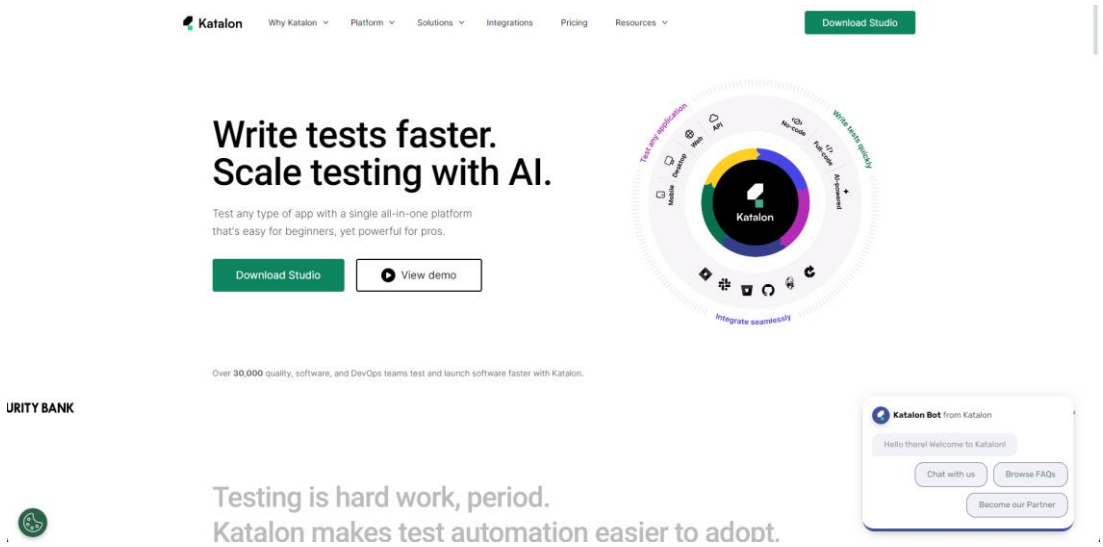


Рисунок 1.2 – Вигляд інтерфейсу застосунку Katalon

BrowserStack [3]

Таблиця 1.3 – Аналіз системи BrowserStack

Виробник	BrowserStack
Архітектура	Web application
Мова реалізації	-
Основні функції	Автоматизоване тестування мобільних додатків, вебінтерфейсів,
Переваги	1) велика кількість варіантів тестування 2) простий інтерфейс
Недоліки	1) вартість платної підписки(Підписки можуть бути досить дорогими, особливо для стартапів або індивідуальних розробників, що обмежує доступ до повного спектру його можливостей для менших компаній.)
Вебсайт	https://www.browserstack.com

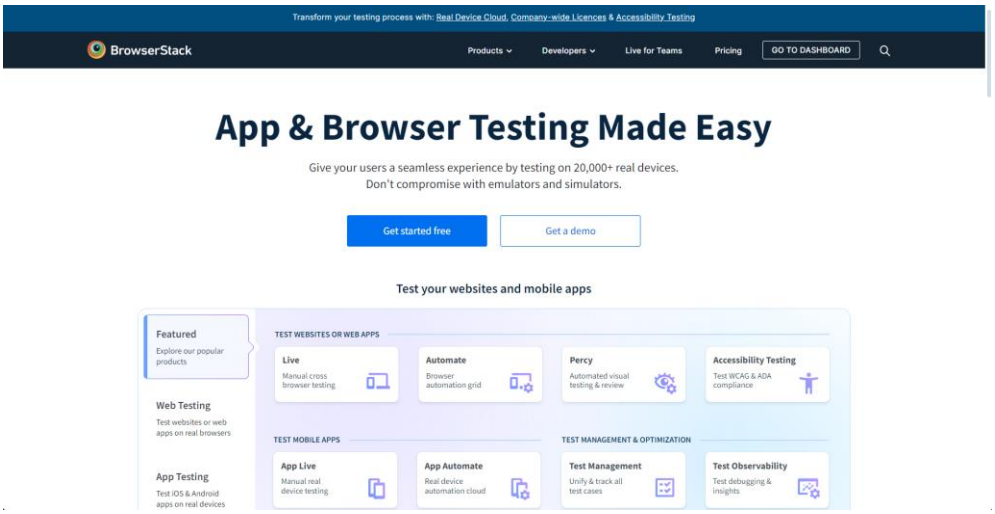


Рисунок 1.3 – Вигляд інтерфейсу застосунку BrowserStack

1.2 Аналіз системи що розроблюється

Під час аналізу будуть враховані переваги та недоліки існуючих рішень, таких як обмежена кількість протягом безкоштовного періоду для Katalon Studio, та інші особливості. Стратегія тестування та управління ризиками також буде розроблена для забезпечення якості та надійності системи. Основна мета аналізу полягає в тому, щоб створити програмний продукт, який буде мати конкурентоспроможні переваги порівняно з існуючими рішеннями.

Таблиця 1.4 – Опис системи що розробляється

Основні задачі	Автоматизоване тестування
Користувачі системи	1) Ззареєстрований користувач 2) гість
Сценарії роботи системи	1) вибір тестових кейсів та запуск тестів 2) збір результатів тестування 3) аналіз результатів тестування

Кінець таблиці 1.4

Засоби апаратної та програмної реалізації	1) MySQL 2) Python 3) Selenium WebDriver
Вихідні дані	4) звіти про результати тестування

Рисунок 1.4 – Макет інтерфейсу застосунку «TestifyWeb»

Mock-up було створено за допомогою програмного забезпечення «Figma». Кольорова гама була підібрана згідно зі стандартами сумісності кольорів. Загалом користувачу надаються наступні розділи застосунку:

1) головна сторінка(вона відповідає за вибір типу тестування)
2) сторінка реєстрації аккаунту
3) про застосунок(короткий опис застосунку)

1.3 Специфікація вимог до розроблюваної системи

Призначення системи, для якої розробляється програмне забезпечення

Призначення застосунку є автоматизація та пришвидшення виправлення помилок на сайті не маючи навичків тестування.

Погодження ухвалені в програмній документації

Було узгоджено що для створення програмного забезпечення будуть використовуватись мова програмування Python та Selenium [4]

Межі проєкту ПЗ

Крайня дата завершення робіт над проєктом – 03.06.2024

Сфера застосування

Дане ПЗ не має особливих меж застосування.

Характеристики користувачів

Користувачу необхідний постійний доступ до інтернет з'єднання

Загальна структура та склад системи

Система має наступні компоненти:

1) клієнт (застосунок на базі Node.js)
2) код(застосунок на базі фреймворку Selenium, PyTest[5] та Flask API)
3) сховище даних(база данних MySQL)

Обмеження

Основне обмеження це постійне інтернет з'єднання

Функції системи автоматизованого тестування вебзастосунків з використанням Selenium та PyTest

Функція пошуку елементу за ідентифікатором (ID)

Опис функції

Функція призначена для знаходження елемента на вебсторінці за його ідентифікатором (ID). Вона допомагає автоматизувати пошук конкретного елемента для подальшої взаємодії з ним.

Вхідна та вихідна інформація

Вхідна інформація – об'єкт WebDriver та ідентифікатор (ID) шуканого елемента.

Вихідна інформація – знайдений елемент або None, якщо елемент не знайдено.

Функціональні вимоги

Функція повинна здійснювати очікування наявності елемента на сторінці протягом певного часу.

Якщо елемент буде знайдено, він повинен бути повернутий.

Якщо елемент не буде знайдено протягом визначеного часу, функція повинна повернути None.

Функція введення тексту в вебелемент

Опис функції

Ця функція призначена для автоматизованого введення тексту в вебелемент на сторінці за допомогою Selenium WebDriver. Вона дозволяє заповнювати тексові поля, текстові області або інші елементи, які приймають текстовий ввід.

Вхідна та вихідна інформація

Вхідна інформація – об'єкт WebDriver, ідентифікатор (ID), ім'я, клас або інший селектор елемента, який приймає текстовий ввід, і текст, який потрібно ввести.

Вихідна інформація – функція не повертає значення.

Функціональні вимоги

Функція повинна шукати вебелемент на сторінці за вказаним селектором.

Після знаходження елемента, текст, який потрібно ввести, повинен бути введений у вебелемент.

Якщо елемент не буде знайдено, функція повинна викинути виняток або

повідомлення про помилку.

Після введення тексту, можуть бути додаткові дії, такі як натискання клавіш Enter або зміна фокусу.

Джерела та зміст вхідної інформації

У програмному забезпеченні для автоматизованого тестування вебзастосунків з використанням Selenium та PyTest головним джерелом інформації є тестувальні сценарії та самі вебзастосунки. Тестувальні сценарії визначають набір дій, які потрібно виконати для перевірки функціональності вебзастосунків.

Нормативно-довідкова інформація (класифікатори, довідники тощо)

Вимоги до даного пункту відсутні.

Вимоги до технічного забезпечення

Вимоги до технічного забезпечення мають певні обмеження. Комп’ютер або ноутбук з якого будуть виконуватись тестування, доступ до інтернет з’єднання також мати підтримку сучасних браузерів.

Вимоги до програмного забезпечення

Архітектура програмної системи

Система складатиметься з вебзастосунку для тестування вебдодатків, серверної частини та бази даних.

Системне програмне забезпечення

Вебзастосунок буде розроблений за допомогою фреймворку Node.js, серверна частина буде побудована на основі Python, а в якості бази даних буде використовуватись MySQL.

Мережне програмне забезпечення

Для створення програмного забезпечення обрана операційна система, Windows 10 редактор коду VS Code, PyCharm. Браузер Google Chrome

Програмне забезпечення ведення інформаційної бази

В якості сховища даних буде використана база даних MySQL

Мова і технологія розробки ПЗ

Система буде розроблена за допомогою Python та його фреймворків PyTest[6] та Selenium для вебзастосунків[7].

Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача

Інтерфейс користувача повинен бути побудований з урахуванням всіх стандартів за для кращого досвіду користування. Навігаційна панель: Розміщена у верхній частині сторінки з вкладками "Welcome", "About", "Register". Головна панель (TestifyWeb): Включає вітаючий текст та опис інструменту, який дозволяє знаходити та виправляти помилки на сайтах. Кнопки дій: "Register testing" та "Main page testing" для запуску тестувань різних аспектів сайту. Футер: Містить іконку зі зв'язком, можливо, для швидкого доступу до підтримки або допомоги.

Апартний інтерфейс

Апаратним інтерфейсом буде пристрій який використовує користувач для роботи з застосунком.

Програмний інтерфейс

Selenium WebDriver для Python: Забезпечує автоматизацію взаємодій із браузером.

PyTest: Використовується для організації та виконання тестів, з можливістю генерування звітів та обробки помилок[8].

Інтерфейс бази даних: Збереження деталей про результати тестів для подальшого аналізу і перегляду.

Властивості програмного забезпечення

Доступність

Інтерфейс простий для користувачів всіх рівнів, без потреби в спеціалізованих знаннях для виконання тестів.

Безпека

Використання JWT для захисту сесій користувачів і забезпечення цілісності даних під час обміну інформацією між клієнтом та сервером.

Продуктивність

Система оптимізована для швидкого та ефективного збору даних з тестувань, мінімізуючи затримки та навантаження на сервер.

Простота використання

Мінімалістичний дизайн з чітко визначеними функціональними зонами забезпечує легке освоєння системи новими користувачами.

Функціональність

Система забезпечує вибір тестових кейсів, дозволяючи користувачам тестувати різні аспекти сайтів, як реєстрація на сайті або головна сторінка.

Висновок до розділу 1

У першому розділі кваліфікаційної роботи було проведено детальний аналіз існуючих інструментів автоматизованого тестування вебзастосунків. Це дозволило виокремити ключові недоліки цих інструментів та розробити стратегії для їх вирішення у власній системі тестування. Окрім того, було проведено аналіз системи, що розробляється, і сформовано список основного функціоналу, який включає модулі для реєстрації, тестування основної сторінки, і інтерфейсів для користувачів. Описано сценарії роботи та визначено технології для реалізації системи, включаючи Selenium WebDriver[9] та PyTest для написання та виконання тестів. Також надано детальну специфікацію вимог до розроблюваної системи, яка охоплює повну характеристику функціональних та нефункціональних вимог, можливі обмеження.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Створення варіантів використання

Короткий use case

Користувач системи тестування вводить URL вебсайту, який бажає протестувати, і обирає опцію "Тестування реєстрації". Система відкриває браузер, переходить на вказану адресу та виконує тест за допомогою PyTest і Selenium, автоматично заповнюючи форму реєстрації[10]. Після відправлення форми система перевіряє відповідь сайту. Якщо реєстрація успішна, результат фіксується як успішний.

Поверхневий use case

Головний сценарій (успішний):

Користувач вводить URL вебсайту для тестування у систему.

Користувач обирає "Тестування головної сторінки" з доступних опцій тестування.

Система використовує Selenium для відкриття браузера та переходу на вказану URL-адресу.

Система перевіряє чи головна сторінка швидко завантажується та чи всі елементи відображаються коректно.

Система збирає дані про час завантаження сторінки та наявність критичних помилок.

Альтернативні сценарії:

Якщо вебсторінка завантажується надзвичайно повільно, система автоматично фіксує це як попередження та надає деталізовані рекомендації для оптимізації.

Якщо сторінка не відкривається або відображає помилки, це фіксується як критична помилка.

Таблиця 2.1 – Повний use case

Scope	Система автоматизованого тесутвання вебзасосунків
Level	Мета користувача (user goal level)
Primary Actor	Користувач
Preconditions	Наявність інтернет підключення
Stakeholders and interests	1) користувач зацікавлений в отриманні звіту про результати тестування. 2) розробник зацікавлений у підтримці злагодженої роботи застосунку.
Main Success Scenario	1) користувач заходить на вебсайт з метою тестування функції реєстрації на бажаному сайті. 2) користувач обирає відповідне тестування на головній сторінці. 3) користувач вводить посилання на бажаний сайт для тестування входу/реєстрації в систему. 4) користувач отримує сформований звіт про результати тестування. 5) користувач аналізує наданий звіт згідно з результати тесту.
Result	Користувач провів тестування обраного вебсайту на елемент входу/реєстрації
Extentions	1) система не може відкрити сторінку: показує повідомлення про помилку. 2) форма не заповнюється через помилку валідації. 3) відповідь сайту вказує на проблему з реєстрацією.
Special Requirements	4) система повинна мати доступ до інтернету 5) система повинна мати підтримку браузера selenium

Кінець таблиці 1.4

Frequency of Occurrence	Часто (залежить від потреб користувача)

Діаграми прецедентів важливі для аналізу автоматизованого тестування вебзастосунків, оскільки вони дозволяють описати основні випадки використання системи тестування, включно з її застосуванням. Такі діаграми чітко показують взаємодію між користувачами (тестерами) та самою системою, окреслюючи контекст та основні вимоги до реалізації тестування. Це забезпечує зрозуміле уявлення про робочі процеси та допомагає визначити область застосування системи у веброзробці.

2.2 Побудова діаграм взаємодії

Unified Modeling Language (UML) — це набір правил та визначень для специфікації системи автоматизованого тестування вебзастосунків, що керується Object Management Group. Ця система визначень надає набір графічних елементів для моделювання частин системи тестування. У наступних підрозділах даного розділу використовуються UML-діаграми для деталізації розроблюваної системи. В процесі аналізу цієї системи важливо продумувати різні сценарії взаємодії користувачів із інструментом тестування. Для моделювання поведінки використовуються діаграми взаємодії (послідовності).

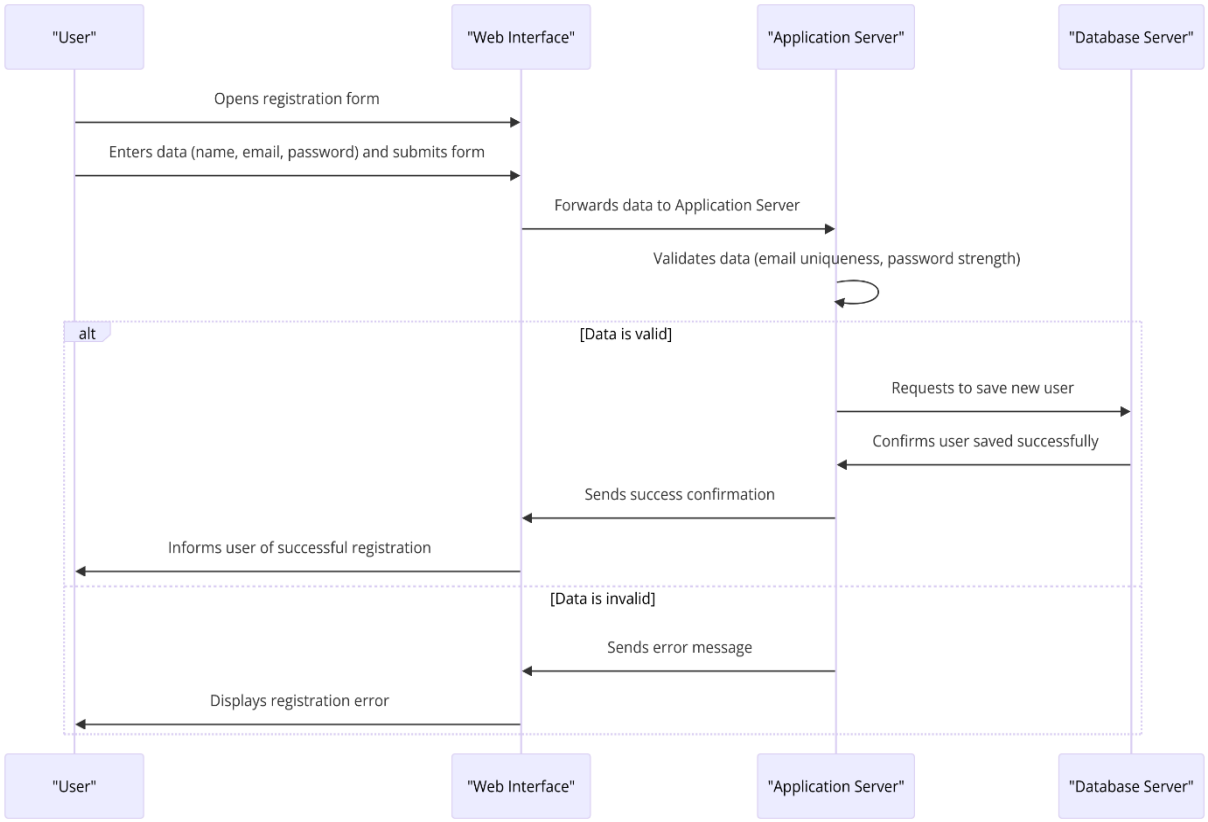


Рисунок 2.1 – Діаграма взаємодії реєстрації користувача

Діграма №1 (рис.2.1) відображає процес реєстрації користувача на веб-сайті з використанням системи, що включає користувача, веб-інтерфейс, сервер застосунків та сервер бази даних. Процес розпочинається з користувача, який відкриває форму реєстрації через веб-інтерфейс. Користувач вводить свої дані, такі як ім'я, електронну адресу та пароль, і подає форму.

Дані пересилаються з веб-інтерфейсу на сервер застосунків, де відбувається їх валідація. Сервер застосунків перевіряє унікальність електронної пошти та силу пароля. Якщо дані визнані валідними, сервер застосунків надсилає запит на сервер бази даних для збереження нового користувача. Після підтвердження успішного збереження користувача, сервер застосунків відправляє підтвердження успішної реєстрації назад через веб-інтерфейс, який інформує користувача про успішну реєстрацію.

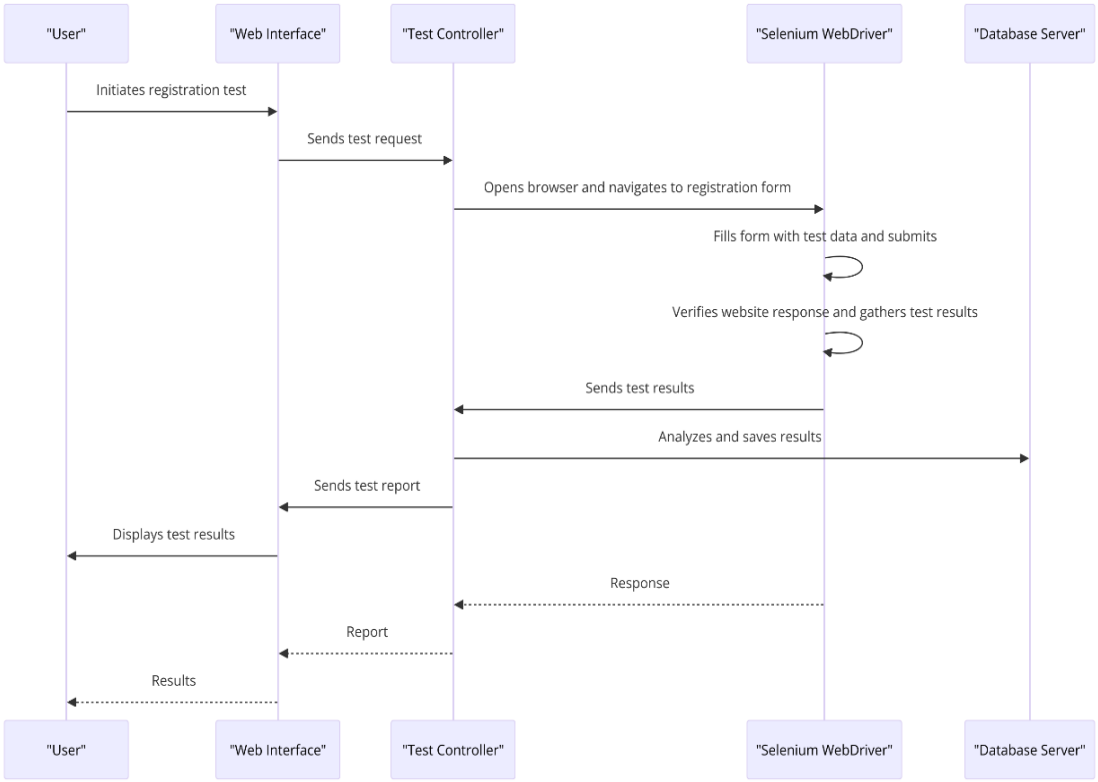


Рисунок 2.2 – Діаграма взаємодії проходження тестування реєстрації

Діаграма №2 (рис.2.2) відображає процес автоматизованого тестування реєстрації користувача на вебсайті, використовуючи Selenium WebDriver. Процес починається, коли користувач ініціює тест реєстрації через веб-інтерфейс, який надсилає запит на тест контролеру. Тест контролер керує Selenium WebDriver, який відкриває браузер, переходить на форму реєстрації, заповнює форму з тестовими даними і подає її. Після подання, WebDriver перевіряє відповідь вебсайту і збирає результати тесту.

Отримані результати тесту надсилаються назад до тест контролера, який аналізує ці результати та зберігає їх у базу даних на сервері бази даних. Одночасно, тест контролер компілює звіт про тест, який відправляється назад через веб-інтерфейс. На заключному етапі, веб-інтерфейс відображає результати тесту користувачу.

Цей процес демонструє інтеграцію між різними компонентами системи, включаючи користувацький інтерфейс, серверні компоненти, і інструменти автоматизації для ефективного виконання тестування реєстрації.

2.3 Побудова діаграм діяльності

Діаграми діяльності в UML використовуються для візуалізації процесів і потоків роботи системи, показуючи різні шляхи, які можуть бути виконані в залежності від умовних виразів.

Для вебзастосунку тестування було розроблено 3 діаграми діяльності:

1) для процесу обробки даних які надходять користувачу після тестування (рис. 2.3);
2) для процесу тестування реєстрації користувача (рис. 2.4);

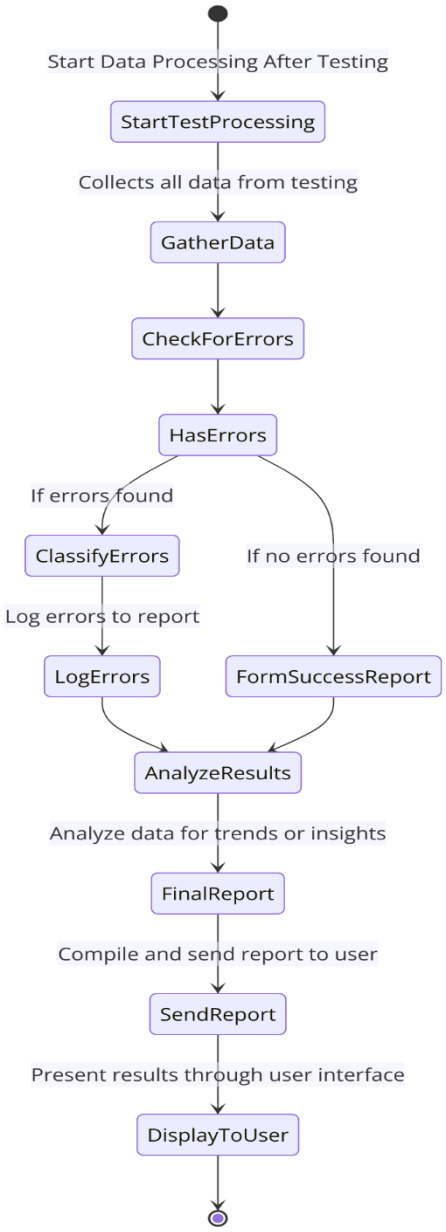


Рисунок 2.3 – Діаграма діяльності для процесу обробки даних

Діаграма №3 (рис. 2.3) описує процес обробки даних після завершення тестування. Процес починається з ініціації обробки даних, де збираються всі дані з тестування. Далі система перевіряє дані на наявність помилок. Якщо помилки знайдені, вони класифікуються і реєструються в звіті. У випадку відсутності помилок формується звіт про успішне виконання. Після цього виконується аналіз зібраних результатів для виявлення тенденцій. На основі аналізу складається звіт,

який надсилається користувачу. Заключний етап включає відображення результатів через користувацький інтерфейс, що дозволяє користувачу переглянути висновки тестування. Ця діаграма забезпечує чітке та структуроване зображення процесу від моменту збору даних до представлення результатів кінцевому користувачу.

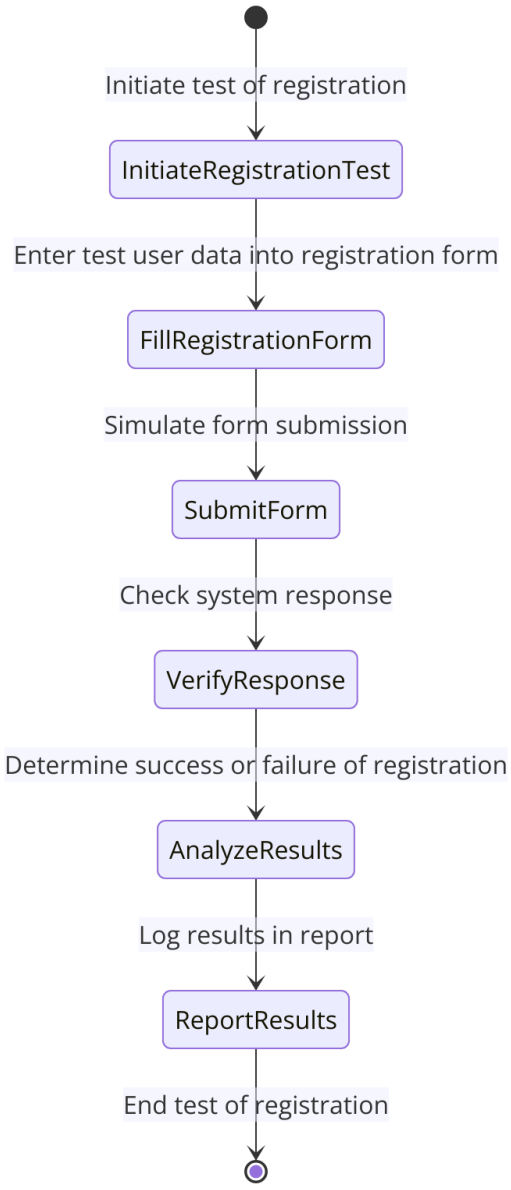


Рисунок 2.4 – Діаграма діяльності для тестування реєстрації користувача

Діаграма №4 (рис. 2.4) детально ілюструє процес тестування реєстрації

користувача на веб-платформі, починаючи з ініціації тесту, введення даних у форму реєстрації, їх подання, перевірки відповіді системи, аналізу результатів, до логування та звітування. Перший крок включає ініціювання тесту реєстрації, де тестовий скрипт активується для запуску процедури. Потім користувач вводить необхідні дані, такі як ім'я, електронна адреса, і пароль у форму реєстрації, що імітується у наступному кроці подання форми. Система обробляє ці дані, і результати цієї обробки перевіряються для визначення, чи була реєстрація успішною або ні. За цим слід аналіз результатів для ідентифікації причин успіху чи невдачі і потенційних помилок у процесі реєстрації. Фінальні результати тесту записуються в звіт, який потім компілюється і відправляється зацікавленим сторонам.

Цей звіт може бути використаний для подальшого вдосконалення реєстраційної системи, гарантуючи її надійність та ефективність.

2.4 Побудова діаграми розгортання

Діаграми розгортання в UML використовуються для візуалізації фізичної архітектури системи, включаючи обладнання, на якому працюють програмні компоненти. Вони показують конфігурацію апаратних засобів, мережеві зв'язки, а також розташування програмного забезпечення. Ці діаграми важливі для розуміння того, як компоненти системи взаємодіють та розміщуються в операційному середовищі.

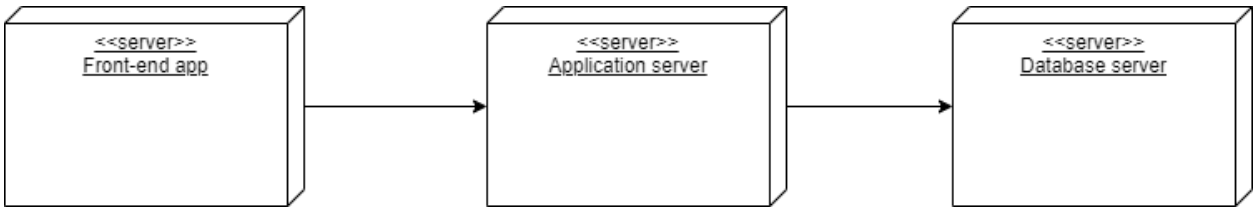


Рисунок 2.4 – Діаграма розгортання системи автоматизованого тестування вебзастосунків з використанням Selenium та PyTest

На діаграмі розгортання представляється архітектуру трьохшарового веб-додатку, де кожен шар виконує певну роль у загальній структурі системи. Перший блок, "Front-end app", відповідає за інтерфейс користувача та представлення інформації, що робить взаємодію з користувачем можливою та зручною. Другий блок, "Application server", є серцем системи, де відбувається обробка даних, бізнес-логіка та інші обчислювальні процеси. Він отримує дані від фронтенду, обробляє їх, і взаємодіє з базою даних. Третій блок, "Database server", використовується для зберігання даних, забезпечуючи надійне збереження та організацію інформації, яка вимагається додатком. Вся інформація між цими серверами передається лінійно: від фронтенду до сервера застосунків, а потім до сервера бази даних, створюючи чіткий і структурований потік даних у системі.

Вони складаються з:

1) клієнтський застосунок(Front-end app)
2) програмне середовище(application server)
3) база даних(Database server)

Саме ці вузли являють собою діаграму розгортання цього проєкту.

Висновки до розділу 2

Другий розділ дипломної роботи зосереджений на моделюванні системи автоматизованого тестування вебзастосунків з використанням Selenium і PyTest. За допомогою раніше створеної діаграми використання представлено ключові сценарії роботи системи, які покликані задовольнити основні потреби користувача. Розроблено діаграми діяльності, які ілюструють алгоритми функціонування системи. Також, за допомогою діаграм взаємодії описано механізми комунікації між клієнтом та сервером під час виконання завдань.

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Побудова діаграми класів

Діаграма класів — це візуальне представлення структури класів у системі, що показує їх атрибути, методи та взаємозв'язки між ними. Вона використовується для моделювання об'єктно-орієнтованого дизайну, допомагаючи зрозуміти логіку та архітектуру додатку. Кожен клас представлений як прямокутник, розділений на три частини: ім'я класу, атрибути та методи. Взаємозв'язки між класами, такі як асоціації, наслідування та реалізації, зображуються стрілками, що вказують на напрямок залежностей або зв'язків між класами. Діаграма класів виступає фундаментом для подальшої розробки системи засобами об’єктно-орієнтованого програмування.

Діаграми класів окремих частин застосунку показані на рисунках 3.1 та 3.2. Відповідні пояснення до цих діаграм можна знайти у таблицях 3.1 та 3.2.

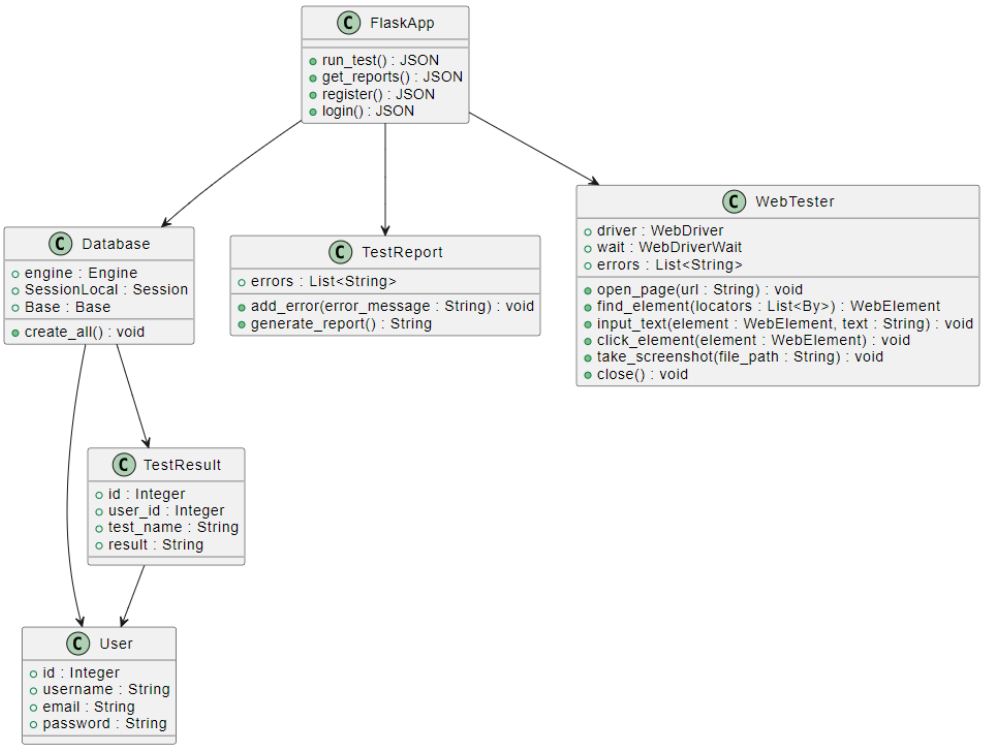


Рисунок 3.1 – Діаграма класів серверного застосунку системи

Таблиця 3.1 – Характеристика класів задіяних у розробці системи

Назва класу	Поля	Методи	Короткий опис
FlaskApp		run_test(): JSON get_reports(): JSON register(): JSON login(): JSON	Клас, який представляє Flask додаток з маршрутами для запуску тестів, отримання звітів, реєстрації та входу в систему.
WebTester	driver: WebDriver wait: WebDriverWait errors: List<String>	open_page(url: String): void find_element(locators: List<By>): WebElement input_text(element: WebElement, text: String): void click_element(element: WebElement):void take_screenshot(file_path:String): void close(): void	Клас для автоматизованого тестування вебсайтів, що використовує Selenium WebDriver для взаємодії з елементами на сторінці, вводу тексту, кліків та створення знімків екрану.
TestReport	errors: List<String>	add_error(error_message: String): void generate_report(): String	Клас для генерації звітів про результати тестування.
TestResult	id: Integer user_id: Integer test_name: String result: String		Клас, який представляє результати тестування, що зберігає інформацію про тест, результат та час виконання.

Кінець таблиці 3.1

User	id: Integer username: String email: String password: String		Клас, який представляє користувача в системі, що містить дані про ім'я користувача, електронну пошту та пароль.
Database	engine: Engine SessionLocal:Session Base: Base	create_all(): void	Клас для роботи з базою даних, який налаштовує з'єднання з базою даних і створює необхідні таблиці.

Варто зауважити, що деякі класи не містять методів та використовуються здебільшого для взаємодії з таблицями бази даних. Це пояснюється застосуванням архітектурного підходу CQRS, який розділяє відповідальності між командами запису (Command) та запитами (Query), що сприяє кращому розподілу навантаження та оптимізації роботи з даними.

3.2 Побудова діаграми компонентів

Діаграма компонентів – це тип UML діаграми, що використовується для моделювання фізичної архітектури системи. Вона ілюструє розбиття програмного забезпечення на компоненти, які можуть бути як фізичними, так і логічними одиницями. Компоненти можуть включати модулі коду, бібліотеки, веб-сервіси, бази даних та інші програмні елементи. Діаграма компонентів представлена на рис. 3.2

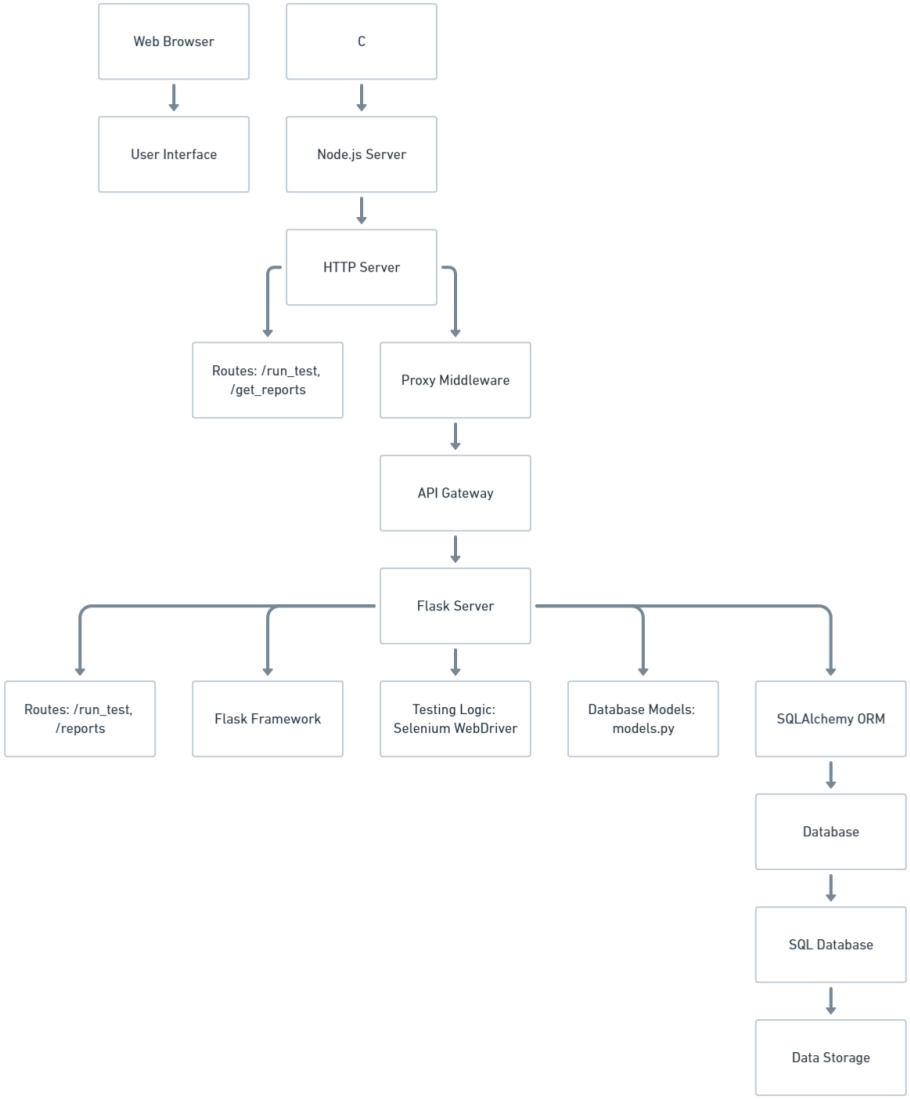


Рисунок 3.2 – Діаграма компонентів серверного застосунку системи

Варто зауважити, що деякі компоненти складаються з менших підкомпонентів та взаємодіють між собою через інтерфейси

3.3 Побудова діаграми пакетів

Діаграма пакетів — це структурний діаграмний інструмент в UML, який використовується для демонстрації організації та залежностей між різними пакетами в системі. Пакети можуть містити інші пакети, класи та інші елементи моделі, що дозволяє групувати їх для кращої організації та управління. Діаграми

пакетів особливо корисні для великих проектів, оскільки вони допомагають зрозуміти високорівневу структуру системи та взаємозв'язки між її різними частинами. Вони також дозволяють виявляти та керувати залежностями, зменшувати складність та підвищувати модульність.

Діаграма пакетів застосунку автоматизованої системи тестування складається з 4 елементів та представлена на рис. 3.3.

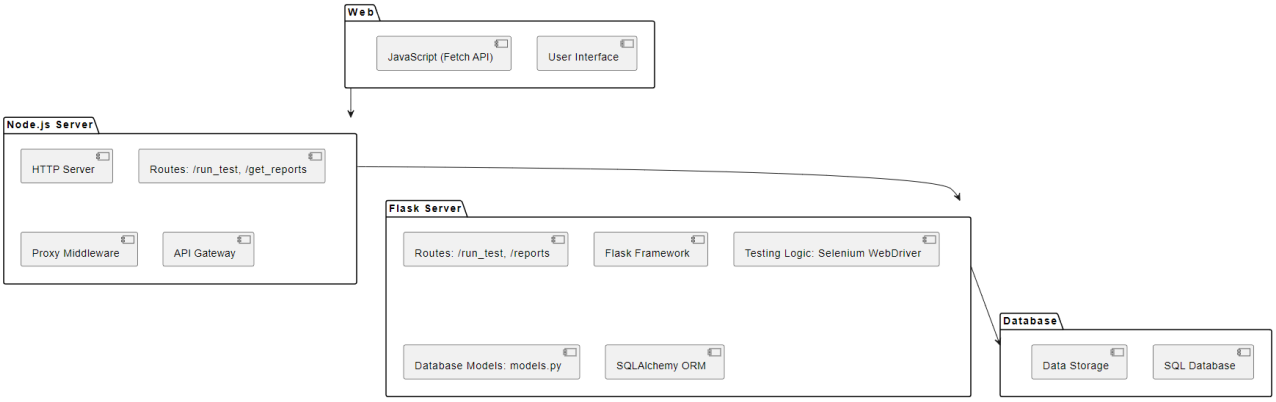


Рисунок 3.3 – Діаграма пакетів

Зазвичай діаграми пакетів використовуються на етапі проектування системи, щоб допомогти архітекторам та розробникам спланувати ефективну та зрозумілу структуру проекту.

3.4 Побудова діаграм станів та переходів

Діаграми станів і переходів — це поведінковий діаграмний інструмент в UML, який використовується для моделювання поведінки об'єктів в системі через їх стани та переходи між ними. Діаграми станів відображають різні стани об'єкта протягом його життєвого циклу, а також події, що спричиняють зміну станів. Вони допомагають зрозуміти, як об'єкти реагують на зовнішні та внутрішні події, і забезпечують наочне представлення логіки роботи системи. Цей тип діаграм сприяють покращенню проектування, тестування та документування системи.

Для майбутньої системи розроблено три діаграми станів і переходів:

1) Загальна діаграма, що демонструє можливості головної сторінки (рис. 3.4);

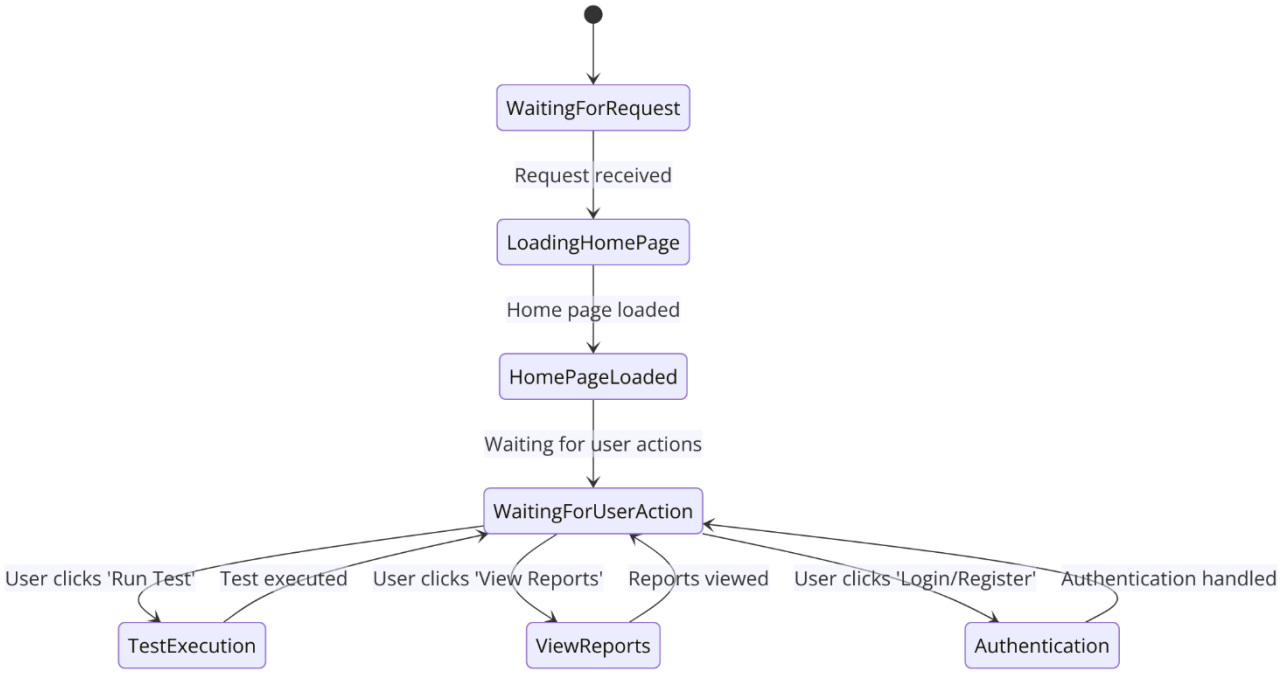


Рисунок 3.4 – Загальна діаграма станів і переходів

2) Запит на виконання тесту та успішне завершення (рис. 3.5);

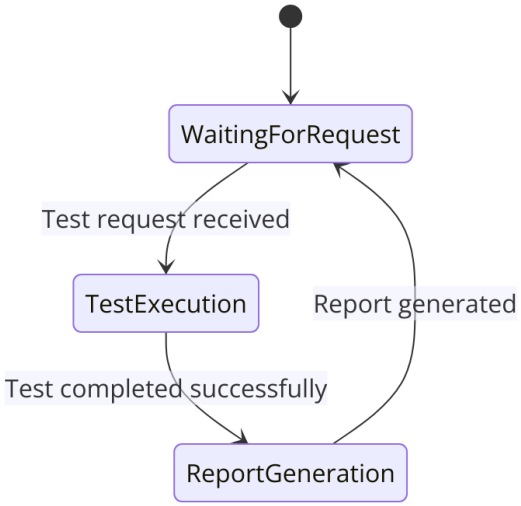


Рисунок 3.5 – Діаграма станів та переходів виконання тесту

3) Перегляд результатів тестування (рис. 3.6);

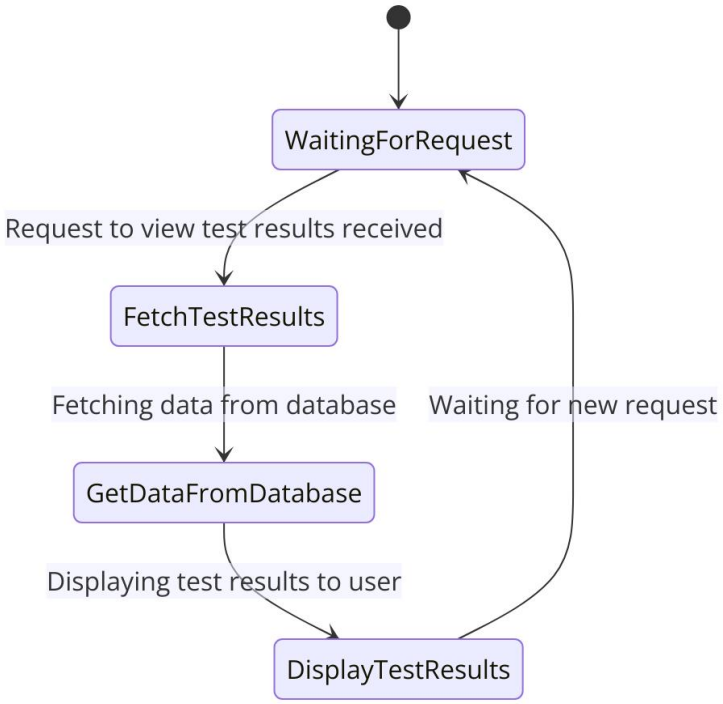


Рисунок 3.6 – Діаграма станів і переходів результатів тестування

Діаграми станів і переходів особливо корисні для моделювання систем з комплексною поведінкою, таких як автоматизовані системи, керування процесами, та взаємодії користувача

3.5 Побудова моделі бази даних

Перед початком розробки системи автоматизованого тестування вебзастосунків, важливим етапом є побудова діаграми моделі бази даних. Ця діаграма дозволяє візуально представити структуру бази даних, визначити основні сутності, їх атрибути та зв'язки між ними. Правильна побудова діаграми допомагає забезпечити логічну цілісність даних, оптимізувати процес зберігання та доступу до інформації, а також спрощує подальшу реалізацію функціоналу системи. Нижче наведено діаграму моделі бази даних, яка складається з двох основних таблиць: User та TestResult(рис. 3.7).

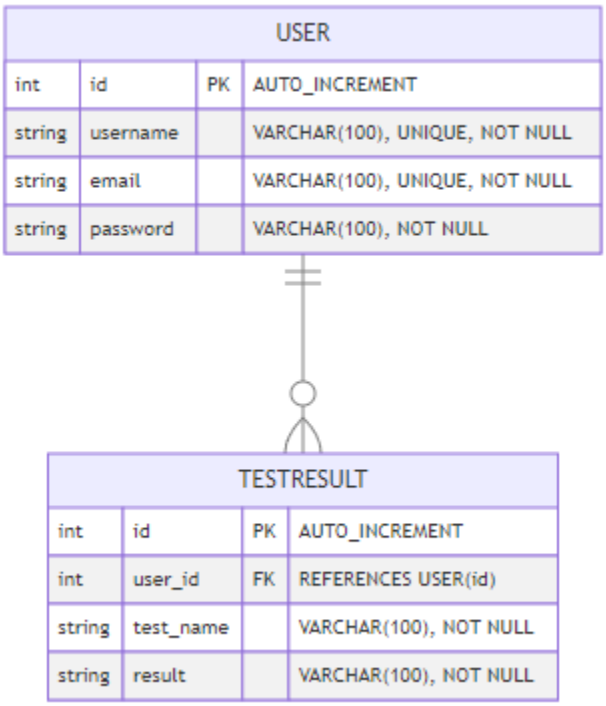


Рисунок 3.7 – Модель бази даних

На діаграмі зображено дві таблиці бази даних: USER та TESTRESULT. Таблиця USER містить інформацію про користувачів системи, включаючи унікальний ідентифікатор (id), ім'я користувача (username), електронну пошту (email) та пароль (password). Таблиця TESTRESULT зберігає результати тестувань і містить поля: унікальний ідентифікатор результату (id), зовнішній ключ користувача (user_id), назву тесту (test_name) та результат тесту (result). Зв'язок між таблицями реалізовано через зовнішній ключ user_id, який посилається на поле id у таблиці USER. Це дозволяє зв'язувати кожен результат тесту з відповідним користувачем, що виконав тест.

3.6 Вибір стеку технологій

Back-end

В умовах сучасних реалій зростає популярність використання Python у створенні бекенд систем. На рисунку 3.8 зображено статистику використання мов

програмування на платформі Stack Overflow, починаючи з 2008 року. Виходячи з отриманих даних, можна зробити висновок, що починаючи з 2017 року Python почав лідирувати серед користувачів. Це не дивно, оскільки Python відомий своєю простотою, легкістю в навчанні та великою кількістю бібліотек, що спрощує розробку додатків.

У порівнянні з іншими мовами, саме Python є найбільш релевантним для розробки бекенд-системи для автоматизованого тестування вебзастосунків, оскільки він забезпечує зручну роботу з базами даних, має потужні фреймворки для розробки вебдодатків, такі як Flask і Django, та підтримує інтеграцію з інструментами тестування, такими як Selenium[11]. Крім того, Python має велику спільноту розробників, які постійно оновлюють та вдосконалюють інструменти і бібліотеки, що робить його ідеальним вибором для нашого проекту.

Важливим фактором також є гнучкість Python[12] у побудові масштабованих систем. Це дозволяє розробникам писати менш об'ємний і більш зрозумілий код для роботи з даними.

Завдяки широкому вибору фреймворків і бібліотек, Python дозволяє швидко реалізовувати функціонал, необхідний для нашої системи, і забезпечує високу продуктивність та стабільність роботи.

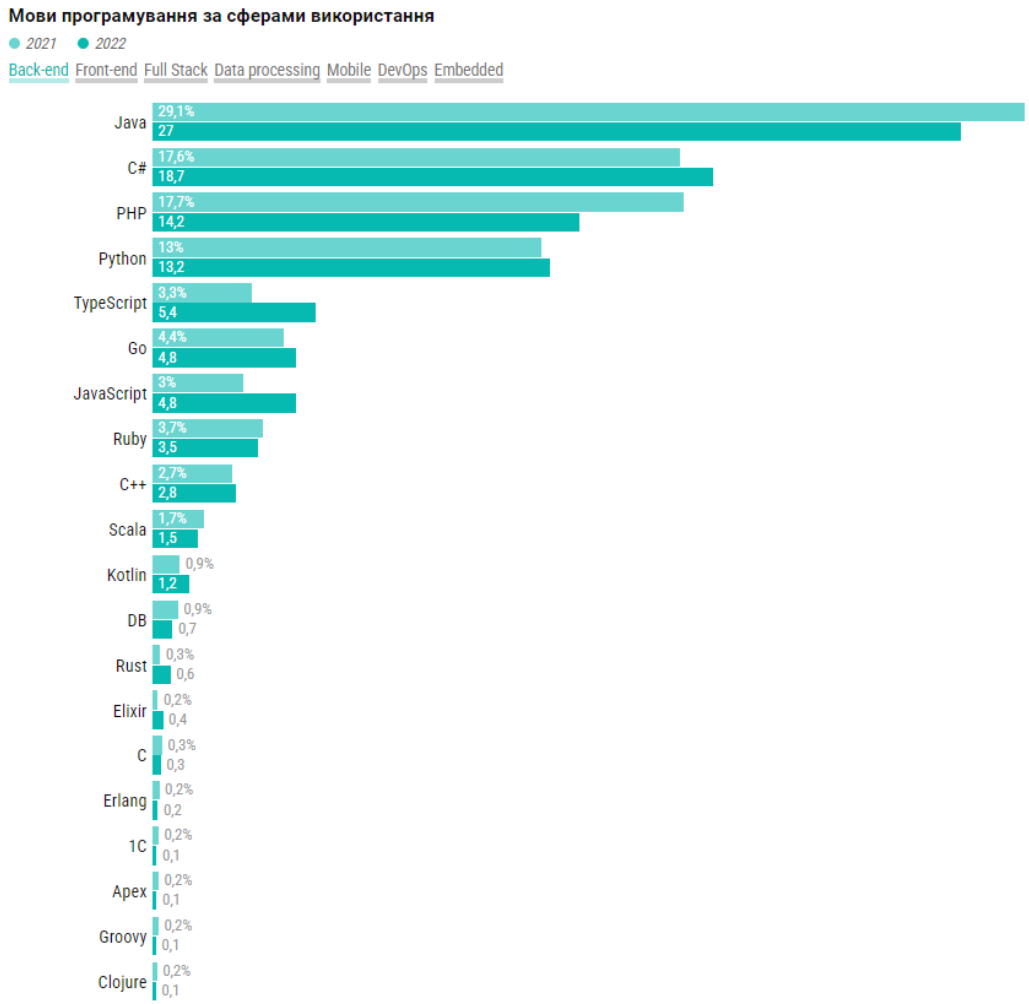


Рисунок 3.8 – Графік популярності мов програмування для Back-end

Front-end

У сучасних реаліях фронтенд-розробка відіграє ключову роль у створенні веб-застосунків, забезпечуючи користувачам зручний та інтуїтивний інтерфейс. Фронтенд включає в себе все, що користувач бачить та з чим взаємодіє на веб-сторінці.

JavaScript-фреймворки[13], такі як React, Angular та Vue.js, значно спрощують процес розробки складних інтерфейсів. React, зокрема, здобув широку популярність завдяки своїй ефективності в управлінні станом додатка та швидкій реакції на зміни в даних. Застосування компонентного підходу в React

дозволяє розробникам створювати багаторазові та легко підтримувані компоненти інтерфейсу(рис. 3.9).

Фронтенд-розробка також включає в себе інструменти для автоматизації та управління проектами, такі як Webpack, Babel та npm, що дозволяють розробникам ефективно працювати з сучасними стандартами та технологіями.

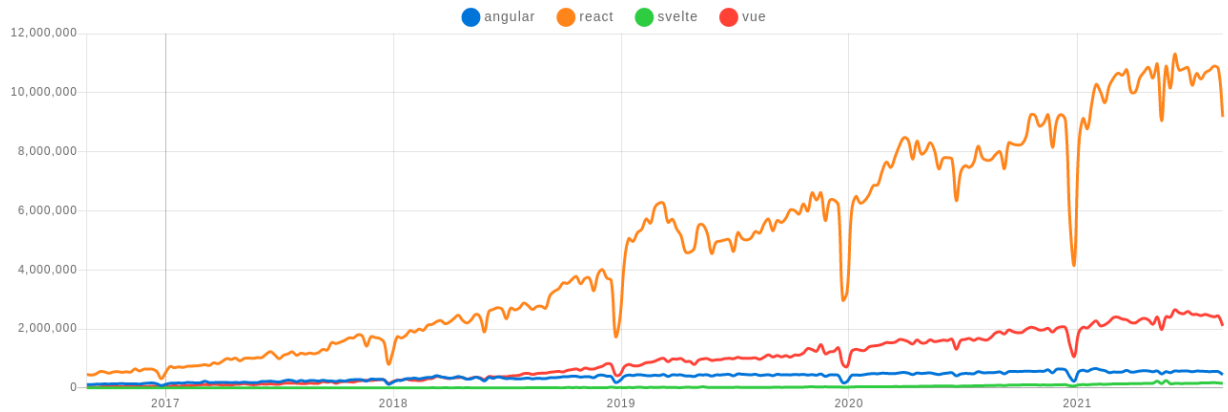


Рисунок 3.9 – Графік популярності фреймворків для JavaScript

Висновки до розділу 3

Третій розділ кваліфікаційної роботи зосереджений на проєктуванні системи. За допомогою раніше створених діаграм представлено ключові сценарії роботи системи. Також проведено аналіз стеку технологій для визначення найбільш оптимальних кандидатів для використання в розробці.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Огляд UI

Проєктування користувацького інтерфейсу здійснювалось за допомогою програмного забезпечення «Figma». Обраний шрифт – Mulish. Легкість сприйняття великих об’ємів тексту, написаних шрифтом такого типу, задовільна і не буде слугувати причиною перенапруження очей користувачів. Будь-який користувач може зайти на головну сторінку і отримати доступ до її повного функціоналу, а саме – запуск тестування веб-сайтів. Як можна побачити з переліку UI-елементів, наявні налаштування темної теми, можливість реєстрації та перегляду інформації про сайт у розділі About. Також, можна виконати запуск тестування шляхом натискання на відповідні кнопки сайтів на головній сторінці (див. рис. 4.1).

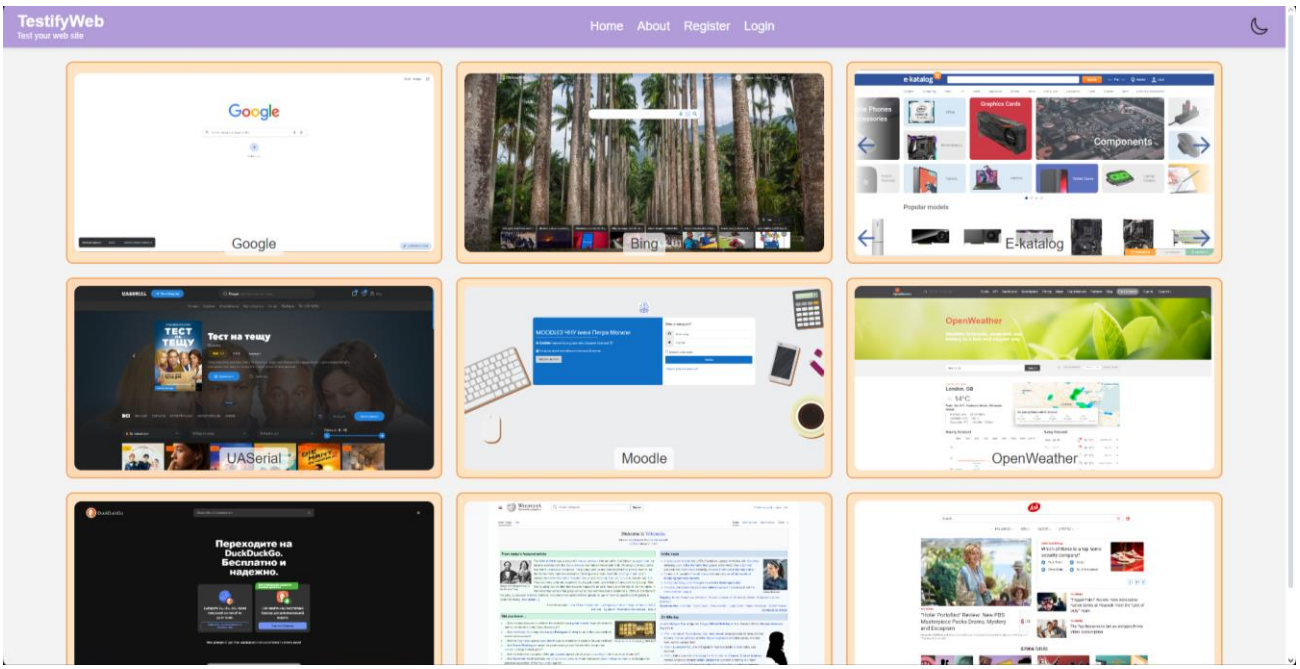


Рисунок 4.1 – Головна сторінка вебзастосунку

Також треба розглянути сторінку реєстрації користувача(див. рис. 4.2). На цій сторінці відбувається реєстрація користувача шляхом введення в необхідні

поля, певних даних. Треба зауважити що поле з назвою Email потребує обов'язкового використання значка собачка, який є необхідним параметром для створення пошти.

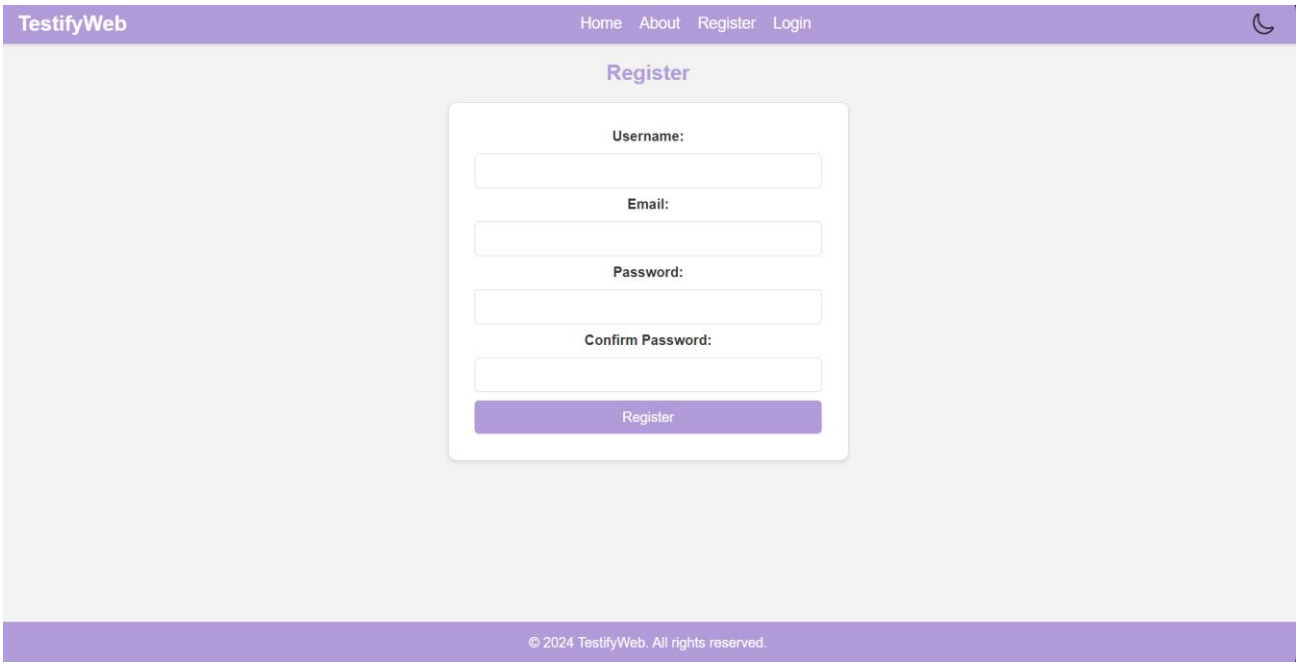


Рисунок 4.2 – Сторінка реєстрації користувача

Після успішної реєстрації можна увійти в аккаунт, за допомогою відповідної форми, яка називається Login(див. рис. 4.3).

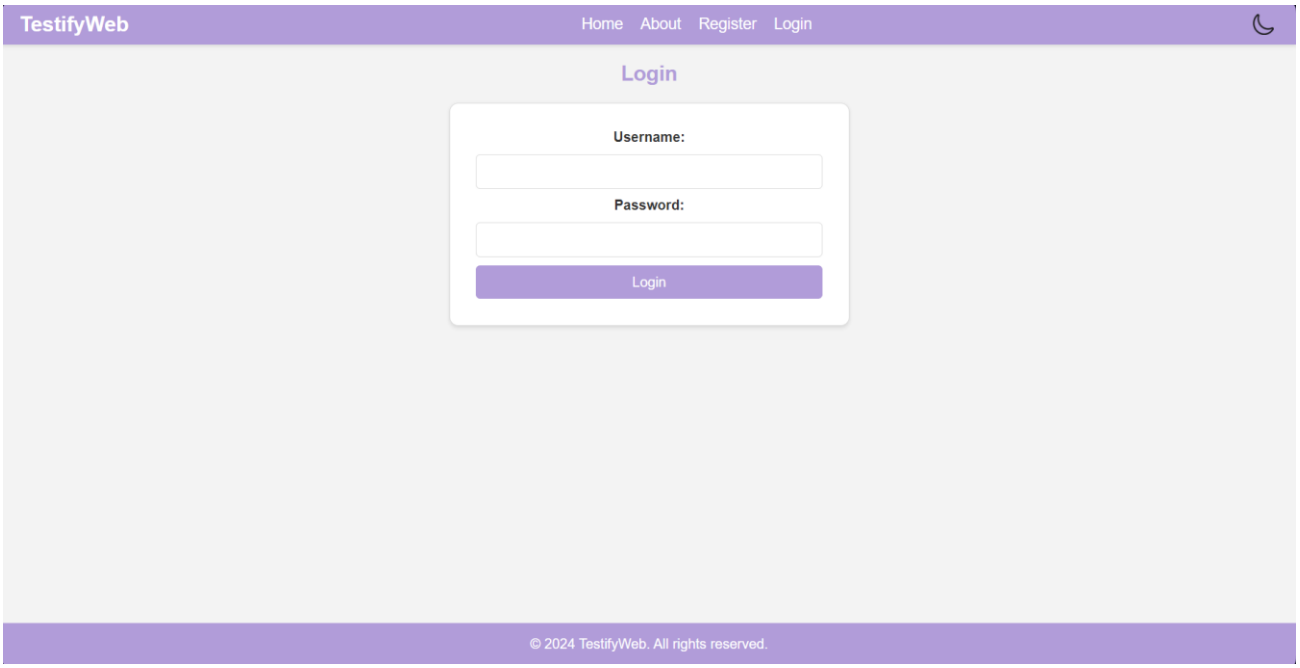


Рисунок 4.3 – Сторінка входу в аккаунт

Для того щоб дізнатися певну інформацію про даний вебзастосунок було створено відповідну сторінку з назвою About (див. рис. 4.4).

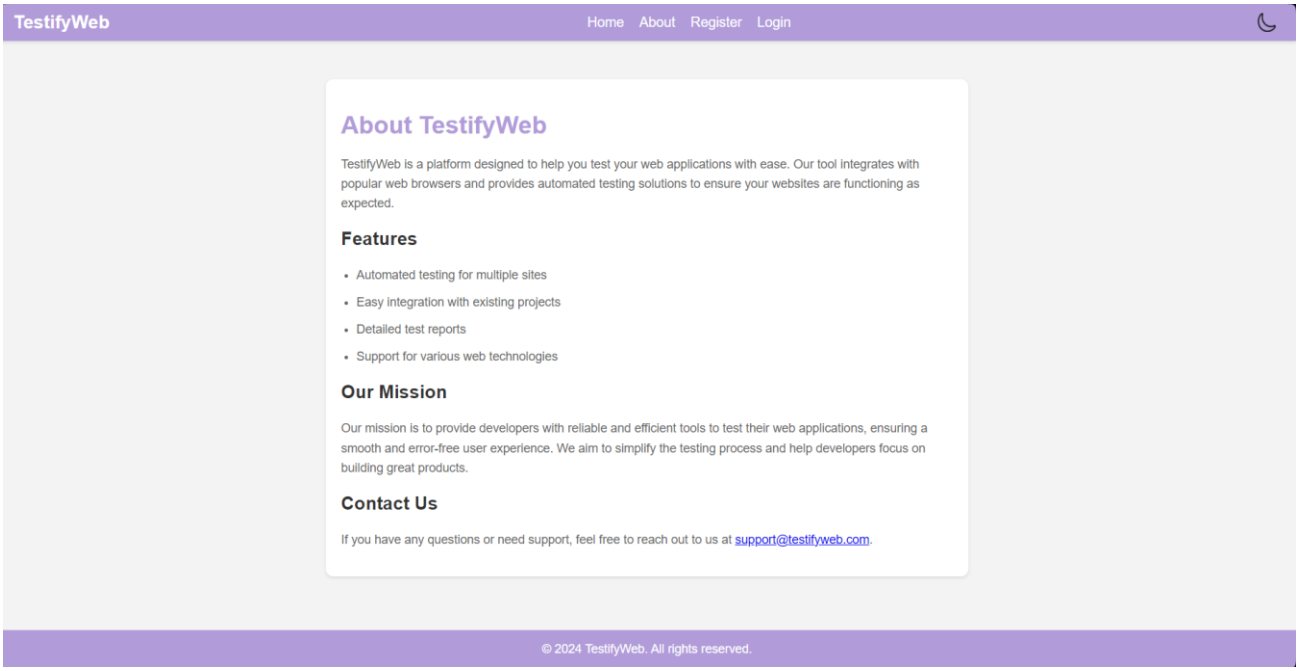


Рисунок 4.4 – Сторінка з інформацією про застосунок

Було додано темну тему для покращення користувацького досвіду на сайті. Темна тема допомагає зменшити навантаження на очі користувачів під час перегляду сайту у темних умовах, забезпечуючи комфортний перегляд контенту. Користувачі можуть легко переключатися між світлою та темною темами, зберігаючи свої налаштування при переході між сторінками. (див. рис. 4.5).

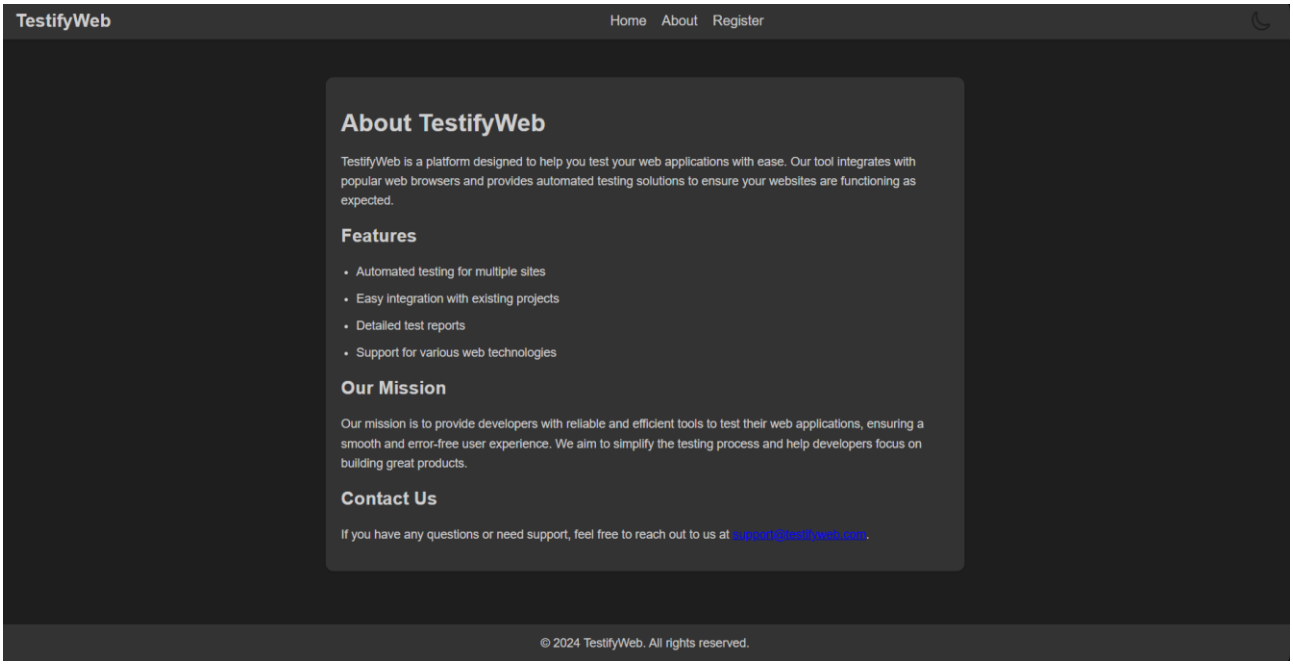


Рисунок 4.5 – Вебзастосунок з використанням темної теми

Таким чином було розглянуто основні елементи користувацького інтерфейсу вебзастосунку.

4.2 Розробка функціоналу автоматизованої системи тестування

4.2.1 Структура front-end застосунку

Перш ніж почати огляду програмного коду варто приділити увагу загальному підходу до структуризації проєкту. Таким чином спрощується майбутній перегляд кодової бази та розуміння принципів, використаних під час

розробки.

Фронтенд частина реалізована за допомогою HTML, CSS та JavaScript, що забезпечує динамічну взаємодію з користувачем і налаштування темного режиму для зручного перегляду в темних умовах. Бекенд частина, побудована на Flask та Node.js[14], відповідає за виконання тестів вебсайтів та обробку запитів[15] від користувачів. Така структура проєкту дозволяє легко розширювати функціонал та підтримувати кодову базу.

Загальна архітектура проєкту розділена на кілька ключових модулів, кожен з яких виконує свою специфічну функцію. Це включає модулі для обробки користувацьких запитів, виконання тестів, генерації звітів та управління користувачами. Кожен модуль має чітко визначені межі відповідальності, що сприяє зниженню зв'язності між компонентами та підвищує масштабованість системи. Ретельне планування та документування структури проєкту також включає створення докладної документації API[16], що дозволяє іншим розробникам легко інтегруватися з системою або розширювати її функціонал. Важливо, щоб код був не тільки функціональним, але й добре організованим та задокументованим, оскільки це значно спрощує подальше обслуговування та розвиток проєкту. На рисунку 4.6 представлено структуру проєкту "TestifyWeb", який включає фронтенд частину. (див. рис. 4.6).

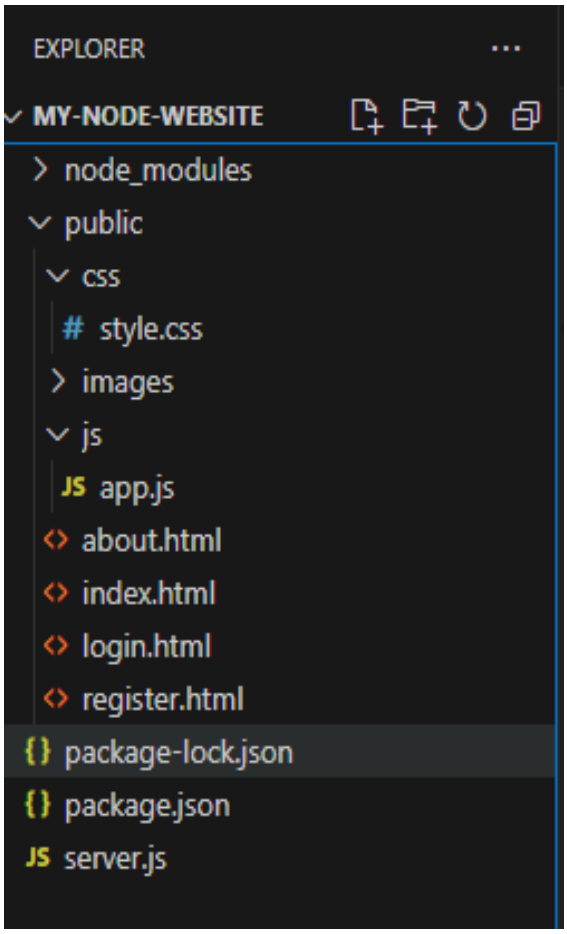


Рисунок 4.6 – Структура front-end проєкту

Кожна з директорій в «public» має наступне значення:

1) node_modules - містить всі залежності та бібліотеки, встановлені за допомогою npm (Node Package Manager). Ці файли автоматично генеруються та управляються npm і не повинні змінюватися вручну;

2) public - основна директорія, що містить ресурси, доступні з клієнтської сторони;

3) css - містить файли стилів, які визначають зовнішній вигляд сторінок. Наприклад, style.css визначає стилі для всіх HTML-елементів на сайті;

4) images - містить зображення, які використовуються на вебсторінках. Це можуть бути логотипи, іконки та інші графічні ресурси;

5) app.js - відповідає за обробку подій та взаємодію з сервером[17];

6) about.html - HTML файл, що представляє сторінку "About" (Про нас). Ця сторінка містить інформацію про проект, його цілі, інші деталі;

7) index.html - HTML файл, що представляє головну сторінку вебсайту. Вона містить основний контент та навігаційні елементи, які користувачі бачать при відвідуванні сайту;

8) register.html - HTML файл, що представляє сторінку реєстрації. Ця сторінка містить форму для реєстрації нових користувачів;

9) login,html - HTML файл, що представляє сторінку входу в аккаунт. Ця сторінка містить форму входу в аккаунт користувачів;

10) package.json - файл конфігурації, що містить інформацію про ваш проект та його залежності. Він використовується npm для управління пакетами та сценаріями;

11) package-lock.json - файл, який автоматично генерується npm для фіксації точних версій встановлених залежностей. Це забезпечує відтворюваність середовища розробки;

12) server.js - файл серверної частини, що використовується для налаштування та запуску локального сервера для розробки. Він може обробляти запити від клієнтів та повертати відповідні ресурси;

Ця структура забезпечує чітку організацію файлів і полегшує підтримку та розширення проєкту.

Особливої уваги з цього списку вимагають два файли а саме server.js та , app.js. Файл server.js виконує функцію основного серверного файлу для Node.js застосунку, який використовує Express.js для обробки запитів та проксі-сервер для перенаправлення API запитів до Flask сервера(див. рис. 4.7).

```
12  app.all('/api/*', (req, res) => {
13      apiProxy.web(req, res, { target: FLASK_SERVER });
14  });
```

Рисунок 4.7 – Перенаправлення API запитів

Цей блок коду перенаправляє всі запити, що починаються з /api/, на Flask сервер. Використовується проксі-сервер для перенаправлення запитів.

Також варто згадати раніше описаний app.js бо він є основним JavaScript файлом для клієнтської частини вашого вебзастосунку. Він відповідає за обробку подій на сторінці, взаємодію з сервером через HTTP-запити та управління темами оформлення. Файл app.js містить обробку запитів на запуск тестування для наданих сайтів(див. рис. 4.8).

```
const buttons = document.querySelectorAll('.site');
buttons.forEach(button => {
    button.addEventListener('click', function() {
        const testName = button.getAttribute('data-test-name');
        fetch('http://localhost:5000/run_test', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ test_name: testName })
        })
        .then(response => response.json())
        .then(data => {
            alert(data.result || 'Test completed successfully');
        })
        .catch(error => console.error('Error:', error));
    });
});
```

Рисунок 4.8 – Обробка запитів на запуск тестування

Тут здійснюється вибір всіх кнопок з класом .site та додавання до них обробників подій click, що відправляють POST-запит на сервер для запуску тестів. Окрім цього присутні форми реєстрації та логіну(див. рис. 4.9).

```
const registerForm = document.getElementById('registerForm');
if (registerForm) {
    registerForm.addEventListener('submit', function(event) {
        event.preventDefault();
        const formData = new FormData(event.target);
        const data = Object.fromEntries(formData.entries());

        fetch('http://localhost:5000/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        })
        .then(response => response.json())
        .then(data => {
            alert(data.message || 'Registration successful');
        })
        .catch(error => console.error('Error:', error));
    });
}
```

Рисунок 4.9 – Обробка форми реєстрації

Цей блок коду відповідає за обробку форми реєстрації. Коли користувач натискає кнопку реєстрації, стандартна поведінка форми перехоплюється, і дані форми збираються за допомогою об'єкта FormData. Дані потім перетворюються у формат JSON і відправляються на сервер за допомогою методу fetch. Відповідь від сервера обробляється для відображення повідомлення про успіх або помилку реєстрації[18]. Якщо запит успішний, на сторінці відображається відповідне повідомлення, якщо ні показується повідомлення про помилку. (див. рис. 4.10).

```
const loginForm = document.getElementById('loginForm');
if (loginForm) {
    loginForm.addEventListener('submit', function(event) {
        event.preventDefault();

        const formData = new FormData(event.target);
        const data = Object.fromEntries(formData.entries());

        fetch('http://localhost:5000/login', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            const resultDiv = document.getElementById('login-result');
            if (resultDiv) {
                resultDiv.innerText = data.message;
            }
            alert(data.message || 'Login successful');
        })
        .catch(error => console.error('Error:', error));
    });
}
```

Рисунок 4.10 – Обробка форми логіну

Цей блок коду відповідає за обробку форми логіну. Коли користувач натискає кнопку логіну, стандартна поведінка форми перехоплюється, і дані форми збираються за допомогою об'єкта FormData. Дані потім перетворюються у формат JSON і відправляються на сервер за допомогою методу fetch[19]. Відповідь від сервера обробляється для відображення повідомлення про успіх або помилку логіну. Якщо запит успішний, на сторінці відображається відповідне повідомлення, якщо ні - показується повідомлення про помилку.

За для покращення умов використання вебзастосунком також було створено можливість перемикання на темну тему(див. рис. 4.11).

```
const themeToggleButton = document.getElementById('theme-toggle');
const body = document.body;
const darkModeClass = 'dark-mode';

if (localStorage.getItem('theme') === darkModeClass) {
    body.classList.add(darkModeClass);
    applyDarkModeToElements();
}

if (themeToggleButton) {
    themeToggleButton.addEventListener('click', () => {
        body.classList.toggle(darkModeClass);

        if (body.classList.contains(darkModeClass)) {
            localStorage.setItem('theme', darkModeClass);
            applyDarkModeToElements();
        } else {
            localStorage.removeItem('theme');
            removeDarkModeFromElements();
        }
    });
}
```

Рисунок 4.11 – Перемикання теми оформлення

Цей код забезпечує функціонал перемикання теми оформлення (світлої та темної). Він перевіряє збережену тему в localStorage, застосовує відповідні класи CSS та зберігає вибір користувача.

4.2.2 Структура back-end застосунку

Окрім front-end частини, застосунок також містить серверну частину. У попередньому розділі вже розглядалась тема пакетів, які формують систему, але це лише верхівка айсберга. Серверна логіка відповідає за обробку запитів від клієнтів, взаємодію з базами даних, а також за забезпечення безпеки та стабільності функціонування застосунку.

Однією з ключових складових серверної частини є архітектура системи, яка визначає, як компоненти взаємодіють між собою. Вибір архітектурного стилю,

будь то монолітна архітектура, мікросервіси або серверлесс підходи, впливає на масштабованість та адаптивність вебзастосунку. Розуміння переваг і недоліків кожного підходу допомагає зробити усвідомлений вибір, який відповідає вимогам проекту.

Крім того, не менш важливим аспектом є використання фреймворків і бібліотек, які спрощують розробку і підтримку серверного коду. Наприклад, використання Flask або Django у Python дозволяє значно прискорити процес розробки завдяки вбудованим функціям для обробки запитів, управління сесіями та інтеграції з базами даних(див. рис. 4.12).

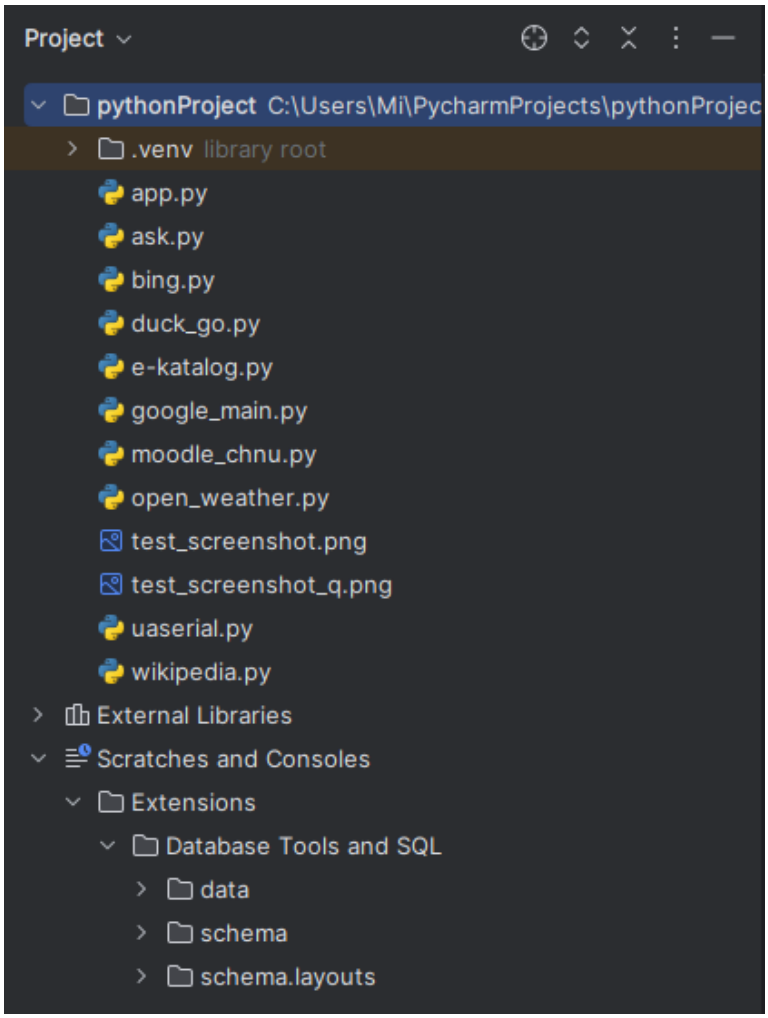


Рисунок 4.12 – Структура back-end проєкту

Проект TestifyWeb має детальну структуру back-end, побудовану на основі Flask, мікрофреймворку для Python, який дозволяє створювати вебдодатки та API швидко і ефективно. Основний файл проекту - app.py, містить головний код для запуску Flask сервера, визначення маршрутів (routes) та обробки запитів. Завдяки Flask, реалізовано серверну логіку, яка відповідає за обробку запитів від клієнтів, виконання необхідних операцій та повернення відповідей. У проекті також містяться файли для тестування різних вебсайтів[20], таких як bing.py, e-katalog.py, google_main.py та uaserial.py. Ці файли дозволяють виконувати автоматизовані тести для відповідних сайтів забезпечуючи їх стабільність та правильність роботи. Проєкт складається з наступних файлів.venv - Це віртуальне середовище Python, яке використовується для ізоляції залежностей проєкту. Воно містить усі необхідні бібліотеки та пакети для вашого проекту. Кожен з файлів в директорії

1) app.py - Основний файл застосунку. Зазвичай містить основну логіку програми, налаштування маршрутизації та конфігурацію сервера.

2) ask.py - Імовірно, скрипт, пов’язаний із певним функціоналом вашого застосунку. Може бути пов'язаний із обробкою запитів користувачів.

3) bing.py - Скрипт для тестування або взаємодії із пошуковою системою Bing.

4) duck_go.py - Скрипт для тестування або взаємодії із пошуковою системою DuckDuckGo.

5) e-katalog.py - Скрипт для тестування або взаємодії із сайтом e-katalog.

6) google_main.py - Скрипт для тестування або взаємодії із пошуковою системою Google.

7) moodle_chnu.py - Скрипт для тестування або взаємодії із системою

Moodle Чернівецького національного університету.

8) open_weather.py - Скрипт для тестування або взаємодії із сервісом OpenWeather.

9) test_screenshot.png - Скриншот, ймовірно, зроблений для перевірки роботи тестів або інтерфейсу вашого застосунку.

10) test_screenshot_q.png - Ще один скриншот, зроблений для перевірки.

11) uaserial.py - Скрипт для тестування або взаємодії із сайтом UAserial.

12) wikipedia.py - Скрипт для тестування або взаємодії із сайтом Wikipedia.

Окремо треба приділити увагу файлу app.py. Він містить код для серверної частини веб-застосунку, написаного на Flask. Він забезпечує функціональність для взаємодії з базою даних, обробки запитів від користувачів, а також запуску тестів. Перш за все після встановлення всіх необхідних для роботи бібліотек, треба зробити налаштування для подальшої роботи(див. рис. 4.13).

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:12345@localhost/dbTestify'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
CORS(app, resources={r"/*": {"origins": "http://localhost:3001"}})
```

Рисунок 4.13 – Налаштування Flask додатку

Тут створюється екземпляр Flask додатку, налаштовується підключення до бази даних MySQL, вимикається відстеження змін (щоб зменшити навантаження), а також додається підтримка CORS для дозволу запитів з іншого домену.

Також в файлі присутнє підключення до бази даних і створення таблиць в

які записуються дані з вебзастосунку(див. рис. 4.14).

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)

class TestResult(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    test_name = db.Column(db.String(50), nullable=False)
    result = db.Column(db.Text, nullable=False)

    def __repr__(self):
        return f'<TestResult {self.test_name}>'

with app.app_context():
    db.create_all()
```

Рисунок 4.14 – Оголошення моделей бази даних

Визначаються дві моделі для бази даних: User (користувач) і TestResult (результат тесту). Кожна модель має свої поля з типами даних і обмеженнями а також виконується створення таблиць у базі даних. Таблиця User має в собі наступний вміст(див. рис. 4.15).

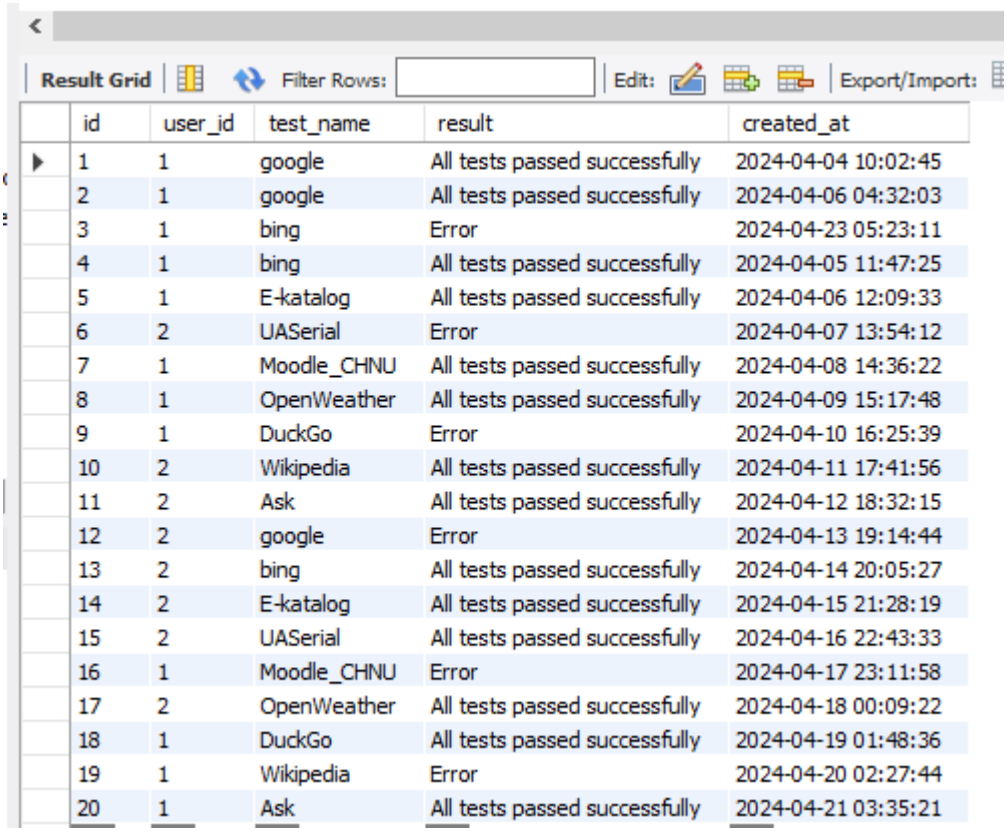
Result Grid | Filter Rows: | Edit:

id	username	email	password
1	123	FollWatetr@ukr.net	password123
2	user2	kolambys002@gmail.com	pssw
3	user3	kolambys004@gmail.com	pssw123
NULL	NULL	NULL	NULL

Рисунок 4.15 – Вміст таблиці User

Також треба подивитись вміст вже заповненої таблиці TestResult яка

містить в собі данні тестування(див. рис. 4.16).



Result Grid | Filter Rows: | Edit: | Export/Import:

id	user_id	test_name	result	created_at
1	1	google	All tests passed successfully	2024-04-04 10:02:45
2	1	google	All tests passed successfully	2024-04-06 04:32:03
3	1	bing	Error	2024-04-23 05:23:11
4	1	bing	All tests passed successfully	2024-04-05 11:47:25
5	1	E-katalog	All tests passed successfully	2024-04-06 12:09:33
6	2	UASerial	Error	2024-04-07 13:54:12
7	1	Moodle_CHNU	All tests passed successfully	2024-04-08 14:36:22
8	1	OpenWeather	All tests passed successfully	2024-04-09 15:17:48
9	1	DuckGo	Error	2024-04-10 16:25:39
10	2	Wikipedia	All tests passed successfully	2024-04-11 17:41:56
11	2	Ask	All tests passed successfully	2024-04-12 18:32:15
12	2	google	Error	2024-04-13 19:14:44
13	2	bing	All tests passed successfully	2024-04-14 20:05:27
14	2	E-katalog	All tests passed successfully	2024-04-15 21:28:19
15	2	UASerial	All tests passed successfully	2024-04-16 22:43:33
16	1	Moodle_CHNU	Error	2024-04-17 23:11:58
17	2	OpenWeather	All tests passed successfully	2024-04-18 00:09:22
18	1	DuckGo	All tests passed successfully	2024-04-19 01:48:36
19	1	Wikipedia	Error	2024-04-20 02:27:44
20	1	Ask	All tests passed successfully	2024-04-21 03:35:21

Рисунок 4.16 – Вміст таблиці TestResult

Окрім бази даних файл містить також функції з реєстрацією та логіном.

«def register» обробляє як GET, так і POST запити для сторінки реєстрації. При GET-запиті вона просто повертає шаблон register.html, який містить форму реєстрації. При POST-запиті вона обробляє введені дані, перевіряє їх та зберігає нового користувача в базу даних(див. рис. 4.17).

```python
@app.route( rule: '/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirm_password']

        if password != confirm_password:
            return jsonify({"message": "Passwords do not match"}), 400

        new_user = User(username=username, email=email, password=password)
        db.session.add(new_user)
        db.session.commit()

        return jsonify({"message": "Registration successful"}), 200

    return render_template('register.html')
```

Рисунок 4.17 – Функція реєстрації користувача

Функція реєстрації (register) обробляє запити на сторінку реєстрації. Вона підтримує два методи: GET і POST. Функція перевіряє, чи співпадають пароль і підтвердження пароля. Якщо паролі не співпадають, повертається повідомлення про помилку. Якщо ж паролі співпадають, створюється новий екземпляр користувача з отриманими даними, додається до сесії бази даних, і зберігається в базі даних. Після успішного збереження нового користувача повертається повідомлення про успішну реєстрацію(див. рис. 4.18).

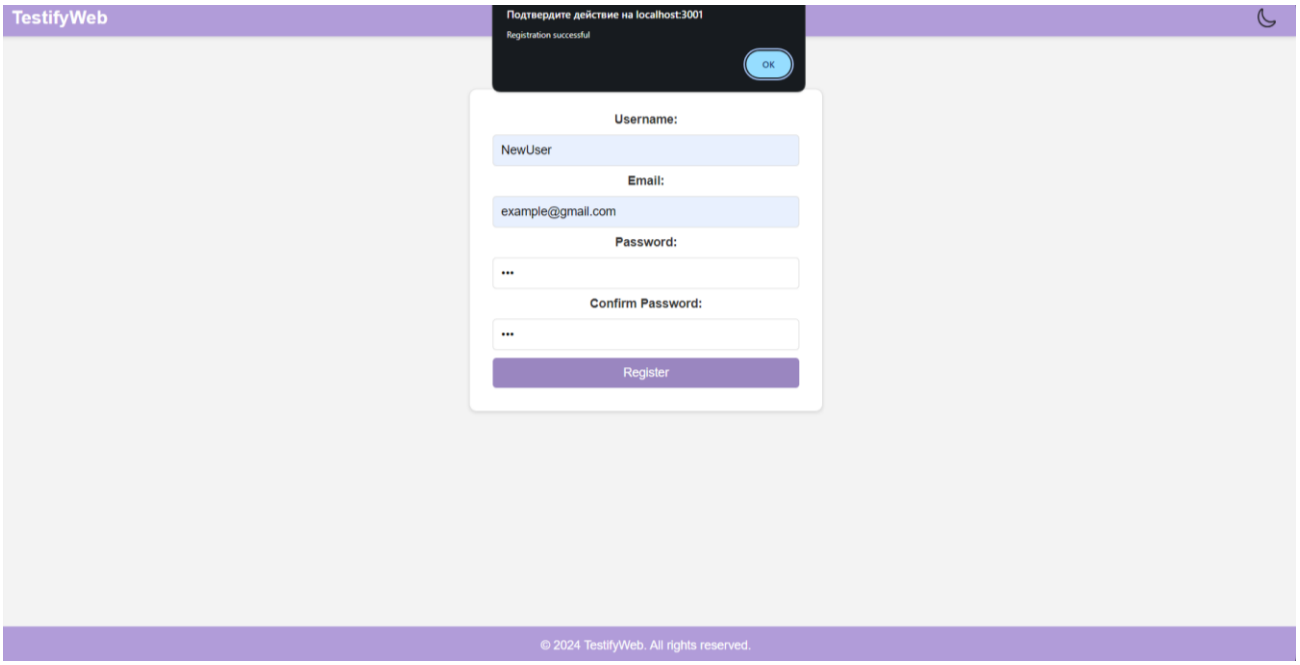


Рисунок 4.18 – Функція реєстрації користувача

Маршрут для входу користувача. Приймає POST-запити з JSON-даними. Перевіряє правильність введених даних і повертає відповідь. (див. рис. 4.19).

```
@app.route( rule: '/login', methods=['POST'])
def login():
    try:
        username = request.json['username']
        password = request.json['password']

        user = User.query.filter_by(username=username, password=password).first()

        if user:
            return jsonify({"message": "Login successful"}), 200
        else:
            return jsonify({"message": "Invalid username or password"}), 401
    except Exception as e:
        app.logger.error(f"Error occurred: {e}")
        return jsonify({"message": str(e)}), 500
```

Рисунок 4.19 – Функція логіну користувача

Функція логіну (login) обробляє POST-запити на сторінку логіну. Вона отримує JSON-дані із запиту, зокрема ім'я користувача та пароль. Потім виконується запит до бази даних для пошуку користувача з вказаними ім'ям і паролем. Якщо користувача з такими даними не знайдено, повертається повідомлення про неправильне ім'я користувача або пароль. У випадку виникнення будь-якої помилки під час обробки запиту, функція повертає повідомлення з текстом помилки. Якщо такий користувач знайдений, повертається повідомлення про успішний вхід(див. рис. 4.20).

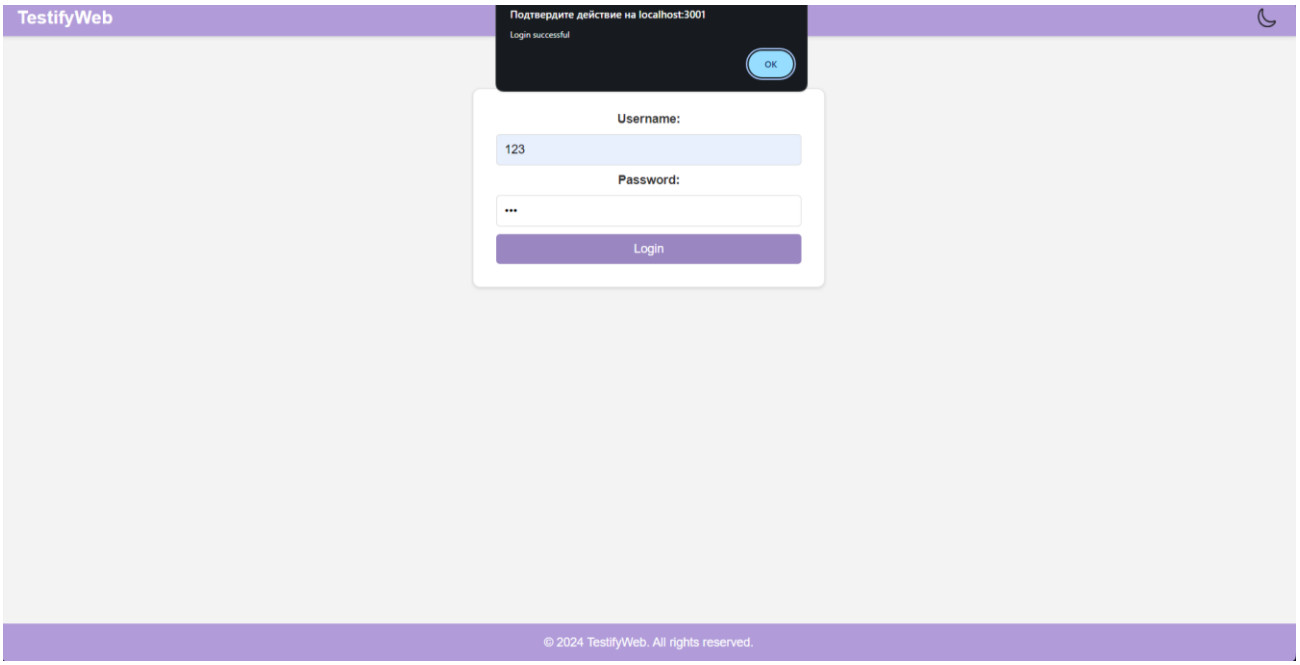


Рисунок 4.20 – Функція реєстрації користувача

Також цей файл містить можливість запуску тестувань(див. рис. 4.21).

```python
@app.route( rule: '/run_test', methods=['POST'])
def run_test():
    data = request.json
    test_name = data.get('test_name')
    result = ""

    if test_name == 'google':
        result = subprocess.run( args: ['python', 'google_main.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'bing':
        result = subprocess.run( args: ['python', 'bing.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'e-katalog':
        result = subprocess.run( args: ['python', 'e-katalog.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'uaserial':
        result = subprocess.run( args: ['python', 'uaserial.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'moodle':
        result = subprocess.run( args: ['python', 'moodle_chnu.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'openweather':
        result = subprocess.run( args: ['python', 'open_weather.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'duckgo':
        result = subprocess.run( args: ['python', 'duck_go.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'wikipedia':
        result = subprocess.run( args: ['python', 'wikipedia.py', '1'], capture_output=True, text=True).stdout
    elif test_name == 'ask':
        result = subprocess.run( args: ['python', 'ask.py', '1'], capture_output=True, text=True).stdout
    return jsonify({"result": result})

if __name__ == '__main__':
    app.run(debug=True)
```

Рисунок 4.21 – Функція запуску тестів

Функція запуску тестів (run_test) обробляє POST-запити для запуску тестів. Вона отримує JSON-дані із запиту, зокрема назву тесту. Залежно від назви тесту, виконується відповідний Python-скрипт за допомогою модуля subprocess. Виконання скрипту проводиться з параметром 1, а результат його виконання зчитується і зберігається в змінній result. Після цього функція повертає результат виконання тесту у форматі JSON. Ця функція дозволяє запускати різні тести для веб-застосунків і отримувати результати їх виконання у вигляді HTTP-відповідей.

Ці три функції є ключовими для забезпечення основної функціональності веб-застосунку: реєстрації нових користувачів, входу зареєстрованих користувачів та запуску тестів для перевірки різних веб-сайтів.

Окрім файлу app.py є також файли які містять в собі скрипти тестування,

на прикладі одного з файлів можливо показати та розібрати принцип роботи, таким файлом є «google_main.py». Спершу треба показати список використаних бібліотек(див. рис. 4.22).

```python
import sys
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import logging
import pytest
```

Рисунок 4.22 – Список бібліотек

Цей блок імпортує необхідні модулі. Бібліотеки Selenium використовуються для взаємодії з веб-браузером, а pytest для організації тестів. В коді присутній класс «TestReport» який відповідає за створення звіту після тесутвання(див. рис. 4.23).

```python
class TestReport:
    def __init__(self):
        self.errors = []

    2 usages
    def add_error(self, error_message):
        self.errors.append(error_message)

    1 usage
    def generate_report(self):
        report = "Test Report\n"
        report += "=" * 20 + "\n"
        if not self.errors:
            report += "All tests passed successfully.\n"
        else:
            report += "Errors:\n"
            for error in self.errors:
                report += f"- {error}\n"
        logging.info(report)
        return report
```

Рисунок 4.23 – Створення звіту тестування

Цей клас збирає та генерує звіт про тести. Якщо тест пройшов успішно, звіт міститиме відповідне повідомлення. У випадку помилок вони додаються до звіту. Завдяки вже існуючому коду в фронтенді на сторінці виводиться повідомлення про проходження тестування(див. рис. 4.24).

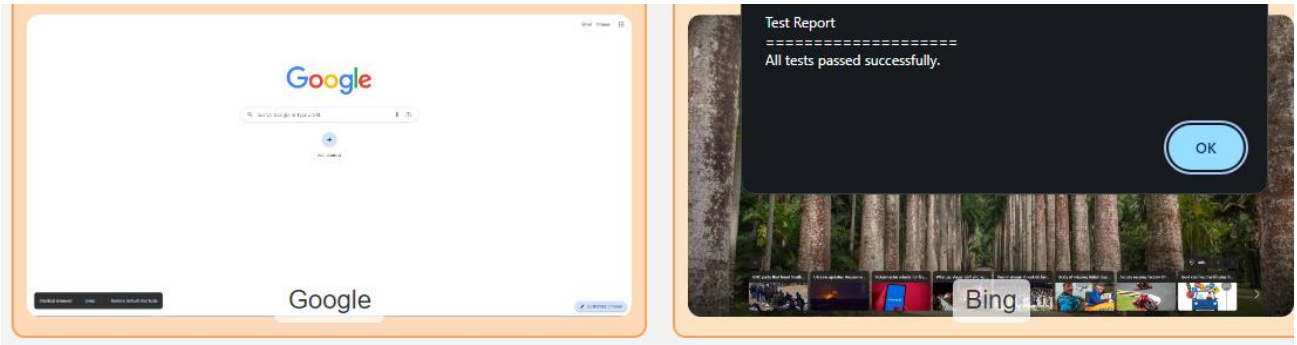


Рисунок 4.24 – Приклад реалізації класу в вебзатсосунку

Клас «WebTester» призначений для автоматизації взаємодії з веб-сторінками за допомогою бібліотеки Selenium. Цей клас забезпечує відкриття веб-сторінок, пошук елементів, введення тексту, клікання на елементи та зняття скріншотів. Нижче наведено детальний опис кожного методу цього класу(див. рис. 4.25).

```python
class WebTester:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(self.driver, timeout: 10)
        self.errors = []

    1 usage
    def open_page(self, url):
        try:
            self.driver.get(url)
            logging.info(f"Opened page: {url}")
        except Exception as e:
            logging.error(f"Failed to open page {url}: {e}")
            self.errors.append(f"open_page: {e}")

    2 usages
    def find_element(self, locators):
        for by, value in locators:
            try:
                element = self.wait.until(EC.presence_of_element_located((by, value)))
                logging.info(f"Found element by {by} with value {value}")
                return element
            except Exception as e:
                logging.warning(f"Failed to find element by {by} with value {value}: {e}")
        error_message = f"Unable to locate element using any of the provided locators: {locators}"
        logging.error(error_message)
        self.errors.append(error_message)
        return None

    1 usage
    def input_text(self, element, text):
        try:
            element.send_keys(text)
            logging.info("Inputted text")
        except Exception as e:
            logging.error(f"Failed to input text: {e}")
            self.errors.append(f"input_text: {e}")
```

Рисунок 4.25 – Клас WebTester

Цей клас включає методи для взаємодії з веб-сторінками через браузер. Він може відкривати сторінки, знаходити елементи, вводити текст, клікати елементи та знімати скріншоти. У випадку помилок вони зберігаються у списку помилок.

Функція run_test відповідає за автоматизацію процесу тестування вебсайту. Вона виконує такі основні кроки: відкриття сторінки, пошук елементів, взаємодія з ними (введення тексту, клікання) і збереження скріншотів. Нижче наведено детальний опис цієї функції(див. рис. 4.26).

```python
def run_test(site_id):
    driver_path = r"C:\Program Files\Google\Chrome\Application\chromedriver-win64\chromedriver.exe"
    service = Service(driver_path)
    driver = webdriver.Chrome(service=service)
    tester = WebTester(driver)
    report = TestReport()

    if site_id == '1':
        url = "https://www.google.com"
        search_term = "q"
        search_button_term = "btnK"
    else:
        return "Unknown site id."

    tester.open_page(url)

    search_box_locators = [
        (By.NAME, search_term),
        (By.XPATH, f"//input[@name='{search_term}']"),
        (By.CSS_SELECTOR, f"input[name='{search_term}']")
    ]

    search_button_locators = [
        (By.NAME, search_button_term),
        (By.XPATH, f"//input[@name='{search_button_term}']"),
        (By.CSS_SELECTOR, f"input[name='{search_button_term}']")
    ]

    search_box = tester.find_element(search_box_locators)
    search_button = tester.find_element(search_button_locators)

    if search_box and search_button:
        random_text = "Test Text"
        tester.input_text(search_box, random_text)
        tester.wait.until(EC.element_to_be_clickable((By.NAME, search_button_term)))
        tester.click_element(search_button)
        tester.take_screenshot(f"test_screenshot_{search_term}.png")
    else:
        report.add_error("Could not find necessary elements on the page")

    for error in tester.errors:
        report.add_error(error)

    final_report = report.generate_report()
    logging.info(f"Final report: {final_report}")
    print(final_report)
```

Рисунок 4.26 – функція run_test

Функція run_test автоматизує процес пошуку в Google, відкриваючи сторінку, вводячи пошуковий запит, клікаючи на кнопку пошуку і зберігаючи скріншот результатів. Вона використовує Selenium WebDriver для взаємодії з браузером і виконує всі дії через методи класу WebTester.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи бакалавра, описано виконання розробки програмного забезпечення. Реалізовано функції реєстрації, логіну користувача, а також запуску тестувань з вебзастосунку. Додано базу даних зберігання даних зареєстрованих користувачів і результатів тестування.

ВИСНОВКИ

При виконанні кваліфікаційної роботи бакалавра було розроблено вебзастосунок та набір тестувань.

Для досягнення поставленої мети було виконано наступні завдання:

1) дослідження предметної області;
2) формування специфікації вимог до програмного забезпечення;
3) визначення архітектури для проектування програмного забезпечення;
4) моделювання та проектування програмного забезпечення;
5) розробка програмного забезпечення;
6) проведення аналізу результатів розробки.

На початковому етапі був проведений детальний аналіз предметної області, що дозволило зрозуміти основні вимоги та очікування від системи. Це дослідження стало фундаментом для подальшого проектування та розробки програмного забезпечення. Воно включало визначення функціональних та нефункціональних вимог, які забезпечують відповідність системи потребам користувачів.

Було обрано найбільш підходящу архітектуру для реалізації програмного забезпечення. Вона забезпечує ефективну роботу системи, її масштабованість та гнучкість для майбутніх розширень. На цьому етапі були створені моделі та діаграми, що відображають структуру та поведінку системи. Це включало діаграми класів, компонування, секвенції тощо, що дозволило візуалізувати всі аспекти роботи програмного забезпечення та забезпечити його правильне функціонування.

Відповідно до сформованих вимог та спроектованої архітектури, було розроблено вебзастосунок. При розробці використовувалися сучасні технології та інструменти, що забезпечують високу продуктивність та надійність системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Analog application: Appium. URL: https://appium.io/docs/en/latest/ (Latest accessed 1.04.2024)

2. Analog application: Katalon. URL: https://katalon.com (Latest accessed 1.04.2024)

3. Analog application: BrowserStack. URL: https://www.browserstack.com (Latest accessed 1.04.2024)

4. Bodea A. Pytest-Smell: a smell detection tool for Python unit tests. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*(2022). P. 793–796.

5. Buelta J. Python Automation Cookbook: 75 Python automation ideas for web scraping, data wrangling, and processing Excel, reports, emails, and more. Packt Publishing Ltd, 2020.

6. Buelta J. Python Automation Cookbook: Explore the world of automation using Python recipes that will enhance your skills. Packt Publishing Ltd, 2018.

7. Cocchiaro C. Selenium Framework Design in Data-driven Testing: Build Data-driven Test Frameworks Using Selenium WebDriver, AppiumDriver, Java, and TestNG. Packt Publishing Ltd, 2018.

8. Fangohr H., Fauske V., Kluyver T. et al. Testing with jupyter notebooks: Notebook validation (nbval) plug-in for pytest. *arXiv preprint arXiv:2001.04808*. 2020.

9. Garg N. Test Automation using Selenium WebDriver with Java: Step by Step Guide. Test Automation Using Selenium with Java, 2014. 342 p.

10. Gundecha U. Learning Selenium Testing Tools with Python: A Pratical Guide on Automated Web Testing with Selenium Using Python. Packt Publishing, 2014.

11. Kovalenko D. Selenium Design Patterns and Best Practices. Packt Publishing Ltd, 2014.

12. Okken B. Python Testing with pytest. Pragmatic Bookshelf, 2022. 98 p.

13. Ahmed N., Ahammed R., Islam M. M. et al. Machine learning based diabetes prediction and development of smart web application. International Journal of Cognitive Computing in Engineering. Vol. 2, 2021

14. Basumatary B., Agnihotri N. Benefits and Challenges of Using NodeJS. International Journal of Innovative Research in Computer Science & Technology, 2022.

15. Grinberg M. Flask Web Development: Developing Web Applications with Python, 2018.

16. Irsyad R. Penggunaan Python Web Framework Flask Untuk Pemula.. Khazanah Informatika : Jurnal Ilmu Komputer dan Informatika.

17. Jadhav G., Gonsalves F. Role of Node.js in Modern Web Application Development.. International Research Journal of Engineering and Technology. Vol. 07, Issue 06.

18. Olanipekun A. T., Mashao D. HardnessTesterV: A Web Machine Learning application for Vickers Hardness Prediction of a Metallic Alloy Using Flask API. 4th International Conference on Electrical, Communication and Computer Engineering, ICECCE DOI:10.1109/ICECCE61019.2023.10442446.

19. Tilkov S., Vinoski S. Node.js: Using JavaScript to build high-performance network programs. © 2006 IEEE. IEEE Internet Computing. Vol. 14, Issue 6. DOI:10.1109/MIC.2010.145.

20. Ussatova O., Nyssanbayeva S., Wójcik W. Modeling of the user's identification security system of on the 2fa base. International Journal of Electronics and Telecommunications. Vol. 67, Issue 2. DOI:10.24425/ijet.2021.135970..

ДОДАТОК А

google_main.py

```
import sys
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected conditions as EC
import logging
import pytest

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

class TestReport:
    def __init__(self):
        self.errors = []

    def add_error(self, error_message):
        self.errors.append(error_message)

    def generate_report(self):
        report = "Test Report\n"
        report += "=" * 20 + "\n"
        if not self.errors:
            report += "All tests passed successfully.\n"
        else:
            report += "Errors:\n"
            for error in self.errors:
                report += f"- {error}\n"
        logging.info(report)
        return report

class WebTester:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(self.driver, 10)
        self.errors = []

    def open_page(self, url):
        try:
            self.driver.get(url)
            logging.info(f"Opened page: {url}")
        except Exception as e:
            logging.error(f"Failed to open page {url}: {e}")
            self.errors.append(f"open_page: {e}")

    def find_element(self, locators):
        for by, value in locators:
            try:
                element = self.wait.until(EC.presence_of_element_located((by,
value)))
                logging.info(f"Found element by {by} with value {value}")
                return element
            except Exception as e:
                logging.warning(f"Failed to find element by {by} with value
```

```
{value}: {e}")
        error_message = f"Unable to locate element using any of the provided
locators: {locators}"
        logging.error(error_message)
        self.errors.append(error_message)
        return None
```

Продовження додатку А

```
    def input_text(self, element, text):
        try:
            element.send_keys(text)
            logging.info("Inputted text")
        except Exception as e:
            logging.error(f"Failed to input text: {e}")
            self.errors.append(f"input_text: {e}")

    def click_element(self, element):
        try:
            element.click()
            logging.info("Clicked element")
        except Exception as e:
            logging.error(f"Failed to click element: {e}")
            self.errors.append(f"click_element: {e}")

    def take_screenshot(self, file_path):
        try:
            self.driver.save_screenshot(file_path)
            logging.info(f"Screenshot saved to {file_path}")
        except Exception as e:
            logging.error(f"Failed to take screenshot: {e}")
            self.errors.append(f"take_screenshot: {e}")

    def close(self):
        self.driver.quit()
        logging.info("Closed the browser")

def run_test(site_id):
    logging.info("Starting Google test")
    driver_path = r"C:\Program Files\Google\Chrome\Application\chromedriver-
win64\chromedriver.exe"
    service = Service(driver_path)
    driver = webdriver.Chrome(service=service)
    tester = WebTester(driver)
    report = TestReport()

    if site_id == '1':
        url = "https://www.google.com"
        search_term = "q"
        search_button_term = "btnK"
    else:
        return "Unknown site id."

    tester.open_page(url)

    search_box_locators = [
        (By.NAME, search_term),
```

```
            (By.XPATH, f"//input[@name='{search_term}']"),
            (By.CSS_SELECTOR, f"input[name='{search_term}']")
        ]
```

Продовження додатку А

```
        search_button_locators = [
            (By.NAME, search_button_term),
            (By.XPATH, f"//input[@name='{search_button_term}']"),
            (By.CSS_SELECTOR, f"input[name='{search_button_term}']")
        ]

        search_box = tester.find_element(search_box_locators)
        search_button = tester.find_element(search_button_locators)

        if search_box and search_button:
            random_text = "Test Text"
            tester.input_text(search_box, random_text)
            tester.wait.until(EC.element_to_be_clickable((By.NAME,
search_button_term)))
            tester.click_element(search_button)
            tester.take_screenshot(f"test_screenshot_{search_term}.png")
        else:
            report.add_error("Could not find necessary elements on the page")

        for error in tester.errors:
            report.add_error(error)

        final_report = report.generate_report()
        logging.info(f"Final report: {final_report}")
        print(final_report)

        tester.close()
        return final_report

@pytest.mark.parametrize("site_id", ['1'])
def test_google(site_id):
    result = run_test(site_id)
    assert "All tests passed successfully." in result, "Test failed!"

if __name__ == '__main__':
    if len(sys.argv) == 2:
        site_id = sys.argv[1]
        result = run_test(site_id)
        print(result)
    else:
        pytest.main([__file__])
```

ДОДАТОК Б

app.py

```
import subprocess
from flask import Flask, request, jsonify, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS

app = Flask(  name  )
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:12345@localhost/dbTestify'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
CORS(app, resources={r"/*": {"origins": "http://localhost:3001"}})

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)

class TestResult(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    test_name = db.Column(db.String(50), nullable=False)
    result = db.Column(db.Text, nullable=False)

    def __repr__(self):
        return f'<TestResult {self.test_name}>'

with app.app_context():
    db.create_all()

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/register', methods=['POST'])
def register():
    if request.method == 'POST':
        data = request.json
        username = data.get('username')
        email = data.get('email')
        password = data.get('password')
        confirm_password = data.get('confirm_password')

        if password != confirm_password:
            return jsonify({"message": "Passwords do not match"}), 400

        new_user = User(username=username, email=email, password=password)
        db.session.add(new_user)
        db.session.commit()

        return jsonify({"message": "Registration successful"}), 200

    return render_template('register.html')


@app.route('/login', methods=['POST'])
```

```
def login():
    try:
        username = request.json['username']
        password = request.json['password']
```

Продовження додатку Б

```
        user = User.query.filter_by(username=username,
password=password).first()

        if user:
            return jsonify({"message": "Login successful"}), 200
        else:
            return jsonify({"message": "Invalid username or password"}), 401
    except Exception as e:
        app.logger.error(f"Error occurred: {e}")
        return jsonify({"message": str(e)}), 500

@app.route('/run_test', methods=['POST'])
def run_test():
    data = request.json
    test_name = data.get('test_name')
    result = ""

    if test_name == 'google':
        result = subprocess.run(['python', 'google_main.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'bing':
        result = subprocess.run(['python', 'bing.py', '1'], capture_output=True,
text=True).stdout
    elif test_name == 'e-katalog':
        result = subprocess.run(['python', 'e-katalog.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'uaserial':
        result = subprocess.run(['python', 'uaserial.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'moodle':
        result = subprocess.run(['python', 'moodle_chnu.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'openweather':
        result = subprocess.run(['python', 'open_weather.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'duckgo':
        result = subprocess.run(['python', 'duck_go.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'wikipedia':
        result = subprocess.run(['python', 'wikipedia.py', '1'],
capture_output=True, text=True).stdout
    elif test_name == 'ask':
        result = subprocess.run(['python', 'ask.py', '1'], capture_output=True,
text=True).stdout
    return jsonify({"result": result})

if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК В

server.js

```
const express = require('express');
const httpProxy = require('http-proxy');
const path = require('path');

const app = express();
const apiProxy = httpProxy.createProxyServer();

const FLASK_SERVER = 'http://localhost:5000';

app.use(express.static(path.join(__dirname, 'public')));

app.all('/api/*', (req, res) => {
    apiProxy.web(req, res, { target: FLASK_SERVER });
});

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
app.get('/about', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'about.html'));
});

app.get('/register', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'register.html'));
});

app.get('/login', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'login.html'));
});

app.listen(3001, () => {
    console.log('Server running at http://localhost:3001');
});
```

ДОДАТОК Г

app.js

```
document.addEventListener('DOMContentLoaded', function() {
    const buttons = document.querySelectorAll('.site');
    buttons.forEach(button => {
        button.addEventListener('click', function() {
            const testName = button.getAttribute('data-test-name');
            fetch('http://localhost:5000/run_test', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ test_name: testName })
            })
            .then(response => response.json())
            .then(data => {
                alert(data.result || 'Test completed successfully');
            })
            .catch(error => console.error('Error:', error));
        });
    });

    const registerForm = document.getElementById('registerForm');
    if (registerForm) {
        registerForm.addEventListener('submit', function(event) {
            event.preventDefault();
            const formData = new FormData(event.target);
            const data = Object.fromEntries(formData.entries());

            fetch('http://localhost:5000/register', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(data)
            })
            .then(response => response.json())
            .then(data => {
                alert(data.message || 'Registration successful');
            })
            .catch(error => console.error('Error:', error));
        });
    }
```

Продовження додатку Г

```
const loginForm = document.getElementById('loginForm');
if (loginForm) {
    loginForm.addEventListener('submit', function(event) {
        event.preventDefault();

        const formData = new FormData(event.target);
        const data = Object.fromEntries(formData.entries());

        fetch('http://localhost:5000/login', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            const resultDiv = document.getElementById('login-result');
            if (resultDiv) {
                resultDiv.innerText = data.message;
            }
            alert(data.message || 'Login successful');
        })
        .catch(error => console.error('Error:', error));
    });
}

const themeToggleButton = document.getElementById('theme-toggle');
const body = document.body;
const darkModeClass = 'dark-mode';

if (localStorage.getItem('theme') === darkModeClass) {
    body.classList.add(darkModeClass);
    applyDarkModeToElements();
}

if (themeToggleButton) {
    themeToggleButton.addEventListener('click', () => {
        body.classList.toggle(darkModeClass);
```

Продовження додатку Г

```
            if (body.classList.contains(darkModeClass)) {
                localStorage.setItem('theme', darkModeClass);
                applyDarkModeToElements();
            } else {
                localStorage.removeItem('theme');
                removeDarkModeFromElements();
            }
        });
    }

    function applyDarkModeToElements() {
        document.querySelectorAll('header, nav ul li a, .site, .site span,
.register-form, .register-form label, .register-form input, .register-form button,
footer, .about-container, .about-container h1, .about-container h2, .about-
container p, .about-container ul').forEach(element => {
            element.classList.add(darkModeClass);
        });
    }

    function removeDarkModeFromElements() {
        document.querySelectorAll('header, nav ul li a, .site, .site span,
.register-form, .register-form label, .register-form input, .register-form button,
footer, .about-container, .about-container h1, .about-container h2, .about-
container p, .about-container ul').forEach(element => {
            element.classList.remove(darkModeClass);
        });
    }
});
```