НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: **Методити автоматизації тестування веб додатків для QA automation на основі бібліотеки Selenium Web Driver**

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-01

БАВОРОВСЬКИЙ Вадим Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
доцент, к.т.н., Ігор КУЗЬМЕНКО _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент: _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *доцент Андрій ДОНЕЦЬ* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту

БАВОРОВСЬКОМУ Вадиму Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи **Методи автоматизації тестування веб додатків для QA automation на основі бібліотеки Selenium Web Driver**

керівник роботи ***Кузьменко Ігор Миколайович, к.т.н., доцент***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування JavaScript, Typescript, бібліотеки React, React Router DOM, UUID, Tailwind CSS, SCSS, classNames, Axios, Redux середовище розробки Visual Studio Code

4. Перелік питань, які потрібно розробити
1) розробка зручного та інтуїтивно зрозумілого користувацького інтерфейсу для створення та управління тестами
2) інтеграція бібліотеки Selenium WebDriver для виконання автоматизованих тестів
3) забезпечення можливості збереження, редагування та повторного використання тестових сценаріїв файлів

4) тестування розробленого програмного забезпечення
5) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють роботу алгоритмів веб-інтерфейсу, архітектуру системи, взаємодію користувачів із системою, реалізація програмного забезпечення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент _____ **Вадим БАВОРОВСЬКИЙ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Ігор КУЗЬМЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 68 сторінках, містить 37 ілюстрацій, 1 додаток, 27 джерел в переліку посилань.

Мета роботи: створити веб-додаток, який допомагає тестувати веб-додатки, з використанням бібліотеки Selenium WebDriver.

Методи та засоби: автоматизацію тестування веб-додатків за допомогою бібліотеки Selenium WebDriver, мови програмування Typescript, ,бібліотека для створення інтерфейсу React, управління станом додатків за допомогою Redux, а також стилі систем CSS і SCSS.

Результат – веб-додаток, для тестування веб-сайтів.

Ключові слова: ТЕСТУВАННЯ, QA, SELENIUM WEB DRIVER, REACT, TYPESCRIPT.

ABSTRACT

The thesis is made on 68 pages, contains 37 illustrations, 1 appendix, 27 sources in the list of references.

Objective**:** to create a web application that helps to test web applications using the Selenium WebDriver library..

Methods and tools: automation of web application testing using the Selenium WebDriver library, Typescript programming language, React interface library, application state management using Redux, and CSS and SCSS system styles.

The result is a web application for testing websites.

Keywords: TESTING, QA, SELENIUM WEB DRIVER, REACT, TYPESCRIPT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ 7

ВСТУП 9

1. ПОСТАНОВКА ПРИКЛАДНОЇ ЗАДАЧІ 10

1.1 Актуальність проблеми автоматизації тестування веб-додатків 10

1.2 Постановка задачі розробки веб-додатку для автоматизації тестування 11

1.3 Аналіз сучасних підходів до автоматизації тестування веб-додатків 12

2. МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧІ 15

2.1 Вибір мови програмування та середовища розробки 15

2.1.1 React JS 15

2.1.2 TypeScript 17

2.2 Огляд бібліотеки Selenium WebDriver та її можливостей 18

2.2.1 Переваги Selenium WebDriver 19

2.2.2 Недоліки Selenium WebDriver 20

2.2.3 Як працює Selenium WebDriver 20

2.3 Використання фреймворків та бібліотек для розробки клієнтської та серверної частин 21

2.3.1 React Router DOM 22

2.3.2 UUID 23

2.3.3 Бібліотеки та фрейворки для стилізації 24

2.3.4 Axios 27

2.3.5 Redux 28

2.4 Система керування версіями Git 29

2.5 Середовище розробки Visual Studio Code 30

3. ПРОГРАМНА РЕАЛІЗАЦІЯ 31

3.1 Архітектура веб-додатку 31

3.1.1 Опис клієнтської частини додатку 32

3.1.2 Опис серверної частини додатку 36

3.2 Взаємодія клієнтської та серверної частин 37

4. ДЕМОНСТРАЦІЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ 43

4.1 Інсталяція програмного забезпечення та вимоги до системних характеристик 43

4.1.1 Системні характеристики 43

4.1.2 Інсталяція програмного забезпечення 43

4.2 Демонстрація функціоналу додатку 46

ВИСНОВКИ 51

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53

ДОДАТОК А 56

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

Quality assurance (QA) — забезпечення якості, це будь-який систематичний процес визначення того, чи відповідає продукт або послуга встановленим вимогам.

Графічний інтерфейс користувача (GUI) — це тип інтерфейсу користувача, який знайомий більшості людей. Це тип інтерфейсу, який використовує візуальні елементи, щоб допомогти нам взаємодіяти з функціями системи.

Фреймворк — це структура, яка забезпечує основу для процесу розробки програми. За допомогою фреймворку можна уникнути написання всього з нуля. Фреймворки надають набір інструментів та елементів, які допомагають прискорити процес розробки. Він діє як шаблон, який можна використовувати і навіть змінювати відповідно до вимог проекту.

Frontend (також відомий як клієнською частиною додтаку) — це все, що користувач бачить або з чим взаємодіє під час відвідування веб-сайту.Він відповідає за загальний вигляд і відчуття від користування сайтом в Інтернеті.

Backend (також відомий як серверна частиною додтаку) — це інфраструктура, яка підтримує інтерфейс і складається з частин програмного забезпечення, які звичайні користувачі не бачать. Бекенд - це, по суті, мозок веб-сайту.

UUID (англ. Universally Unique Identifier) — стандарт ідентифікації, який використовується при створенні програмного забезпечення, затверджений Open Software Foundation (OSF) як частина Розподіленого комп'ютерного середовища (DCE).

UI (англ User Interface) — Інтерфейс користувача це точка взаємодії людини з комп'ютером, веб-сайтом або додатком.

API (англ Application Programming Interface) — Інтерфейс прикладного програмування - це набір правил або протоколів, які дозволяють програмним додаткам взаємодіяти один з одним для обміну даними, функціями та функціоналом.

DOM (англ Document Object Model) — модель об'єктів документа і являє собою JavaScript-представлення вашої веб-сторінки. DOM бере HTML-елементи та CSS-стилі і представляє їх за допомогою JavaScript-об'єктів.

HTTP (англ HyperText Transfer Protocol) — Протокол передачі гіпертексту — це протокол зв'язку, який використовується для більшості комунікацій у всесвітній павутині. Протокол являє собою набір правил, які забезпечують взаємодію між клієнтом і сервером між вашим веб-браузером і цільовою веб-сторінкою. Як дворецький, він приймає ваші запити, а потім отримує відповідь від відповідного сервера.

JSON — це абревіатура від JavaScript Object Notation, це відкритий стандартний формат, який є легким і текстовим, розробленим спеціально для обміну даними, що читаються людиною. Це незалежний від мови формат даних. Він підтримує майже всі мови, фреймворки та бібліотеки.

URL — це веб-адреса, яка веде на потрібний веб-сайт. URL-адреса знаходиться у верхній частині вікна веб-браузера. Повна URL-адреса часто не відображається, поки користувач не клацне в адресному рядку.

HTML — це стандартизована система розмітки текстових файлів, яка створює структуру практично кожної сторінки, яку ми знаходимо і використовуємо в Інтернеті. Саме HTML додає розриви сторінок, абзаци, жирний шрифт, курсив тощо. HTML створює цю структуру за допомогою тегів, які вказують браузерам, що робити з текстом.

CSS розшифровується як Cascading Style Sheets (каскадні таблиці стилів). Це мова кодування, яка визначає зовнішній вигляд та макет веб-сайту. Поряд з HTML, CSS має фундаментальне значення для веб-дизайну. Без нього веб-сайти так і залишилися б звичайним текстом на білому фоні.

ВСТУП

Актуальність теми: В сучасному світі важко уявити життя без Інтернету та веб-сайтів зокрема. Вони стають все складнішими, а тому вимагають великою уваги до тестування під час їх розробки. Практично усі бізнеси або установи впроваджують нові можливості та функції до своїх веб-сайтів, які повинні полегшити роботу своїм користувачам або клієнтам. Під час розширення функціоналу не можна забувати слідкувати над тим, щоб існуючий функціонал продовжував працювати стабільно. Автоматизація тестування є ключовим рішенням для подолання цих викликів, що робить цю тему актуальною та важливою для сучасної індустрії розробки програмного забезпечення.

Мета: є розробка веб-додатку для автоматизації тестування веб-сайтів за допомогою бібліотеки Selenium WebDriver. Розробка зручний інтерфейс для створення та виконання тестових сценаріїв, які можна зберігати та використовувати повторно.

Завдання:

1. Розробка зручного та інтуїтивно зрозумілого інтерфейсу для створення та керування тестами.
2. Інтеграція бібліотеки Selenium WebDriver для виконання автоматизованих тестів.
3. Створення функціоналу збереження, редагування та повторного використання тестових сценаріїв.
4. Використання додатку для тестування функціоналу форм, натискання кнопок та введення даних.

1. ПОСТАНОВКА ПРИКЛАДНОЇ ЗАДАЧІ

Сьогоднішнє життя не можливо уявити без веб-додатків. Вони займають центральне місце сферах життя від електронної комерції до соціальних мереж та корпоративних систем управління. Веб-додатки швидко розвиваються, стають все складнішими та більш функціональними. Також постійно ростуть потреби та вимоги користувачів до якості. Відповідно і еквівалентно росте потреба у тестуванні та випробувані веб-додатків.

1.1 Актуальність проблеми автоматизації тестування веб-додатків

Ручне тестування (manual testing) тестувальники вручну виконують тести, не використовуючи ніяких засобів автоматизації [1]. Воно залишається важливим етапом перевірки якості, програмного забезпечення, але коли ми говоримо про великий продукт, який потрібно потім підтримувати та розвивати стає менш ефективним. Тому, що це призводить до зростання витрат на тестування, затримок у випуску продуктів та підвищеного ризику помилок, які можуть залишитися непоміченими.

Автоматизоване тестування припускає використання спеціального ПЗ (окрім того, що тестується) для контролю виконання тестів та порівняння очікуваного і фактичного результату роботи програми. Не важко зрозуміти, автоматизація тестування веб-додатків дає можливість для підвищення ефективності цього процесу. Вона дозволяє суттєво зменшити час, необхідний для виконання тестів, підвищує точність і повторюваність тестових сценаріїв, що знижує ризик помилок.

Отже автоматизація тестування веб-додатків зумовлена необхідністю у підвищені якості тестування та мінімізувати ризики виникнення помилок під час тестування, пов’язаних з людським фактором. Таким чином, використовуючи та популяризуючи автоматизованих рішень для тестування веб-додатків дає

можливість для розвитку програмного забезпечення, що є важливим фактором до успіху в конкурентному середовищі.

1.2 Постановка задачі розробки веб-додатку для автоматизації тестування

У сфері технологій, що постійно розвивається, складність сучасних веб-додатків невпинно зростає, створюючи величезні труднощі для QA-інженерів, перед якими стоїть кропітка задача виявлення та усунення помилок. Ця зростаюча складність підкреслює необхідність інноваційних рішень для оптимізації процесу тестування. Так з'являється ключова роль Selenium WebDriver, надійної бібліотеки, яку шанують за її досконаліст в автоматизації тестування веб-додатків.

Суть цього рішення полягає в розробці комплексної системи, розробленої для безперешкодної інтеграції Selenium WebDriver, тим самим організовуючи автоматизованих тестів, які перетинають складну павутину функціональних можливостей додатку. Також потрібно звернути увагу на ретельну розробку інтерфейсу веб-додатків, наділеного інтуїтивно зрозумілими принципами дизайну. Спрощуючи взаємодію з елементами веб-сторінок, такими як кнопки, поля введення та форми.

Ключові задачі проекту:

1. Розробка зручного та інтуїтивно зрозумілого інтерфейсу для створення та керування тестами.

2. Інтеграція бібліотеки Selenium WebDriver для виконання автоматизованих тестів.

3. Створення функціоналу збереження, редагування та повторного використання тестових сценаріїв.

4. Використання додатку для тестування функціоналу форм, натискання кнопок та введення даних.

Продукт має на меті спростити та прискорити процес тестування, також зменшити кількість помилок, що трапляються під час ручному тестуванні.

Розробка охоплює клієнтську частину та серверну, використовуючи сучасні технології.

1.3 Аналіз сучасних підходів до автоматизації тестування веб-додатків

Основними підходами автоматизації тестування є два. Перший це тестування на рівні коду і GUI-тестування або їх ще називаються функціональні та нефункціональні[2].

Функціональне тестування сюди входить перевірка функціоналу ПЗ. Воно підтверджує, що система працює належним чином і не містить помилок. QA інженер моделює реальний сценарій кінцевого користувача та порівнює результати очікуванні з реальними. Функціональне тестування може використовуватися вручну або автоматизовано.

Нефункціональне тестування, перевіряє, як продукт працює. За цією назвою ховаються такі типи тестування програмного забезпечення:

- продуктивності
- інтерфейсу користувача
- захищеності
- інсталяційне – тут QA-інженер повинен перевірити, чи не виникнуть проблеми при встановленні та оновленні продукту;
- сумісності – як продукт працює з іншими продуктами;
- локалізації – коректність мовного та культурного аспекту, якщо ПЗ буде використовуватися на різних ринках.

Також виділяють 4 основних рівня.

1. Модульне тестування (Unit testing) [3] – передбачає тестування кожної атомарної функціональності програми окремо у штучно створеному середовищі. Необхідність створення такого синтетичного робочого середовища для

конкретного модуля вимагає від тестувальника знань в області автоматизації тестування ПЗ та певних навичок програмування.

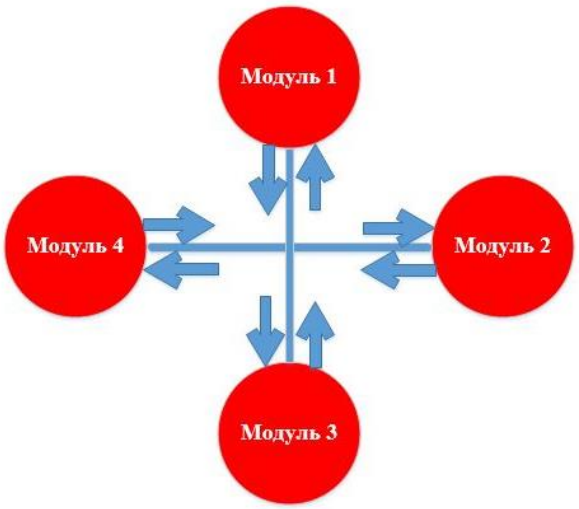

Рисунок 1.1 — Модульне тестування

Об’єкт тестування виділений червоним кольором.

2. Інтеграційне тестування[4] — вид тестування перевіряється інтеграція модулів, як вони взаємодія між собою також, чи правильно відбувається інтеграція підсистем в одну загальну систему або між собою.

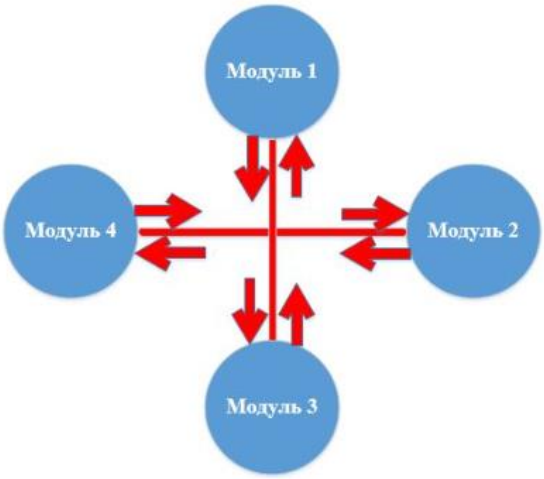

Рисунок 1.2 — Інтеграційне тестування

Об’єкт тестування виділений червоним кольором.

3. Системне тестування[5] – це тестування ПЗ, що виконується на повній, інтегрованій системі, тобто всієї системи з метою перевірки відповідності системи вихідним вимогам, як функціональним, так і не функціональним.

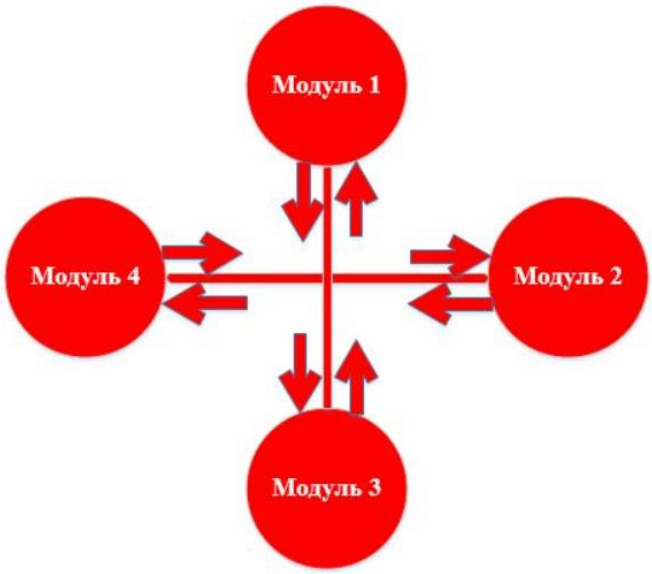

Рисунок 1.3 — Системне тестування

Об’єкт тестування виділений червоним кольором.

4. Приймальне тестування (acceptance) [6] – вид тестування, який проводиться на етапі здачі готового продукту (або готової частини продукту) замовнику. Метою є перевірити чи відповідає готовий продукт поставленій задачі. Результати тесту може бути здача проєкту або до працювання.

Автоматизація тестування на кожному з цих рівнів забезпечує більш швидке виявлення помилок та збоїв, що сприяє підвищенню якості кінцевого продукту. Використання сучасних інструментів і методів автоматизації дозволяє значно скоротити час тестування та зменшити витрати на розробку. Також, автоматизоване тестування забезпечує стабільність і повторюваність результатів, що є ключовим фактором у процесі розробки складних програмних продуктів.

2. МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

В процесі вирішення задачі, що ставиться перед проектом, було використано ряд методів та алгоритмів, які забезпечують ефективність та надійність кінцевого продукту. Вибір відповідних інструментів та технологій для розробки веб-додатка обумовлений їхніми перевагами та сумісністю з вимогами проекту.

2.1 Вибір мови програмування та середовища розробки

Для розробки frontend (клієнтської) частини веб-додатка, який буде відповідати вимогам проекту, було обрано комбінацію React[7] з TypeScript[8]. Для backend (серверної) частини TypeScript.

Давайте по черзі розглянемо, що таке React та TypeScript, чому саму ці технології було обрано для нашого веб-додатку.

2.1.1 React JS

React захопив світ веб-розробки , він один з найпотужніших JavaScript-бібліотек для створення інтерактивних користувацьких інтерфейсів. За даними Statista, це другий за поширеністю веб-фреймворк[9] у 2023 році, яким користуються 40% розробників в усьому світі.

Розробники активно використовуємо React для створення веб-додатків та UI під найрізноманітніші завдання. Розгляньмо особливості та переваги цієї бібліотеки.

React був розроблений у компанії Facebook (тепер вже відома як Meta) у 2013 році і активно розвивається. Найпопулярніша соцмережа світу потребувала розширених можливостей робити з користувацьким інтерфейсом, тож інженер Facebook Джордон Уолк створив прототип бібліотеки, що сьогодні відома як React, або React JS.

Технічно, React JS являє собою відкриту JavaScript-бібліотеку для побудови користувацьких інтерфейсів з компонентів UI. Тобто, на React реалізується та частина сайту, з якою користувач безпосередньо взаємодіє (frontend) через браузер. React робить розробку на frontend в рази простішою, дозволяючи створювати інтерактивні елементи, які можна з легкістю повторно використовувати в іншій частині продукту. Цей принцип називається компетентність. У React ми створюємо компоненти, потім ми можемо повторно їх використати. Тобто, розробнику не потрібно писати код на JavaScript двічі, аби створити дві однакові кнопки різного кольору. У великих проектах таких кнопок можуть бути сотні.

Архітектура React розв'язує цю проблему, дозволяючи розробнику визначити окремий шматочок UI (ту саму кнопку), як окремий компонент. Тож, якщо розробнику необхідно реалізувати в інтерфейсі безліч однакових кнопок різного кольору, він може, скажімо, використовувати свій компонент Button, модифікуючи його колір через “властивості” props. В цьому механізмі криється уся суть бібліотеки React: вона робить роботу над складним інтерактивним фронт-ендом простою та організованою.

React JS не варто плутати з React Native. Це споріднені фреймворки, що поділяють спільний синтаксис, але це не одне й те саме. React Native був створений для створення кросплатформових додатків, він використовує нативні компоненти UI та API, зокрема для мобільної розробки.

Переваги React JS у фронтенд-розробці:

1. Доступність: React легко освоїти навіть розробникам із базовими знаннями JavaScript. Досвідчені JS-розробники можуть опанувати його функції за кілька вечорів.

2. Гнучкість: Код, написаний на React, легко розуміється та підтримується. Так як це бібліотека він є гнучким та сумісним з іншими фреймворками, що значно економить час розробників і скорочує витрати бізнесу.

3. Швидкодія: React забезпечує швидку та ефективну роботу фронтенду завдяки віртуальній об'єктній моделі документа (virtual DOM) та можливості серверного рендерингу. Це дозволяє складним сторінкам та додаткам працювати

швидко. Тому, React не звертається напряму до DOM, а робить це через копію, через це операції, які ми створює значно швидші.

4. Перевикористання компонентів: Окремі автономні UI-компоненти зв'язують разом код HTML та JavaScript. Ця логіка робить розробку наочнішою, покращує якість коду та заощаджує час розробника.

5. Браузерні інструменти: Розробники мають доступ до безкоштовних плагінів React development tools для популярних браузерів. Ці інструменти дозволяють легко перевіряти ієрархію компонентів React та забезпечують зручності для відладки.

6. Величезне ком’юніті: Світова спільнота розробників постійно розвиває фреймворк, створює нові бібліотеки та ділиться досвідом з новачками. Завжди можна знайти потрібні інструменти та інформацію. Крім того, знайти фахівців для проекту на React не складно.

2.1.2 TypeScript

TypeScript одна з найпопулярніших мов програмування, за даними dou.ua вона посіла друге місце у рейтингу 2024 році (перше JavaScript) [10] Розроблений корпорацією Microsoft, цей інструмент став втіленням прагнення до поняттю чистого кода та його структурування. Він є надбудова над JavaScript шляхом додавання синтаксису для оголошень типів, класів та інших об'єктно-орієнтованих функцій з перевіркою типів.

TypeScript легкий у вивчені та з його знаннями можна працювати у будь-якому напряму інформаційних технологій також він забезпечує сумісність із вже що існує JavaScript-бібліотеками та фреймворками, дозволяючи плавно інтегрувати новітні розробки в старі проєкт. Тому часто використовують TypeScript зі згаданим раніше React. Це поєднання є дуже популярне серед розробників. Така гармонія новацій і традицій робить TypeScript ідеальним вибором для компаній, які прагнуть технологічного прогресу без втрати часу на переписування коду з нуля.

Ключові переваги TypeScript для розробників

1. TypeScript дорівнює JavaScript: Typescript використовує базовий синтаксис і будівельні блоки JavaScript. Розробники, які знають JS, можуть почати використовувати Typescript негайно. Під час виконання, файли typescript перетворюються у звичайний Javascript. Typescript підтримує бібліотеки, фреймворки та інструменти JS. Крім того, файли .js можуть бути перетворені в .ts для компіляції з файлами typescript.

2. Покращена підтримка IDE (Visual Studio Code): Оскільки і TS, і VSCode розроблені Microsoft, то зв'язки між ними просто дивовижні, що робить процес розробки надзвичайно приємним. Редактор коду реагує на типізований код, надаючи підказки та автодоповнення, що полегшує процес написання та відгадки коду.

3. Сумісність із JavaScript: TypeScript є надбудовою над JavaScript, тому він сумісний зі всіма що існує бібліотеками та фреймворками JavaScript.

4. Передові можливості програмування: Використання дженериків, декораторів та інших сучасних функцій мови робить код більш гнучким та потужним.

5. Об'єктно-орієнтована мова сценаріїв: TypeScript - це об'єктно-орієнтована мова програмування. Вона підтримує такі поняття, як класи, інтерфейси, успадкування тощо. І представляє переваги об'єктно-орієнтованих мов програмування, такі як легше усунення несправностей, повторне використання коду, продуктивність, надмірність даних, гнучкість коду, інкапсуляція, успадкування та поліморфізм.

2.2 Огляд бібліотеки Selenium WebDriver та її можливостей

Selenium WebDriver[11] - це об'єктно-орієнтований, потужний інструмент для автоматизації тестування веб-додатків. Він керує браузером так, як це зробив би користувач, надаючи розробникам можливість взаємодіяти з веб-елементами напряму. Підтримка декількох мов програмування робить його ще більш універсальним.

Мови програмування інтегруються через драйвери Selenium. Це

бібліотеки, спеціально створені для кожної мови, які перетворюють команди з Selenium API у методи або функції, доступні для використання розробниками.

Selenium часто використовується для автоматизації веб-додатків з метою тестування. Проте, важливо зазначити, що він не включає в себе фреймворк для тестування. Це комплексне рішення, яке об'єднує різноманітні інструменти та бібліотеки для автоматизації веб-браузерів, пропонуючи розширення для емуляції взаємодії з браузерами.

Окрім цього, Selenium має сервер для масштабованого розподілу браузерів і інфраструктуру для реалізації специфікації W3C WebDriver. Ця специфікація дозволяє писати універсальний код, що працює у всіх основних веб-браузерах. Основою Selenium є WebDriver — інтерфейс, який дозволяє створювати набори інструкцій, придатних для використання в різних браузерах.

2.2.1 Переваги Selenium WebDriver

Серед багатьох переваг основні можна виділити такі[12]:

1. Підтримка безлічі операційних систем: Windows, Mac, Linux, Unix та інших. Така універсальність гарантує, що розробники та тестувальники можуть безперешкодно працювати у своїх улюблених середовищах.

2. Мовна сумісність - ще одна перевага Selenium WebDriver. Він охоплює безліч мов програмування, від Python і Java до Perl і Ruby, задовольняючи різноманітні уподобання розробників з усього світу.

3. Сумісність з сучасними браузерами: Selenium WebDriver, який розширює свою підтримку на Chrome, Firefox, Safari та Internet Explorer. Така сумісність з сучасними браузерами підкреслює його актуальність у цифровому ландшафті, що постійно розвивається.

4. Швидкість має вирішальне значення в тестуванні, і тут Selenium WebDriver перевершує всі очікування. Він завершує виконання тестових скриптів зі швидкістю, яка випереджає багато конкуруючих інструментів, тим самим підвищуючи продуктивність і ефективність.

5. API (інтерфейс прикладного програмування) Selenium WebDriver більш лаконічний, ніж у його попередника, Selenium RC. Цей оптимізований API

спрощує взаємодію та скорочує час навчання, роблячи його більш зручним для тестувальників усіх рівнів.

6. Сумісність з IPhone та Android: Selenium WebDriver забезпечує сумісність з мобільними та безголівковими браузерами завдяки підтримці iPhoneDriver, HtmlUnitDriver та AndroidDriver. Це гарантує, що тести можуть бути виконані на широкому спектрі пристроїв і середовищ, що відображає різноманітні способи взаємодії користувачів з додатками сьогодні.

2.2.2 Недоліки Selenium WebDriver

Однак, важливо розуміти, що жоден інструмент не позбавлений обмежень. Незважаючи на свої потужні можливості, Selenium WebDriver має свої обмеження. Розуміння цих обмежень має вирішальне значення для встановлення реалістичних очікувань і забезпечення оптимального використання інструменту.

1. Налаштування у різних середовищах: Selenium WebDriver підтримує декілька браузерів та операційних систем, його налаштування для різних середовищ може бути складним і трудомістким. Процес налаштування може вимагати значних знань і досвіду, що може стати перешкодою для початківців.

2. Генерація звітів: Selenium WebDriver не має вбудованих можливостей для створення звітів. Користувачі повинні покладатися на зовнішні фреймворки або інструменти для створення тестових звітів, що може ускладнити процес тестування і вимагати додаткового налаштування та обслуговування.

2.2.3 Як працює Selenium WebDriver

Selenium WebDriver працює в три етапи:

1. Тестові команди перетворюються в HTTP-запит за допомогою протоколу JSON.

2. Перед виконанням будь-яких тестових кейсів кожен браузер має свій власний драйвер, який ініціалізує сервер.

3. Потім браузер починає отримувати запит через свій драйвер.

Давайте розглянемо приклад з фрагментом коду нижче:

```
const driver = new Builder().forBrowser('chrome').build();

driver.get("https://www.google.com/");
```

Рисунок 2.1 — Selenium WebDriver

Після виконання коду у новому вікні відкриється Chrome, який перейде на сайт google.com.

Тепер давайте розберемося, що відбувається за кадром, коли ви натискаєте на кнопку Виконати до запуску браузера Chrome.

Після виконання програми кожен рядок коду буде перетворено на URL-адресу. Це стає можливим завдяки протоколу JSON Wire через HTTP. Потім ця URL-адреса передається драйверам браузера (у нашому прикладі - ChromeDriver). На цьому етапі наша клієнтська бібліотека (у нашому прикладі TypeScript) переводить код у формат JSON і взаємодіє з ChromeDriver.

URL після перетворення JSON виглядає наступним чином:

```
`https://localhost:8080/{"url":https://www.google.com"}`
```

Рисунок 2.2 — Selenium WebDriver

Для отримання HTTP-запитів кожен драйвер браузера використовує HTTP-сервер. Як тільки драйвер браузера отримує URL-адресу, він обробляє запит, передаючи його реальному браузеру по HTTP. І тоді всі ваші команди в Selenium-скриптах будуть виконані.

2.3 Використання фреймворків та бібліотек для розробки клієнтської та серверної частин

Для розробки веб-додатка було використано ряд фреймворків та бібліотек, які забезпечили ефективну та продуктивну роботу над проектом. Детальніше розглянемо кожний фреймворк та бібліотеку.

Ці фреймворки та бібліотеки забезпечують швидку та ефективну розробку

веб-додатка, а також забезпечують його стабільність та підтримку.

Швидко створювати та налаштовувати різні компоненти та функціональність веб-додатка, прискорюючи процес розробки та забезпечуючи більшу продуктивність.

Крім того, ці інструменти мають велику та активну спільноту розробників, яка підтримує їх розвиток та надає корисні ресурси та документацію для допомоги розробникам у вирішенні проблем та оптимізації веб-додатка.

Таким чином, використання цих фреймворків та бібліотек допомагає забезпечити якісну та стабільну розробку веб-додатка, що відповідає сучасним стандартам розробки.

2.3.1 React Router DOM

React Router DOM[13]: - це npm-пакет, який дозволяє реалізувати динамічну маршрутизацію у веб-додатку. Він дозволяє відображати сторінки та надавати користувачам можливість переходити між ними. Це повнофункціональна клієнтська та серверна бібліотека маршрутизації для React. React Router Dom використовується для створення одно сторінкових додатків, тобто додатків, які мають багато сторінок або компонентів, але сторінка ніколи не оновлюється, натомість вміст динамічно завантажується на основі URL-адреси. Цей процес називається маршрутизацією, і він стає можливим за допомогою React Router Dom.

За своєю суттю, React Router DOM є бастіоном універсальності, пропонуючи надійний фреймворк як для клієнтської, так і для серверної маршрутизації в React-додатках. Він слугує стрижнем для побудови односторінкових додатків (SPA), де сторінка залишається статичною, в той час як контент динамічно змінюється у відповідь на зміну URL-адреси.

Перевага React Router DOM полягає не лише у його функціональності, але й у його трансформаційному впливі на взаємодію з користувачем. Минули часи різких перезавантажень сторінок; натомість користувачі переміщуються по додатку з безперешкодною плавністю. Ця плавність у поєднанні з вродженою

швидкістю піднімає користувацький досвід на нові висоти, виводячи його за межі традиційної навігації сторінками.

React Router DOM з його блискавичною навігацією надає додаткам неперевершену маневреність, підвищуючи загальну продуктивність і підносячи подорож користувача до рівня неперевершеної ефективності.

React Router Dom v6 має багато корисних компонентів, і для створення повноцінної маршрутизації вам знадобиться більшість з них.

1. Router (зазвичай імпортується як BrowserRouter): Це батьківський компонент, який використовується для зберігання всіх інших компонентів. Все, що знаходиться всередині нього, буде частиною функціональності маршрутизації

2. Routes: маршрути використовуються для визначення шляхів навігації в односторінковому додатку (SPA). Маршрути визначають, які компоненти слід відображати на основі поточної URL-адреси або місця розташування додатку.

3. Route: Цей компонент перевіряє поточну URL-адресу і відображає компонент, пов'язаний з цим шляхом. Усі маршрути розміщуються у компонентах Routes.

4. Link: Компонент Link використовується для створення посилань на різні маршрути.

2.3.2 UUID

Пошук унікальності часто необхідно коли розробляєш веб-додаток на React. З цим нам допоможе бібліотека uuid[14], бібліотеку, яка є має рішення у цьому пошуку, пропонуючи безпроблемний шлях до створення унікальних ідентифікаторів.

Корисність цієї бібліотеки виходить далеко за межі простої функціональності; вона слугує для створення унікальних ключів в межах React-компонентів. Крім того, вона виступає надійним захисником від конфліктів, забезпечуючи недоторканність даних, наділяючи їх унікальністю. Та оптимізовує React додаток.

2.3.3 Бібліотеки та фрейворки для стилізації

Розглянемо, які саме технології ми використали, та чому саме їх ми обрали для стилізації нашого проекту.

2.3.3.1 Tailwind CSS

Перший та напервно основний фрайворк для стилізації нашиш комопонентів це Tailwind CSS[15]. Це фреймворк CSS, який спрощує веб-розробку, надаючи набір заздалегідь розроблених утилітарних класів. Ці класи дозволяють швидко змінювати стиль без написання власного CSS, сприяючи узгодженості та масштабованості. Підхід Tailwind зміщує акцент з традиційних CSS-компонентів на функціональні класи, дозволяючи розробникам ефективно створювати адаптивні та візуально привабливі інтерфейси з мінімальними зусиллями.

Чому Tailwind CSS? Пришвидшення процесу створення інтерфейсів. Це CSS-фреймворк, орієнтований на утиліти, що означає, що ми можемо використовувати утилітарні класи для створення користувацьких дизайнів без написання CSS, як при традиційному підході.

Переваги Tailwind CSS:

1. Не потрібно придумувати унікальні назви для класів та ідентифікаторів CSS.

2. Мінімум рядків коду у файлі CSS.

3. Ми можемо налаштувати дизайн для створення компонентів.

4. Робить сайт адаптивним.

5. Вносить зміни бажаним чином.

6. CSS є глобальним за своєю природою, і якщо ви вносите зміни в файл, властивості змінюються у всіх HTML-файлах, які на нього посилаються. Але за допомогою Tailwind CSS ми можемо використовувати утилітарні класи і вносити локальні зміни.

Чому варто використовувати Tailwind, а не інші CSS фреймворки. Методологія Tailwind, орієнтована насамперед на утиліти, пропонує детальний контроль над стилями, що дозволяє точно стилізувати і швидше створювати прототипи без перевизначення стилів фреймворку.

На відміну від традиційних фреймворків, Tailwind дозволяє широкі можливості стилізації та уникає попередньо створених стилів компонентів, пропонуючи гнучкість у дизайні.

Утилітарні класи Tailwind усувають необхідність писати власний CSS, що призводить до зменшення розміру файлів та пришвидшення часу завантаження.

Tailwind спрощує адаптивний дизайн завдяки вбудованим класам, полегшуючи створення мобільних макетів без додаткових медіа-запитів або складних стилів.

Завдяки великій документації та інтуїтивно зрозумілому синтаксису Tailwind прискорює розробку, спрощуючи процес стилізації та скорочуючи час ітерацій.

2.3.3.2 SASS

Але не все можна стилізувати через Tailwind, де що потрібно писати саме на чистому CSS, у цій роботі ми користали препроцесор SASS[16].

Що таке препроцесор CSS? CSS-препроцесори схожі на компілятори, які допомагають вам компілювати CSS-код з власного унікального синтаксису. Це означає, що ви можете написати код, використовуючи синтаксис препроцесора, і він буде використовувати цей код для генерації коду CSS. HTML-файл може посилатися на CSS-файл для стилізації веб-сторінки.

Щоб використовувати препроцесори CSS, вам не потрібно вивчати нову мову, оскільки ви можете писати код так само, як CSS. Крім того, за допомогою препроцесорів ви можете робити набагато більше. Наприклад, ви можете створювати змінні за допомогою препроцесорів і використовувати їх багато разів.

Існує декілька препроцесорів для CSS. Однак SASS все частіше стає мейнстрімом у веб-розробці. Давайте дізнаємося більше про SASS CSS.

Скорочення від Syntactically Awesome Style Sheets, SASS - це популярний препроцесор CSS. Код SASS обробляється програмою і компілюється в код CSS, який можна використовувати для стилізації елементів HTML. SASS контролює те, як він відображається на веб-сторінці.

Також одна з особливостей SASS це його синтаксис. SASS CSS пропонує два різних синтаксиси. синтаксис з відступами та SCSS (Sassy CSS).

```
$font-stack: Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

Рисунок 2.3 — Синтаксис з відступами

Як випливає з назви, синтаксис з відступами використовує відступи, подібно до мови програмування Python. Ви можете опустити фігурні дужки і крапку з комою під час роботи з синтаксисом з відступами.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Рисунок 2.4 — SCSS

SCSS, з іншого боку, є новішою версією, яка ідентична CSS і використовує фігурні дужки для позначення блоків і крапку з комою для розділення правил. Саме тому ми будемо використовувати саме цей синтаксис у нашому веб-додатку.

2.3.3.3 Classnames

За своєю суттю, classnames[17] - це легка бібліотека JavaScript, яка допомагає керувати класами CSS у більш інтуїтивно зрозумілий та організований спосіб. Вона особливо популярна в екосистемі React для роботи з динамічними іменами класів. Замість того, щоб конкатенувати імена класів або

використовувати складну умовну логіку для їх перемикання, «classnames» спрощує процес, роблячи ваш код чистішим та ефективнішим.

Переваги classnames:

1. Читабельність: Це значно покращує читабельність коду, спрощуючи керування іменами класів.

2. Масштабованість: У міру зростання вашого проекту підтримка та розширення класів CSS стає все меншим головним болем.

3. Гнучкість: «classnames» працює не тільки з React, але й з різними іншими JavaScript-фреймворками, що робить його універсальним та адаптивним.

2.3.4 Axios

Axios[18] - це популярна бібліотека JavaScript, яка використовується для створення HTTP-запитів з веб-браузера. Вона спрощує процес надсилання асинхронних HTTP-запитів до сервера, а також обробляє відповідь. Axios підтримує такі функції, як перехоплювачі, обробка заголовків запитів і відповідей, а також обробка різних типів даних, таких як JSON. Він широко використовується у веб-розробці для отримання даних з API та взаємодії з серверами.

Бібліотека широко використовується в різних проектах, включаючи React, Vue.js, Angular та інші фреймворки. Вона дозволяє розробникам легко взаємодіяти з API та обробляти дані, отримані від сервера.

Переваги Axios:

1. Простота використання: Вона має простий і зрозумілий інтерфейс, що робить її легкою у використанні навіть для початківців. Вона дозволяє розробникам швидко надсилати HTTP-запити до сервера та отримувати від нього відповіді.

2. Універсальність: Підтримує браузери та Node.js, що робить його ідеальним для використання як на стороні клієнта, так і на стороні сервера в React-проектах.

3. Підтримка Promises: Повертає об'єкт Promises, що робить його ідеальним для роботи з сучасним синтаксисом JavaScript та асинхронними операціями.

4. Перехоплювачі: Дозволяє використовувати перехоплювачі, щоб запити та відповіді оброблялися до їх відправлення та після отримання. Це корисно для додавання заголовків, обробки помилок тощо.

5. Скасування запитів: Бібліотека дозволяє скасовувати запити, покращуючи продуктивність програми та запобігаючи непотрібним запитам.

6. Підтримка заголовків і параметрів запитів: Дозволяє додавати заголовки і параметри запитів, що корисно при передачі інформації на сервер.

7. Широка функціональність: Підтримує всі основні методи HTTP-запитів: GET, POST, PUT, DELETE і т.д., а також багато типів даних, таких як JSON, форми і текст.

8. Обробка помилок: Має вбудовану обробку помилок, яка дозволяє виявляти і обробляти помилки, що виникають під час виконання запитів.

2.3.5 Redux

Redux[19] - це просто сховище для зберігання стану змінних у нашому додатку. Redux створює процес і процедури для взаємодії зі сховищем, щоб компоненти не просто оновлювали або читали сховище випадковим чином. Подібно до банку. Це не означає, що якщо у вас є гроші в банку, ви можете піти в будь-який час, відкрити сховище і забрати гроші. Ви повинні пройти певні кроки, щоб зняти гроші.

Якщо коротко, то Redux - це спосіб управління «станом», або ми можемо сказати, що це кеш або сховище, до якого всі компоненти можуть мати структурований доступ. Доступ до нього здійснюється через «Reducer» та «Actions».

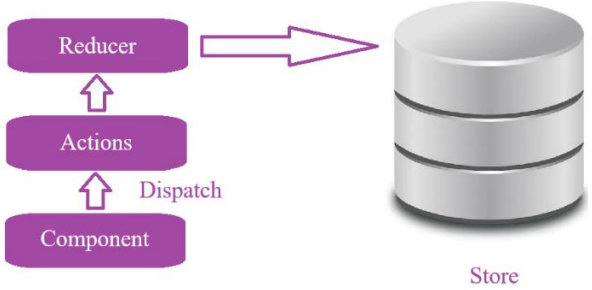

Рисунок 2.5 — Redux

Redux дозволяє вам керувати станом вашого додатку в єдиному місці, а зміни у вашому додатку стають більш передбачуваними та відстежуваними, що полегшує розуміння змін, які відбуваються у вашому додатку. Але всі ці переваги супроводжуються певними проблемами. Деякі розробники стверджують, що Redux вводить непотрібні шаблони, що потенційно ускладнює прості завдання. Однак це залежить від архітектурних рішень проекту.

Redux використовується для підтримки та оновлення даних у ваших додатках для спільного використання декількома компонентами, залишаючись при цьому незалежним від них.

У великому додатку є потреба зберігати стан у центральному місці та ділитися ним між різними компонентами. Саме тут і з'являється сховище Redux:

2.4 Система керування версіями Git

Git[20] - це найпоширеніша система контролю версій. Git відстежує зміни, які ви вносите до файлів, тому у вас є запис про те, що було зроблено, і ви можете повернутися до певних версій, якщо вам коли-небудь знадобиться

Git працює локально, зберігаючи ваші файли та історію змін на вашому комп'ютері. Це означає, що ви можете працювати автономно, без необхідності постійного доступу до інтернету. Однак, для зберігання копій файлів та їхньої історії, ви можете використовувати онлайн-сервіс, такий як GitHub. Це особливо корисно при роботі в команді, оскільки забезпечує централізоване місце для зберігання та обміну змінами.

Спільна робота з GitHub значно полегшується завдяки централізованому репозиторію, куди всі члени команди можуть завантажувати свої зміни та завантажувати зміни інших. Це сприяє ефективній співпраці та координації роботи між розробниками.

Git також може автоматично об'єднувати зміни, що дозволяє двом людям працювати над різними частинами одного файлу, а потім з'єднувати ці зміни без

втрати роботи один одного. Це унікальна можливість значно підвищує продуктивність і спрощує процес розробки.

Сховище (або скорочено репозиторій) містить усі файли проекту та всю історію ревізій. Ви берете звичайну папку з файлами (наприклад, кореневу папку веб-сайту) і наказуєте Git зробити з неї репозиторій. Це створить підтеку .git, яка містить всі метадані Git для відстеження змін.

2.5 Середовище розробки Visual Studio Code

Visual Studio Code[21] - це безкоштовний багато функціональний редактор вихідного коду, який доступний для Windows, macOS, Linux та Raspberry Pi OS. Підтримує замовчуванням JavaScript, TypeScript та Node.js, але також можна і працювати з іншими мовами програмування. Має велику кількість різноманітних розширень, які можна встановлювати та налаштовувати відповідно до задач та вимог.

Можна виділити декілька переваг у використані Visual Studio Code:

1. Легкість Visual Studio Code як редактора в поєднанні з можливістю перевіряти синтаксис, завершувати код, рефакторити код, налагоджувати та перевіряти в репозиторії.

2. Хмарні можливості, підтримка основних хмар, Docker і Kubernetes

3. Повна інтеграція з GIT, що дає можливість роботи у команді та контролювати версіями проєктів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

В цьому розділі ми розглянемо програмну реалізацію веб-додатку з поділом на різні компоненти та архітектурні вирішення.

3.1 Архітектура веб-додатку

Архітектура веб-додатку розділена на дві основні частини: frontend та backend. Кожна з цих частин має свою структуру та відповідальність за певні аспекти розробки.

Frontend

У директорії frontend/src знаходиться основний код клієнтської частини веб-додатку. Основні піддиректорії та файли включають:

- types: В цій папці містяться TypeScript типи та інтерфейси, які використовуються в клієнтській частині.
- api: Тут знаходяться модулі, які взаємодіють з сервером за допомогою HTTP запитів.
- assets: У цій папці знаходяться ресурси, такі як зображення, шрифти тощо.
- components: Компоненти React, які використовуються для побудови користувацького інтерфейсу. Кожен компонент зазвичай має свою власну папку з файлами компоненту.
- constants: Тут містяться константи, такі як рядки тексту, кольори тощо.
- page-component: Ця папка містить компоненти, які представляють окремі сторінки веб-додатку.
- pages: Реалізація сторінок веб-додатку.
- redux: Конфігурація та файли стану Redux, які використовуються для керування станом додатка.

- routers: Файли для маршрутизації веб-додатку за допомогою React Router DOM.
- styles: CSS або SCSS файли для стилізації компонентів.
- utils: Допоміжні утиліти та функції, які можуть використовуватися в різних частинах додатку.
- app.tsx: Основний файл додатка, де відбувається ініціалізація та монтаж кореневого компоненту.
- index.tsx: Файл, який монтує кореневий компонент в DOM та запускає додаток.

Backend

У директорії backend/src знаходиться основний код серверної частини веб-додатку. Основні елементи включають:

- test: Тут знаходяться модулі для тестування серверної частини.
- types: TypeScript типи та інтерфейси, які використовуються в серверній частині.
- index.ts: Основний файл, де відбувається конфігурація та запуск сервера.
- api: Модулі, які надають API для взаємодії з клієнтською частиною.

3.1.1 Опис клієнтської частини додатку

Pages

У додатку існують три основні сторінки:

1. Home: Це головна сторінка додатку, яка може містити загальну інформацію, огляд функцій, або будь-яку іншу релевантну інформацію для користувача. Головна сторінка часто використовується для першого враження про додаток та навігації до інших сторінок.

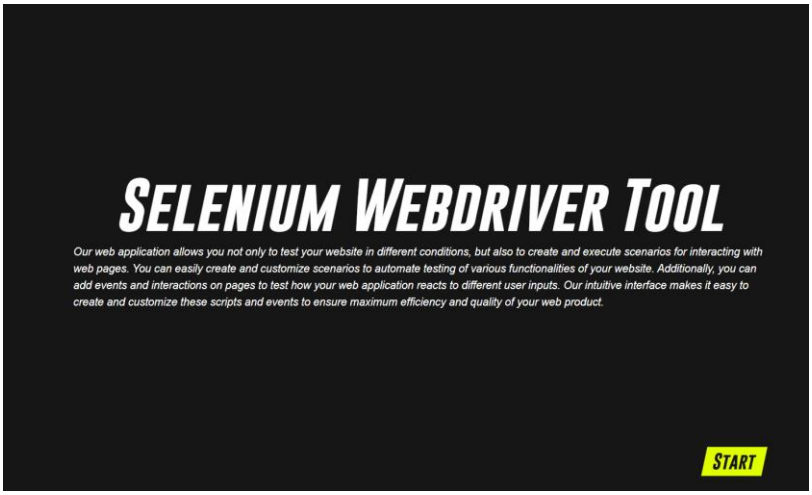

Рисунок 3.1 — Сторінка “Домашня”

2. User: Ця сторінка призначена для керування користувачем та правилами в додатку. У неї є підсторінки:

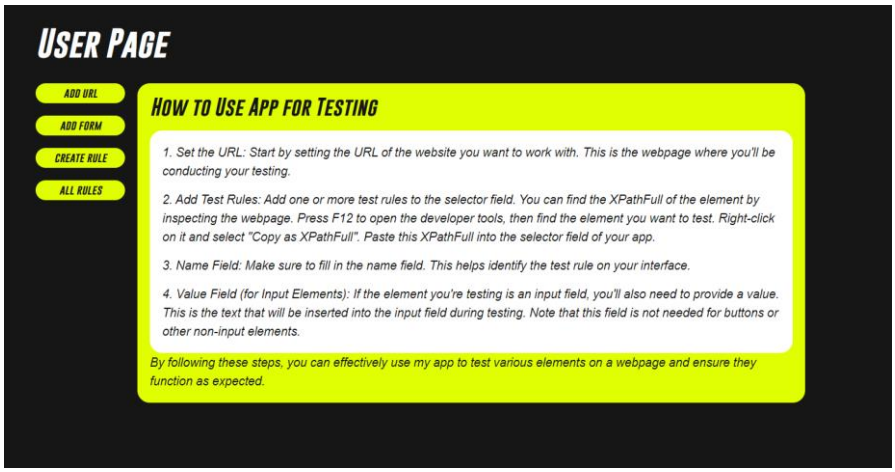

Рисунок 3.2 — Сторінка “Користувача”

- Add URL: Форма для додавання нового URL, що потрібно відстежувати або аналізувати.

Рисунок 3.3 — Сторінка “Додати посилання”

- Add Form: Форма для створення нової форми або шаблону, що буде використовуватися для збору даних.

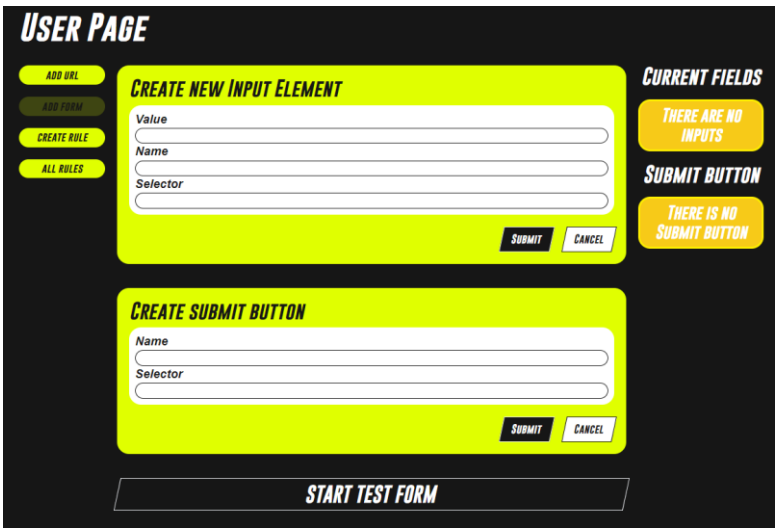

Рисунок 3.4 — Сторінка “Додати форму”

- Add Rule: Форма для додавання нового правила або умови, яка визначає, які дії слід виконати в певних ситуаціях.

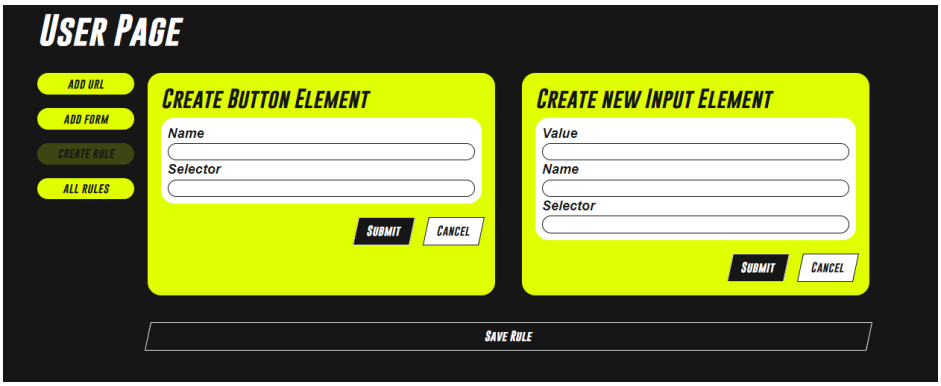

Рисунок 3.5 — Сторінка “Додати правило”

- All Rules: Сторінка, на якій відображаються всі правила, що визначені в системі. Це дозволяє користувачу переглядати, видаляти наявні правила.

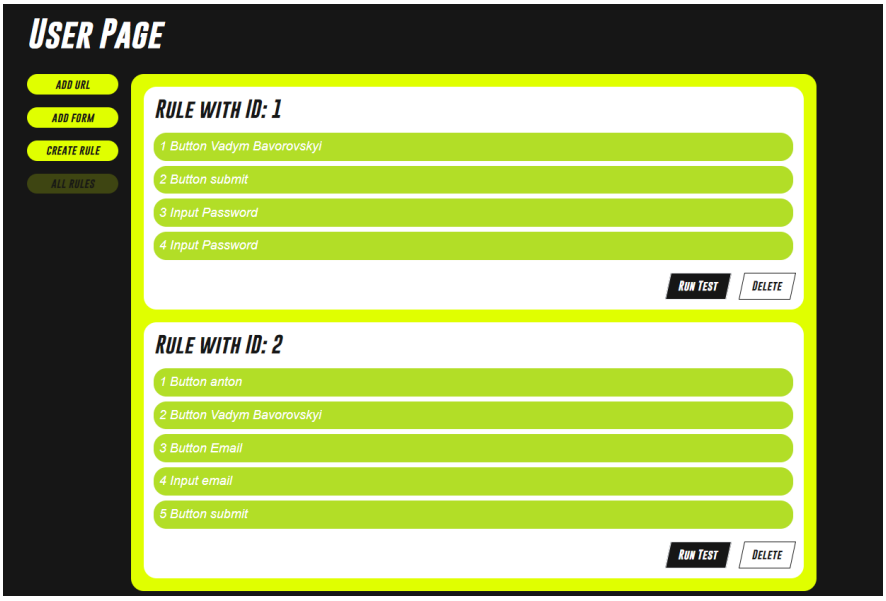

Рисунок 3.6 — Сторінка “Всі правила”

3. NotFound: Ця сторінка відображається, коли користувач звертається до неіснуючої сторінки. Вона надає повідомлення про помилку та можливості перейти на інші сторінки.

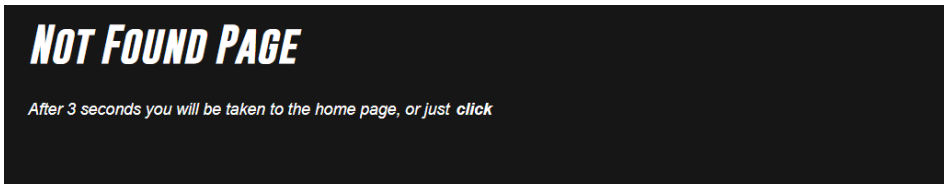

Рисунок 3.7 — Сторінка “Невідома”

Pages-Components

У директорії pages-components знаходяться компоненти, які використовуються для побудови сторінок. Деякі загальні компоненти, для User page включають:

- CreateButton: Цей компонент представляє кнопку, яка дозволяє користувачам створювати нові об'єкти або виконувати певні дії.
- CreateInput: Використовується для введення даних під час створення нового об'єкта або встановлення параметрів.
- EditInput: Цей компонент дозволяє користувачам редагувати існуючі дані або параметри.

Ці компоненти використовуються на відповідних сторінках для створення та редагування об'єктів, що використовуються в додатку. Їхнє використання

спрощує розробку та забезпечує функціонал інтерфейсу користувача.

3.1.2 Опис серверної частини додатку

У директорії test реалізовані функції, які використовуються для автоматизації тестування веб-додатка з використанням Selenium WebDriver. Ось деякі особливості цих функцій:

1. testRule: Ця функція виконує тестування правила на веб-сторінці. Вона приймає URL-адресу та об'єкт правила, який містить дані про елементи сторінки та їхні значення. Функція виконує послідовність дій на сторінці відповідно до визначеного правила.
2. testForm: Ця функція призначена для тестування форм на веб-сторінках. Вона отримує URL-адресу та масив даних форми, які потрібно ввести. Функція автоматично вводить ці дані у відповідні поля форми та натискає кнопку відправки.

Ці функції дозволяють автоматизувати процес тестування веб-додатка, забезпечуючи швидке та ефективне перевірку функціоналу.

API-маршрути (API Routes)

У директорії api реалізовані API-маршрути для взаємодії з серверною частиною веб-додатка. Основні можливості API включають:

1. Робота з URL-адресами: API дозволяє зберігати та отримувати URL-адреси для використання в тестуванні. Це дозволяє легко змінювати адреси та повторно використовувати їх у тестах.
2. Маніпуляція введеними даними: Користувач може зберігати, видаляти та оновлювати дані полів форми за допомогою API. Це дозволяє здійснювати тестування з різними наборами вхідних даних.
3. Маніпуляція правилами: API також дозволяє зберігати, видаляти та оновлювати правила тестування. Це дозволяє налаштовувати тести згідно з потребами та змінювати їх у відповідності до змін у веб-додатку.

Ці API-маршрути забезпечують зручний інтерфейс для взаємодії з сервером та керування тестами. Вони дозволяють легко виконувати тестування та змінювати

його параметри у відповідності до потреб користувача.

Більш детальніше ми їх розглянемо коли будемо розглядати, як взаємодіють клієнтська та серверна частини між собою.

3.2 Взаємодія клієнтської та серверної частин

Під час роботи веб-додатка його дві частини постійно мають зв’язок. Це спілкування відбувається за допомогою згаданого вище, HTTP протоколу. Веб-додатки, які розробляються з використанням сучасних технологій, потребують надійного та ефективного механізму для взаємодії між клієнтською та серверною частинами. Для цього широко використовуються різні бібліотеки та інструменти, що забезпечують зручність у створенні та обробці запитів.

На клієнтській частині додатку ми створюємо HTTP-запити за допомогою бібліотеки Axios. Axios є популярною бібліотекою для здійснення HTTP-запитів з браузера або Node.js, яка забезпечує простий і зрозумілий інтерфейс для відправки запитів і обробки відповідей. Кожен HTTP-запит, створений на клієнтській частині, відправляється до серверної частини, де він обробляється, і відповідно повертається результат у вигляді повідомлення.

Також варто сказати, що сервер може повернути помилку, якщо він знайшов відповідний протокол для взаємодії з переданим API. Тому для таких випадків була розроблена система обробки помилок, яка дозволяє серверу продовжувати роботу після виникнення помилок і підтримувати зв’язок з клієнтом. Це забезпечує стабільність і безперервність роботи веб-додатку, навіть у разі непередбачених обставин.

На клієнті, так само як і на сервері, реалізовано обробку помилок. Це важливий аспект з точки зору безпеки, оскільки пряме відображення помилок з серверу може бути небезпечним через можливість розкриття конфіденційної інформації або структури системи стороннім зловмисникам. Обробка помилок на клієнтській частині дозволяє коректно відобразити користувачеві повідомлення про помилку без розкриття технічних деталей.

Перший файл який ми подивимось знаходиться на стороні клієнта це url-api.ts у ньому зберігається об’єкт, з усіма можливими API, які можна використовувати:

```
export const API = {
  GET_INPUTS: `${BASE_URL}/get-inputs`,
  POST_INPUT: `${BASE_URL}/post-input`,
  DELETE_INPUT: `${BASE_URL}/delete-input`,
  UPDATE_INPUT: `${BASE_URL}/update-input`,

  GET_RULES: `${BASE_URL}/get-rules`,
  POST_RULE: `${BASE_URL}/post-rule`,
  DELETE_RULE: `${BASE_URL}/delete-rule`,

  GET_SUBMIT_BTN: `${BASE_URL}/get-submitBtn`,
  POST_SUBMIT_BTN: `${BASE_URL}/post-submitBtn`,

  GET_URL: `${BASE_URL}/get-url`,
  POST_URL: `${BASE_URL}/post-url`,

  TEST_FORM: `${BASE_URL}/test-form`,
  TEST_RULE: `${BASE_URL}/test-rule`,
}
```

Рисунок 3.8 — Файл url-api.ts

Також на клієнтській частині знаходиться файл client.ts тут описані всі функції за допомогою яких виконуємо запити на сервер. У цьому файлі реалізована обробка помилок які у нас можуть виникнути під час запиту на сервер.

Розглянемо цей файл по частинам:

Файл починається з того, що підключаємо бібліотеку Axios та допоміжні файли, такі як вище згаданий url-api.ts та інші.

```
import axios from "axios";
import { API } from "./url-api";
import { InputElement } from "src/@types/inputElement";
import { UrlObj } from "src/@types/url";
import { AllRules, Rule } from "src/@types/rule";
```

Рисунок 3.9 — Файл client.ts imports

Наступна йде функція за допомогою якої ми моделюємо затримку (ніби ми відправляємо запит не на локальний сервер), це потрібно для того, щоб ми могли побачити UI компонент такий як, спінер при загрузці наших даних.

```
function wait(delay: number) {
  return new Promise(resolve => setTimeout(resolve, delay));
}
```

Рисунок 3.10 — Файл client.ts wait() function

Далі йдуть загальні функції роблять без посередньо HTTP-запит до сервера, вони реалізовані за допомогою бібліотеки axios та використовують її методи такі, як get, post, put.

```
async function getData<T>(url: string): Promise<T> {
  await wait(1500);

  try {
    const response = await axios.get<T>(url);
    return response.data;
  } catch (error) {
    console.error(`Error fetching data from ${url}:`, error);
    throw error;
  }
}

async function postData<T>(url: string, data?: T): Promise<void> {
  try {
    await axios.post(url, data);
  } catch (error) {
    console.error(`Error posting data to ${url}:`, error);
    throw error;
  }
}

async function putData<T>(url: string, data: T): Promise<void> {
  try {
    await axios.put(url, data);
  } catch (error) {
    console.error(`Error updating input with id ${url}:`, error);
    throw error;
  }
}
```

Рисунок 3.11 — Файл client.ts axios functions

До цих функцій передаються відповідні url запити з файлу url-api.ts, до прикладу розглянемо одну з функції, яка повертає результат виконання функції getData

```
export async function getInputElementsFromServer(): Promise<Rule> {
  return getData<Rule>(API.GET_INPUTS);
}
```

Рисунок 3.12 — Файл client.ts getData() function

Таким чином ми викликаємо функцію getInputElementsFromServer() у тому місці, де ми хочемо отримати доступ до наших елементів, після чого ми їх відображаємо або виконуємо інші маніпуляції.

Тепер перейдемо до серверної частини додатку. Файл api.ts тут ми обробляємо наші вхідні запити, що йдуть з клієнтської частини додатку. Розглянемо на прикладі даних, Input що використовуються при виконані тесту для перевірки форми.

Для кожного типу запити реалізована свою функція, яка передається у відповідний метод (post, get, delete, put) другим параметром. Першим параметром ми передаємо сам url, який потрібно використовувати при запиті до сервера.

Відповідно до кожного типу даних, який ми використовуємо, створена подібна реалізація.

Метод post, приймає першим параметром '/post-input', а друг callback функцію, яка у свою чергу приймає два обє’кта req – запит та res – відповідь. Записуємо у змінну data наші передані input, потім додаємо їх до наших збережених на сервері storeInputs, виводимо повідомлення про успішне збереження та повертаємо відповідь на клієнт.

```
app.post('/post-input', (req, res) => {
  const data: Input = req.body;
  storedInputs.push(data);

  console.log('Input saved:', data);
  res.send('Input saved successfully');
});
```

Рисунок 3.13 — Файл api.ts Inputs метод post

Метод get приймає першим параметром рядок '/get-inputs', а другим - callback функцію, яка отримує два об'єкти: req (запит) та res (відповідь). Він повертає усі збережені введені дані у форматі JSON та виводить ці дані у консоль.

```
app.get('/get-inputs', (req, res) => {
  res.json(storedInputs);
  console.log('Inputs get:', storedInputs);
});
```

Рисунок 3.14 — Файл api.ts Inputs метод get

Метод delete приймає першим параметром рядок '/delete-input/:id', а другим - callback функцію з об'єктами req та res. Спочатку, з параметрів запиту витягується id введеного даних, після чого проводиться пошук індексу цього введеного даних у масиві storedInputs. Якщо введене дані знайдено, воно видаляється з масиву, виводиться повідомлення про успішне видалення у консоль та відповідь відправляється на клієнт. Якщо ж введене дані не знайдено, виводиться повідомлення про відсутність даних та відправляється статус 404.

```
app.delete('/delete-input/:id', (req, res) => {
  const id = parseInt(req.params.id);

  const index = storedInputs.findIndex(input => input.id === id);

  if (index !== -1) {
    storedInputs.splice(index, 1);
    console.log('Input deleted:', id);
    res.send(`Input with id ${id} deleted successfully`);
  } else {
    console.log(`Input with id ${id} not found`);
    res.status(404).send(`Input with id ${id} not found`);
  }
});
```

Рисунок 3.15 — Файл api.ts Inputs метод delete

Метод put приймає рядок '/update-input/:id' та callback функцію. З параметрів запиту витягується id, а з тіла запиту - оновлені дані. Проводиться пошук індексу введеного даних у масиві storedInputs. Якщо введене дані знайдено, його значення оновлюється, виводиться повідомлення у консоль та відправляється відповідь на клієнт. Якщо ж введене дані не знайдено, виводиться повідомлення про відсутність даних та відправляється статус 404.

```
app.put('/update-input/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const updatedInput: Input = req.body;

  const index = storedInputs.findIndex(input => input.id === id);

  if (index !== -1) {
    storedInputs[index] = updatedInput;
    console.log('Input updated:', id, updatedInput);
    res.send(`Input with id ${id} updated successfully`);
  } else {
    console.log(`Input with id ${id} not found`);
    res.status(404).send(`Input with id ${id} not found`);
  }
});
```

Рисунок 3.16 — Файл api.ts Inputs put

Таким чином, взаємодія клієнтської та серверної частин веб-додатку забезпечується за допомогою HTTP-запитів, оброблених бібліотекою Axios на клієнті та відповідних методів на сервері. Це дозволяє ефективно обробляти дані та забезпечувати безперервну роботу додатку, навіть у разі виникнення помилок.

Дотримуючись принципів безпеки, обробка помилок реалізована на обох частинах додатку, що зменшує ризик витоку конфіденційної інформації та підвищує загальний рівень захисту від потенційних загроз. Завдяки структурованому підходу до реалізації функцій, процес взаємодії між клієнтом та сервером залишається прозорим та ефективним, що сприяє стабільній роботі веб-додатку та покращує користувацький досвід.

4. ДЕМОНСТРАЦІЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

Розробка веб-додатка передбачає не лише створення функціональних модулів, але й їх демонстрацію та оцінку кінцевих результатів. Для забезпечення надійності та ефективності роботи продукту важливо провести інсталяцію програмного забезпечення, а також перевірку його роботи на відповідність системним вимогам.

4.1 Інсталяція програмного забезпечення та вимоги до системних характеристик

Для того, щоб веб-додаток працював стабільно необхідно дотримуватися рекомендацій щодо його встановлення та використання. Також потрібно враховувати характеристики комп’ютера на якому буде використовуватися веб-додаток.

4.1.1 Системні характеристики

Для встановлення веб-додатку на пристрої мають виконуватись такі технічні умови:

— вільне місце на диску в розмірі ~ 460 Мб

— оперативна пам’ять ~ 4 Гб

— операційна система Windows або macOS

4.1.2 Інсталяція програмного забезпечення

Для того, щоб встановити веб-додаток першим кроком потрібно встановити декілька додаткових пакетів, Node.js[22] та Git[23]. Їх можна встановити за відповідними посиланням.

Також для зручної роботи нам потрібно встановити два пакет менеджера для Node.js. Це NPM[24] та YARN[25], відповідно їх також можна встановити за

вказаними посиланням там і знайдете інструкцію. Встановити також потрібно Visual Studio Code [26].

Після виконання підготовчих кроків переходимо до встановлення веб-додатку. Для цього створіть папку на своєму комп'ютері, важливо щоб папка мала шлях без кириличних та спеціальних символів та без пробілів у назвах кореневих папках.

Відкрийте термінал у створеній папці та введіть команду git clone та додайте посилання на віддалений репозиторій[27] (в кінці допишіть .git).

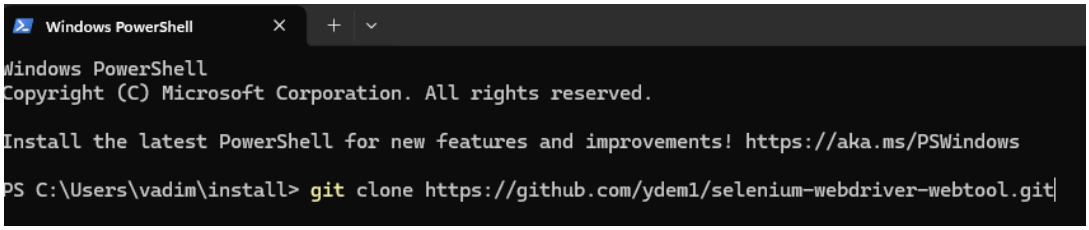

Рисунок 4.1 — Термінал (git clone)

Після успішної загрузки введіть команду "code .". Після чого відкриється Visual Studio Code.

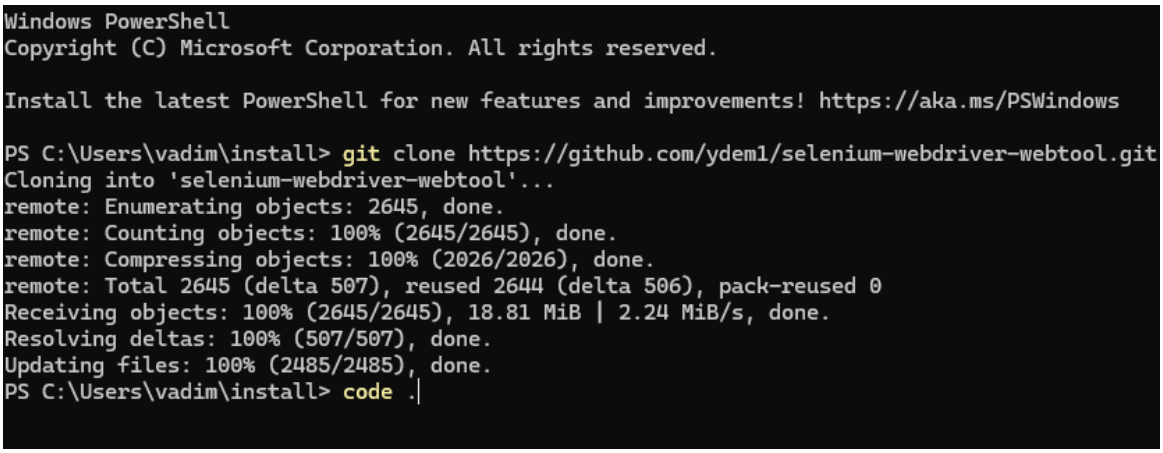

Рисунок 4.2 — Термінал (code .)

У Visual Studio Code вам потрібно встановити всі залежності та відповідні бібліотеки за допомогою пакет менеджерів npm та yarn. Для цього відкрийте термінал вже у Visual Studio Code за допомогою клавіш Ctrl + Shifth + `.

У терміналі введіть cd selenium-webdriver-webtool/backend/ ця команда відкриває термінал у нашій серверній частині (backend) веб-додатку

```
vadim@mi-notebook-pro MINGW64 ~/install
$ cd selenium-webdriver-webtool/backend/

vadim@mi-notebook-pro MINGW64 ~/install/selenium-webdriver-webtool/backend (main)
$
```

Рисунок 4.3 — Термінал VS Code (cd)

Далі вводимо команду “npm i “, встановлюємо всі залежності.

```
$ npm i

up to date, audited 128 packages in 999ms

26 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 4.4 — Термінал VS Code для сервера (npm i)

Після цього запускаємо наш сервер за допомогою команди npm start. Після чого побачимо повідомлення про, що сервер запущений.

```
vadim@mi-notebook-pro MINGW64 ~/install/selenium-webdriver-webtool/backend (main)
$ npm start

> backend@1.0.0 start
> tsc && node dist/index.js

Server is running on port 5000
```

Рисунок 4.5 — Термінал VS Code для сервера (npm start)

Та аналогічно робимо для клієнтської частини (frontend) веб-додатку. Спочатку вводимо cd selenium-webdriver-webtool/frontend_new/, але для встановлення залежності використовуємо команду yarn install. А для запуску yarn start.

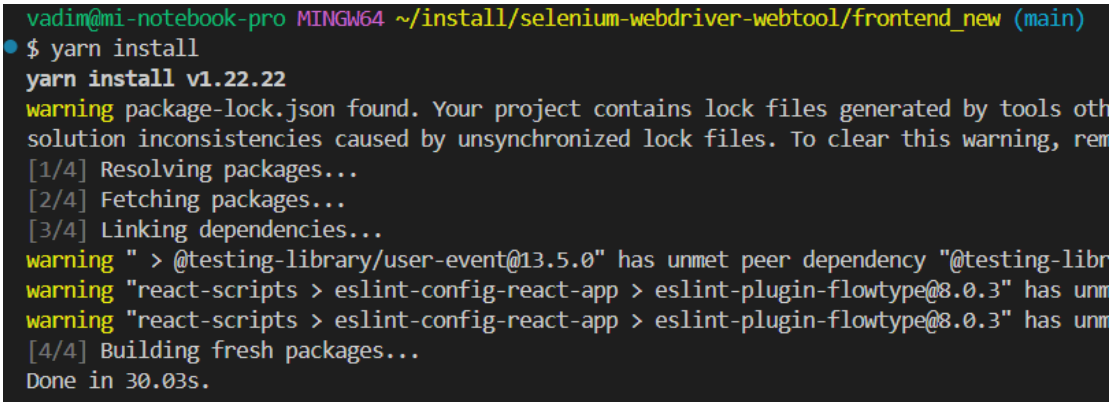

Рисунок 4.6 — Термінал VS Code для клієнта (yarn install)

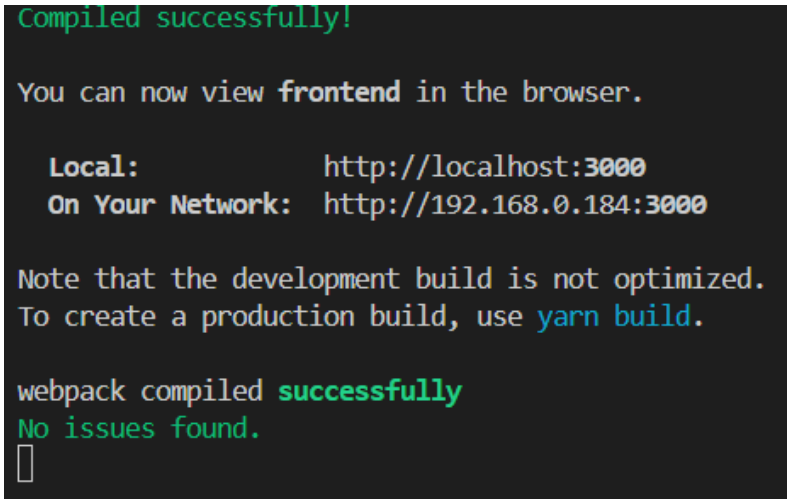

Рисунок 4.7 — Термінал VS Code для клієнта (result)

Дотримуючись цих інструкцій, ви забезпечите коректну роботу веб-додатку та уникнете можливих проблем, пов’язаних із невідповідністю системних вимог або неправильною інсталяцією. Надалі, при виникненні питань чи проблем, рекомендується звертатися до офіційної документації або підтримки відповідних інструментів та пакетів.

4.2 Демонстрація функціоналу додатку

Розроблений веб-додаток для автоматизації тестування веб-додатків має зручний та інтуїтивно зрозумілий інтерфейс. Головна мета цього інтерфейсу — забезпечити легкий доступ до функцій тестування, зокрема натискання на кнопки, введення даних у поля та робота з формами. Це дозволяє користувачам

без знань програмування ефективно використовувати інструмент для автоматизованого тестування.

1. Введення URL-адреси: Користувач вводить URL-адресу веб-сайту, з яким він бажає працювати, та зберігає її у додатку.

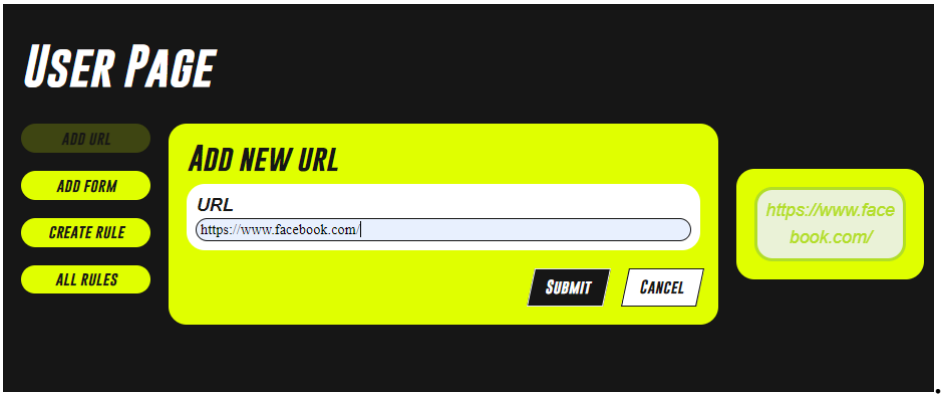

Рисунок 4.8 — Введення URL-адреси

2. Збереження URL-адреси: Сервер отримує введену URL-адресу від клієнта та зберігає її. Після успішного збереження сервер відправляє відповідь про успішне отримання даних.
3. Створення правила або форми: Користувач може створити правило для тестування конкретного сценарію на сторінці веб-сайту. Це правило включає послідовність дій, які потрібно виконати на сторінці. Також користувач може створити форму для тестування, що включає в себе поля для введення даних та кнопку відправки.

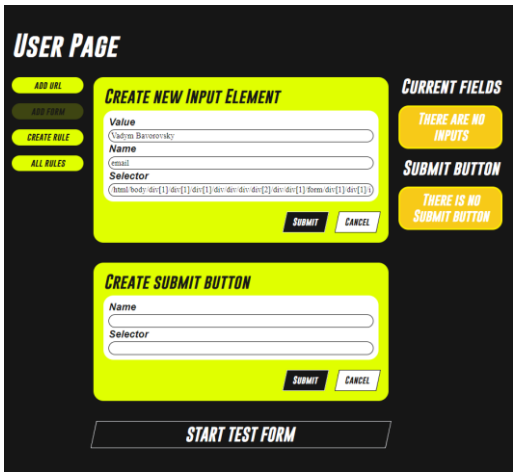

Рисунок 4.9 — Створення форми

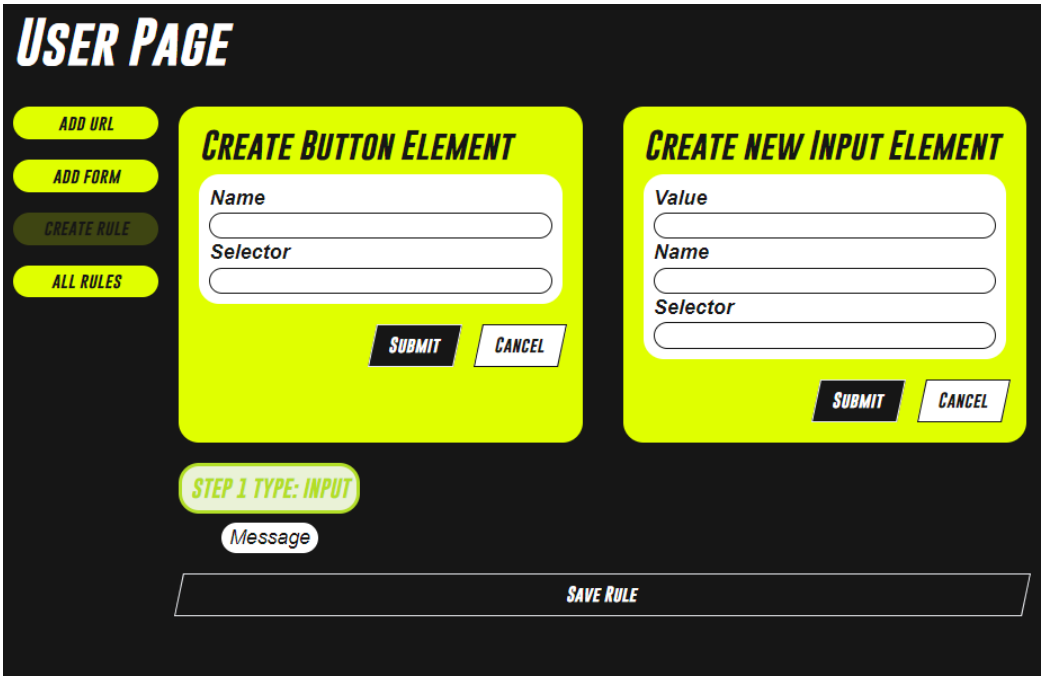

Рисунок 4.10 — Створення правила

4. Збереження правила або форми: Сервер отримує від клієнта створене правило або форму та зберігає їх. Після успішного збереження сервер відправляє відповідь про успішне збереження даних.
5. Редагування полів: Після створення форми користувач має можливість редагувати поля для введення даних, налаштовуючи їх під свої потреби.

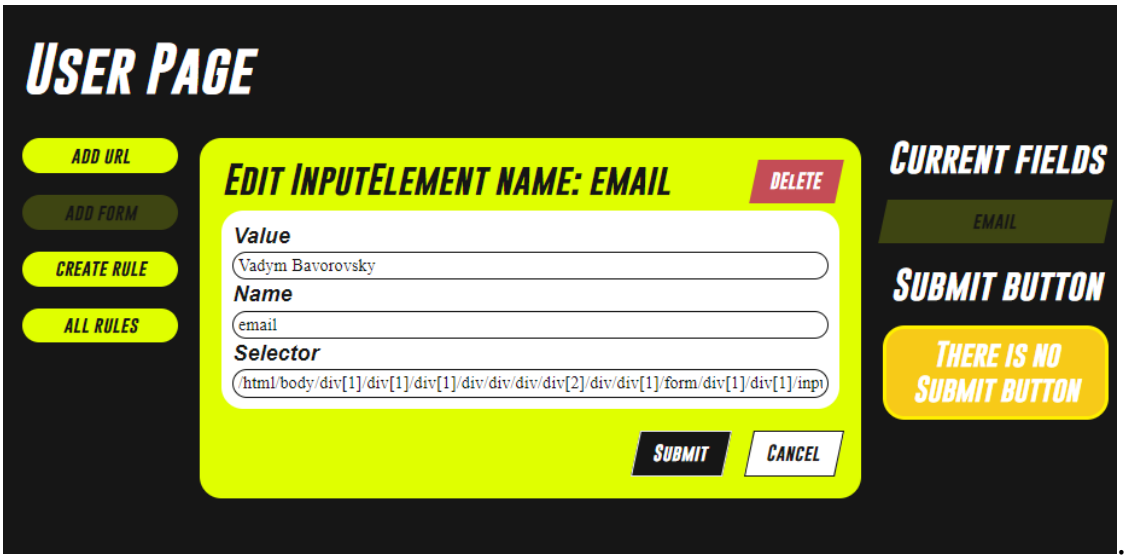

Рисунок 4.11 — Редагування полів форми

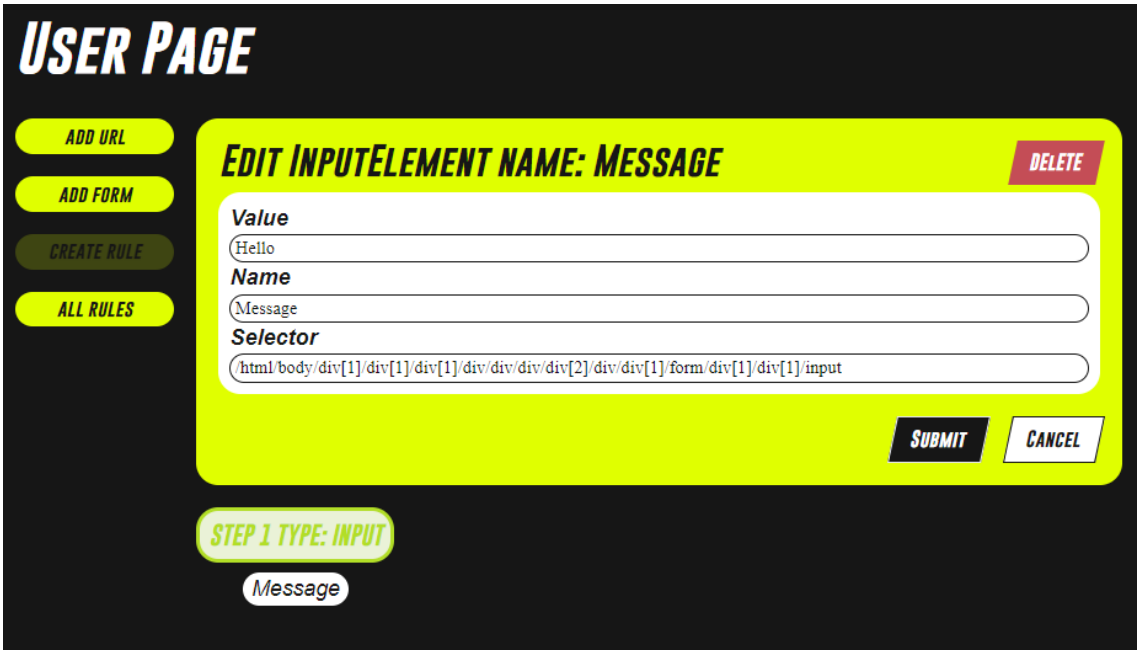

Рисунок 4.12 — Редагування полів правило

6. Запуск тесту: Після збереження правила або форми користувач може запустити тест. Для цього додаток використовує Selenium WebDriver, щоб відкрити веб-сторінку та виконати вказані дії згідно з правилом чи формою.
7. Виконання тесту: Selenium WebDriver перевіряє веб-сторінку на відповідність зазначеним у правилі діям. Якщо елемент, що потрібно знайти, не знайдено на сторінці, виводиться повідомлення про помилку. Якщо тест успішно пройшов, процес продовжується.

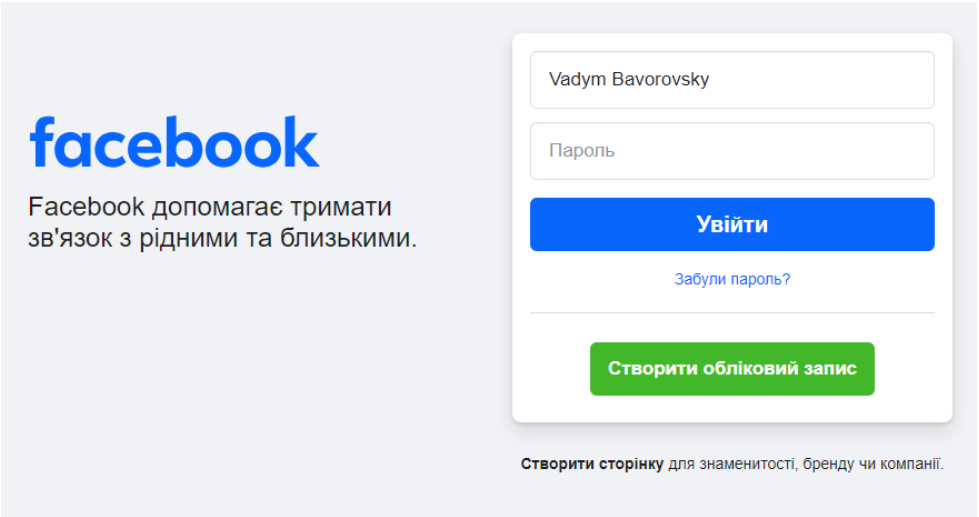

Рисунок 4.13 — Результат виконання тесту

Цей підхід до автоматизації тестування веб-додатків надає можливість значно спростити та прискорити процес перевірки функціональності веб-сайтів. Завдяки інтуїтивно зрозумілому інтерфейсу, навіть користувачі без спеціальних знань у сфері програмування можуть налаштовувати та проводити тести, що робить наш інструмент доступним для широкого кола користувачів.

У подальшому можливі покращення та розширення функціоналу додатку, зокрема додавання нових можливостей для автоматизації тестування, що дозволить охопити ще більше аспектів роботи веб-додатків.

ВИСНОВКИ

У процесі практики було створено веб-додаток для автоматизації тестування веб-додатків, який використовує бібліотеку Selenium WebDriver. Основною метою цього проекту було створення інструменту, який би значно спрощував процес тестування шляхом автоматизації рутинних завдань, які зазвичай виконуються вручну.

Цей розроблений додаток надає користувачам зручний інтерфейс, який дозволяє легко вводити URL-адреси веб-сайтів, створювати тестові сценарії, редагувати їх та запускати тести. Завдяки використанню Selenium WebDriver, стало можливим автоматизувати взаємодію з веб-сторінками, включаючи введення даних у форми, натискання кнопок та перевірку результатів. Ця автоматизація не тільки знижує ризик людських помилок, але й скорочує час на проведення тестів, підвищуючи загальну ефективність процесу тестування.

Додаток є гнучким і масштабованим завдяки використанню сучасних фреймворків і бібліотек, таких як React, Redux, Tailwind CSS та SCSS. Це дозволяє легко адаптувати його до змін та розширювати для підтримки нових функцій. Важливою перевагою є можливість інтеграції з системами безперервної інтеграції та доставки (CI/CD), що дозволяє автоматично запускати тести при кожному оновленні коду. Крім того, користувачі можуть редагувати створені правила та форми, а також використовувати їх повторно для тестування після змін на веб-сайті, що забезпечує ефективність і зручність у підтримці тестів.

Проект продемонстрував значні покращення у процесі тестування веб-додатків, забезпечивши автоматизацію багатьох рутинних завдань та підвищивши ефективність роботи QA-команди. Використання сучасних технологій та бібліотек дозволило створити гнучкий і зручний інструмент, який може бути легко адаптований під різні потреби проектів. Недоліками реалізованого рішення є потреба у початкових знаннях для налаштування інструментів автоматизації та обмежена підтримка специфічних сценаріїв, що можуть вимагати додаткової адаптації. В цілому, реалізований проект став

важливим кроком у вдосконаленні процесів тестування веб-додатків, забезпечивши підвищення якості та стабільності програмного забезпечення.

А тепер додам ще більше інформації для повнішого розуміння. Додаток розроблено з урахуванням потреб різних користувачів, від розробників до тестувальників, які можуть мати різні рівні технічних знань. Для розробників додаток пропонує можливість глибокої інтеграції з існуючими CI/CD пайплайнами, що дозволяє безперервно моніторити якість коду та швидко виявляти дефекти. Тестувальники, з іншого боку, оцінять можливість створення складних тестових сценаріїв за допомогою зручного графічного інтерфейсу, який мінімізує необхідність написання коду вручну.

Щоб додатково підвищити зручність користування, додаток підтримує функцію запису і відтворення дій користувача на веб-сторінці. Це дозволяє тестувальникам записати свої дії на сайті та автоматично згенерувати тестовий сценарій на основі цих дій. Такий підхід значно скорочує час на створення тестів та знижує ймовірність пропущених перевірок.

Значну увагу було приділено і безпеці. Додаток забезпечує захищене зберігання даних тестування, використовуючи сучасні методи шифрування. Це особливо важливо для проектів, де тести включають конфіденційні дані або проводяться у середовищах з високими вимогами до безпеки.

Проект продемонстрував значні покращення у процесі тестування веб-додатків, забезпечивши автоматизацію багатьох рутинних завдань та підвищивши ефективність роботи QA-команди. Використання сучасних технологій та бібліотек дозволило створити гнучкий і зручний інструмент, який може бути легко адаптований під різні потреби проектів. Недоліками реалізованого рішення є потреба у початкових знаннях для налаштування інструментів автоматизації та обмежена підтримка специфічних сценаріїв, що можуть вимагати додаткової адаптації. В цілому, реалізований проект став важливим кроком у вдосконаленні процесів тестування веб-додатків, забезпечивши підвищення якості та стабільності програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Qalight. Ручне та автоматизоване тестування. URL: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/ (дата звернення: 01.05.2024).
2. Lemon School. Види тестування програмного забезпечення. URL: https://lemon.school/blog/vydy-testuvannya-programnogo-zabezpechennya (дата звернення: 02.05.2024).
3. Qalight. Модульне тестування. URL: https://qalight.ua/baza-znaniy/modulne-testuvannya/ (дата звернення: 03.05.2024).
4. Qalight. Інтеграційне тестування. URL: https://qalight.ua/baza-znaniy/integratsijne-testuvannya/ (дата звернення: 04.05.2024).
5. Qalight. Системне тестування. URL: https://qalight.ua/baza-znaniy/sistemne-testuvannya/ (дата звернення: 05.05.2024).
6. Qalight. Приймальне тестування. URL: https://qalight.ua/baza-znaniy/prijmalne-testuvannya/ (дата звернення: 06.05.2024).
7. React Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 07.05.2024).
8. Typescript. URL: https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-0.html (дата звернення: 08.05.2024).
9. Worldwide developer survey: most used frameworks for web. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (дата звернення: 09.05.2024).
10. Рейтинг мов програмування 2024. URL: https://dou.ua/lenta/articles/language-rating-2024/ (дата звернення: 10.05.2024).
11. Selenium WebDriver. URL: https://www.selenium.dev/ (дата звернення: 11.05.2024).
12. Selenium WebDriver Tutorial. URL: https://www.browserstack.com/guide/selenium-webdriver-tutorial (дата звернення: 12.05.2024).

13. React Router DOM. URL: https://reactrouter.com/en/main (дата звернення: 13.05.2024).

14. UUID. URL: https://www.npmjs.com/package/uuid (дата звернення: 14.05.2024).

15. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 15.05.2024).

16. SCSS Documentation. URL: https://sass-lang.com/documentation (дата звернення: 16.05.2024).

17. classNames Documentation. URL: https://www.npmjs.com/package/classnames (дата звернення: 17.05.2024).

18. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 18.05.2024).

19. Redux Documentation. URL: https://redux.js.org/introduction/getting-started (дата звернення: 19.05.2024).

20. Git. URL: https://www.git-scm.com/ (дата звернення: 20.05.2024).

21. Documentation for Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення: 21.05.2024).

22. Node.js installation. URL: https://nodejs.org/en/download/package-manager (дата звернення: 22.05.2024).

23. Git installation. URL: https://www.git-scm.com/downloads (дата звернення: 23.05.2024).

24. Downloading and installing Node.js and npm. URL: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm (дата звернення: 24.05.2024).

25. Yarn installation. URL: https://classic.yarnpkg.com/lang/en/docs/install/#windows-stable (дата звернення: 25.05.2024).

26. Visual Studio Code installation. URL: https://code.visualstudio.com/download (дата звернення: 26.05.2024).

27. Віддалений репозиторій на веб-додаток Selenium Webdriver. URL: https://github.com/ydem1/selenium-webdriver-webtool (дата звернення: 27.05.2024).

ДОДАТОК А

Реалізація логіки взаємодії клієнтської та серверної частин для запуску тестів з використання Selenium Webdriver

Текст програми

Аркушів 12

Київ 2024

```
import express from 'express';
import cors from 'cors';
import { testForm } from './test/test-form.js';
import { Input } from './types/input.js';
import { AllRules } from './types/rule.js';
import { testRule } from './test/test-rule.js';

const app = express();
const PORT = 5000;

app.use(cors());
app.use(express.json());

// Url
let storedUrl = "";

app.post('/post-url', (req, res) => {
  const { url } = req.body;

  storedUrl = url;

  console.log('Url saved:', url);
  res.send('Url saved successfully');
});

app.get('/get-url', (req, res) => {
  res.json(storedUrl);
  console.log('Url get:', storedUrl);
});
```

```
// Inputs
let storedInputs: Input[] = [];

app.post('/post-input', (req, res) => {
  const data: Input = req.body;
  storedInputs.push(data);

  console.log('Input saved:', data);
  res.send('Input saved successfully');
});

app.get('/get-inputs', (req, res) => {
  res.json(storedInputs);
  console.log('Inputs get:', storedInputs);
});

app.delete('/delete-input/:id', (req, res) => {
  const id = parseInt(req.params.id);

  const index = storedInputs.findIndex(input => input.id === id);

  if (index !== -1) {
    storedInputs.splice(index, 1);
    console.log('Input deleted:', id);
    res.send(`Input with id ${id} deleted successfully`);
  } else {
    console.log(`Input with id ${id} not found`);
    res.status(404).send(`Input with id ${id} not found`);
  }
});
```

```
app.put('/update-input/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const updatedInput: Input = req.body;

  const index = storedInputs.findIndex(input => input.id === id);

  if (index !== -1) {
    storedInputs[index] = updatedInput;
    console.log('Input updated:', id, updatedInput);
    res.send(`Input with id ${id} updated successfully`);
  } else {
    console.log(`Input with id ${id} not found`);
    res.status(404).send(`Input with id ${id} not found`);
  }
});

// submit button
let storedSubmitBtn: Input = {
  id: 0,
  name: '',
  selector: '',
};

app.post('/post-submitBtn', (req, res) => {
  const data: Input = req.body;

  storedSubmitBtn = data;

  console.log('Submit button saved:', storedSubmitBtn);
  res.send('Submit button saved successfully');
});
```

```
app.get('/get-submitBtn', (req, res) => {
  res.json(storedSubmitBtn);
  console.log('Submit button get:', storedSubmitBtn);
});

// Array rules
let storedRules: AllRules[] = [];

app.post('/post-rule', (req, res) => {
  const data: AllRules = req.body;
  storedRules.push(data);

  console.log('Rule saved:', data);
  res.send('Rule saved successfully');
});

app.get('/get-rules', (req, res) => {
  res.json(storedRules);
  console.log('Rules get:', storedRules);
});

app.delete('/delete-rules/:id', (req, res) => {
  const id = parseInt(req.params.id);

  const index = storedRules.findIndex(input => input.id === id);

  if (index !== -1) {
    storedRules.splice(index, 1);
    console.log('Rule deleted:', id);
    res.send(`Rule with id ${id} deleted successfully`);
```

```
  } else {
    console.log(`Rule with id ${id} not found`);
    res.status(404).send(`Rule with id ${id} not found`);
  }
});

// test form
app.post('/test-form', async (req, res) => {
  try {
    await testForm(storedUrl, storedInputs, storedSubmitBtn)

    console.log(`
      testForm run with url: ${storedUrl}
      and Inputs: ${storedInputs.map(input => input.name)}
      and Submit Button ${storedSubmitBtn.name}
      `);

    res.send('Page is accessible');
  } catch (error) {
    res.status(500).send('Page is not accessible');
  }
});

// test rule
app.post('/test-rule', async (req, res) => {
  const { id } = req.body;

  const ruleId = parseInt(id);

  const currentRule = storedRules.find(rule => rule.id === ruleId).rule;
```

```
  if (!currentRule) {
    return res.status(404).send('Rule not found');
  }

  try {
    await testRule(storedUrl, currentRule);

    console.log(`
      testRule run with url: ${storedUrl}
      and Inputs: ${currentRule.map(input => input.name)}
    `);

    res.send('Page is accessible');
  } catch (error) {
    res.status(500).send('Page is not accessible');
  }
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

```
import { Builder, By } from "selenium-webdriver";
import { Rule } from "../types/rule.js";
export async function testRule(url: string, data: Rule) {
  const driver = await new Builder().forBrowser('chrome').build();
  try {
    await driver.manage().window().maximize();

    await driver.get(url);

    for (const element of data) {
      try {
        const inputElement = await driver.findElement(By.xpath(element.selector));

        if (element.value !== undefined) {
          await inputElement.sendKeys(element.value);
        } else {
          inputElement.click();
        }

        await new Promise(resolve => setTimeout(resolve, 3000));
      } catch (error) {
        console.error(`Error filling form element:`, error);
        throw new Error(`Error filling form element with selector ${element.selector}:
${error.message}`);
      }
    }

  } catch (error) {
    console.error('Error in testRule:', error);
    throw new Error(error);
  }}
```

```
import { Builder, By } from "selenium-webdriver";
import { Input } from "../types/input.js";

export async function testForm(url: string, formDataArray: Input[], formSubmitBtn:
Input) {
  const driver = await new Builder().forBrowser('chrome').build();
  try {
    await driver.get(url);

    for (const formData of formDataArray) {
      const inputElement = await driver.findElement(By.xpath(formData.selector));

      await inputElement.sendKeys(formData.value);
    }

    const submitBtn = await driver.findElement(By.xpath(formSubmitBtn.selector));

    submitBtn.click();

  } catch (error) {
    console.error(`Error filling form element:`, error);
  }
}
```

```
import axios from "axios";
import { API } from "./url-api";
import { InputElement } from "src/@types/inputElement";
import { UrlObj } from "src/@types/url";
import { AllRules, Rule } from "src/@types/rule";

function wait(delay: number) {
  return new Promise(resolve => setTimeout(resolve, delay));
}

async function getData<T>(url: string): Promise<T> {
  await wait(1500);

  try {
    const response = await axios.get<T>(url);
    return response.data;
  } catch (error) {
    console.error(`Error fetching data from ${url}:`, error);
    throw error;
  }
}

async function postData<T>(url: string, data?: T): Promise<void> {
  try {
    await axios.post(url, data);
  } catch (error) {
    console.error(`Error posting data to ${url}:`, error);
    throw error;
  }
}
```

```
async function putData<T>(url: string, data: T): Promise<void> {
  try {
    await axios.put(url, data);
  } catch (error) {
    console.error(`Error updating input with id ${url}:`, error);
    throw error;
  }
}

export async function getInputElementsFromServer(): Promise<Rule> {
  return getData<Rule>(API.GET_INPUTS);
}

export async function postInputElement(newInputElement: InputElement):
Promise<void> {
  await postData<InputElement>(API.POST_INPUT, newInputElement);
}

export async function updateInputElementToServer(id: number,
updatedInputElement: InputElement): Promise<void> {
  await putData<InputElement>(`${API.UPDATE_INPUT}/${id}`,
updatedInputElement);
}

export async function deleteInputElement(id: number): Promise<void> {
  try {
    await axios.delete(`${API.DELETE_INPUT}/${id}`);
  } catch (error) {
    console.error(`Error deleting input with id ${id}:`, error);
    throw error;
  }
```

```
}

export async function getUrlFromServer(): Promise<string> {
  return getData<string>(API.GET_URL);
}

export async function postUrlToServer(newUrl: UrlObj): Promise<void> {
  await postData<UrlObj>(API.POST_URL, newUrl);
}

export async function getSubmitBtnFromServer(): Promise<InputElement> {
  return getData<InputElement>(API.GET_SUBMIT_BTN);
}

export async function postSubmitBtnToServer(submitBtn: InputElement):
Promise<void> {
  await postData<InputElement>(API.POST_SUBMIT_BTN, submitBtn);
}

export async function getAllRulesFromServer(): Promise<AllRules[]> {
  return getData<AllRules[]>(API.GET_RULES);
}

export async function postRule(newRule: AllRules): Promise<void> {
  await postData<AllRules>(API.POST_RULE, newRule);
}

export async function deleteRule(id: number): Promise<void> {
  try {
    await axios.delete(`${API.DELETE_RULE}/${id}`);
  } catch (error) {
```

```
    console.error(`Error deleting input with id ${id}:`, error);
    throw error;
  }
}

export async function testForm(): Promise<void> {
  await postData(API.TEST_FORM);
}

interface FiledId {
  id: number;
}

export async function testRule(filedId: FiledId): Promise<void> {
  await postData(API.TEST_RULE, filedId);
}
```