НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Інструментарій автоматизованого тестування веб-сайтів»

Виконала:
студентка IV курсу, групи ТМ-82
Савчук Анна Іванівна _____

Керівник:
старший викладач,
Дацюк Оксана Антонівна _____

Консультант

_____

Рецензент:

_____

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

Київ – 2022

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший

спеціальність 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
”___”_____2022р.

ЗАВДАННЯ
на дипломну роботу студенту
Савчук Анні Іванівні

1. Тема роботи «Інструментарій автоматизованого тестування веб-сайтів», керівник роботи Дацюк Оксана Антонівна, старший викладач, затверджена наказом вищого навчального закладу від ”25” травня 2022р. № _____

2. Строк подання студентом роботи __10 червня 2022 р._____

3. Вихідні дані до роботи _інструменти автоматизованого тестування веб-сайтів

4.Зміст розрахунково-пояснювальної записки
- 4.1 Провести аналіз проблеми автоматизованого тестування веб-сайтів
- 4.2 Огляд існуючих рішень
- 4.3 Сформувати вимоги до системи
- 4.4 Розробити алгоритм системи для автоматизованого тестування веб-сайтів
- 4.5 Обґрунтувати вибір технологій та програмних засобів для розробки
- 4.6 Описати взаємодію користувача з системою
- 4.7 Зробити висновки

5. Перелік ілюстративного матеріалу
- 5.1 Use-case діаграма

5.2 Контекстна діаграма IDEF0
5.3 Діаграма класів
5.4 Діаграма активностей(видів діяльності)
5.5 Mind map

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”10” жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.2022	виконано
2.	Вивчення та аналіз задачі	01.03.2022-01.04.2022	виконано
3.	Розробка архітектури та загальної структури системи	02.04.2022-08.04.2022	виконано
4.	Розробка структур окремих підсистем	09.04.2022-15.04.2022	виконано
5.	Програмна реалізація системи	16.04.2022-25.05.2022	виконано
6.	Оформлення пояснювальної записки	26.05.2022–28.05.2022	виконано
7.	Захист програмного продукту	20.05.2022	виконано
8.	Передзахист	08.06.2022	виконано
9.	Захист		виконано

Студент _____ _____Савчук А.І._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Дацюк О.А._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета розробки – система для автоматизації тестування веб-сайтів, яка допомагає уникнути регресії та надає можливість проаналізувати кроки раніше запущених тест кейсів.

У дипломному проекті розроблено систему для автоматизації тестування веб-сайтів. Інженер створює тест кейс на основі тестового сценарію та набору даних для будь-якого веб-сайту за його URL-адресою. Наявні тест сценарії та набори даних можуть бути використані для створення тест кейсів різних проектів.

Для реалізації даної системи використані мова програмування Java, фреймворк Spring Boot, система управління базами даних PostgreSQL. Клієнтська частина використовує фреймворк Angular.

Пояснювальна записка складається зі вступу, п’яти розділів, висновку списку використаних джерел; містить 74 сторінки тексту, 38 рисунків та 1 додаток. Список використаних джерел включає 17 бібліографічних найменувань та 3 електронних ресурси.

Ключові слова: тест кейс, автоматизована система, автоматизоване тестування, функціонал веб-сайту, тест сценарій, особистий кабінет користувача, Java, PostgreSQL, SpringFramework, Angular.

SUMMARY

The purpose of the development is to automate the testing of websites, which helps to avoid regression and provides an opportunity to analyze the steps of previously launched test cases.

The diploma project developed a system for automating website testing. The engineer creates a test case based on a test script and data set for any website at its URL. Available test scenarios and data sets can be used to create a test case of various projects.

To implement this system used Java programming language, Spring Boot framework, PostgreSQL database management system. The client part uses the Angular framework.

The explanatory note consists of an introduction, five chapters, a conclusion of the list of sources used; contains 74 pages of text, 38 figures and 1 appendix. The list of used sources includes 17 bibliographic titles and 3 electronic resources.

Keywords: test case, automated system, automated testing, website functionality, test scenario, personal user account, Java, PostgreSQL, SpringFramework, Angular.

ЗМІСТ

ВСТУП

Сучасне програмне забезпечення є складним багатофункціональним об’єктом. Його якість забезпечується тестуванням на кожному етапі життєвого циклу. Зі зростанням підсистем проекту, збільшується кількість тестів, для перевірки кожного модулю програми, що значного ускладнює процес розробки програмного забезпечення. Виникає проблема підтримки життєдіяльності системи за умови обмежених ресурсів, таких як кількість QA інженерів в команді, часу, програмного та апаратного забезпечення. Способом їх вирішення та оптимізації даного процесу є автоматизація тестування ПЗ.

Одним із головних завдань автоматизаці процесу тестування є підвищення ефективності, збільшення варіантів роботи систем та прискорення тестування. Автотести дозволяють запускати тестові сценарії регулярно та у будь-який час, завдяки чому можна якнайшвише виявляти помилки після додавання чи змінення функціоналу системи. Надають розробникам звіт про стан продукту одразу після завершення тестування, забезпечують підтримку Agile, крім того здатні виявляти помилки, які були пропущені на стадії ручного тестування.

Такий вид тестування доцільно використовувати для регресійного тестування, тестування з різним набором конфігурацій та на різних платформах. Існує думка, що регресійне тестування є досить складним та заплутаним видом тестування програмного забезпечення. Але наявність сучасних інструментів для автоматизації регресійного тестування та коректне використання таких інструментів дозволяє автоматизувати проведення регресійних тестів.

Регресійне тестування дозволяє перевірити функціональність програми після внесення в неї змін. Це досить складна та кропітка робота, яка вимагає

детального аналізу. Автоматизація регресійних тестів дозволить суттєво полегшити роботу.

Проте, найчастіше існуючі інструменти автоматизованого тестування відповідають лише декільком з наведених пунктів.

Тому виникає необхідність створення програмного забезпечення для вирішення цієї проблеми. В основі обраного підходу є створення інструментарію для автоматизованого тестування, що дозволило б інженерам уникнути запуску тест сценаріїв вручну, зконцентрувавшись на створенні складніших тест-кейсів для перевірки функціональності сайту. Така система допомагає уникнути регресії та надає можливість проаналізувати виконані кроки тест сценаріїв.

1 ЗАДАЧА СТВОРЕННЯ ІНСТРУМЕНТАРІЮ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-САЙТІВ

Метою роботи є розробка системи автоматизованого тестування веб-сайтів. На основі URL-адреси сайту поставлена задача розробки функціоналу для тестування сайту та пошуку несправностей.

Розроблена система повинна відповідати наступним вимогам:

1. Можливість запуску тест кейсу різного рівня складності.
2. Написання тест сценарію та можливість його запуску з різним набором даних.
3. Тест сценарії та дата сети можуть бути використані для створення тест кейсу в інших проектах з різними URL-адресами.
4. Веб-інтерфейс та функціональність системи повинні бути однаковими в останніх версіях веб-браузерів.
5. Адаптивний дизайн та інтуїтивно зрозумілий інтерфейс користувача.
6. Перегляд результатів проходження тест кейсу.
7. Перегляд результатів всіх запущених тест кейсів користувача.

Система має бути реалізована на основі клієнт-серверної архітектури, з можливістю доповнення та масштабування в майбутньому. Усі створені та наявні тестові сценарії та набори даних, можуть бути використані в різних проектах для тестування різних веб-сайтів.

2 МЕТОДИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-САЙТІВ

З розвитком технологій, збільшуються і потреби користувачів, тому для вдосконалення існуючих рішень у сфері програмного забезпечення розробники все більше намагаються автоматизувати кожен з етапів розробки та тестування продукту, що розробляється.

2.1 Опис предметної області

Важливим фактором автоматизації будь-якого процесу є зменшення часу, який витрачається на виконання процесу, підвищення надійності. У сучасній розробці широко використовується конвеєр CI/CD. Він відіграє важливу роль у забезпеченні успішного переміщення змін коду за допомогою автоматизованих циклів тестування та подальших етапів. Тому всі інструменти автоматизації тестування розраховані на швидку інтеграцію з інструментами CI/CD — GitHub, Jenkins, Bambo, хмарними сервісами на зразок AWS або Azure.

Автоматизувати можна практично будь-який процес, проте перш ніж змінювати систему під таке рішення, необхідно проаналізувати його доцільність. Більшу частину перевірок, можливо реалізувати програмно, створивши алгоритм. Що в свою чергу вимагає часу на імплементаію, виконання, аналіз отриманих результатів, окрім цього, технічний спеціаліст повинен володіти вищим рівнем кваліфікації для його втілення.

Розрізняють два види тестування: ручне та автоматизоване. Ручне тестування проводиться тестувальником вручну взаємодіючи з інтерфейсом

програми чи його API за допомогою відповідних інструментів. Автоматизовані тести виконуються згідно заздалегідь написаному тест сценарію. Ці тести можуть бути різними за складністю, від перевірки одного методу в класі до виконання послідовності складних дій в інтерфейсі користувача. Вони є надійнішими ніж ручні тести, але якість автоматизованих тестів залежить від якості сценаріїв тестування.

Тестування програмного забезпечення має свій життєвий цикл, кроки, які виконуються в певній послідовності для досягання поставлених цілей. У життєвому циклі тестування програмного забезпечення (STLC), кожна діяльність здійснюється планомірно і систематично. STLC включає як верифікацію, так і валідацію. Тестування складається з ряду заходів, які проводяться методично, щоб допомогти сертифікувати програмні продукти (рисунок 2.1).

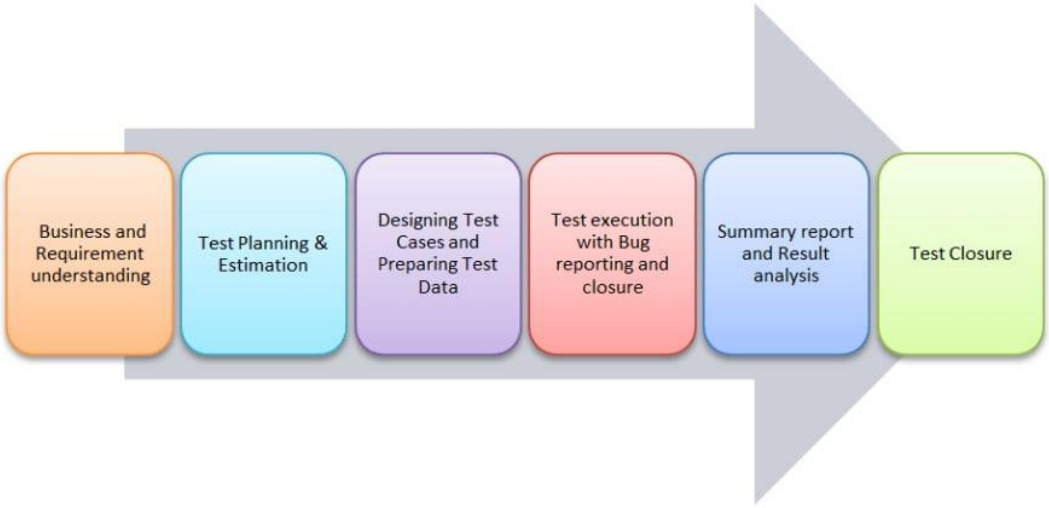

Рисунок 2.1 — Життєвий цикл тестування

Нижче наведено етапи STLC:

- Requirements phase. Цей етап допомагає визначити обсяг тестування, проаналізувавши вимоги
- Planning Phase. Етап визначення діяльності та ресусів для досягнення цілей тестування. Враховуються також такі фактори як тестова стратегія та аналіз ризиків

- Analysis phase. Фаза визначає що буде перевірено. На це впливає рівнь та глибина тестування, складність продукту, ризики життєвий цикл розробки
- Design Phase. Етап визначає як тестувати. Включає в себе завдання: налаштування тестового середовища, показники покриття тестами
- Implementation Phase. Основним завданням є створення детальних тест кейсів
- Execution Phase. Виконання тестування
- Conclusion Phase. Звітність результатів
- Closure Phase. Перевірка виконаних сценаріїв

2.2 Методи тестування програмного забезпечення

Існує багато різних типів тестування, які використовують для перевірки того, що зміни у програмному коді працюють належним чином.

Unit tests. Полягає у тестуванні окремих методів і функцій класів, компонентів чи модулів, їх часто використовують для автоматизації за допомогою севера безперервної інтеграції.

Integration tests. Це тип тестування при якому декілька модулів програми логічно згруповані і тестуються як одне ціле. Метою є виявлення дефектів зв’язків та потоків даних між моулями.

Functional tests. Зосереджені на бізнес вимогах програми. Використовує методи тестування «чорного ящика», тобто не перевіряють проміжні стани системи під час виконання цієї дії. Перевіряють лише результат дій, чи працює система належним чином.

End-to-end tests. Наскрізне тестування імітує поведінку користувача, його взаємоію з системою у повному прикладному середовищі. Перевіряє

дієздатність системи навіть при великому навантаженні. Такі тести складно підтримувати якщо вони автоматизовані.

Acceptance testing. Тести для перевірки того, чи задовольняє система бізнес вимогам. Вимагають того, щоб програма була зосереджена на поведінці користувачів.

Performance testing. Тести продуктивності перевіряють поведінку системи при великому навантаженні, швидкості реагування системи, часу відгуку, надійності, стабільності, масштабованості та використання ресурсів. Основною метою є виявлення та усунення вузьких місць продуктивності. Ця підмножина продуктивності розробки також відома як «Perf Testing».

Кожен тип програмного тестування має свої переваги та недоліки, використовується з певною метою, має свою стратегію тестування та його результат. Крім того, необхідно брати до уваги основні принципи тестування, усі тести, які мають бути проведені повинні бути сплановані до їх впровадження та відповідати вимогам замовника, необхідний оптимальний обсяг тестування на основі ризику програми, починати тестування необхідно з невеликих модулів, поступово збільшуючи його обсяг.

2.3 Аналіз існуючих рішень тестування програмного забезпечення

Існує багато інструментів що дозволяють автоматизувати тестування та проводити його аналіз. Усі вони відносяться до різних типів тестування, дозволяють протестувати різні модулі програмної системи, надати звіти чи графіки результатів. Розглянемо детальніше найпопулярніші варіанти тестування веб-додатків.

2.3.1 Allure

Allure Framework — це гнучкий і легкий інструмент для багатомовних звітів про тестування, для формування звітів про проведене тестування. Allure надає чітку «загальну картину» того, які функції були охоплені, де згруповані дефекти, як виглядає графік виконання та багато інших зручних речей. Модульність і розширюваність Allure гарантує, що ви завжди зможете щось налаштувати, щоб Allure підходила вам краще.

Однією з її головних переваг є розширюваність: можна адаптувати Allure відповідно до ваших потреб, використовуючи систему плагінів. Крім того, він має багато цікавих функцій, починаючи від тестового середовища та повторних тестів до текстових приладів.

Точкою входу для кожного звіту буде сторінка "Огляд" з інформаційними панелями та віджетами (рисунок 2.2).

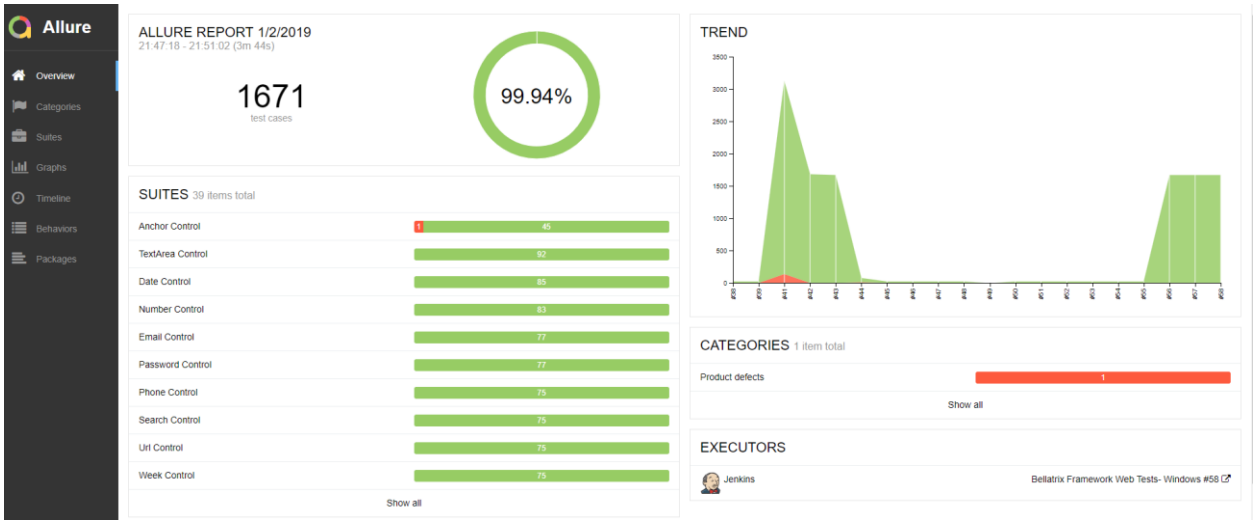

Рисунок 2.2 — Фрагмент роботи Allure

З деяких сторінок огляду результатів, можна перейти на сторінку тестового прикладу, натиснувши окремі тести. Ця сторінка зазвичай містить багато індивідуальних даних, пов’язаних із тестовим прикладом: кроки, виконані під час тестування, час, вкладення, мітки категоризації тесту, описи та посилання (рисунок 2.3).

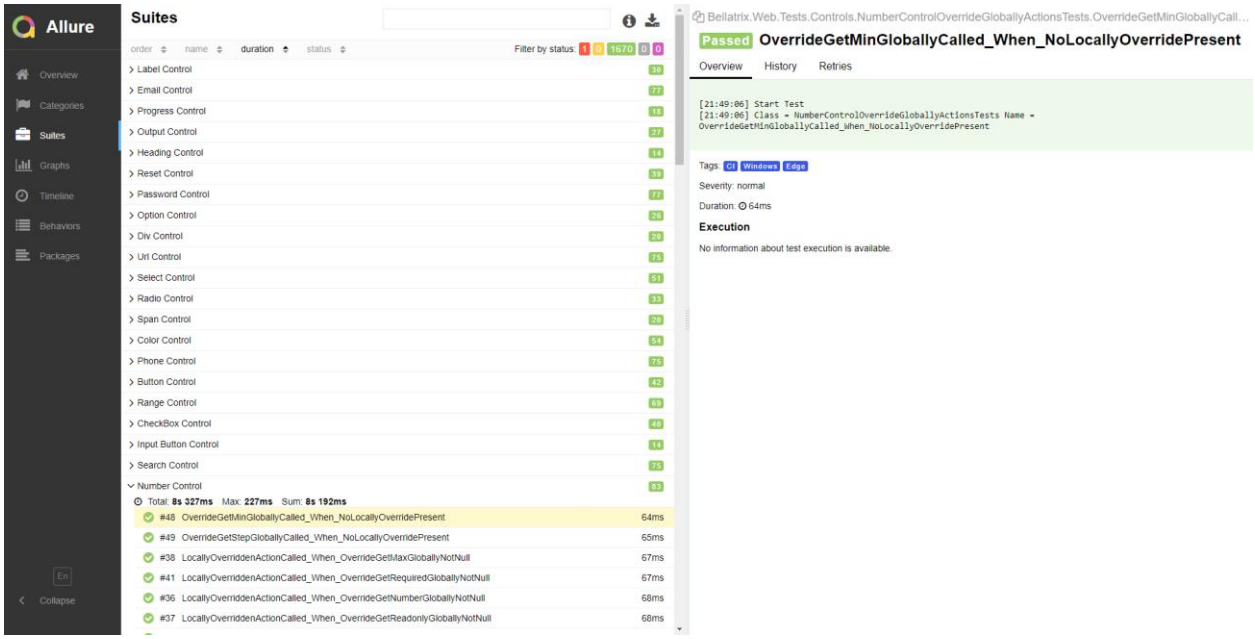

Рисунок 2.3 — Test case Allure page

2.3.2 Cypress

Cypress — платформа для проведення end-to-end тестування для автоматизації веб-додатків. Доповнюючи провідний фреймворк Selenium WebDriver, який має різні мовні прив’язки та побудований на архітектурі сітки, Cypress приносить користь своїм користувачам на етапі створення тесту завдяки своїм можливостям виконання тестів.

Cypress є більш універсальним,у порівнянні з іншими фреймворками, оскільки він написаний на JavaScript і заснований на Mocha і Chai. Він також використовує Node.js під час роботи у браузерах. Cypress може запускати тести у Firefox та браузерах сімейства Chrome, таких як Edge та Electron.

Cypress відомий своїм швидким виконанням тестів — з часом відповіді менше 20 мс. У Cypress вбудовано автоматичне очікування, що означає, що вам не потрібно визначати неявні та явні очікування. Фреймворк автоматично чекає таких речей, як завантаження DOM, анімація, елементи тощо. Крім того, фреймворк також запускає наступні тести автоматично після першого.

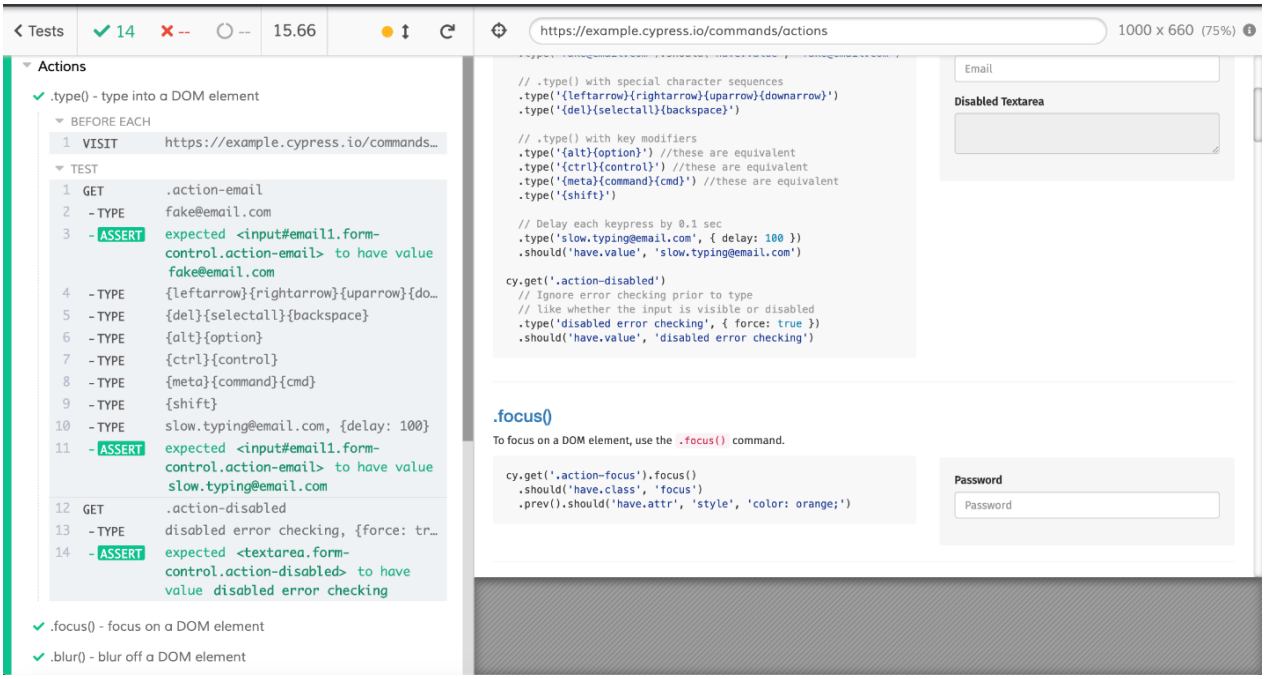

Рисунок 2.4 — Приклад роботи Cypress

2.3.3 Cucumber

Cucumber — інструмент тестування, що підтримує Behavior Driven Development (BDD). Він зчитує специфікації та перевіряє що програма виконує набір цих прикладів або сценаріїв, які описують поведінку системи з точки зору клієнта.

Інструмент Cucumber відіграє важливу роль у розробці приймальних тестових випадків для автоматизованого тестування. В основному він використовується для написання Acceptance тестів для веб-додатків відповідно до поведінки їх функціональних можливостей.

Інструмент Cucumber спочатку був написаний на мові програмування Ruby. Він використовувався виключно для тестування Ruby як доповнення до фреймворку RSpec BDD.

Переваги Cucumber інструменту:

- Open-source BDD (Behavior Driven Development) інструмент.
- Забезпечує наскрізне тестування, на відміну від інших інструментів.

- Підтримує майже всі популярні різні мови, такі як Java.net, JavaScript Ruby, PHP тощо. У Java він підтримує власний JUnit.
- Налаштування та виконання середовища тестування дуже швидкі та легкі.

До недоліків можна віднести:

- Cucumber використовується для тестування лише веб додатків.
- Cucumber тестування є менш надійним рішенням в порівнянні з Selenium та QTP.

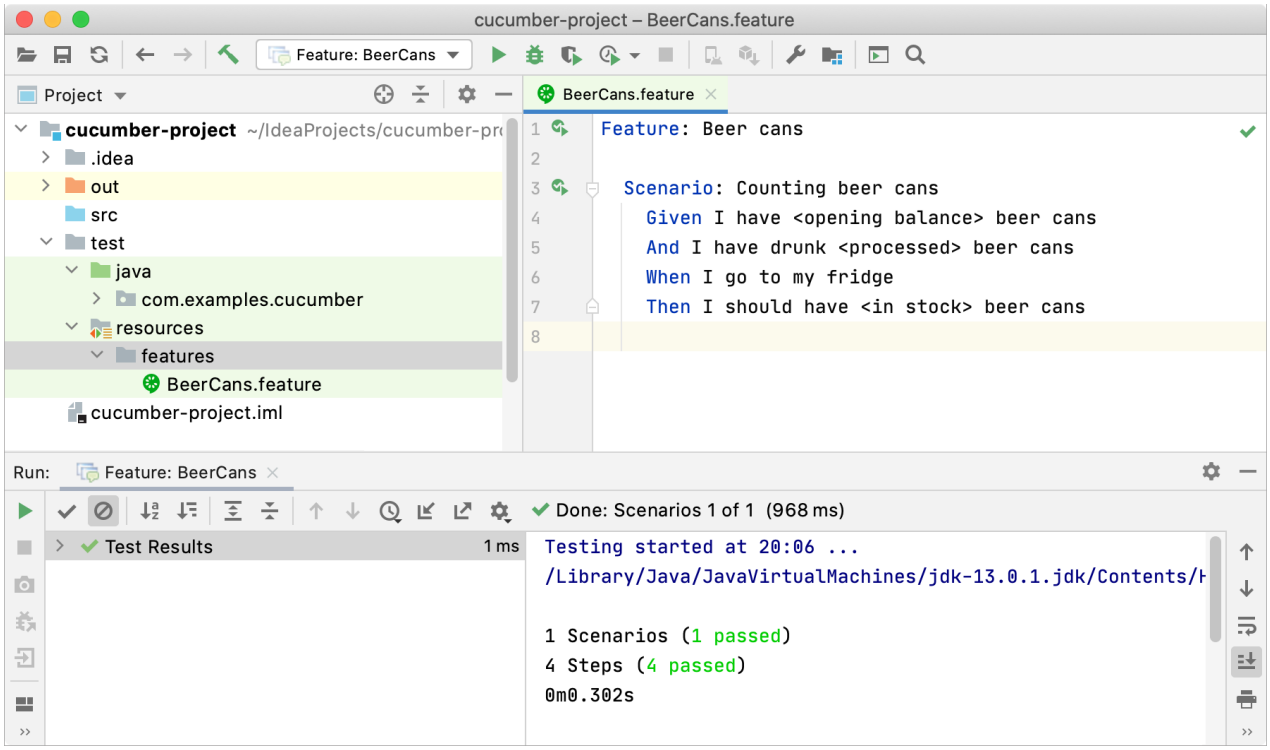

Рисунок 2.5 — Приклад роботи Cucumber

2.4 Висновки

У розділі розглянуто основні підходи тестування веб-додатків, види тестів, що виділяють з їх цільовим призначенням. Наведено аргументи вибору автоматизації тестування чи ручних тестів для системи.

Проаналізовано існуючі рішення, зокрема найпопулярніші інструменти, які найчастіше використовуються для тестування, їх переваги та недоліки. Проведений аналіз допоможе спроектувати програмне рішення для тестування за визначеними вимогами.

3 РОЗРОБКА ІНСТРУМЕНТАРІЮ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-САЙТІВ

В основі створення системи автоматизованого тестування веб-сайтів лежить взаємодія з інструментом автоматизованого тестування, що дозволило би створити надійне рішення, і як результат, покращувало якість веб-додатків.

3.1 Основні дії Selenium

Створена система повинна взаємодія з інструментом Selenium WebDriver, а основна ідея полягає у використанні його функціоналу дій (actions) та створення власних сценаріїв для тестування.

Selenium надає можливість обробки подій клавіатури та миші, що зберігаються у класі Actions. Це доступне API для користувача, що дозволяє емулювати складні події, зокрема Drag-n-Drop і подвійне клацання, які неможливо виконати за допомогою простих команд WebElement.

Найчастіше використовуються наступні дії:

- Click() - клацання на поточне розташування миші
- DoubleClick() – Виконує подвійне клацання в поточному місці розташування миші.
- ClickAndHold() – Клацання (не відпускаючи) у поточному місці розташування миші.
- DragAndDrop(source, target) – Виконує клацання й утримування на місці вихідного елемента, переміщення до розташування цільового елемента, потім відпускає мишу. Параметри: source-element (для емуляції кнопки вниз) та target-element, до якого потрібно перейти та відпустити мишу.

- dragAndDropBy(source, x-offset, y-offset) - Виконує клацання й утримування на місці вихідного елемента, переміщення на задане зміщення, потім відпускає мишу. Параметри: source-element для емуляції кнопки вниз. x-offset- зміщення горизонтального переміщення. y-offset- зміщення вертикального переміщення.
- MoveToElement() – Переміщує вказівник миші на центр елемента
- ContextClick() – Performs right-click on the mouse
- Release() – Відпускає ліву кнопку миші в поточному місці миші.
- SendKeys() – Надсилає ряд ключів елементу
- KeyUp() – Виконує звільнення ключа
- KeyDown() – Виконує натискання клавіш без відпускання

3.2 Взаємодія системи з бібліотекою Selenium

Інженер має можливість переглянути наявні в системі набори дій чи створити нові. Саме за їх використання, можна протестувати дієздатність веб-додатку. Система зможе емулювати дії користувача на сайті.

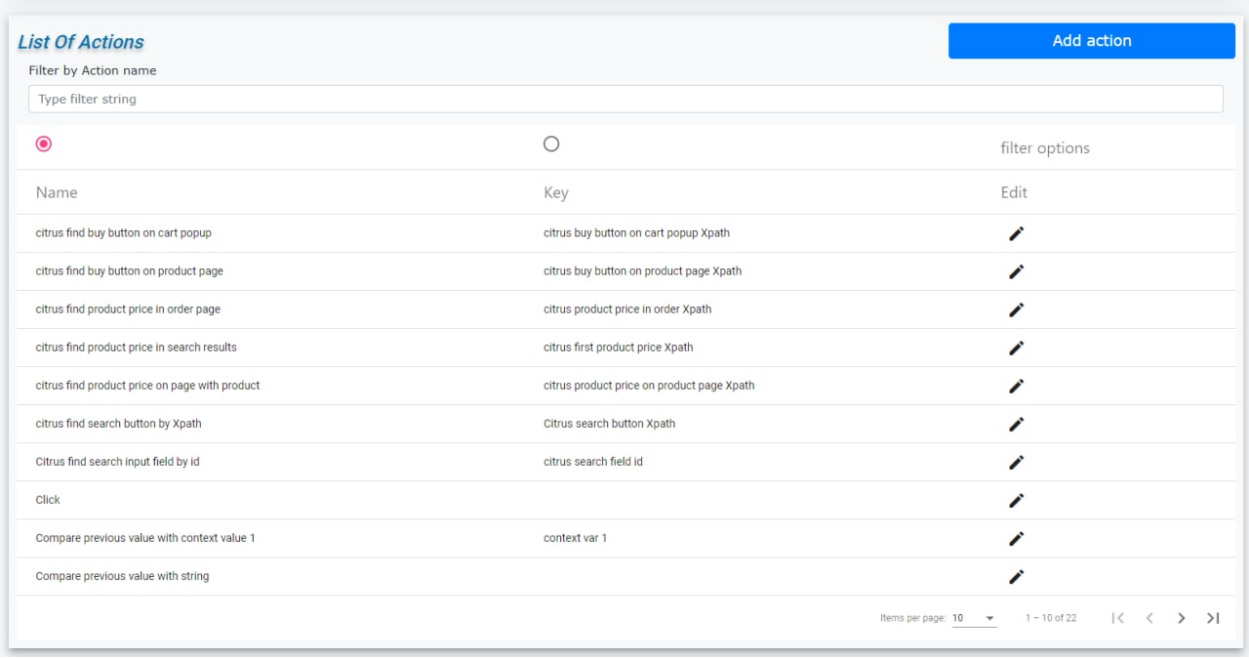

Рисунок 3.1 — Сторінка наборів дій

Кожному набору дій присвоюється ім’я, необхідно також вказати його ключ, та опис. Тип дії (action type) — параметр, який відповідає наборам дій з бібліотеки Selenium, та використовується для створення власних. Це дозволяє використовуючи об’єкт (instance) веб-драйвера для обраного браузера, виконувати прописані дії.

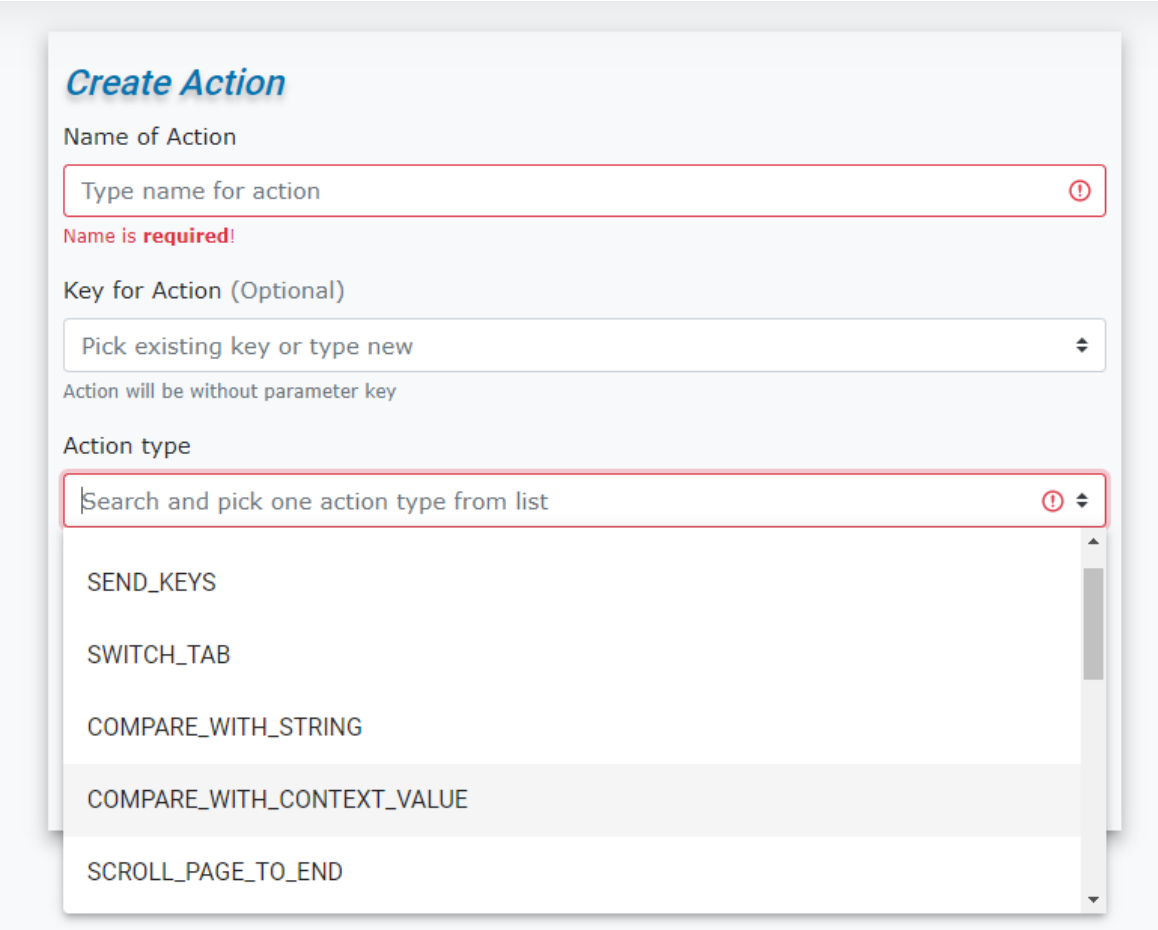

Рисунок 3.2 — Створення дії

З будь-якого набору дій можна створити їх послідовність, тобто декілька наборів дій з визначеним порядком виконання. Вони будуть додані до бібліотеки дій системи. Це дозволяє прискорити створення тест сценаріїв.

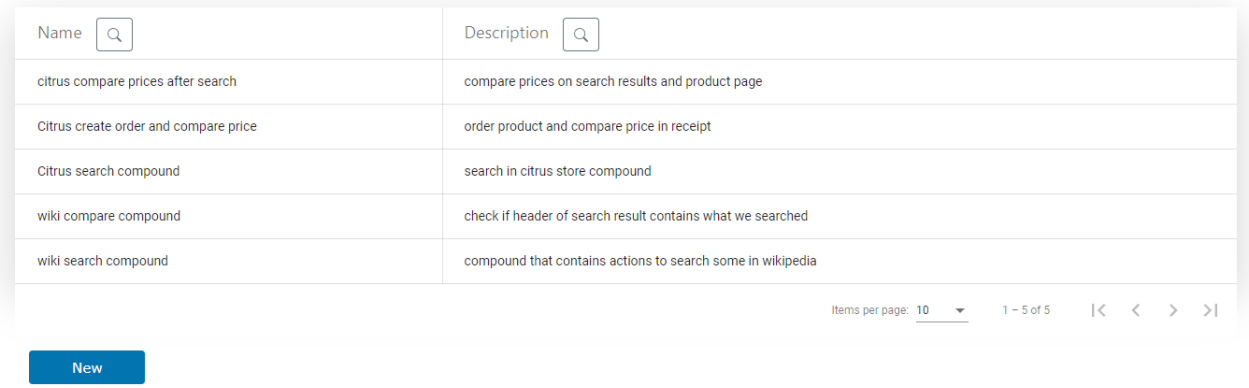

Рисунок 3.3 — Список наборів дій

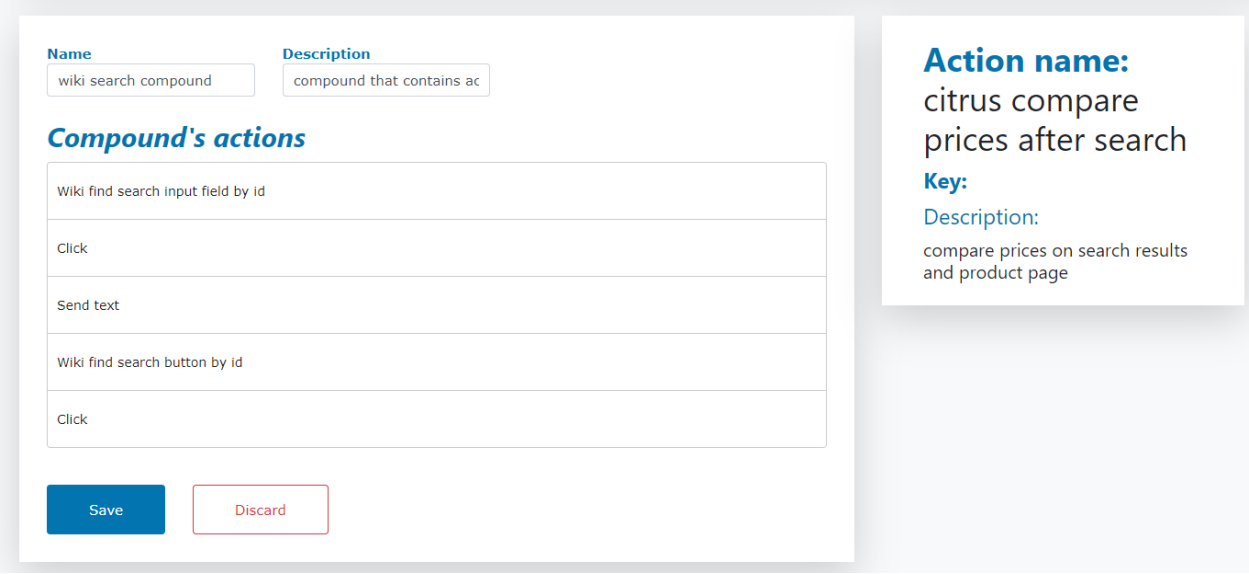

Рисунок 3.4 — Набір дій для пошуку елементів

Всі описані вище компоненти, використовуються для створення тест сценаріїв, написаних для проекту.

3.3 Основні функції системних компонентів

Проектування системи для автоматизованого тестування, полягає у визначенні основних компонентів майбутньої системи та функцій кожного з них.

Інтелект-карта (mind map) — діаграма, яка використовується для візуального впорядкування інформації в ієрархію, що показує взаємозв’язки між частинами цілого.

Діаграма описує основні функції кожного компонента системи, який використовується для побудови тест кейсів за визначеним алгоритмом.

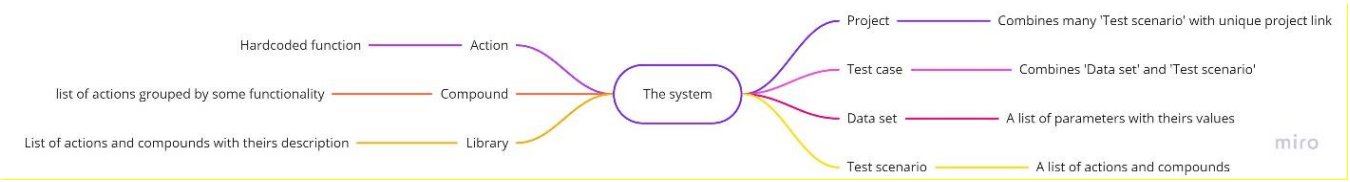

Рисунок 3.5 — Інтелект-карта

Система містить об’єктні класи:

- Action. Функції, що виконуються за сценарієм
- Compound. Набір дій, що виконуються для певних сценаріїв та у визначеній послідовності
- Library. Бібліотека зберігає списки наборів дій та їх опис. Використовується при створенні тест кейсу
- Project. Кожен проект має унікальне посилання, може комбінувати декілька тест сценаріїв для одного проекту
- Test case. Поєднує тест сценарій та набір даних
- Data set. Список параметрів з їх значеннями. Це можуть бути дані, які вводять в пошуковий рядок
- Test scenario. Сценарій виконання тестування

Діаграма активностей (activity diagram) (рисунок 3.6) описує алгоритм дій, які виконує система.

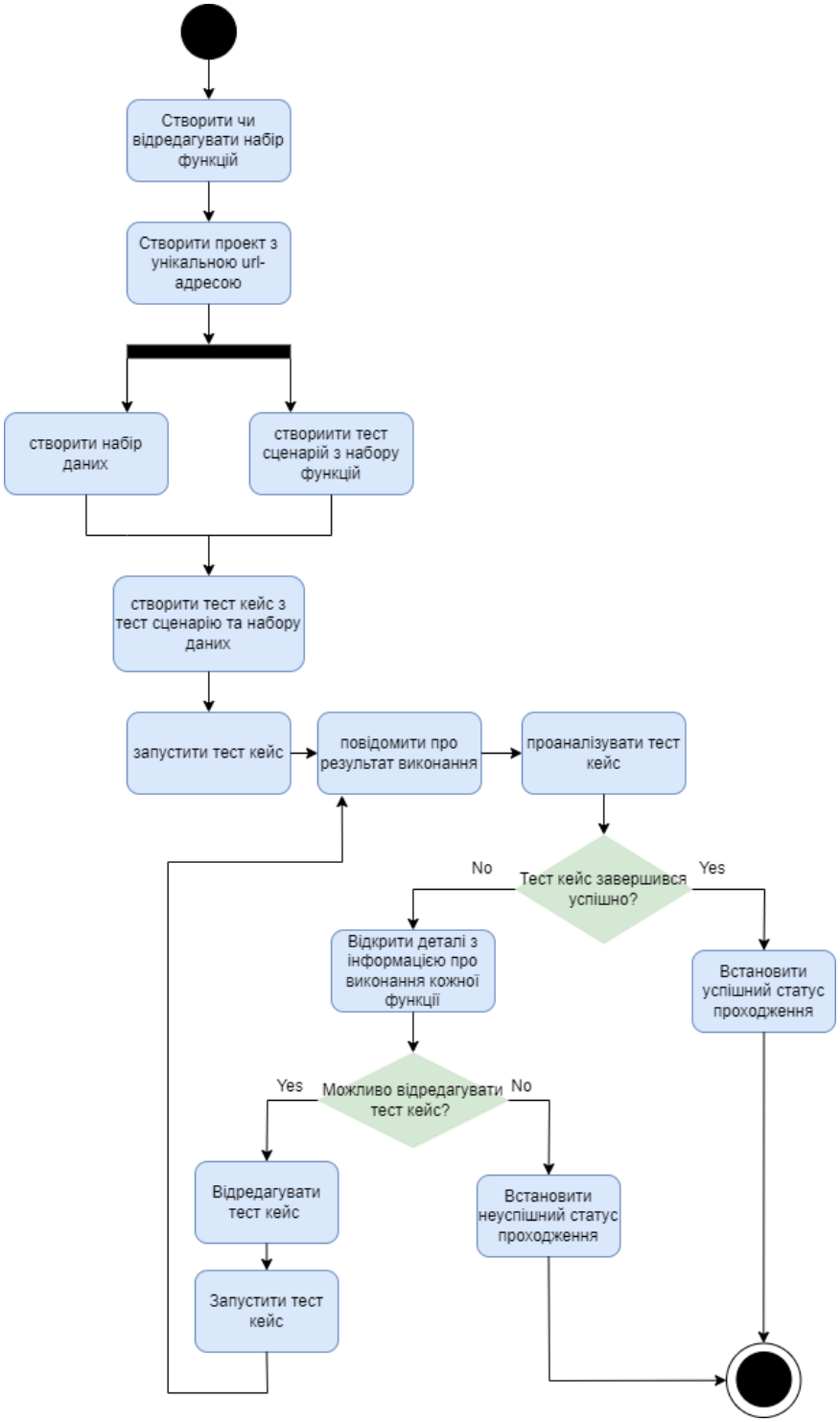

Рисунок 3.6 — Діаграма активностей

3.4 Засоби програмної реалізації

Для реалізації програмної системи було обрано інструменти та засоби розробки, що найкраще підходять для створення та підтримки даної системи. Мовою програмування для серверної частини обрано Java та фреймворк Spring

Framework. Клієнтська частина розроблена з використанням фреймворку Angular.

3.4.1 REST API

В системі спілкування між клієнтом та сервером відбувається по протоколу REST.

REST — це абревіатура від REpresentational State Transfer і архітектурний стиль для розподілених гіпермедійних систем. RESTful системи характеризуються тим, що вони не зберігають стану та відокремлюють клієнтську та серверну частини.

Для того, щоб API вважався RESTful, він повинен відповідати цим критеріям:

- Запити в клієнт-серверній архітектурі керуються через HTTP
- Незалежність від стану, тобто інформація про клієнта не зберігається між запитами на отримання даних (stateless)
- Багаторівнева архітектура (layered system)
- Єдиний уніфікований програмний інтерфейс (uniform interface)
- Кеш архітектура, що спрощує взаємодію клієнт-сервер (cacheable)
- Зручне представлення даних, єдиний інтерфейс між компонентами

Запит від клієнта передається на сервер через RESTful API у вигляді представлення. Ця інформація надається в одному з кількох форматів через HTTP: JSON, HTML, XLT, Python, PHP або звичайний текст. JSON є найбільш популярним форматом файлів, оскільки, незважаючи на свою назву, він не залежить від мови.

Основні HTTP запити:

- GET — отримати ресурс за ідентифікатором або набір ресурсів
- POST — створення нового ресурсу
- PUT — оновлення ресурсу за ідентифікатором

- DELETE — видалити ресурс за ідентифікатором.

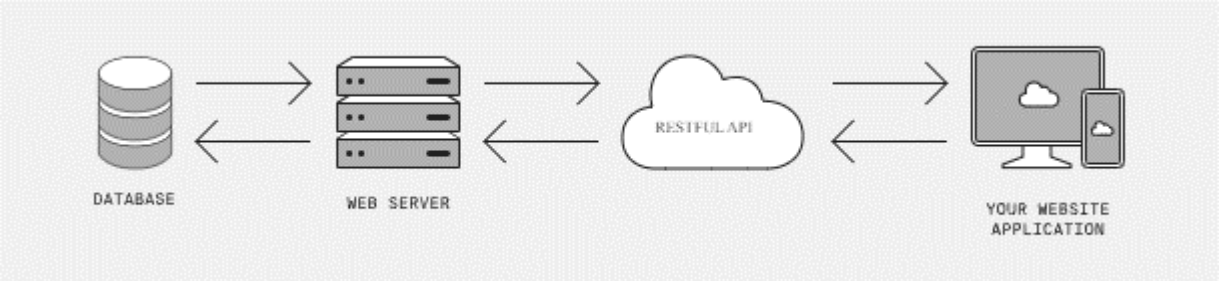

Рисунок 3.7 — REST API

REST API є легкими (lightweight), що робить їх ідеальними для нових контекстів, таких як the Internet of Things (IoT), розробка мобільних додатків і serverless computing. Крім того, багато загальнодоступних API, як-от API Карт Google, використовують REST.

3.4.2 Мова програмуваня Java

Основною мовою для розробки була обрана Java. Однією з головних переваг та особливостей мови є віртуальна машина Java (JVM). Написаний код можна запускати на різних операційних системах (Windows, Linux, MacOS та ін). При компіляції Java програма компілюється в байт-код, що дозволяє реалізувати принцип «написана для одного —працює скрізь».

Java - об'єктно-орієнтована мова програмування. Тому в ній широко використовуються основні принципи ООП: абстракція, інкапсуляція, наслідування та поліморфізм.

Інкапсуляція є властивістю системи, що дозволяє об'єднати дані та методи в класі, та приховати деталі реалізації від користувача, відкриваючи лише те що необхідно для подальшого використання. Головна мета інкапсуляції — уникнути залежності зовнішнього інтерфейсу від реалізації. Абстракція — концепція об’єктно-орієнтованого програмування, яка приховує неважливу інформацію для користувача, показуючи лише важливі атрибути. Наслідування дозволяє дочірньому класу розширити інший клас,

успадкувавши його особливості. Наслідування реалізує принцип програмування DRY, покращує можливість повторного використання коду, множинне наслідування у Java не підтримується. Поліморфізм надає класу здатність мати різні реалізації методу залежно від типу об’єкта, який передається як вхідний параметр метода. У Java існує 2 типи поліморфізму: compile-time та runtime.

Java дозволяє писати багатопоточні програми. Потоки виконання є необхідним функціоналом мови Java, оскільки можуть спростити розробку системи, перетворюючи складний асинхронний код у більш простий і прямолінійний. Багатопоточність дозволяє ефективно використовувати обчислювальні потужності процесорів, забезпечує спільний доступ всіх потоків до пам’яті, мінімізує час виконання програми. Здебільшого використовується в складних програмах, де потоки дозволяють легко створювати модулі програми, що значно підвищує їх продуктивність.

Java створена для масштабованості, тому є такою популярною серед великих проектів і стартапів. Оскільки Java є мовою статичної типізації, її простіше та швидше підтримувати з меншою кількістю помилок. Java підтримує зворотну сумісність, тому старі версії програм, як і раніше, працюватимуть навіть після переходу на нові версії мови.

3.4.3 Spring Framework

Spring Framework (Spring) — це програма з відкритим вихідним кодом, яка забезпечує підтримку інфраструктури для розробки Java-додатків. Одна з найпопулярніших фреймворків Java Enterprise Edition (Java EE), Spring допомагає розробникам створювати високопродуктивні програми за допомогою простих старих об’єктів Java (POJO). Видаляючи більшу частину стандартного коду та конфігурації, пов’язаної з веб-розробкою, спрощує розробку додатків на стороні сервера.

Технологія, з якою Spring найбільше ототожнюють, — це інверсія керування (DI). Dependency Inversion — це набір патернів та принципів

розробки програмного забезпечення, які дозволяють писати слабозв’язний код.

До основних переваг можна віднести:

- Lightweight. Дозволяє створювати масштабовані програми за допомогою POJO(простий Java об'єкт)
- Flexibility. Пропонує гнучкі бібліотеки для використання розробникам. Для параметрів конфігурації розробник може вибрати анотації на основі XML або Java. Функції IoC і DI створюють основу для широкого набору функцій і функціональних можливостей
- Loose Coupling. Забезпечує слабку з’язаність завдяки dependency injection
- Powerful Abstraction. Забезпечує потужну абстракцію до специфікацій JEE, таких як JMS, JDBC, JPA і JTA.
- Productive. Дозволяє інгегрувати різні модулі в програму, додаючи депенденсі в файл конфігурації.
- Secure. Spring Security забезпечує безпеку даних і програм, крім того випускаються постійні оновленняng Security
- Declarative Support. Надає декларативну підтримку кешування, перевірки, транзакцій і форматування.
- Portable. Є можливість використовувати серверну сторону у веб-додатку/EJB, а бізнес-логіку на клієнтській стороні
- Cross-cutting behavior. Управління ресурсами є наскрізною проблемою, яку легко копіювати та вставляти скрізь.
- Configuration. Забезпечує можливість послідновного налаштування конфігурації, відокремлення її від логіки програми
- Lifecycle. Відповідає за керування життєвим циклом всіма компонентами програми
- Dependency Injection. Інверсія залежностей є реалізацією Ioc. Набір патернів та принципів розробки програмного забезпечення,

у відповідності з якими, об’єкт передає створення необхідних йому залежностей іншому, зовнішньому об’єкту, спеціально призначеному для цього

До недоліків, які можуть ускладнити використання фреймворку можна віднести:

- Complexity. Складний фреймровк через велику кількість бібліотек, необхідних конфігурацій та їх налаштування для роботи
- Parallel Mechanism. Розробники повинні знати, які функції будуть корисними, і прийняття неправильних рішень може призвести до значних затримок.
- Lots of XML. Для розробки програми Spring необхідно багато файлів конфігурацій XML. Для того аби уникнути цього, можна використовувати Spring Boot

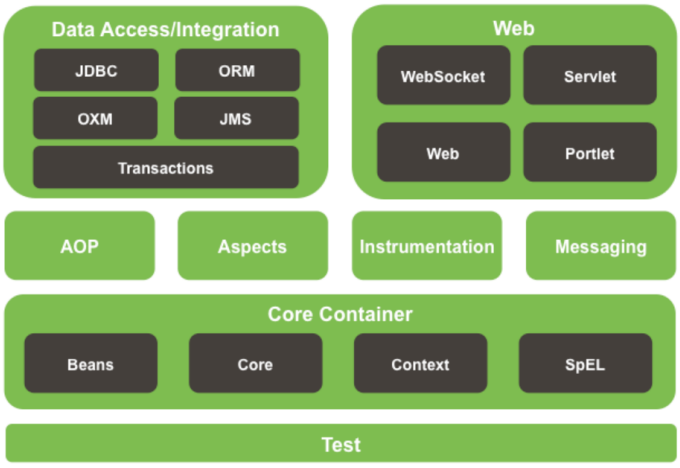

Рисунок 3.8 – Spring Framework components

3.4.4 PostgreSQL

PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД), яка використовує SQL як основну мову запитів.

Основні переваги PostgreSQL:

- Користувачі PostgreSQL можуть безпосередньо брати участь у спільноті та публікувати та ділитися незручностями та помилками.
- Функції SQL, збережені процедури, можна використовувати для серверного середовища.
- Підтримує мови, подібні до PL/SQL в Oralcle, такі як PL/pgSQL, PL/Python, PL/Perl, C/C++ і PL/R.
- PostgreSQL підтримує ACID (атомність, послідовність, ізоляція, довговічність).
- PostgreSQL надає не тільки методи індексу дерева B+, але й різні види техніки, такі як GIN (узагальнений перевернутий індекс) і GiST (узагальнене дерево пошуку) тощо.
- Повнотекстовий пошук доступний при пошуку рядків з виконанням векторної операції та пошуку рядків.

Тому для розробки системи було обрано SQL базу даних, яка надає великі переваги для транзакційних даних, структура яких не змінюється часто (або взагалі) і де цілісність даних є першорядною. Це також найкраще рішення для швидких аналітичних запитів.

3.4.5 Selenium

Для запуску тест кейсу, необхідно було обрати інструмент, для взаємодії з браузером, який на основі складеного тест сценарію, виконував би прописані інструкції. Тому таким інструментом було обрано Selenium WebDriver, один з серії програмних продуктів Selenium, що залишається одним найпопулярніших рішень.

Selenium WebDriver – гнучкий інструмент для автоматизованого тестування веб-проектів на базі набору бібліотек для різних мов програмування, таких як Java, .Net(C#), Python, PHP, Ruby, Perl, JavaScript. Даний інструмент підтримує роботу на базі Windows, macOS та Linux, а також найпоширеніші браузери Google Chrome, Firefox, Safari, Edge, Internet Explorer та навіть деякі браузери без графічного інтерфейсу. Використовується

Selenium WebDriver найчастіше з такими видами тестування, як регресійне та функціональне.

Для того щоб не повторювати одноманітних операцій в браузері при тестуванні, використовується webdriver браузера, який звертається до браузера, драйвер якого використано, та виконує послідовність дій через певний скрипт з бібліотеки Selenium. Прикладами дій можуть бути команди по знаходженню елементів, перехід за посиланнями, збір великих об'ємів даних (парсинг), натискання кнопок та ін. Взаємодія бібліотеки з веб-драйвером відбувається через JSON Wire Protocol.

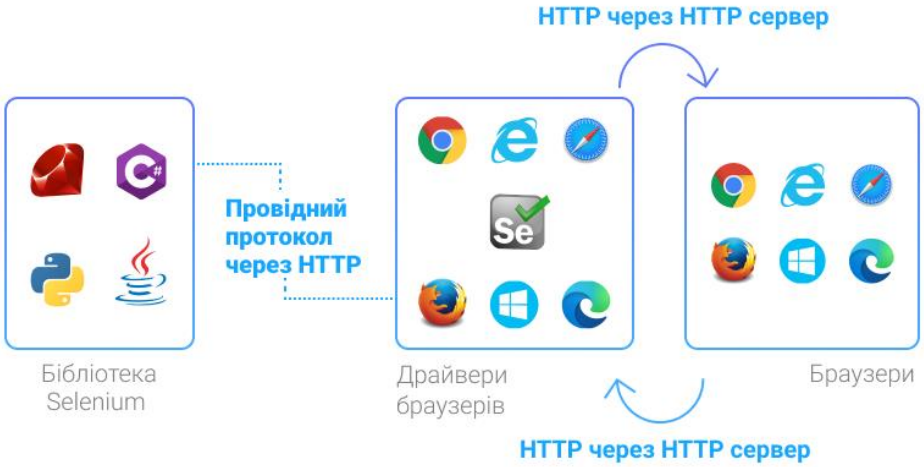

Рисунок 3.9 – Архітектура Selenium WebDriver

3.5 Висновки

У розділі розглянуто деталі реалізації основних елементів системи, а саме процес використання наборів функцій бібліотеки Selenium та їх комбінування. Описано основні функції бібліотеки Selenium. Проаналізовано кроки для складання тест сценарію тестування. Розроблено діаграму активностей та інтелект-карту основних об'єктів системи.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В проектуванні будь-якого програмного продукту важливо провести аналіз існуючих технологій розробки програмного забезпечення та обрати ті, які найкраще підійдуть для реалізації. На основі зазначених вимог до програмної системи, побудованих UML-діаграм розроблено алгоритм роботи системи.

4.1. Вибір архітектурного підходу

Вибір архітектурного підходу системи є одним із найважливіших критеріїв успішності проекту в майбутньому, можливості додавання нового функціоналу, додавання нових модулів.

Мікросервіси, які з’явилися на світ лише кілька років тому, сьогодні набувають стрімкого розвитку. Адже цей підхід має суттєві переваги, такі як масштабованість та гнучкість. Netflix, Google, Amazon та інші технічні лідери використовують мікросервісну архітектуру, здійснивши перехід від монолітної.

Однією з головних причин, для чого з’явились мікросервіси та стали часто використовуватись це здатність масштабувати програму, вносити зміну до її окемих модулів не змінюючи при цьому інші. Розподіл додатку на декілька сервісів, дає змогу швидко їх розробляти та підтримувати в майбутньому. Дозволяє розгортати кожен мікросервіс незалежно. Проте такий підхід спричиняє додаткову складність, адже необіхдно налаштувати зв’язок між усіма модулями та базами даних в розподіленій системі.

Монолітна архітектура вважається традиційною, система вважається єдиною та неподільною, містить клієнтський інтерфейс користувача, серверну програму та базу даних.

Основними перевагами монолітної архітектури є:

- End-to-end тестування можна реалізувати запуском програми та тестуванням UI за допомогою Selenium чи іншого інструменту.
- Простий у розгортанні. Архів додатку необхідно скопіювати на сервер.
- Легко масштабувати по горизонталі, запускаючи декілька копій через load balancer.
- Якщо проект на першому етапі SDLC, далі він зростатиме. Монолітна архітектура дозволяє виконувати швидкі ітерації.

Недоліками використання монолітної архітектури є:

- Scalability. У разі необхідності, масштабувати доводиться всю програму, а не окремі її компоненти
- Making changes. Складно запроваджувати зміни в проекті з сильною зв’язаністю компонентів
- New technology barriers. Можуть виникнути складнощі з застосуванням нових технологій, через переписування програмного коду

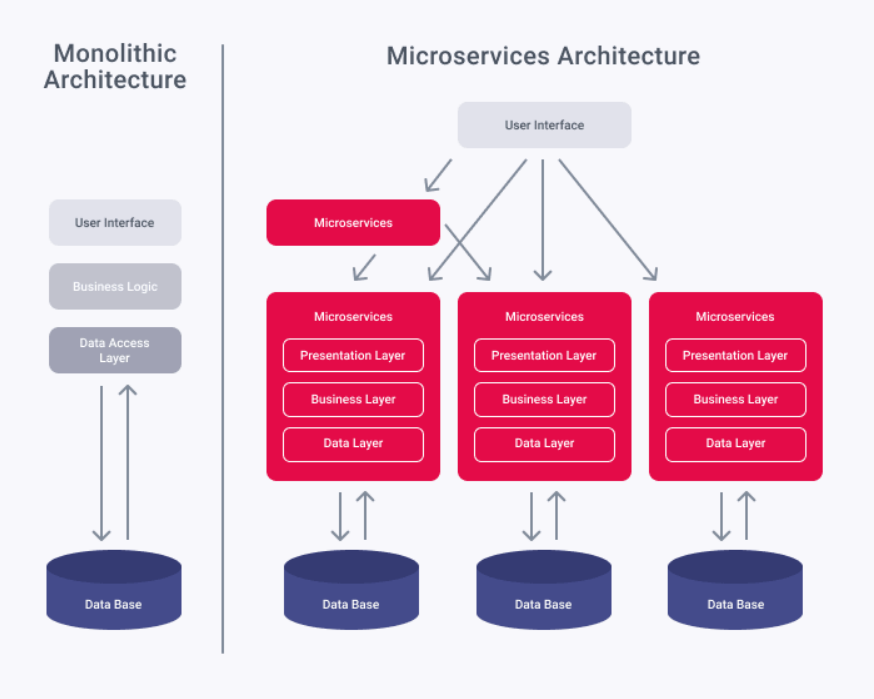

Рисунок 4.1 — Монолітна та мікросервісна архітектури

Проаналізувавши вимоги до системи, та особливості архітектурних рішень, було обрано монолітну архітектуру, побудову за шаблоном MVC. Адже програма складається з простих логічних компонентів, якимим легше керувати не розділяючи на окремі мікросервіси. Основна ідея шаблону MVC полягає у розділенні системи на рівні приймання запитів від користувачів, рівень бізнес логіки системи та рівень взаємодії з базою даних (рисунок 4.1).

Система складається з двох основних модулів, серверної та клієнтської частин. Серверна частина реалізована з використанням об'єктно-орієнтованої мови Java та фреймворку Spring. Клієнтська частина використовує фреймворк Angular. Для доступу до даних та їх зберігання було обрано систему управління базами даних PostgreSQL (рисунок 4.2).

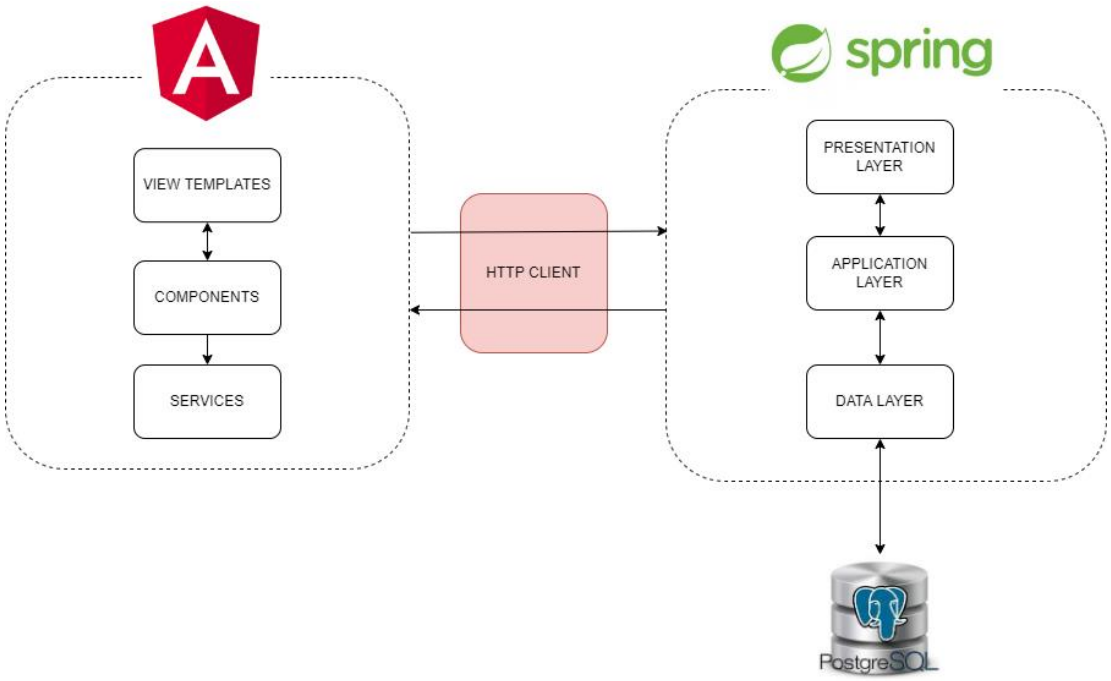

Рисунок 4.2 – Архітектура системи

4.2 Архітектура системи автоматизованого тестуваннння веб сайтів

Система реалізована на основі клієнт-серверної архітектури. Сховищем

даних було обрано СУБД PostgreSQL. Обмін даними між клієнтською та серверною частиною здійснюється за протоколом REST.

На рисунку 4.3 зображено схему основних модулів системи. Система підтримує багатопоточність, тому здатна приймати запити від великої кількості користувачів.

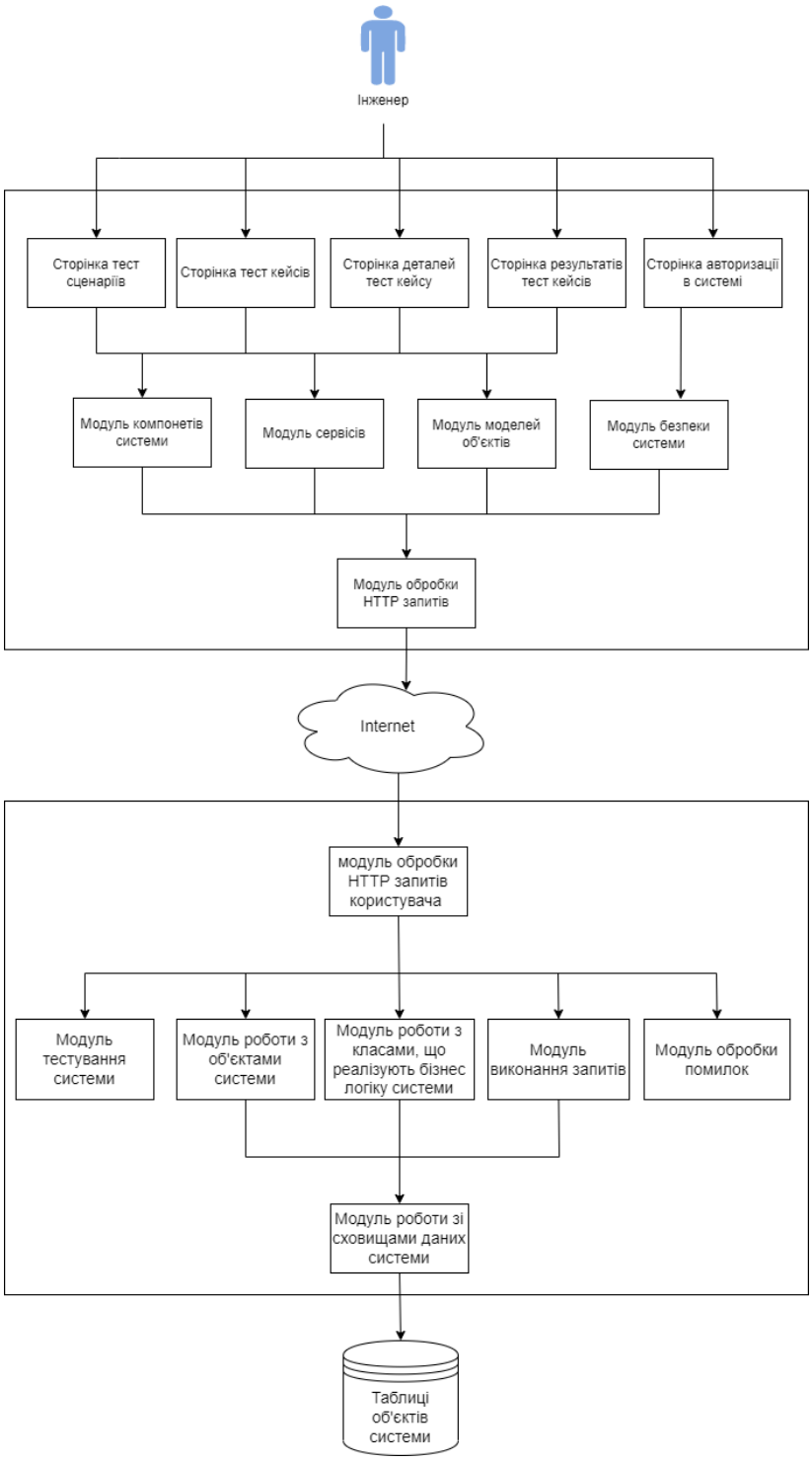

Рисунок 4.3 – Структура модулів програмної системи

Серверна частина поділена на модулі що проводять тестування роботи програми, модуль, де описані об’єкти системи, що потім будуть збережені до бази даних, модуль що виконує запити та модуль обробки помилок. Основним з цих модулів є класи сервісів системи, де реалізована логіка створення тест кейсів, тестових сценаріїв, наборів даних, проектів та профілю корисувача.

Клієнтська частина представляє інтерфейс системи та дозволяє відправляти запити користувача на сервер.

Діаграма класів (class diagram) відображає структуру системи шляхом моделювання її класів, операцій, атрибутів та зв’язків між об’єктами.

Серед основних класів системи:

- TestCaseServiceImpl – сервіс, що дозволяє створювати новий тест кейс за обраним тест сценарієм та дата сетом.
- DataSetServiceImpl – сервіс, що описує логіку створення дата сетів
- TestScenarioServiceImpl – сервіс, що дозволяє створювати тест сценарій з обраного списку дій
- ActionsServiceImpl – сервіс, що реалізує інтерфейс ActionsService і його методи для створення нових дій, редагування та видалення.
- ActionsService - інтерфейс, що містить методи для створення редагування, видалення дій
- TestScenarioService – інтерфейс, що містить методи для створення, редагування та тест сценарій з обраного списку дій.
- DataSetService – інтерфейс, що містить методи створення, редагування, видалення та пошуку дата сетів.
- TestCaseService – інтерфейс, що містить методи для створення тест кейсу, його запуску, отримання деталей тест кейсу та отримання статистики успішних та неуспішних тест кейсів.
- Invoker – сервіс, який реалізує шаблон проектування «команда» (Commander) та виконує послідовний запуск кожної функції.

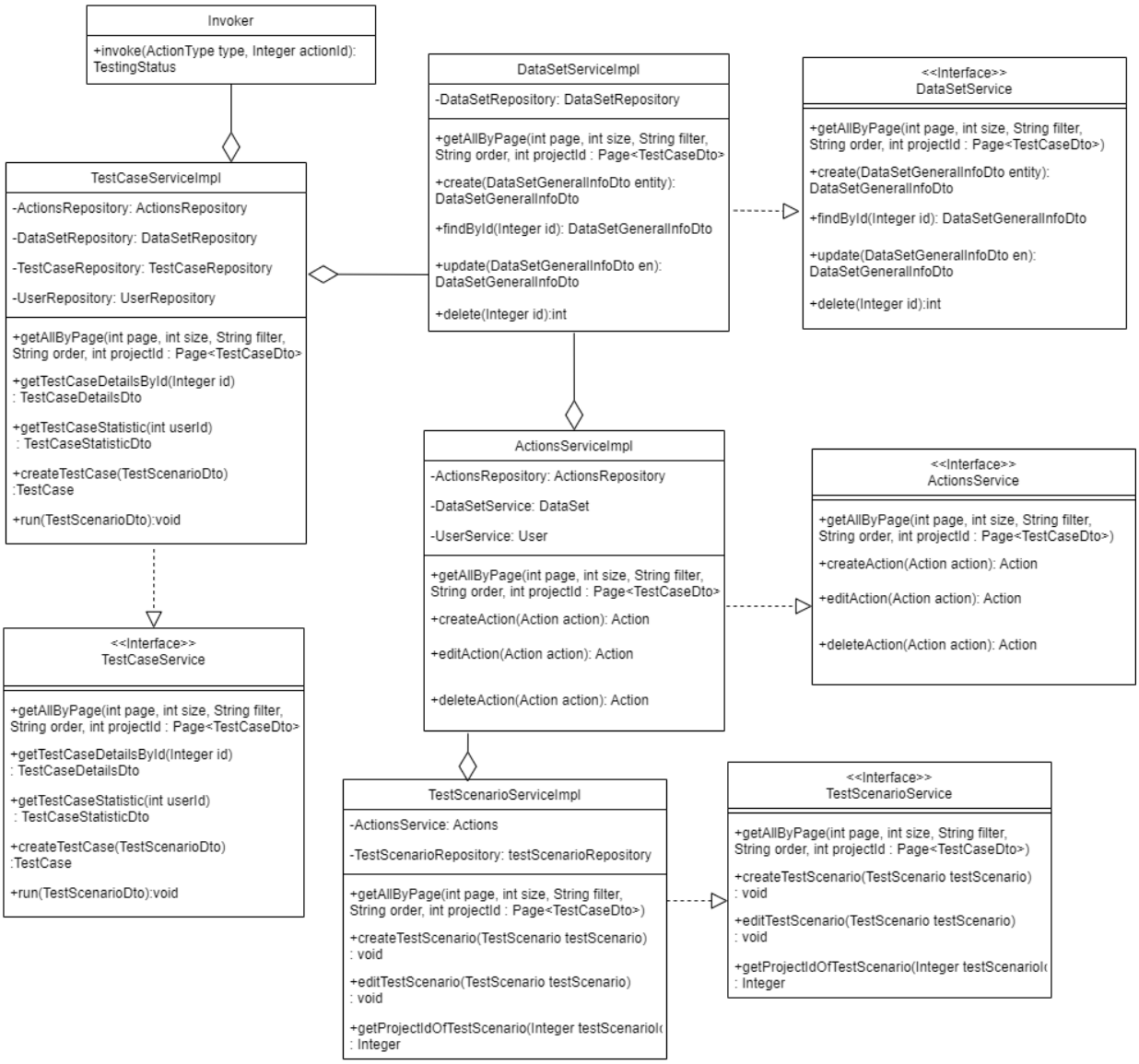

Рисунок 4.4 — Діаграма класів

Діаграма прецедентів (use-case) (рисунок 4.5) використовується для опису варіантів використання (набору дій), які система може виконувати у співпраці з одним або декількома зовнішніми користувача системи (акторами). Інженеру доступний весь функціонал системи, зокрема створення тест кейсів для подальшого тестування обраного проекту.

Адміністратору доступні усі функції інженера, а також можливість додавати нових працівників, переглядати інформацію про кожного працівника.

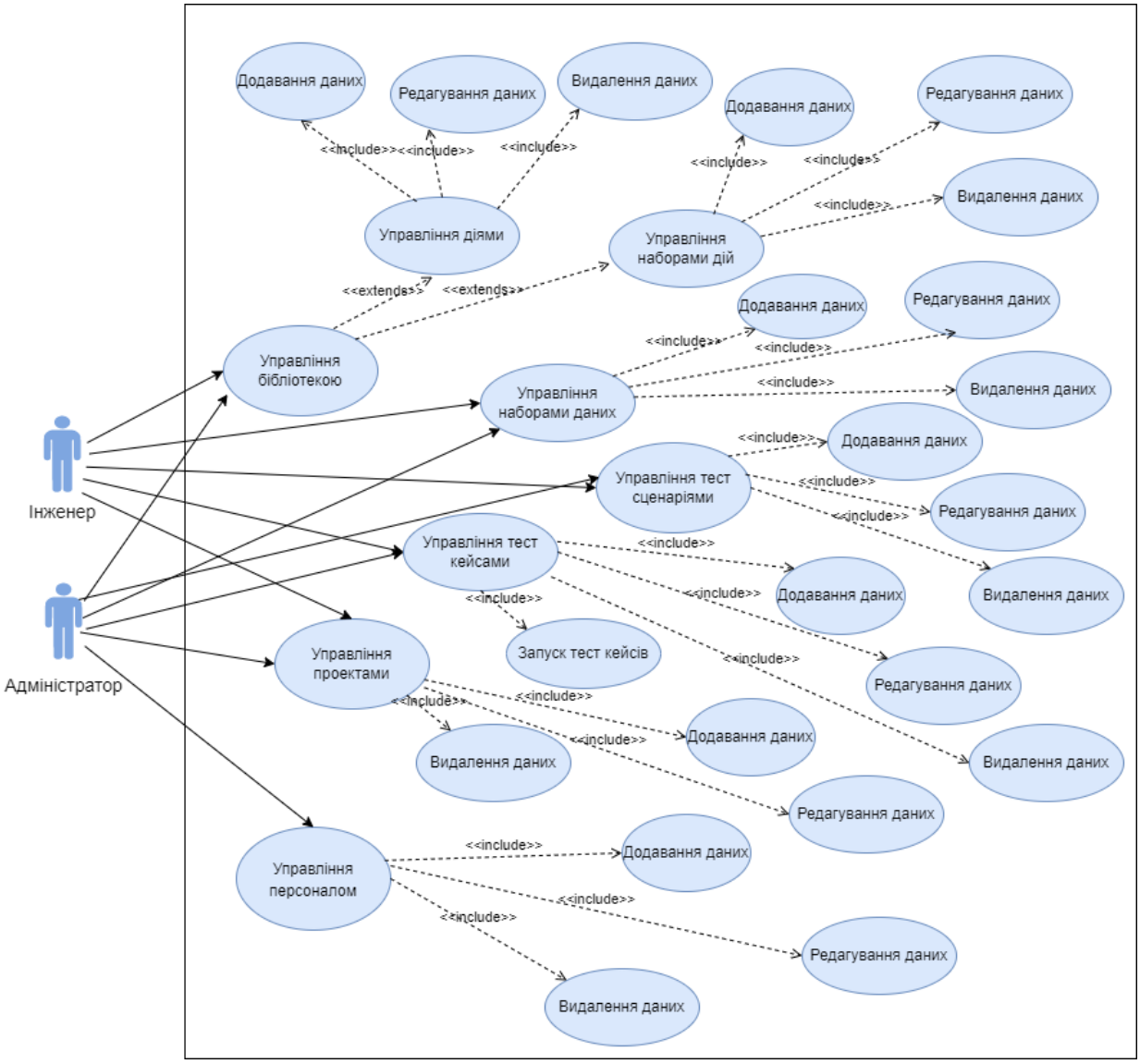

Рисунок 4.5 — Діаграма прецедентів

4.3 Опис структури бази даних

Для зберігання даних про користувачів системи, набори даних, тест сценарії та тест кейси, що є основними об’єктами функціоналу, було обрано SQL базу даних PostgreSQL, розроблено структуру таблиць та зв’язки між ними.

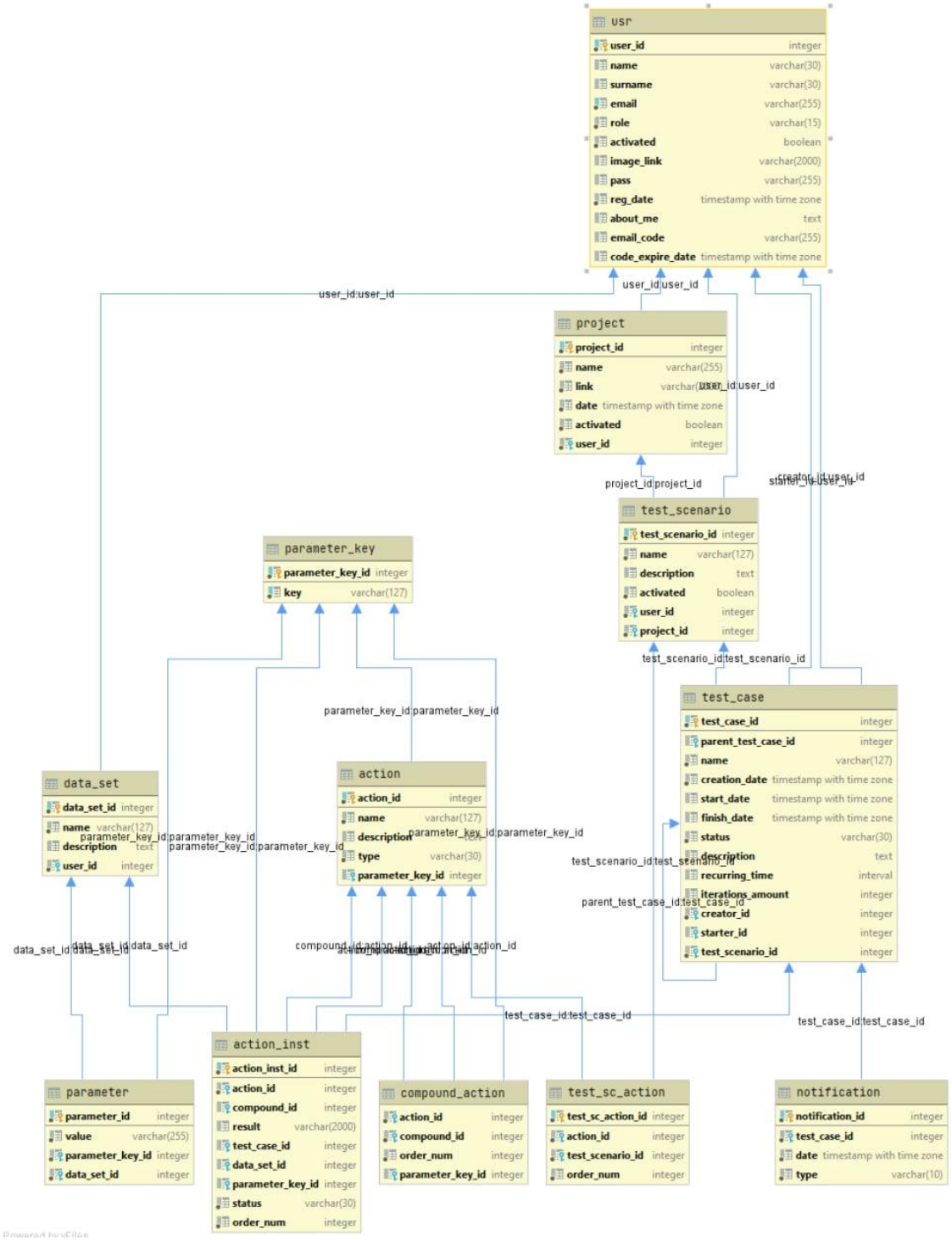

Рисунок 4.6 Фізична модель даних

Таблиця usr відповідає за зберігання інформації про користувачів:

- user_id(ідентифікатор користувача)
- name(ім’я користувача)
- surname(прізвище користувача)
- email(пошта для входу в систему)

- role(роль користувача)
- activated(статус користувача)
- image_link(посилання на зображення в профілі)
- pass(пароль акаунту)
- reg_date(дата реєстрації)
- about_me(дані про користувача)
- email_code(код активації профілю)
- code_expire_date(час тривалості сесії користувача)

Таблиця project проекту, яка зберігає інформацію про сайт, що тестується, зв'язок з таблицею користувачів один до багатьох, так як один користувач може створювати різні проекти:

- project_id(ідентифікатор проекту)
- name(назва проекту)
- link(посилання на сайт)
- date(дата створення проекту)
- activated(статус проекту)
- user_id(індентифікатор користувача)

Таблиця test_scenario зберігає інформацію про тест сценарій та проект, в якому він використовується:

- test_scenario_id(ідентифікатор тест сценарію)
- name(назва тест сценарію)
- description(опис сценарію)
- activated(статус тест сценарію)
- user_id(ідентифікатор користувача)
- project_id(ідентифікатор проекту)

Таблиця test_case є основною таблицею, що зберігає дані про тест кейс, який буде запущено:

- test_case_id(індентифікатор тест кейсу)
- parent_test_case_id(ідентифікатор батьківського тест кейсу)

- name(назва тест кейсу)
- creation_date(дата створення тест кейсу)
- start_date(дата запуску тест кейсу)
- finish_date(дата отримання результату тест кейсу)
- status(статус тест кейсу)
- description(опис тест кейсу)
- recurring_time(час виконання тест кейсу)
- iterations_amount(кількість запущених)
- creator_id(ідентифікатор інженера, що створив тест кейс)
- starter_id(ідентифікатор інженера, що запустив тест кейс)
- test_scenario_id(ідентифікатор тест сценарію)

Таблиця test_sc_action :

- test_sc_action_id(ідентифікатор дії в тест сценарії)
- action_id(ідентифікатор набору дій)
- test_scenario_id(ідентифікатор тест сценарія)
- order_num(порядок виконання дій)

Таблиця action зберігає інформацію про наявні дії в системі:

- action_id(ідентифікатор набору дій)
- name(назва набору дій)
- description(опис набору дій)
- type(тип набору дій)
- parameter_key_id(ідентифікатор ключа параметру)

Таблиця parameter_key:

- parameter_key_id(ідентифікатор ключа параметрів)
- key(ключ параметру)

Таблиця data_set зберігає інформацію про набори даних системи:

- data_set_id(ідентифікатор дата сету)
- name(назва набору даних)
- description(опис набору параметрів набору даних)

- user_id(ідентифікатор користувача)

Таблиця parameter:

- parameter_id(індентифікатор параметру)
- value(значення параметру)
- parameter_key_id(ідентифікатор ключа параметру)
- data_set_id(ідентифікатор набору даних)

Таблиця action_inst:

- action_inst_id(ідентифікатор об’єкту дії)
- action_id(ідентифікатор дії)
- compound_id(ідентифікатор набору дій, якому може належати дія)
- result(результат виконання дій)
- test_case_id(ідентифікатор тест кейсу)
- data_set_id(ідентифікатор набору даних)
- parameter_key_id(ідентифікатор ключа параметрів)
- status(статус дії)
- order_num(порядок дії)

Таблиця compound_action:

- action_id(ідентифікатор дії)
- compound_id(ідентифікатор набору дій)
- order_num(порядок виконання компаундів)
- parameter_key_id(ідентифікатор параметру ключа)

Таблиця notification зберігає дані про сповіщення проходження тест кейсу:

- notification_id(ідентифікатор сповіщення)
- test_case_id(ідентифікатор тест кейсу)
- date(дата отримання сповіщення)
- type(тип сповіщення)

4.4 Обробка запитів користувача

Основною завданням системи що розроблюється, є запуск тест кейсів, створених за запитами користувача системи: інженера чи адміністратора. Побудуємо контекстну діаграму IDEF0 для детального моделювання роботи системи. Визначимо вхідні та вихідні параметри об’єкта моделювання “Інструментарій автоматизованого тестування веб-сайтів” (рисунок 4.7).

Схеми IDEF0 зазвичай включають наступні компоненти:

- Контекстна схема — сама верхня схема в моделі IDEF0.
- Ієрархія декомпозиції IDEF0 з використанням батьківських і дитячих зв'язків.
- Дерева вузлів — структура вузлів у вигляді дерева, корневих на обраному вузлі, які використовуються для представлення повної декомпозиції IDEF0 в одній схемі.

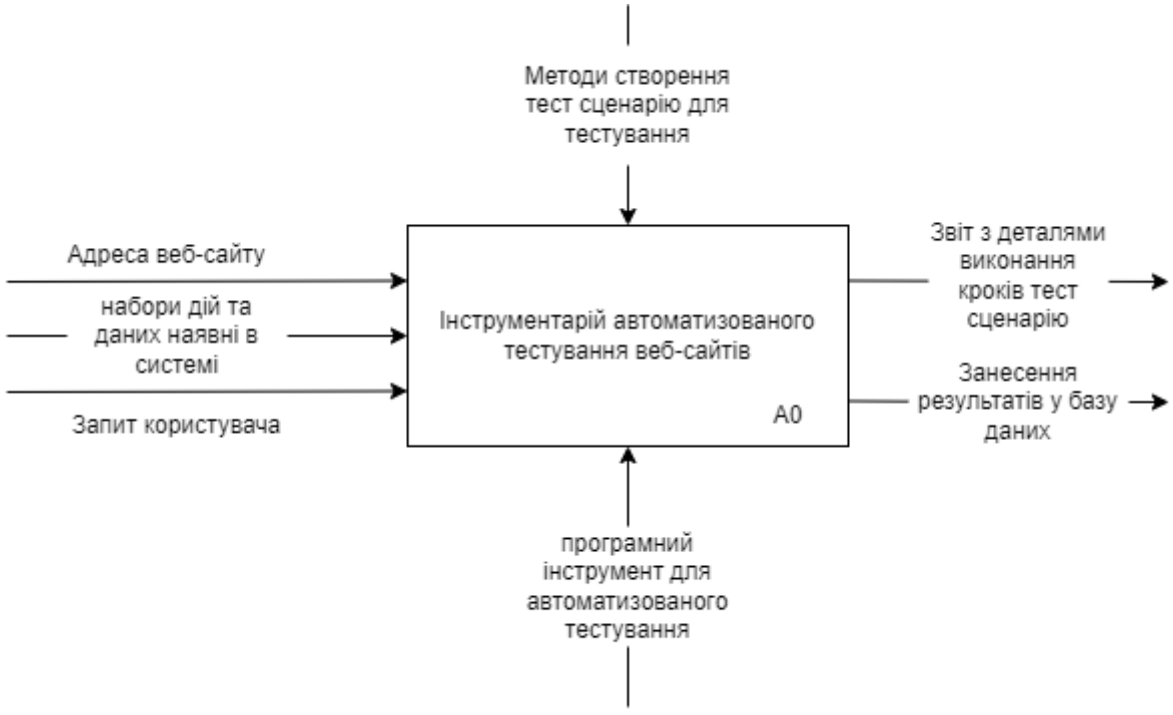

Рисунок 4.7 — Контекстна діаграма IDEF0 “Інструментарій автоматизованого тестування веб-сайтів”

Нотація IDEF0 підтримує послідовну декомпозицію процесу до необхідного рівня деталізації. Дочірня діаграма, що створюється при декомпозиції, охоплює ту саму область, що і батьківський процес, але описує її докладніше.

Для даної системи, доцільно провести декомпозиціюю наступних процесів:

— Підготовка даних та сценаріїв для тестування

— Обробка запиту користувача

— Отримання результату та занесення до бази даних

Побудуємо декомпозицію контекстної діаграми (рисунок 4.8).

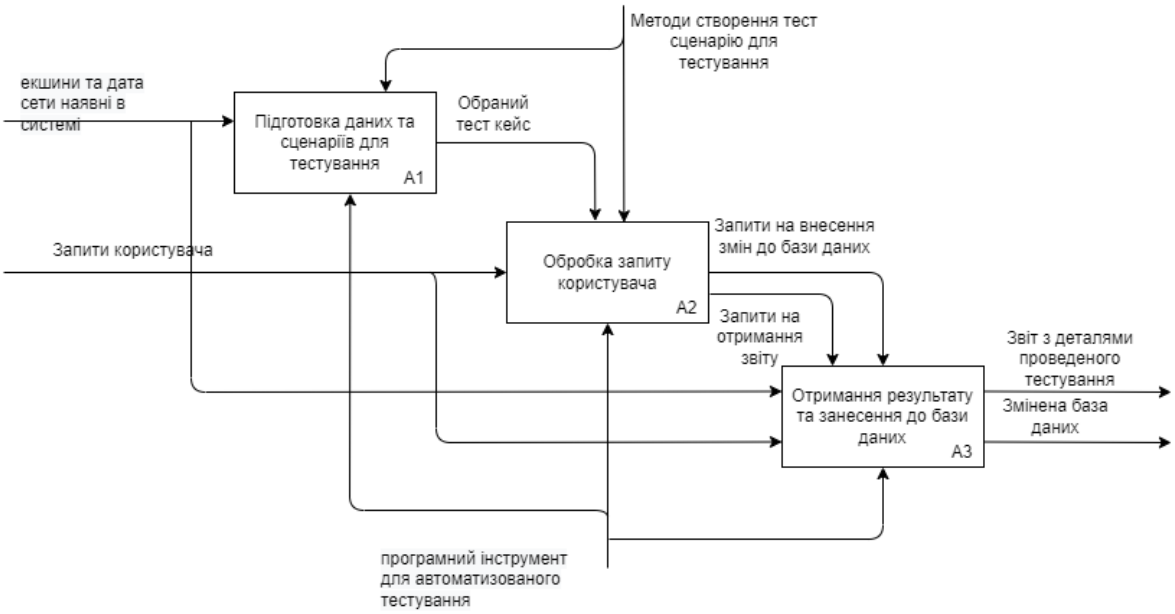

Рисунок 4.8 — Декомпозиція контекстної діаграми

4.4.1 Підготовка даних та сценаріїв для тестування

Перший процес описує підготовку даних та сценаріїв для тестування. Система містить набір дій, які можуть бути використані для складання тест кейсу, та дата сети, які можуть бути доповнені, або створені нові. Вони є вхідними даними процесу. На даному етапі відбувається формування тест кейсу. За методами тестування, формується сценарій, далі необхідно обрати

набір дій, після чого зі списку дата сетів, підібрати дані для кожної дії. Скомбіновані таким чином тест сценарій та дата сет дають можливість створити тест кейс, який потім буде запущено для перевірки сайту.

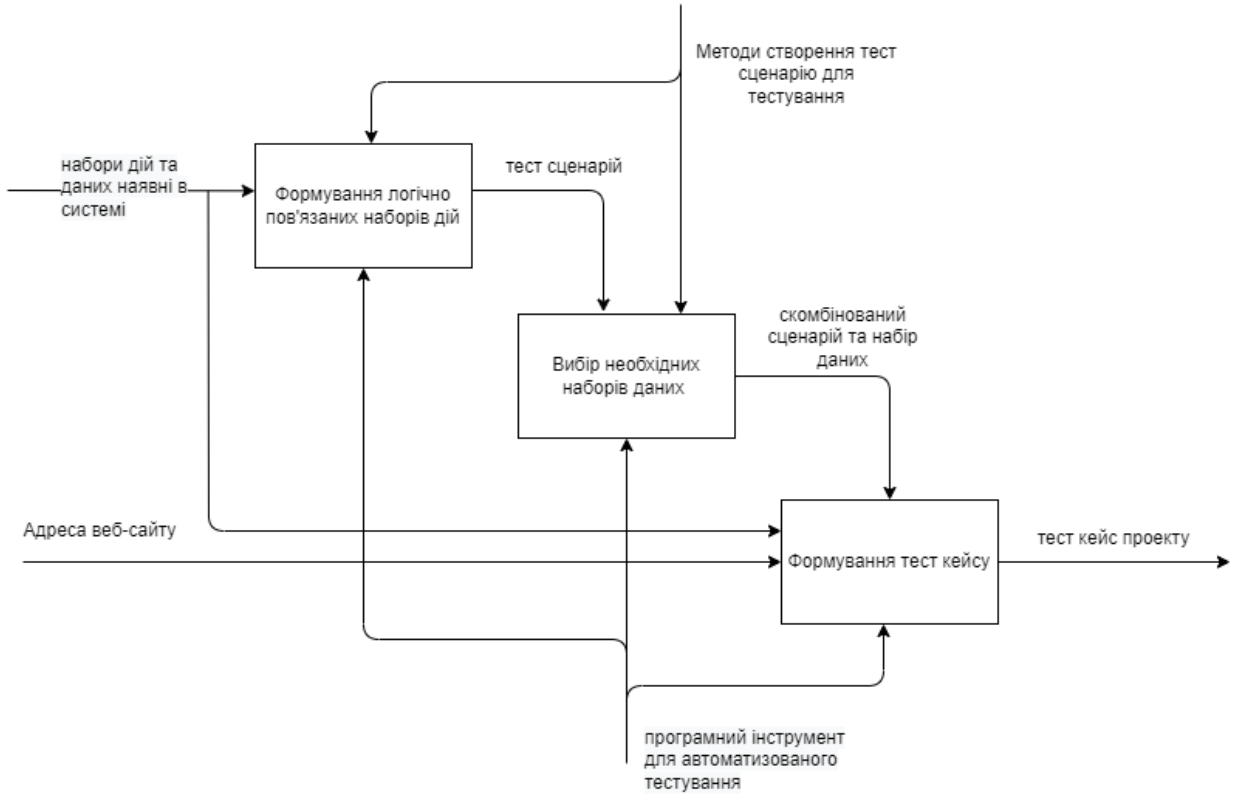

Рисунок 4.9 — Декомпозиція модуля “Підготовка даних та сценаріїв для тестування”

4.4.2 Виконання запиту

Далі на основі обраного тест кейсу, формується запит на його запуск. Усі запити користувачів виконуються асинхронно. Кожен тест кейс належить проекту, тому виконується перевірка на правильність введеної URL-адреси, наявність тест сценарію та дата сету в тест кейсі. У програмі створюється інстанс необхідного веб-браузера для тестування сайту, викликається метод на запуск тест кейсу, в який передається інстанс та тест кейс. Виконання тест кейсу відбувається за обраним сценарієм, результат автоматично оновлюється.

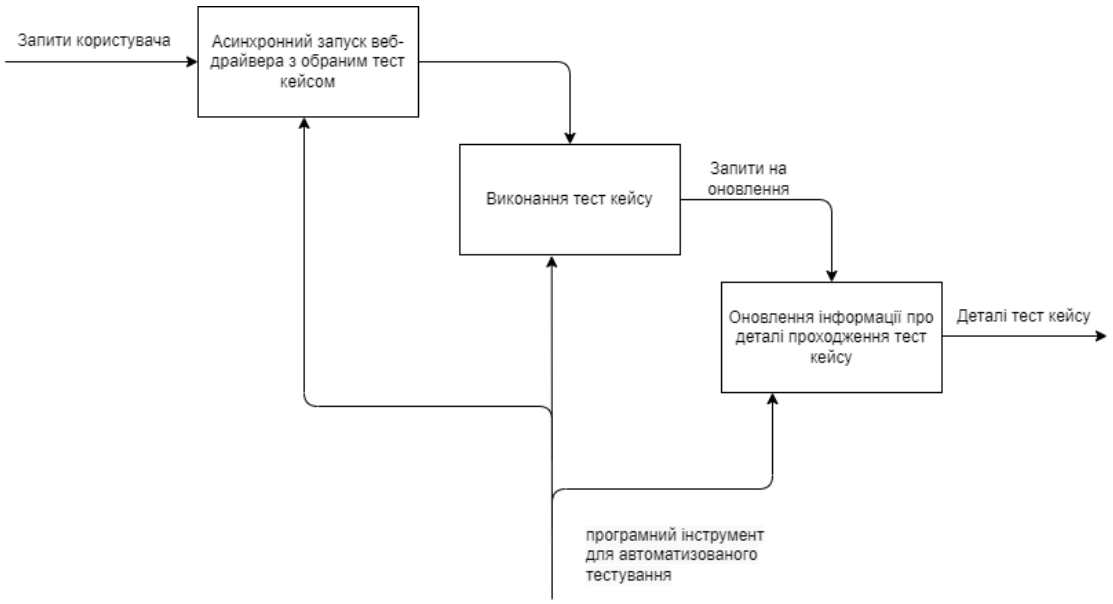

Рисунок 4.10 — Декомпозиція модуля “Обробка запиту користувача”

4.4.3 Отримання результату та занесення до бази даних

На останньому етапі винанання тестування, формується остаточний звіт деталей тест кейсу. До бази даних записуються відповідні зміни, про кількість запусків даного тест кейсу та результат його виконання. Результат декомпозиції (рисунок 4.11).

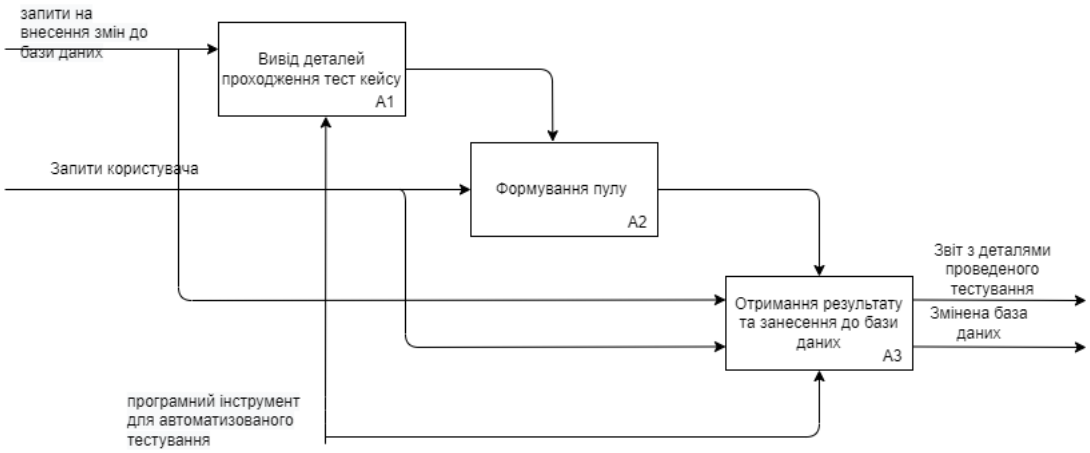

Рисунок 4.11 — Декомпозиція модуля “Отримання результату та занесення до бази даних”

4.5 Висновки

У розділі детально розглянуто архітектуру системи. Розроблена система складається з клієнтської та серверної частин. Спілкування між якими відбувається по REST API. Серверна частина реалізована на мові програмування Java та фреймворку Spring. Для розробки клієнтської частини, було використано фреймворк Angular. Наведено діаграму класів та діаграму прецедентів для відображення функціоналу системи. Побудовано декомпозицію модулів системи.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Створена система дозволяє інженерам розробляти тест сценарії для тестування різних веб-сайтів, використовуючи функції наявні в системі та набори даних та має інтуїтивно зрозумілий інтерфейс. Розроблена система для автоматизованого тестування підтримує роботу на базі Windows, macOS та Linux, а також найпоширеніших браузерів Google Chrome, Mozilla Firefox, Safari, Edge, Internet Explorer.

5.1 Робота користувача з системою

Після запуску серверної та клієнтської частини, переходимо за посиланням на сторінку авторизації.

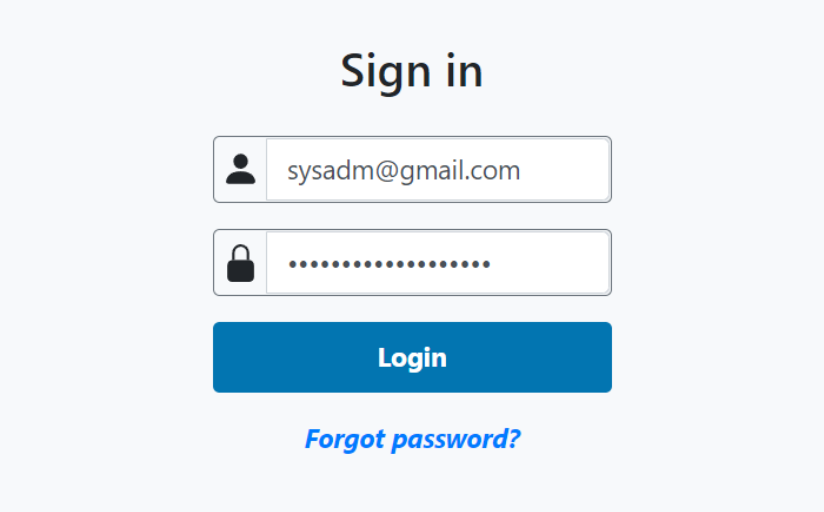

Рисунок 5.1 – Сторінка авторизації

Профіль користувача відображає статистику даного користувача, а саме кількість запущених ним тест кейсів та особову інформацію.

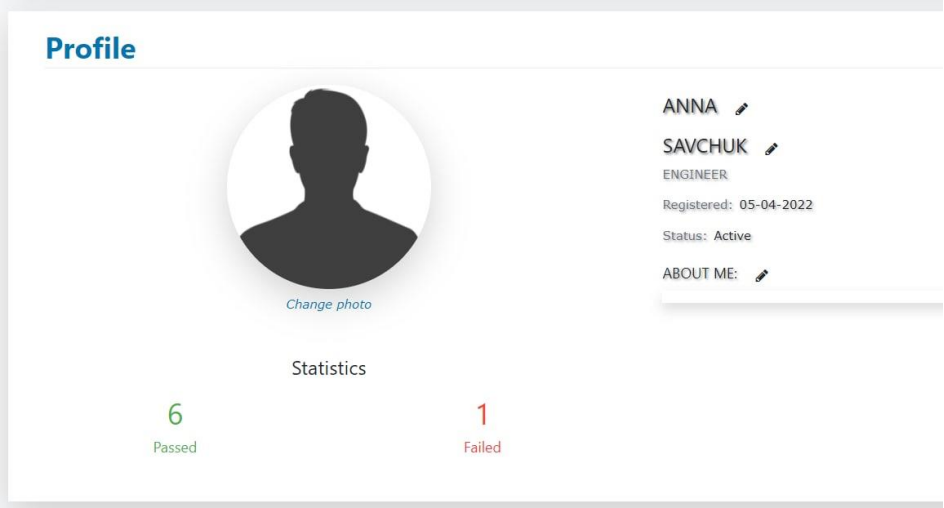

Рисунок 5.2 – Профіль користувача

5.1.1 Створення проекту для тестування

Проект, є основним елементом системи, що ідентифікує веб-сайт на якому проводиться тестування. Користувач повинен створити новий проект, вказавши його URL-адресу та унікальне ім’я. Кожен проект містить список тест кейсів, які написані для його перевірки.

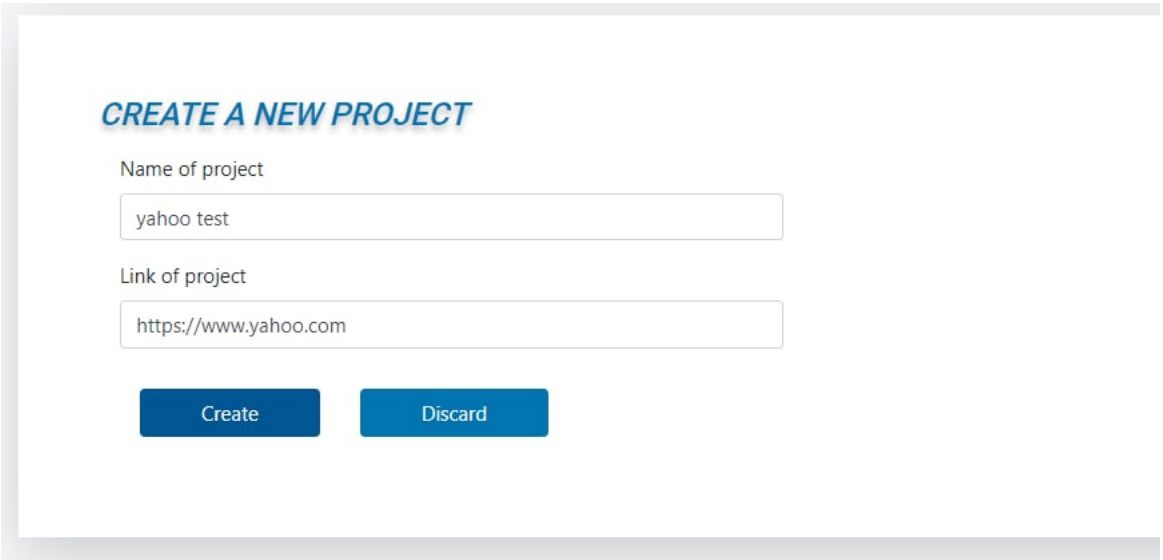

Рисунок 5.3 – Створення проекту

Якщо проект вже створений, його можна знайти на сторінці з усіма наявними проектами в системі. Можна відфільтрувати значення за ім’ям та

посиланням проекту. Відредагувати назву проекту чи посилання на веб-сайт.

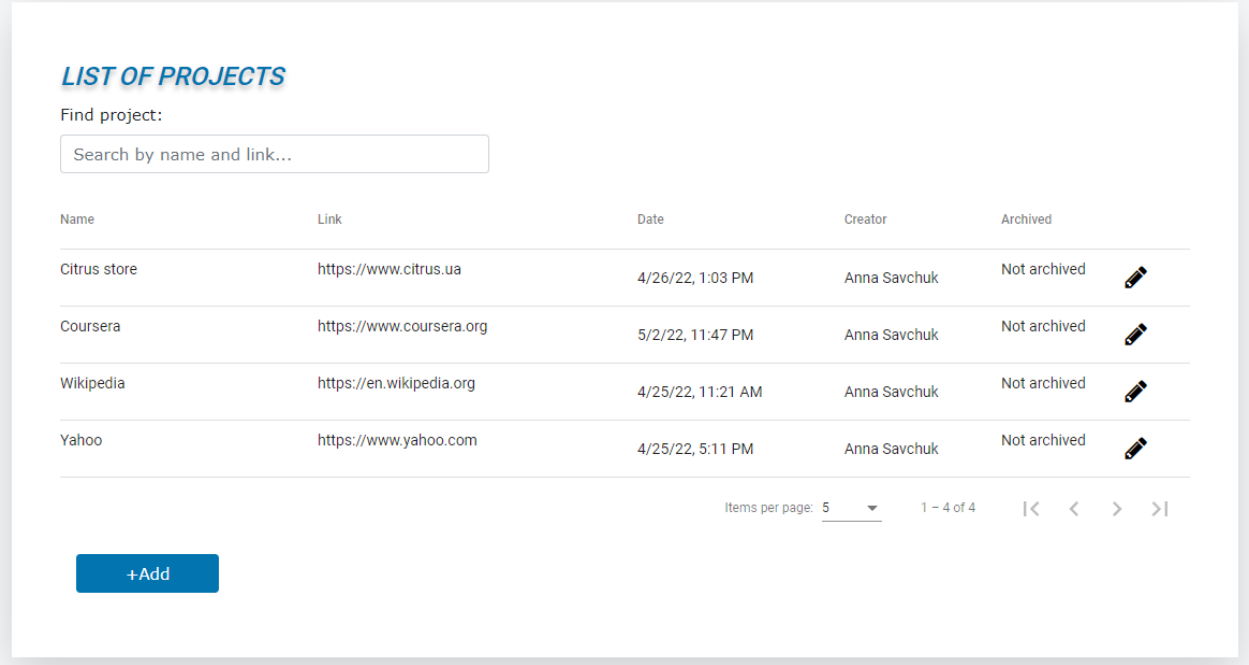

Рисунок 5.4 – Список проектів

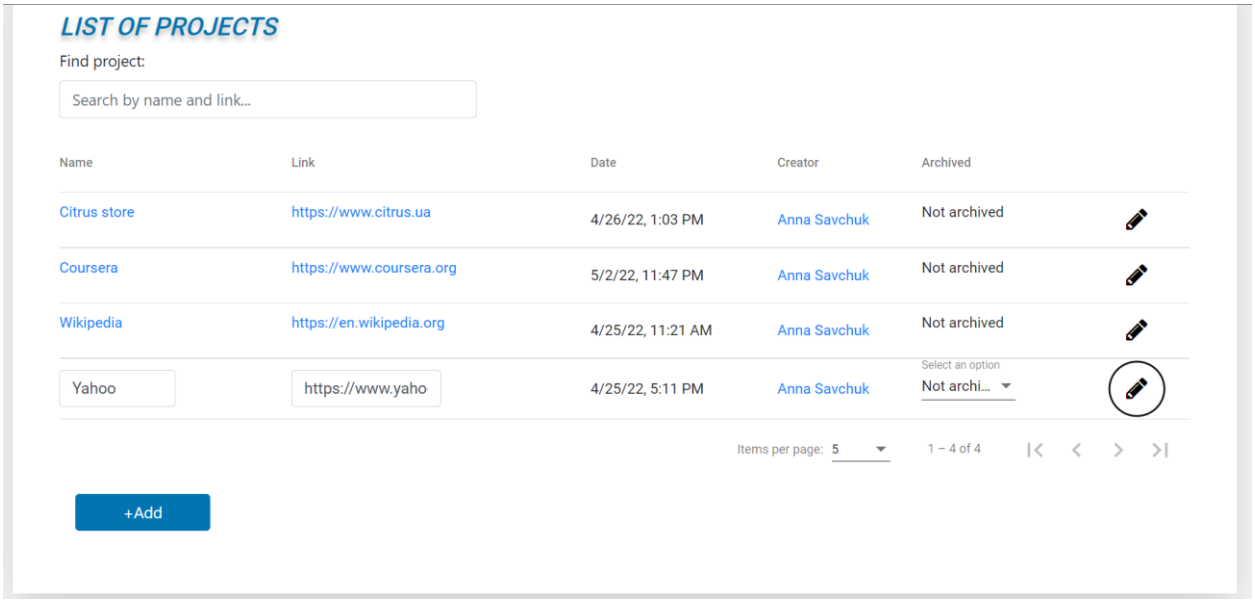

Рисунок 5.5 – Редагування проекту

5.1.2 Створення тестового сценарію

Користувач повинен створити тест сценарій за яким буде проведено тестування. Тест сценарій складається з набору дій (actions). Вони є

універсальними та можуть бути використані для тестування різних сайтів. Тому, необхідно з бібліотеки яка відображає наявні в системі дій (actions), обрати потрібні та розмістити їх у правильному порядку. Далі вказати унікальне ім’я тест сценарію та його короткий опис.

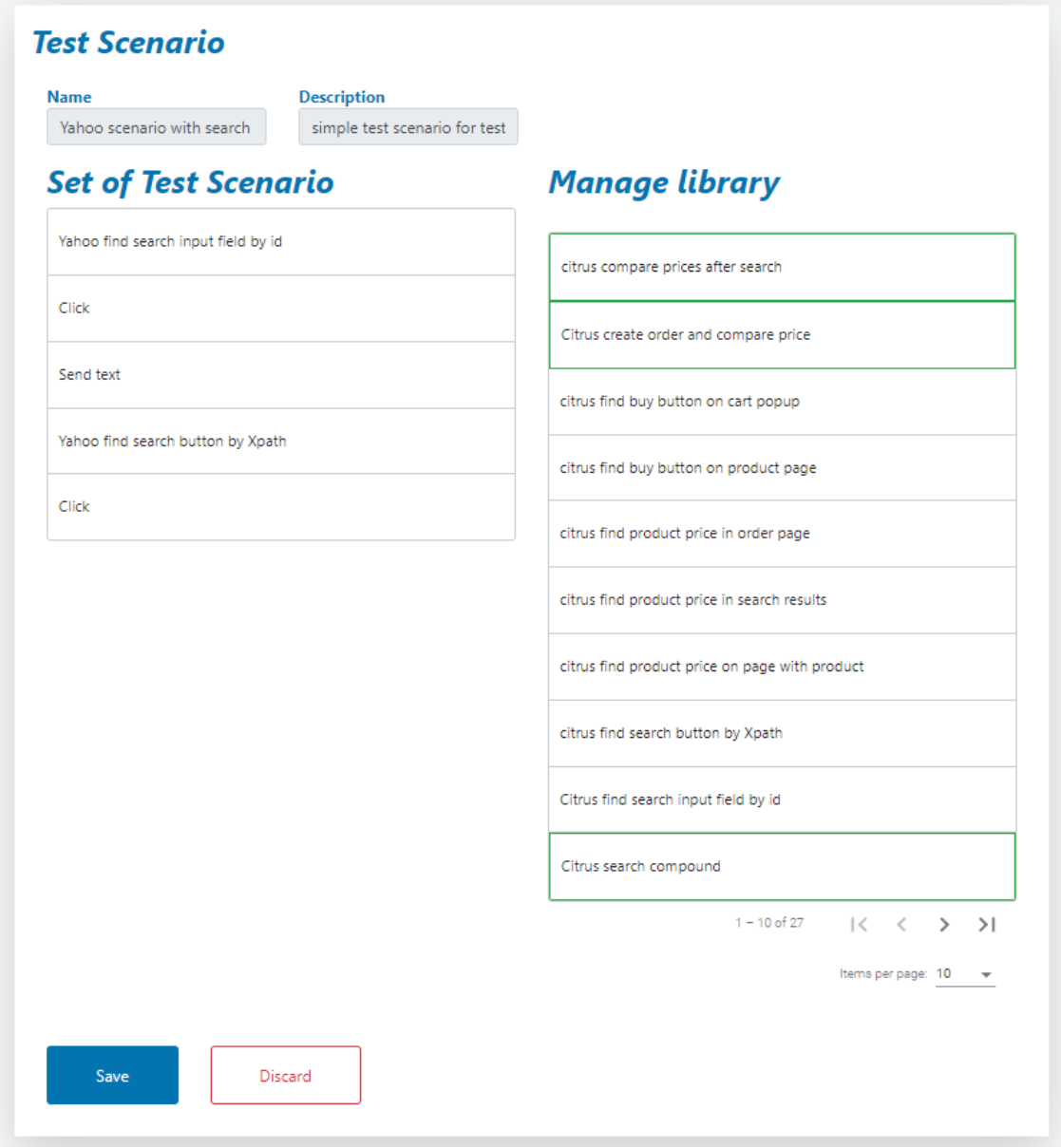

Рисунок 5.6 – Створення тестового сценарію

5.1.3 Створення тестового сценарію

Далі переходимо на сторінку створення нового тест кейсу. В таблиці відображається створений попередньо тест сценарій, його автор та короткий

опис. Користувач натискає на кнопку “New Test Case” та переходить на сторінку створення нового тест кейсу.

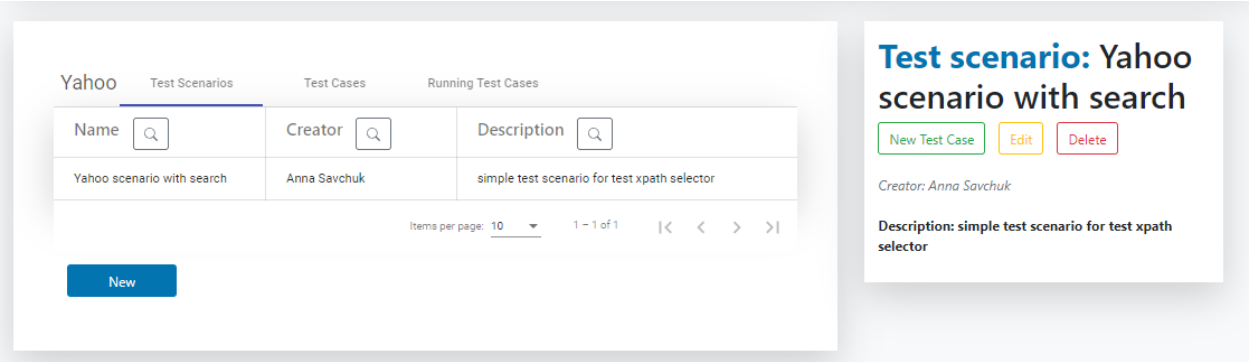

Рисунок 5.7 – Сторінка створення тест кейсу

Тест кейс для будь-якого проекту повинен містити набір даних з якими буде проведено тестування. Обираємо дата сет, або декілька, які підходять до функцій, що описані в тест сценарії.

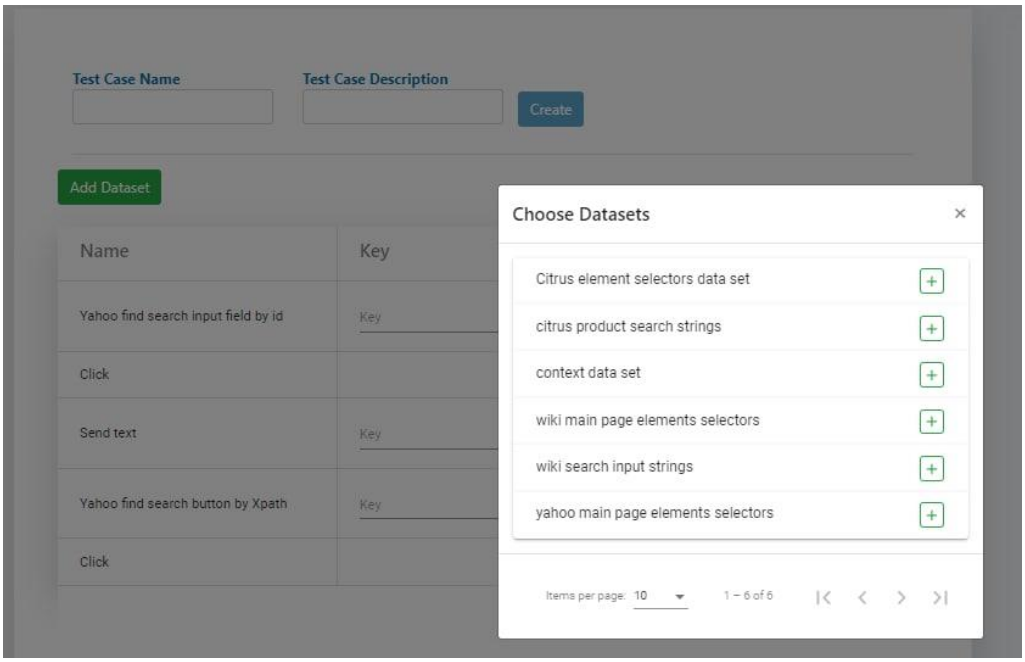

Рисунок 5.8 – Вибір набору даних

Користувач має список функцій з тестового сценарію, до якого обирає потрібний набір даних з списку всіх наявних у системі. Між значеннями кожної функції та набором даних встановлюється відповідність за вказаним

ключем.

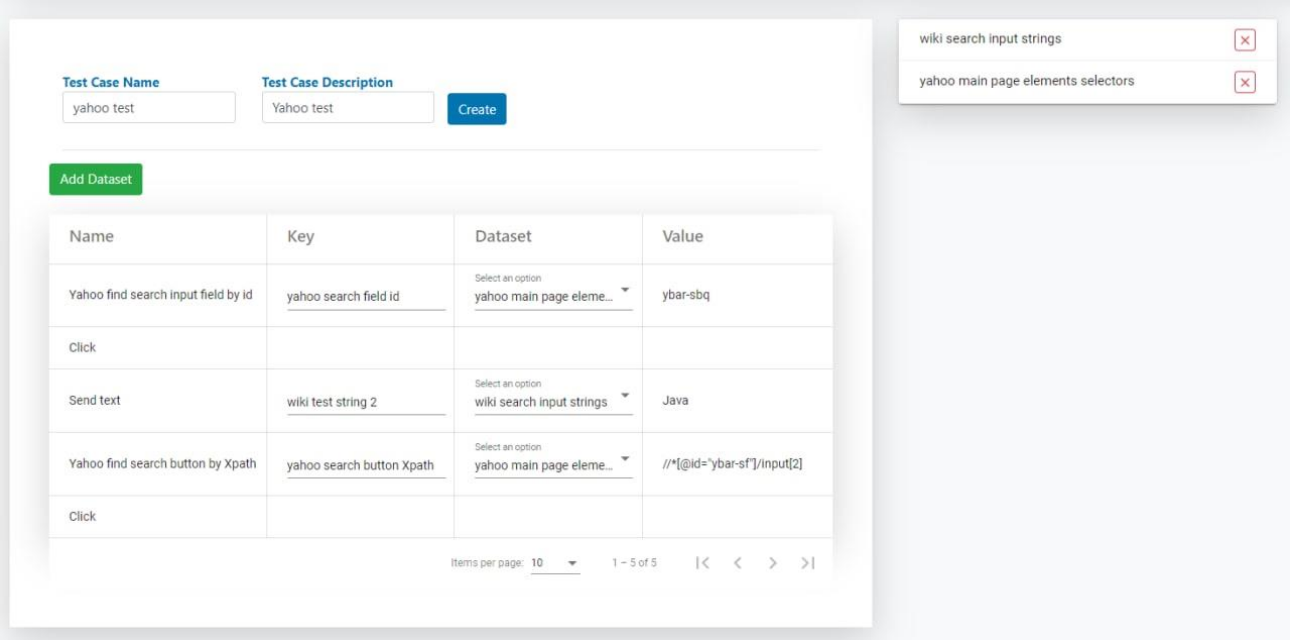

Рисунок 5.9 – Формування тест кейсу

5.1.4 Запуск тест кейсу

Після створення тест кейсу користувач обирає тест кейс для запуску та натискає “Run”.

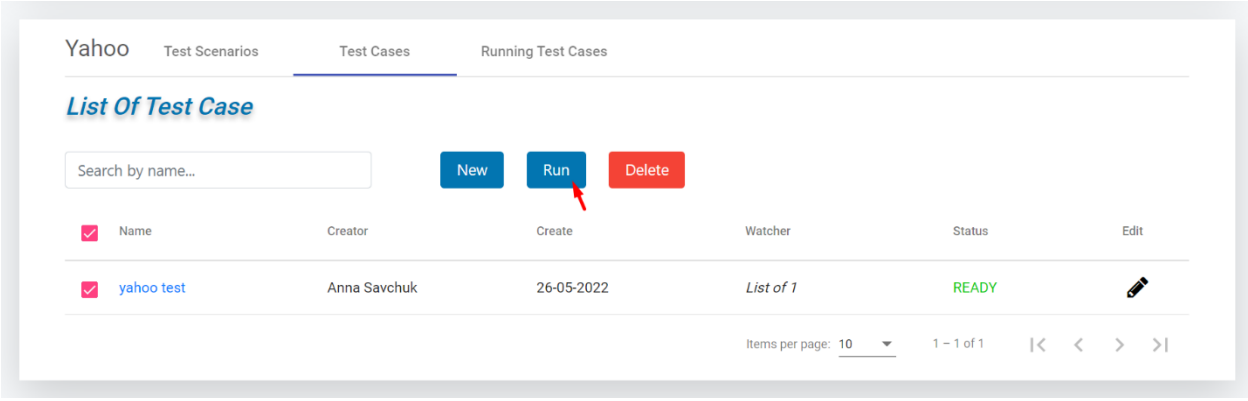

Рисунок 5.10 Запуск тест кейсу

Далі система посилає запит за вказаною адресою, відкривається вікно браузера, Selenium знаходить в структурі DOM дерева сайту поле пошуку,

вводить текст, який необхідно знайти, шукає кнопку пошуку для відправлення запиту. На кожному кроці виконання, оновлюється сторінка з деталями тест кейсу, де інформується про успішність чи неуспішність виконання кожного кроку.

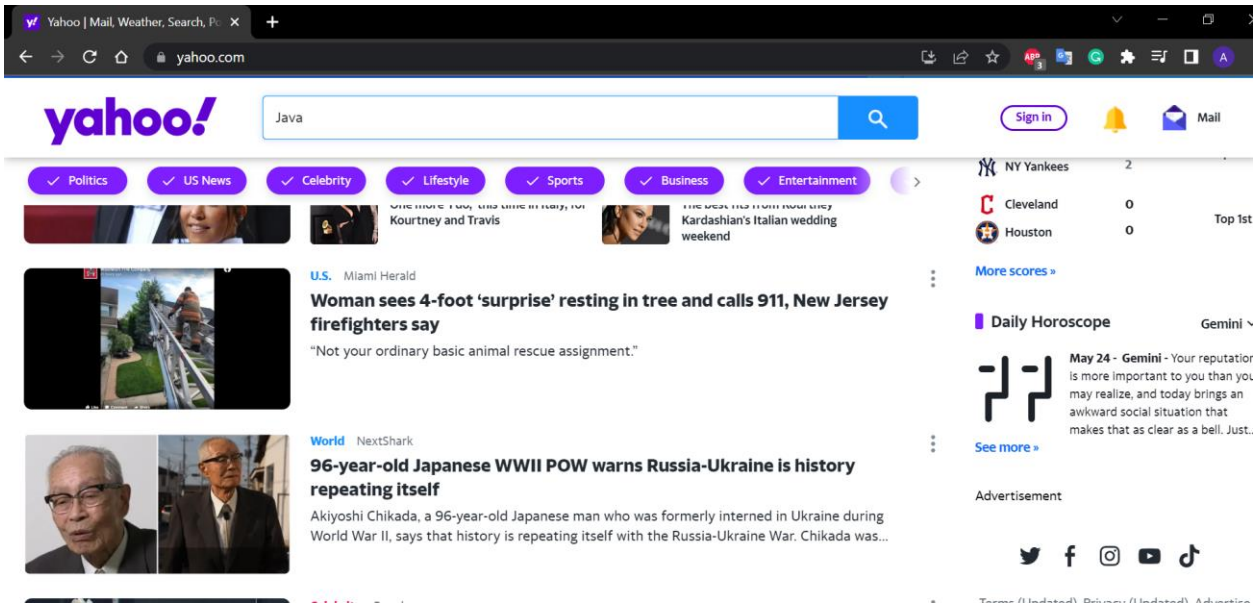

Рисунок 5.11 – Вікно браузера

5.1.5 Аналіз виконання тест кейсу

Перейшовши на вкладку з запущеними тест кейсами, можемо детально подивитись на результат виконання тест кейсу покроково. На сторінці динамічно змінюється відсоток виконання запущеного тест кейсу. Інформація про автора тест кейсу та інженера який запустив тест кейс, дата початку та завершення його виконання. У таблиці нижче відображаються крокі з тест сценарію та статус їх виконання. Отримані результати, дозволяють проаналізувати виявлені помилки, у випадку якщо одна з дій пройшла неуспішно. Наступний тест кейс повинен бути створений з врахуванням цих результатів.

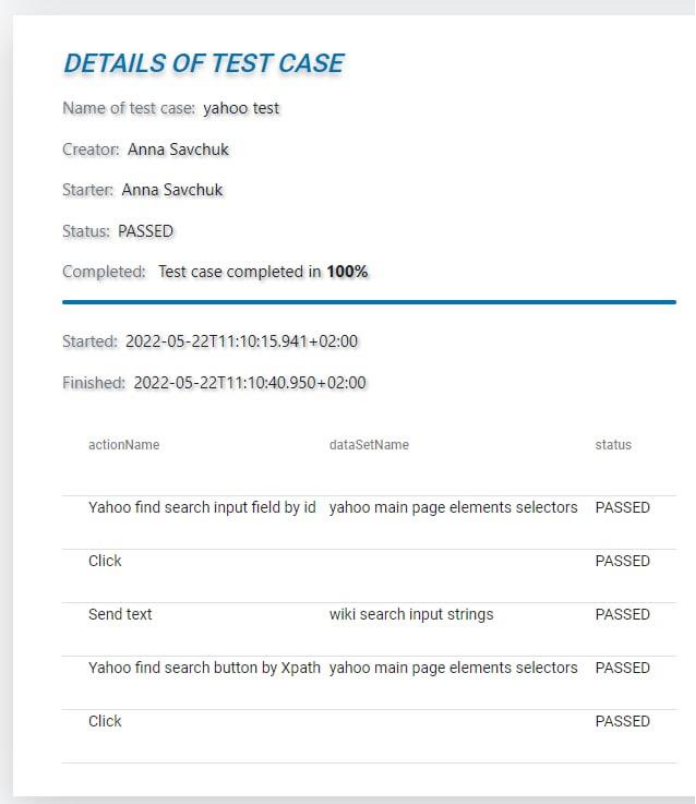

Рисунок 5.12 – Сторінка деталей тест кейсу

Результати запущених тест кейсів можна переглянути на головній сторінці, в таблиці розміщено основну інформацію про результати виконання тест кейсів та діаграму, побудовану на статистиці запущених тест кейсів.

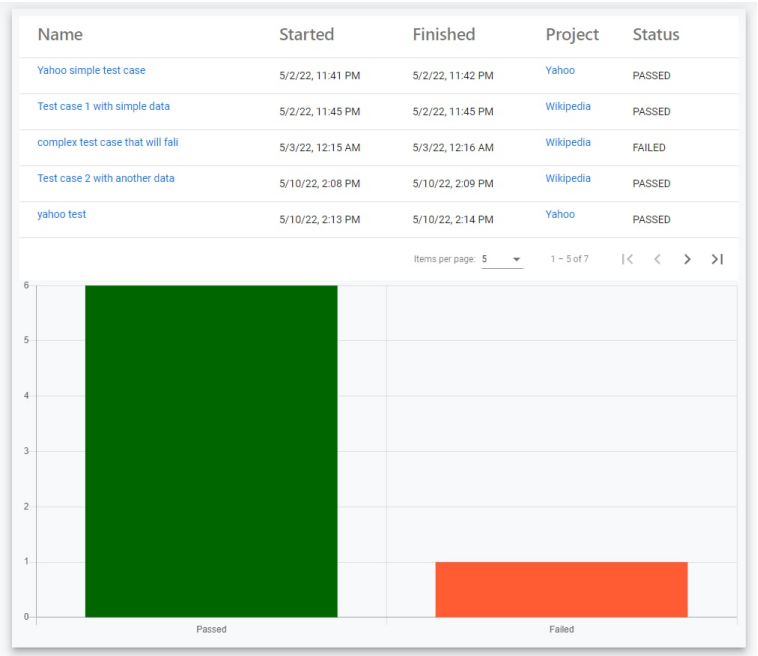

Рисунок 5.13 Сторінка результатів попередніх тестувань

5.2 Висновки

У п’ятому розділі описано системні вимоги для запуску та роботи з системою. Розглянуто основний сценарій роботи користувача з програмною системою. Продемонстровано створення основних компонентів у системі, набір можливих даних для тестування за створеним тест сценарієм.

ВИСНОВКИ

У дипломній роботі реалізовано систему для автоматизованого тестування веб-сайтів, яка допомагає уникнути регресії та надає можливість проаналізувати кроки раніше запущених тест кейсів. Розглянуті основні види тестування програмного забезпечення та їх використання в залежності від проектів. Зважаючи на це, обрано інструмент автоматизованого тестування, що є одним із основних компонентів системи.

Розроблено алгоритм та обрано інструменти для реалізації програмного продукту. Створено модель для проведення тестування веб-сайтів та аналізу отриманої інформації після проходження тестів.

В процесі розробки програмного рішення, було розглянуто існуючі системи, що дозволяють проводити тестування програмного забезпечення на базі різних інструментів. Проведено перевірку дієздатності системи шляхом тестування різних веб-сайтів, проаналізовано результати виконання тест кейсів на кожному кроці описаному в тест сценарії.

Розроблено систему, яка дозволяє проводити автоматизоване тестування веб-сайтів, покращуючи якість та надійність програмного забезпечення. Описано архітектуру системи, основні модулі програмної реалізації, обрані технології та інструменти. Розглянуто основні сценарії використання системи.

При реалізації програмної системи було обрано мову програмування Java та фреймворк Spring Framework для серверної частини. Клієнтська частина розроблена з використанням фреймворку Angular. Середовищем розробки було обрано Intellij IDEA.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Angular Development with TypeScript 2nd Edition/ Yakov Fain. Manning; 2nd edition 2017. 560p.
2. Angular: Up and Running: Learning Angular/ Seshadri S.. Step by Step. O'Reilly Media, 2018. 312p.
3. Clean Code: A Handbook of Agile Software Craftsmanship/Robert C. Martin. Pearson 2008. 464p.
4. Effective Java, 3rd Edition/ Joshua Bloch. Upper Saddle River, NJ : Addison-Wesley, 2017. 416p.
5. Fowler M. Monolith First / Martin Fowler [Електронний ресурс] – режим доступу: https://martinfowler.com/bliki/MonolithFirst.html#footnotetypical-monolith.
6. Hands-On Software Architecture with Java: Learn key architectural techniques and strategies to design efficient and elegant Java applications/ Giuseppe Bonocore. Packt Publishing 2021. 510p.
7. Introduction to Algorithms, fourth edition 3th Edition/ Thomas H. Cormen. MIT Press 2009. 1292p.
8. Java Design Patterns: A Hands-On Experience with Real-World Examples 2nd ed. Edition/ Vaskaran Sarcar. Apress 2018. 533p.
9. Java: The Complete Reference, Eleventh Edition/Herbert Schildt. McGraw Hill Education 2018. 1248p.
10. Modern Java in Action: Lambdas, streams, functional and reactive programming 2nd Edition/ Raoul-Gabriel Urma, Mario Fusco, Alan Mycroft. Manning – 2018. 592p.
11. Practical PostgreSQL/Joshua D. Drake, John C. Worsley. O'Reilly Media 2002. 640p.
12. PostgreSQL: Documentation: 14: Index Types [Електронний ресурс]. —режим доступу: https://www.postgresql.org/docs/14/functions.html

13. RESTful Java Web Services: A pragmatic guide to designing and building RESTful APIs using Java, 3rd Edition 3rd Revised edition/ Bogunuva Mohanram Balachandar. Packt Publishing 2017. 420p.
14. Spring in Action, 5th edition/Craig Walls. Manning Publications 2018. 520p.
15. Spring Framework 5.3.20: [Електронний ресурс]. – режим доступу: https://spring.io/projects/spring-framework
16. Spring Security 5.7.1: [Електронний ресурс]. – режим доступу: https://spring.io/projects/spring-security
17. Test Driven Development: By Example 1st Edition/ Kent Beck. Addison-Wesley Professional 2002. 240p.
18. Angular Development with TypeScript 2nd Edition/ Yakov Fain, Anton Moiseev. Manning; 2nd edition 2018. 560p.
19. Angular: Up and Running: Learning Angular, Step by Step 1st Edition/
20. Web Engineering: Modelling and Implementing Web Applications / G. Rossi, P. Oscar, S. Daniel. – New York: Springer-Verlag, 2008. 476p.

ДОДАТОК А

Інструментарій автоматизованого тестування веб-сайтів

Текст програмного модулю
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ_82311

Аркушів 12

Київ — 2022

```
@RestController
@RequestMapping("api/test-scenario")
public class TestScenarioController {
   private final TestScenarioService testScenarioService;

   public TestScenarioController(TestScenarioService testScenarioService) {
      this.testScenarioService = testScenarioService;
   }

   @GetMapping("{id}")
        @ApiOperation("Get test scenario by ID")
   public TestScenario getTestScenarioById(@PathVariable int id) {
      return testScenarioService.getTestScenarioById(id);
   }

   @GetMapping("decomposed/{id}")
        @ApiOperation("Get decomposed test scenario by ID")
   public TestScenario getDecomposedTestScenarioById(@PathVariable int id) {
      return testScenarioService.getDecomposedTestScenarioById(id);
   }

   @PostMapping
        @ApiOperation("Add new test scenario")
   @ResponseStatus(value = HttpStatus.OK)
   public void create(@RequestBody TestScenario testScenario) {
      testScenarioService.createTestScenario(testScenario);
   }

   @GetMapping("list/{projectId}")
        @ApiOperation("Get all test scenarios by defined page and project ID")
   public ResponseEntity<Page<TestScenarioDto>> getAllByProject(
         @RequestParam(defaultValue = "10") int pageSize,
         @RequestParam(defaultValue = "0") int pageIndex,
         @RequestParam(defaultValue = "") String filterBy,
         @RequestParam(defaultValue = "") String filter,
         @RequestParam(defaultValue = "") String orderBy,
         @RequestParam(defaultValue = "") String order,
         @PathVariable int projectId
   ) {
      Page<TestScenarioDto> testScenarioList =
testScenarioService.getAllByPage(pageIndex, pageSize, filterBy, filter, orderBy,
order, projectId);
      return new ResponseEntity<>(testScenarioList, HttpStatus.OK);
   }
```

```
    @GetMapping("{testScenarioId}/projectId")
        @ApiOperation("Get project ID by provided test scenario ID")
    public Integer getProjectIdOfTestScenario(@PathVariable Integer
testScenarioId) {
        return testScenarioService.getProjectIdOfTestScenario(testScenarioId);
    }

    @GetMapping("/list")
        @ApiOperation("Return all test scenarios by page")
    public ResponseEntity<Page<TestScenarioDto>> getAll(
            @RequestParam(defaultValue = "10") int pageSize,
            @RequestParam(defaultValue = "0") int pageIndex,
            @RequestParam(defaultValue = "") String filterBy,
            @RequestParam(defaultValue = "") String filter,
            @RequestParam(defaultValue = "") String orderBy,
            @RequestParam(defaultValue = "") String order

    ) {

        Page<TestScenarioDto>                    testScenarioList                    =
testScenarioService.getAllByPage(pageIndex, pageSize, filterBy, filter, orderBy,
order, 0);
        return new ResponseEntity<>(testScenarioList, HttpStatus.OK);
    }

    @PutMapping
        @ApiOperation("Edit project by test scenario")
        @ResponseStatus(value = HttpStatus.OK)
    public void editProjectByName(@RequestBody TestScenario testScenario) {
        testScenarioService.editTestScenario(testScenario);
    }

    @DeleteMapping("{testScenarioId}")
        @ApiOperation("Delete test scenario by test scenario ID")
    public void deleteTestScenario(@PathVariable int testScenarioId) {
        testScenarioService.deleteTestScenario(testScenarioId);
    }
}

@Service
public class TestScenarioServiceImpl implements TestScenarioService {
    private final TestScenarioDao testScenarioDao;
    private final CompoundDao compoundDao;

    public          TestScenarioServiceImpl(TestScenarioDao          testScenarioDao,
```

```
CompoundDao compoundDao) {
        this.testScenarioDao = testScenarioDao;
        this.compoundDao = compoundDao;
    }

/**
     * Створення тестового сценарію.
     *
     * @param testScenario тестовий сценарій, який буде створено в системі.
     */
    @Override
    @Transactional
    public void createTestScenario(TestScenario testScenario) {
        int testScenarioId = testScenarioDao.create(testScenario);

testScenarioDao.addManyActionOrCompound(testScenario.getListActionCompou
ndId().stream().mapToInt(i -> i).toArray(), testScenarioId);
    }

         /**
     * Повертає всі тестові сценарії на сторінці.
     *
     * @param page номер сторінки.
     * @param size розмір сторінок.
     * @param filterBy параметр, за яким фільтруються значення.
     * @param orderBy параметр, що визначає сортування елеметнів.
     * @param order визначає порядок елементів.
     * @param projectId ID ідентифікатор проекту в якому використовується
тестовий сценарій.
     */

    @Override
    public Page<TestScenarioDto> getAllByPage(int page, int size, String filterBy,
String filter, String orderBy, String order, int projectId) {
        orderBy                                                              =
TestScenarioFilterAndOrderByEnum.getEnum(orderBy).getValue();
        filterBy                                                             =
TestScenarioFilterAndOrderByEnum.getEnum(filterBy).getValue();
        if (!order.toLowerCase().equals("desc")) {
            order = "";
        }
        if (projectId != 0) {
            return new Page<>(testScenarioDao.getAllByPageAndProject(page, size,
filterBy, filter, orderBy, order, projectId),
testScenarioDao.getSizeOfProjectResultSet(filterBy, filter, projectId));
```

```
        }
        return new Page<>(testScenarioDao.getAllByPage(page, size, filterBy, filter,
orderBy, order), testScenarioDao.getSizeOfResultSet(filterBy, filter));
    }

/**
     * Змінює поля тест сценарію.
     *
     * @param testScenario ідентифікатор тестового сценарію.
     */
    @Override
    @Transactional
    public void editTestScenario(TestScenario testScenario) {
        int testScenarioId = testScenario.getTestScenarioId();
        if (!testScenarioDao.checkForTestCaseOnIt(testScenarioId)) {
            testScenarioDao.dropActionOrCompound(testScenarioId);

testScenarioDao.addManyActionOrCompound(testScenario.getListActionCompou
ndId().stream().mapToInt(i -> i).toArray(), testScenarioId);
            testScenarioDao.edit(testScenario);
        } else {
            this.createTestScenario(testScenario);
        }

    }

/**
     * Видаляє тестовий сценарій за ідентифікатором.
     *
     * @param testScenarioId ідентифікатор тест сценарію
     */
    @Override
    @Transactional
    public void deleteTestScenario(int testScenarioId) {
        if (!testScenarioDao.checkForTestCaseOnIt(testScenarioId)) {
            testScenarioDao.dropActionOrCompound(testScenarioId);
            testScenarioDao.delete(testScenarioId);
        } else {
            testScenarioDao.makeUnactivated(testScenarioId);
        }
    }

/**
     * Повертає тест сценарій за вказаним ідентифікатором.
     *
```

```
     * @param id ідентифікатор тестового сценарію.
     */
    @Override
    public TestScenario getTestScenarioById(int id) {
        Optional<TestScenario> testScenarioOptional = testScenarioDao.getById(id);
        if (testScenarioOptional.isPresent()) {
            TestScenario testScenario = testScenarioOptional.get();
            testScenario.setActions(getComponents(id));
            return testScenario;
        }
        return new TestScenario();
    }

    private List<TestScenarioComponent> getComponents(int id) {
        List<TestScenarioComponent> components =
testScenarioDao.getComponents(id);
        for (TestScenarioComponent component : components) {
            if (component.getAction().getType() == ActionType.COMPOUND) {
                component
                        .getAction()

.setActions(compoundDao.getActionsOfCompound(component.getAction().getId()
));
            }
        }
        return components;
    }

    @Override
    public TestScenario getDecomposedTestScenarioById(int id) {
        Optional<TestScenario> testScenarioOptional = testScenarioDao.getById(id);
        if (testScenarioOptional.isPresent()) {
            TestScenario testScenario = testScenarioOptional.get();
            testScenario.setActions(getDecomposedComponents(id));
            return testScenario;
        }
        return new TestScenario();
    }

/**
     * Повертає ідентифікатор проекту для тест сценарію.
     *
     * @param testScenarioId ідентифікатор тест сценарію.
     */
    @Override
```

```
    public Integer getProjectIdOfTestScenario(Integer testScenarioId) {
        return this.testScenarioDao.getProjectIdOfTestScenario(testScenarioId);
    }

    private List<TestScenarioComponent> getDecomposedComponents(int id) {
        List<TestScenarioComponent> components = new ArrayList<>();
        int order_num = 1;
        for (TestScenarioComponent component : testScenarioDao.getComponents(id)) {
            if (component.getAction().getType() == ActionType.COMPOUND) {
                for (ActionOfCompound action : compoundDao.getActionsOfCompound(component.getAction().getId())) {
                    components.add(new TestScenarioComponent(action.getAction(), order_num++));
                }
            } else {
                component.setOrderNum(order_num++);
                components.add(component);
            }
        }
        return components;
    }
}

@Repository
public class TestScenarioDaoImpl implements TestScenarioDao {

    private final JdbcTemplate jdbcTemplate;
    private final QueryService queryService;

    public TestScenarioDaoImpl(JdbcTemplate jdbcTemplate, QueryService queryService) {
        this.jdbcTemplate = jdbcTemplate;
        this.queryService = queryService;
    }

/**
     * Створення тест сценарію.
     *
     * @param testScenario об’єкт, який повинен бути створений.
          */
    @Override
    public Integer create(TestScenario testScenario) {
        String sql = queryService.getQuery("testScenario.create");
        return jdbcTemplate.queryForObject(sql, new Object[]{
```

```
                testScenario.getName(),
                testScenario.getUser().getUserId(),
                testScenario.getProject().getProjectId(),
                testScenario.getDescription()},
            (rs, rowNum) -> rs.getInt("test_scenario_id")
        );
    }

/**
     * Додавання наборів дій до тестового сценарію.
     *
     * @param action_compound_id ідентифікатор дії в наборі дій.
         * @param ts_id ідентифікатор тест кейсу.
     * @param order_num порядок дії в наборі дій .

         */
    @Override
    public void addActionOrCompound(int action_compound_id, int ts_id, int
order_num) {
        String sql = queryService.getQuery("testScenario.addActionOrCompound");
        jdbcTemplate.update(sql,
                action_compound_id,
                ts_id,
                order_num
        );
    }

/**
     * Додавання дій до тест сценарію.
     *
     * @param action_compound_id ідентифікатор дії в наборі дій.
         * @param ts_id ідентифікатор тест кейсу.

         */
    @Override
    public void addManyActionOrCompound(int[] action_compound_id, int ts_id) {
        String                                    sql                                    =
queryService.getQuery("testScenario.addManyActionOrCompound");
        jdbcTemplate.update(sql,
                action_compound_id,
                ts_id,
                IntStream.range(1, action_compound_id.length + 1).toArray());
    }

        /**
```

```
    * Повертає всі тестові сценарії
    *
    * @param page номер сторінки.
    * @param size розмір сторінки.
    * @param filterBy параметр, що дозволяє фільтрувати значення.
    * @param orderBy визначає порядок сортування елементів.
    * @param order визначає порядок елементів.
    * @param projectId ID ідентифікатор проекту, в якому використовується тестовий сценарій.
    */
   @Override
   public List<TestScenarioDto> getAllByPage(int page, int size, String filterBy, String filter, String orderBy, String order) {
      String query = queryService.getQuery("testScenario.getAllByPage");
      query = String.format(query, filterBy, orderBy, order);
      return jdbcTemplate.query(query,
            new Object[]{"%" + filter + "%", size, size, page},
            new TestScenarioDtoRowMapper()
      );
   }

/**
    * Редагування тестового сценарію в проекті.
    *
    * @param testScenario тест сценарій, який буде змінений.
       */
   @Override
   public void edit(TestScenario testScenario) {
      String sql = queryService.getQuery("testScenario.edit");
      jdbcTemplate.update(sql,
            testScenario.getName(),
            testScenario.getDescription(),
            testScenario.isActive(),
            testScenario.getTestScenarioId()
      );
   }

/**
    * Отримання розміру вихідного результату.
    *
    * @param filterBy параметр, за яким буде відфільтровано тестовий сценарій.
       * @param filter поле, яке буде відфільтроване.
       */
   @Override
   public Integer getSizeOfResultSet(String filterBy, String filter) {
```

```
        String sql = queryService.getQuery("testScenario.getSizeOfResultSet");
        sql = String.format(sql, filterBy);
        return jdbcTemplate.queryForObject(sql,
                new Object[]{"%" + filter + "%"},
                (rs, rowNum) -> rs.getInt("count"));
    }

        /**
     * Повертає усі тестові сценарії.
     *
     * @param page номер сторінки.
     * @param size розмір сторінки.
     * @param filterBy параметр, що дозволяє фільтрувти значення.
     * @param orderBy визначає порядок сортування елементів.
     * @param order визначає порядок елементів.
     * @param projectId ID ідентифікатор проекту, в якому використовується тестовий сценарій.
     */

    @Override
    public List<TestScenarioDto> getAllByPageAndProject(int page, int size, String filterBy, String filter, String orderBy, String order, int projectId) {
        String query = queryService.getQuery("testScenario.getAllByPageAndProject");
        query = String.format(query, filterBy, orderBy, order);
        List<TestScenarioDto> testScenarioList = jdbcTemplate.query(query,
                new Object[]{"%" + filter + "%", projectId, size, size, page},
                new TestScenarioDtoRowMapper()
        );

        return testScenarioList;
    }

/**
     * Повертає розмір даних проекту.
     *
     * @param filterBy параметр, за яким буде відфільтровано тестовий сценарій.
         * @param filter поле, за яким фільрується тестовий сценарій
     * @param projectId ідентифікатор проекту

         */
    @Override
    public Integer getSizeOfProjectResultSet(String filterBy, String filter, int projectId) {
        String sql =
```

```
queryService.getQuery("testScenario.getSizeOfProjectResultSet");
        sql = String.format(sql, filterBy);
        return jdbcTemplate.queryForObject(sql,
                new Object[]{"%" + filter + "%", projectId},
                (rs, rowNum) -> rs.getInt("count"));
    }

/**
     * Перевірка для тест кейсу .
     *
     * @param testScenarioId ідентифікатор тестового сценарію.
          */
    @Override
    public Boolean checkForTestCaseOnIt(int testScenarioId) {
        String sql = queryService.getQuery("testScenario.checkForTestCaseOnIt");
        return jdbcTemplate.queryForObject(sql,
                new Object[]{testScenarioId},
                (rs, rowNum) -> rs.getBoolean("case"));
    }

/**
     * Видаляє набори дій.
     *
     * @param testScenarioId ідентифікатор тестового сценарію.
          */
    @Override
    public void dropActionOrCompound(int testScenarioId) {
        String sql = queryService.getQuery("testScenario.dropActionOrCompound");
        jdbcTemplate.update(sql, testScenarioId);
    }

/**
     * Видаляє тестовий сценарій.
     *
     * @param testScenarioId ідентифікатор тестового сценарію.
          */
    @Override
    public void delete(int testScenarioId) {
        String sql = queryService.getQuery("testScenario.deleteById");
        jdbcTemplate.update(sql, testScenarioId);
    }

/**
     * Змінює статус тестового сценарію на неактивний.
     *
```

```
     * @param testScenarioId ідентифікатор тестового сценарію.
         */
    @Override
    public void makeUnactivated(int testScenarioId) {
        String sql = queryService.getQuery("testScenario.makeUnactivated");
        jdbcTemplate.update(sql, testScenarioId);
    }

/**
     * Повертає тестовий сценарій за ідентифікатором.
     *
     * @param id ідентифікатор тестовоо сценарію.
         */
    @Override
    public Optional<TestScenario> getById(int id) {
        String sql = queryService.getQuery("testScenario.findById");
        TestScenario ts = jdbcTemplate.queryForObject(sql,
                new Object[]{id},
                (rs, rowNum) -> new TestScenario(
                        id,
                        rs.getString("name"),
                        rs.getString("description"),
                        new User(rs.getInt("user_id"),
                                rs.getString("user_email"),
                                rs.getString("user_name"),
                                rs.getString("user_surname")),
                        new Project(rs.getInt("project_id"),
                                rs.getString("project_name"),
                                rs.getString("project_link"),
                                rs.getTimestamp("project_date")),
                        rs.getBoolean("activated")
                )
        );
        return Optional.ofNullable(ts);
    }

/**
     * Повертає всі компоненти тестового сценарію за ідентифікатором.
     *
     * @param id ідентифікатор тестового сценарію.
         */
    @Override
    public List<TestScenarioComponent> getComponents(int id) {
        String sql = queryService.getQuery("testScenario.getComponents");
        return jdbcTemplate.query(sql,
```

```
            new Object[]{id},
            (resultSet, i) -> new TestScenarioComponent(
                    resultSet.getInt("order_num"),
                    resultSet.getInt("action_id"),
                    resultSet.getString("name"),
                    resultSet.getString("description"),
                    ActionType.valueOf(resultSet.getString("type")),
                    resultSet.getString("key"),
                    resultSet.getInt("parameter_key_id")
            )
        );
    }

/**
     * Повертає ідентифікатор проекту за тест сценарієм.
     *
     * @param testScenarioId ідентифікатор тестового сценарію.
          */
    @Override
    public Integer getProjectIdOfTestScenario(Integer testScenarioId) {
        String sql =
queryService.getQuery("testScenario.getProjectIdOfTestScenario");
        return jdbcTemplate.queryForObject(sql, new Object[]{testScenarioId},
Integer.class);
}
}

@Data
@AllArgsConstructor
@NoArgsConstructor
public class TestScenarioDto {
    private int testCaseId;
    private int testScenarioId;
    private String name;
    private String description;
    private User user;
    private Integer projectId;
    private List<ActionInstDto> actions;

    public TestScenarioDto(int testScenarioId, String name, String description) {
        this.testScenarioId = testScenarioId;
        this.name = name;
        this.description = description;
    }
```

```
    public TestScenarioDto(int testScenarioId, String name, String description, User
user) {
        this.testScenarioId = testScenarioId;
        this.name = name;
        this.description = description;
        this.user = user;
    }

    public TestScenarioDto(int testScenarioId, String name, String description, User
user, List<ActionInstDto> actions) {
        this.testScenarioId = testScenarioId;
        this.name = name;
        this.description = description;
        this.user = user;
        this.actions = actions;
    }
```