КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

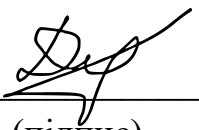
на тему:

СИСТЕМА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ І КОНТРОЛЮ ДОСТУПУ ДО ВЕБ-САЙТІВ

Виконав студент 4-го курсу
Богрєєв Євгеній Вікторович

(підпис)

Науковий керівник:
доцент кафедри математичної інформатики, кандидат фіз.-мат. наук
Деревянченко Олександр Валерійович



(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент 

(підпис)

Київ-2022

РЕФЕРАТ

Звіт до дипломної роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (12 найменувань), 28 рисунків і чотирьох таблиць.

В роботі розроблено автоматизовану програму для тестування веб-сайтів, при цьому досліджено предметну область, застосовано, систематизовано та закріплено знання з технологій програмування (Spring MVC, Spring Boot, PostgreSQL, Heroku).

Об’єкт дослідження – тестова система. Система дозволяє автоматизувати тестування. Інженерам не доведеться запускати кожен сценарій вручну. Система допомагає уникати регресії аналізувати проблемні етапи.

Метою дипломної роботи є розробка системи для автоматизованого тестування веб-сайтів. Методи розроблення: розробка програмного продукту на основі еволюційної моделі, спостереження порівняння, аналіз та синтез. Інструменти розроблення: інтегроване середовище розробки IntelliJ IDEA, RDDMS PostgreSQL, JDBC Template, Spring Framework, мова програмування Java 8.

Результати роботи: виконано загальний огляд систем, призначених для автоматизації тестування, проаналізовано переваги та недоліки використаних технологій, розроблено програмний продукт «Система для автоматизованого тестування веб-сайтів».

ЗМІСТ

ВСТУП

У наш час інформаційні технології відіграють величезну роль у житті людини. Сфера IT на ринку фахівців розвивається дуже швидко. Люди вивчають і використовують системи зберігання даних, знаходять і надсилають інформацію, цікавляться програмним забезпеченням, апаратним забезпеченням і додатками.

Звичайно, є потреба в тестуванні всього програмного забезпечення, оскільки воно визначає якість програмного забезпечення після його розробки програмістом. Цей процес передбачає оцінку інформації, пов’язаної з продуктом. Цей процес тепер є невід’ємною частиною компаній, які будуть ефективнішими у своїй повсякденній діяльності, якщо впровадять процедури тестування програмного забезпечення.

Актуальність цього проекту полягає в тому, що коли ми розробляємо програмне забезпечення, ми всі робимо помилки, деякі з цих помилок незначні, але деякі з них дорогі або навіть небезпечні. Тестування програмного забезпечення необхідне, тому що нам потрібно перевірити все, що ми робимо, тому що завжди може бути щось не так.

Завданням цього проекту є розробка системи, а після її вдосконалення використання спеціальних тестових структур та об’єктів. Все це зроблено для того, щоб автоматизація перевірених проектів для інженерів була ще швидшою та зручнішою. Для успішного написання розробленого дипломного проекту необхідно дотримуватися наступних пунктів:

• Ознайомтеся та детально вивчіть наукову літературу

• Детально сплануйте алгоритми тестування цієї роботи

• Розробити структурований проект для тестування різних типів веб-ресурсів користувачів.

• Розробити програму проекту зі зручним інтерфейсом.

Цей проект містить чотири розділи, кожен з яких описує конкретну область дипломної роботи.

Перший розділ описує основні структури та алгоритми тестування програмного забезпечення та захисту веб-сервісів від різних атак.

Другий розділ містить опис сучасних технологій, які будуть використані при розробці цього проекту. А також передбачає вибір мов програмування та алгоритмів, на яких буде написана робота.

Третій розділ описує реалізацію програми та інтерфейси для зручності використання.

У четвертому розділі показано демонстрацію цього проекту в дії. Передбачається надання результатів у вигляді картинок, а також пояснення програми.

В кінці кожного розділу наводяться висновки, в кінці дипломного проекту висновок до всієї роботи, а також список використаної літератури, який був необхідний для виконання даної роботи.

РОЗДІЛ 1

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Тестування. Загальні відомості

Тестування сайту — це загальний термін для процесів, які тестують сайти та програмне забезпечення для виявлення впливу на широкий спектр небажаних пристроїв. Він включає тестування функціональності, придатності, доступності та продуктивності. Веб-тестування покращує якість ваших сайтів і програм, відкриваючи великі можливості та переваги для вашого бізнесу.

Тестування програмного забезпечення – це метод перевірки того, що фактичний програмний продукт відповідає очікуваним вимогам, і підтвердження того, що програмний продукт не має дефектів. Він включає програмне забезпечення та компоненти системи, що використовують ручні або автоматизовані інструменти для виконання певних властивостей, що цікавлять. Метою тестування програмного забезпечення є виявлення помилок, прогалин або відсутніх вимог на відміну від фактичних вимог.

відповідає стандарту ANSI / IE 1059, тестування розробки програмного забезпечення — це процес оцінки програмного продукту, щоб визначити, чи відповідає поточний програмний продукт необхідним умовам. Процес тестування включає оцінку характеристик програмного продукту з точки зору вимог щодо будь-яких недоліків, помилок чи помилок, безпеки, надійності та продуктивності.

Тестування програмного забезпечення важливо, якщо в програмному забезпеченні є помилки або помилки, які можна виявити на ранній стадії та виправити до того, як програмне забезпечення буде відправлено. Правильний перевірений програмний продукт забезпечує надійність,

безпеку та високу продуктивність, що за своєю функцією дозволяє заощадити час, налаштувати та задовольнити клієнтів.

Тестування важливе, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними. Помилки в можливостях програмного забезпечення можуть призвести до фінансових втрат і людських жертв, і історія рясніє такими прикладами.

1.2. Види тестування

На сьогоднішній день існує багато різних типів програмного забезпечення, програм і веб-сайтів для тестування. Дуже важливо вибрати якісний інструмент або планувати, щоб заощадити час і гроші.

1.2.1. Функціональне тестування

Як функціональне, так і нефункціональне тестування засноване на концепції тестування чорного ящика. Цей тест не є кодом для вашої програми. Процес функціонального тестування полягає в тестуванні кожної функції вашої системи.

Функціональне тестування для кожної функції забезпечує відповідні системи введення. Тест починається з правильного результату або результату. Порівняння результатів є кроком пошуку для розуміння та можливих помилок тестування.

Функціональне тестування гарантує, що ваш сайт або веб-сервер керуються правильними умовами. Він перевіряє помилки та інші проблеми, щоб забезпечити максимальну продуктивність.

Типові перевірки:

• Форми – чи працюють? І чи працюють вони на настільних і мобільних додатках?

• Поточні URL-адреси – чи працює переспрямування сторінок?

Функціональне веб-тестування відрізняється від інших форм тестування в цілому. Він зосереджений на тестуванні ключових аспектів

програми. Дуже важливо перед початком процесу мати список вимог до програмного забезпечення. Цей лист містить тестові сценарії, кейси та інші елементи стратегічного плану.

Потрібно ефективно розуміти всі вхідні дані, тому що це допомагає правильно розпочати процес. Використання правильного процесу допоможе вам уникнути небажаних помилок під час тестування програм.

Функціональне веб-тестування є одним із найпоширеніших процесів з кількох причин. Сьогодні на ринку доступно багато різноманітних програмних продуктів. Якщо ви знайдете правильний процес, ви досягнете кращих результатів і продуктивності.

Процес тестування включає інтерфейс користувача, безпеку баз даних, програм тощо. Функціональне тестування зазвичай використовується для тестування клієнтського або веб-серверного програмного забезпечення. Ще однією величезною перевагою функціонального тестування є зручність. Цей процес дозволяє користувачам працювати як вручну, так і автоматично. Легкий вибір керування допомагає людям заощадити час.

В даний час більшість тестувальників віддають перевагу автоматизованому тестуванню. Оскільки функціональне тестування підтримує автоматизовані стратегії, очевидно, що люди знаходять переваги та розуміють, що ними легше користуватися.

1.2.2. Локалізаційне тестування

Локалізаційне тестування забезпечує доступність сайту для широкої аудиторії користувачів. Він охоплює лінгвістичні, технічні та візуальні тести для вирішення проблем перекладу та діалекту. Деякі тестування також пропонують зміни для покращення оптимізації локалізації.

Типові перевірки:

- Переклад - якщо потрібно ввести спеціальний діалект або спеціальні символи.

- Різні формати адрес для різних місць
- Продуктивність – чи працюватиме ваш сайт у різних місцях?

1.2.3. Тестування доступності

Трошки менше 21% дорослих потребують у тій чи іншій формі для розуміння, розуміння, навігації та взаємодії з Інтернетом.
Тестування доступності гарантує, що ваш веб-сайт буде доступним для всіх користувачів. Чи порушують користувачі, чи мають погане інтернет-з'єднання.

1.2.4. Тестування і контролю доступу і безпеки

Тестування безпеки гарантує, що ваш веб-сайт не піддається вразливості щодо загроз і ризиків, які можуть завдати великої шкоди вашому бізнесу.

1.2.5. Автоматизоване тестування

Автоматичне тестування базується на попередньо підготовлених тестах, які запускаються автоматично, а потім порівнюють фактичний результат з очікуваним. У багатьох випадках термін «тестування» вводить в оману, оскільки автоматизовані тести насправді просто перевіряють, чи ваш веб-сайт працює належним чином за певними сценаріями.

Навпаки, ручне тестування виконується людиною, яка сидить перед комп'ютером і ретельно виконує кроки тестування. Програмне забезпечення для автоматизації тестування також може вводити дані тестування в тестову систему, порівнювати очікувані та фактичні результати та створювати детальні звіти про випробування. Автоматизація тестування вимагає значних вкладень грошей і ресурсів.

Для послідовних циклів розробки знадобиться кілька запусків одного і того ж набору тестів. Використовуючи інструмент автоматизації тестування, ви можете записати цей набір тестів і відтворити його, якщо необхідно. Після автоматизації набору тестів втручання людини не

потрібно. Метою автоматизації є скорочення кількості тестових випадків, які необхідно запускати вручну, замість того, щоб повністю виключати ручне тестування.

Автоматизація тестування — найкращий спосіб підвищити ефективність, охоплення тестуванням і швидкість тестування програмного забезпечення. Автоматичне тестування програмного забезпечення важливо з таких причин:

- Ручне тестування всіх поточних амбасадорів, всіх течій, всіх негативних сценаріїв вимагає часу і грошей.
- Вручну перевірити багатомовні сайти важко
- Автоматизація тестування не вимагає втручання людини. Ви можете запустити автоматичне тестування без нагляду (за ніч)
- Автоматизація тестування збільшує швидкість тестування
- Автоматизація допомагає збільшити тестування охоплення
- Ручне тестування може бути виснажливим і, отже, схильним до помилок.

1.3. Основні положення ручного тестування

Мануальне тестування — це тип тестування програмного забезпечення, при якому тестувальник виконує тести вручну без використання будь-яких автоматизованих інструментів. Метою ручного тестування є виявлення помилок, проблем і недоліків у програмному забезпеченні. Ручне тестування програмного забезпечення є найбільш примітивним методом з усіх видів тестування, який допомагає виявити критичні помилки в програмному забезпеченні.

Кожна нова програма повинна бути перевірена вручну, перш ніж її можна буде автоматизувати. Ручне тестування програмного забезпечення вимагає більше зусиль, але необхідне для автоматизованого тестування. Концепції ручного тестування не вимагають знання будь-яких інструментів

тестування. Щоб ручне тестування було успішним, тестер повинен спочатку зрозуміти вимоги, а це означає, як має працювати програмне забезпечення. Документи, які містять всю необхідну інформацію про тестований додаток, називаються запитами або історіями користувачів, якщо вони написані в такому форматі.

Вони допомагають особам, які тестують, зрозуміти призначення програмного забезпечення, усі тестові частини, що потрібно зробити тестеру та що класифікується як несправність. Знати цю інформацію перед підготовкою до тестування дуже корисно, оскільки, як і в будь-якому тестуванні програмного забезпечення, головна мета — максимально наблизити програмне забезпечення до відсутності помилок. Після того, як вимоги вивчені та зрозумілі, слід написати приклади тестів. Тестові приклади служать посібниками для осіб, які тестують, які містять кроки та інструкції для тестування різних функцій і сценаріїв у програмному додатку. Написання докладних тестових прикладів дуже важливо, оскільки вони допомагають вам плавно виконувати тести та забезпечують якомога ширше охоплення тестами. Тестові набори також повинні містити достатню кількість деталей, щоб тести можна було повторити за потреби. Це дозволяє потенційним тестувальникам запускати або перезапускати будь-які тести, не задаючи додаткових запитань.

Після написання тестових випадків і підготовки тестового середовища настав час почати тестування. В кінці кожного тесту має бути позначено як пройдено, не пройдено або пропущено. Виконуючи тестування вручну, запишіть, що саме стало причиною невдачі тесту, оскільки корисно мати доступ до цих показників для майбутнього планування.

Крім тестування, людина, яка тестує, також несе відповідальність за запис інформації про будь-які виявлені помилки чи упущення. Зберігання інформації про помилку пізніше принесе користь команді розробників.

Готові якісні попередньо написані звіти про помилки допоможуть вам і вашій команді та заощадять ваш час, якщо вам потрібно відповісти на будь-які запитання про помилки.

Модульне тестування — це тестування окремого програмного компонента або модуля, яке називається модульним тестуванням. Зазвичай цим займаються розробники, а не експерти з контролю якості, оскільки це вимагає детального знання внутрішнього дизайну та програмного коду.

Інтеграційне тестування — це тестування підсистеми, що складається з двох або більше інтегруючих компонентів. Це буде зроблено на основі того, як окремі компоненти тестуються та як вони працюють належним чином. Виконується для пошуку недоліків інтерфейсу та взаємодії інтегрованих компонентів.

Тестування системи означає тестування системи в цілому. Усі розроблені компоненти проходять модульне тестування, а потім інтегруються в додаток. Після цього ми ретельно тестуємо всю систему, щоб переконатися, що програма відповідає всім стандартам якості.

Прийнятне тестування користувача – це тип тестування, яке проводить клієнт для сертифікації системи відповідно до вимог за попередньою домовленістю. Тестування проводиться на заключній фазі тестування перед передачею програмного забезпечення на ринок або виробниче середовище. Замовник проводить цей тип тестування в спеціальному середовищі (подібному до виробничого середовища) і підтверджує, чи відповідає система технічним вимогам.

Тестування чорного ящика - цей метод тестування проводиться без знання внутрішніх кодів і структури програми. Тестування проводиться з точки зору клієнта, і особа, яка тестує, знає лише вхідні дані та очікувані результати програми. Тестер не знає, як програмне забезпечення обробляє запити і як видаються початкові результати.

1.3.1. Плюси ручного тестування

- Ручне тестування програми дозволяє виявити більшість проблем, у тому числі проблеми з компоновкою програми.
- Тестер може легко отримати доступ до візуальних компонентів, таких як текст, макет, інші компоненти, а також можуть бути виявлені проблеми з користувачем та інтерфейсом користувача. Вартість робіт, як правило, низька, оскільки ми не використовуємо високоякісні інструменти чи навички.
- Він добре підходить, якщо ми вносимо незаплановані зміни в програму, оскільки вона адаптується. Люди можуть спостерігати, судити та забезпечувати інтуїцію у випадку ручних тестів, і це корисно, коли мова йде про зручність користувача або великий досвід користувача.

1.3.2. Мінуси ручного тестування

Ручне тестування досить складне, тому воно, безсумнівно, має свої недоліки:

- Мануальне тестування часо затратне.
- Використання ручного тесту, щоб знайти різницю в розмірі та комбінації кольорів об'єктів GUI, нелегко.
- Навантажувальне тестування та тестування продуктивності є непрактичними при ручних тестах.
- При великій кількості тестів виконання тестів вручну займає багато часу.
- Випадки регресійного тестування, виконаного ручними тестами, забирають багато часу.

1.3.3. Порівняння ручного і автоматичного тестування

Таблиця 1.1 Порівняння тестування (початок)

Параметр порівняння	**Ручне тестування**	**Автоматизоване тестування**
Виконання	Особи, які тестують, вручну виконують тестові кейси.	Використовуються інструменти для планування і виконання тестових випадків.
Час і вартість	Мануальний тест займає багато часу і вимагає великих витрат.	Економія часу
Тип додатка	Теоретично можливо вручну протестувати будь-який додаток.	Автоматичне тестування вигідно тільки для стабільних систем.
Процес	Повторюємий процес і і доволі нудний	Через те, що перевірка автоматизована, особа, яка тестує, пропускає нудну частину.
Надійність і точність	Надійність досить низка через людський фактор	Висока надійність

Таблиця 1.1 Порівняння тестування (закінчення)

Параметр порівняння	**Ручне тестування**	**Автоматизоване тестування**
Користувальницький інтерфейс	Більш зручний і гарантує поліпшення якості обслуговування клієнтів	Чи не гарантує зручність використання або хороше обслуговування клієнтів.

1.4. Захист та веб-сайтів від атак

Якщо сайт зламано та заблоковано, ви втратите до 98% інформації про трафік. Відсутність безпеки для вашого сайту може бути так само поганою, як і незахищеність. Втрата даних клієнтів може призвести до судових позовів, значних штрафів і поганої репутації.

У 2019 році налічувалося понад 1,94 мільярда онлайн-сайтів. Власники та адміністратори часто думають, що їхній сайт ніколи не буде атакований, оскільки він дуже маленький і виконує багато роботи.

Хакери проникають на дуже великі веб-сайти, щоб викрасти важливу інформацію. Для особистих цілей їм просто потрібні невеликі сайти.

Може бути кілька причин, чому ваш сайт не працює

- Робота користувачів сайту
- Крадіжка інформації, що зберігається на сервері
- Зловживання ресурсами сервера
- Хуліганство

Автоматизовані атаки часто використовують горезвісні вразливості для взаємодії з багатьма веб-сайтами, а іноді й без відома власників. Автоматичні атаки засновані на здібностях. Навпаки, автоматизовані атаки

є більш актуальними, ніж деякі спеціально розроблені атаки через їхню простоту та зручність використання.

Майже 60% Інтернету працює на системі керування вмістом. Тепер деяким веб-сайтам стало легко використовувати системи керування вмістом з відкритим кодом (CMS), такі як WordPress, Joomla або Drupal, для швидкого доступу до Інтернету. Хоча ці платформи постійно оновлюють свої компоненти безпеки, використання інших доповнень може призвести до вразливостей на сайтах, які незабаром будуть атаковані..

SQL-Injection - атаки виконуються шляхом вставки шкідливого коду в запит SQL. Зловмиснику довіряють додати спеціально створений запит до бази даних у повідомленні, надісланому веб-сайтом. Успішна атака змінює запит до бази даних, повертаючи інформацію, яку шукав зловмисник, замість інформації, яку очікував сайт. Ін’єкції SQL все ще можуть змінювати або додавати шкідливу інформацію до бази даних.

Багатосторінковий сценарій (CSSS) складається з вбудовування шкідливого клієнтського коду на веб-сайт і використання веб-сайту для власних цілей. Це надає зловмисник під час завантаження сторінки.

DoS / DDoS атаки – ці атаки призначені для зупинки або сповільнення вашого сайту шляхом заповнення мережі або сервера непотрібним трафіком або інформацією. DDoS-атаки — це загрози, на які власники веб-сайтів повинні звернути особливу увагу, оскільки вони важливі. Коли DDoS-атака атакує вразливу кінцеву точку, що вимагає багато ресурсів, навіть невеликого обсягу трафіку достатньо для успіху. безпека веб-сайту.

Безліч веб-сайтів було зламано через застаріле та шкідливе програмне забезпечення. Дуже важливо завантажувати нові оновлення в міру появи нових доповнень або версій CMS. Ці оновлення можуть включати посилені заходи безпеки або виправляти вразливості.

Більшість атак на сайти відбуваються автоматично. Роботи постійно шукають сайти для виявлення вразливостей, щоб найближчим часом

використовувати ці сайти. Негайного оновлення сторінок щомісяця чи щотижня недостатньо, тому що роботи знайдуть уразливі місця до їх усунення. Тому дуже важливо використовувати брандмауер для усунення вразливостей безпеки під час оновлення сайту.

Щоб відновити заражені або зламані веб-сайти, спеціалісти з відновлення повинні увійти на клієнтський сервер зі своєю особистою інформацією адміністратора. Зазвичай користувачі використовують небезпечні або слабкі паролі. Сьогодні існує багато списків зламаних паролів в Інтернеті. Хакери об’єднують їх із багатьма солістами, щоб створити ще ширші списки можливих паролів. Якщо паролі, які ви використовуєте, знаходяться в одному з цих списків, злом вашого сайту – це лише питання часу.

Щоб забезпечити надійну безпеку, потрібно дотримуватися таких правил:

- Не повторюйте свої паролі.
- Кожен використаний пароль повинен бути унікальним. Ваш менеджер паролів може допомогти вам у цьому.
- Створіть довгий пароль. Спробуйте використовувати пароль довше 12 символів. Чим довший пароль, тим важче комп’ютерній програмі його зламати.
- Використовуйте випадковий пароль. Зломщик паролів може вгадати мільйони паролів за хвилину, коли ви знайдете слова в Інтернеті або в словниках.
- Використання правильних слів у паролі не є випадковим. Якщо ви можете легко прочитати свій пароль, це означає, що він недостатньо складний. Навіть якщо ви використовуєте розрив символу (наприклад, змінюєте літеру на 0), цього недостатньо.

- В Інтернеті є програми, такі як LastPass або KeePass, які зберігають усі паролі в зашифрованому форматі та можуть бути згенеровані випадковим чином одним натисканням кнопки.

Хакерів іноді цікавлять не веб-сайти, а користувачі. Запис IP-адрес та загальної історії активності буде корисним пізніше при криміналістичному аналізі. Значне збільшення кількості зареєстрованих користувачів може свідчити про збій у процесі реєстрації та дозволити спамерам завантажити сторінку фейковим вмістом.

Надання привілеїв певним ролям визначає, що вони можуть, а що не можуть робити. В ідеальній системі роль заважає будь-кому намагатися вжити заходів, які виходять за межі наміченого.

Ретельно визначені ролі користувачів і правила доступу обмежують кількість можливих помилок. Це також знижує ймовірність компрометації облікового запису та може забезпечити захист від шахрайства, спричиненого шахрайськими користувачами.

Сертифікати SSL використовуються для шифрування даних, коли вони передаються між хостом (веб-сервером або брандмауером) і клієнтом (браузером). Це допомагає гарантувати, що інформація надсилається на відповідний сервер і не може бути перехоплена.

Деякі типи сертифікатів SSL, наприклад сертифікат SSL організації або спеціально перевірений SSL, забезпечують додатковий рівень довіри, оскільки відвідувач може бачити інформацію про вашу організацію і знає, що ви є законною особою. Використання одного сертифіката SSL недостатньо для запобігання доступу зловмисників до конфіденційної інформації. Уразливість у веб-додатках може дозволити зловмиснику перехоплювати трафік, спрямовувати відвідувачів на підроблені веб-сайти, відображати неправдиву інформацію, захоплювати веб-сайт у заручники (вимагати) або видаляти всі дані.

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі ми обговорюємо загальну інформацію про тему дисертації, ключові поняття, визначення, а також підходи до роботи з веб-додатками.

Проведено детальний аналіз видів тестування та захисту та їх застосування в багатофункціональній сфері обчислювальної техніки та програмування. Ми представили різні структури та архітектури в цій області і дійшли висновку, що на сьогоднішній день проблема захисту веб-додатків є дуже важливою в області програмування. Система, яка контролює продуктивність будь-якого веб-додатка, повинна бути відповідною.

РОЗДІЛ 2

2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ФУНКЦІОНАЛУ

2.1. Постановка задачі

Проєкт забезпечує автоматизування тестування та перевірку помилок під час обслуговування сайту. Інженерам не потрібно вручну запускати та тестувати сценарії. Система допомагає уникнути регресій і аналізує операції, які відбуваються в тестових сценаріях. Користувачі можуть перевірити, чи їх веб-програми працюють належним чином.

Створіть систему із простим інтерфейсом та функціональними можливостями, щоб було легше тестувати свої сайти. Для цієї системи повинні бути реалізовані такі функції:

- Створіть простий, зручний інтерфейс.
- Досягти легкої сумісності із системою, щоб покращити тестування та автоматизувати перевірку сайту.
- Після завершення тесту надайте користувачеві чіткі результати на екрані системи або електронною поштою.
- Зручне повторне використання тестових випадків і скриптів для швидкого тестування ресурсів користувача.

Метою цієї системи є створення програми, яка демонструє покрокове тестування продуктивності, а також налагодження автоматизованого процесу.

2.1.1. Опис моделі функціональної

Система забезпечує можливість користувачеві спілкуватися з системою, розробляючи інтерфейс користувача на стороні веб-додатка.

Користувач отримує доступ до системи з посиланням на потрібну веб-сторінку і надає всі необхідні ресурси та імена компонентів для успішного завершення процесу. Тож система перевіряє наявність такого

веб-ресурсу і, якщо він є, починає візуально тестувати сайт з цими компонентами. У будь-якому випадку, незалежно від того, вдалий це процес чи невдалий, система інформує користувача та звітує про процес.

2.1.2. Основний функціонал

Система тестування та перевірки сайту повинна автоматизувати наступні процеси:

- Якщо клієнт надає ресурс для тестування на бажаному веб-сайті, система повинна знайти цей ресурс і перевірити, чи ці посилання є в Інтернеті.
- Під час тестування сайту система повинна використовувати всі компоненти, визначені користувачем
- Виконання процесу має базуватися на введенні клієнта
- Записуйте процес тестування, щоб користувач звітував про виконану роботу та результати, не забудьте надати всю необхідну інформацію в листі

Система повинна мати такі характеристики:

- Робота з користувачами, тобто розпізнавання користувачів у системі, збереження вхідних параметрів для подальшого тестування веб-ресурсів
- Створюйте різні сценарії та тестові випадки на основі вхідних ресурсів, які можуть початися з різних наборів даних у майбутньому
- Мати механізм роботи з поштою для надсилання результатів користувачеві після процесу тестування
- Намалювати тест роботи в реальному часі, скріншоти, скріншоти.
- У цьому проекті є 3 ключові ролі:
- Адміністратор – особа, відповідальна за додавання, видалення або редагування особистої інформації користувача. У нього є

справжній «Супер користувач», тобто найважливіша роль у цій системі, такий користувач має бути один у системі.

- Інженер – особа, відповідальна за створення, редагування та запис тестових випадків і сценаріїв. Такі права можуть мати декілька користувачів.
- Менеджер - особа, відповідальна за перегляд звітів або результатів після процесу тестування. Такі права можуть мати декілька користувачів.

На ілюстрації «рис.2.1» зображено детальні обов’язки користувачів в системі. На ілюстрації «рис.2.2» зображено взаємодію ролей в системі. Адміністратор є найголовнішою особою в системі.

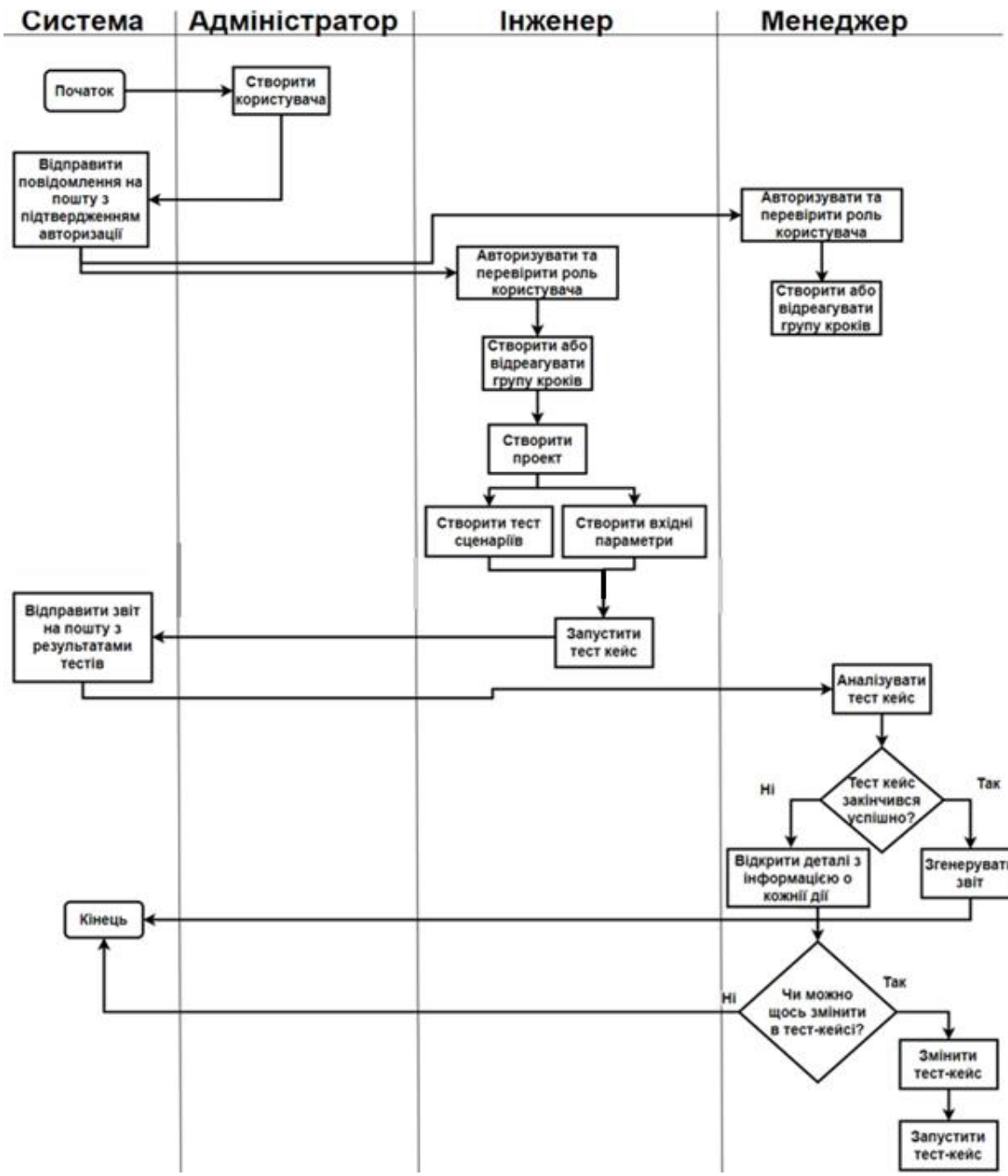


Рис 2.1 Обов’язки користувачів системи

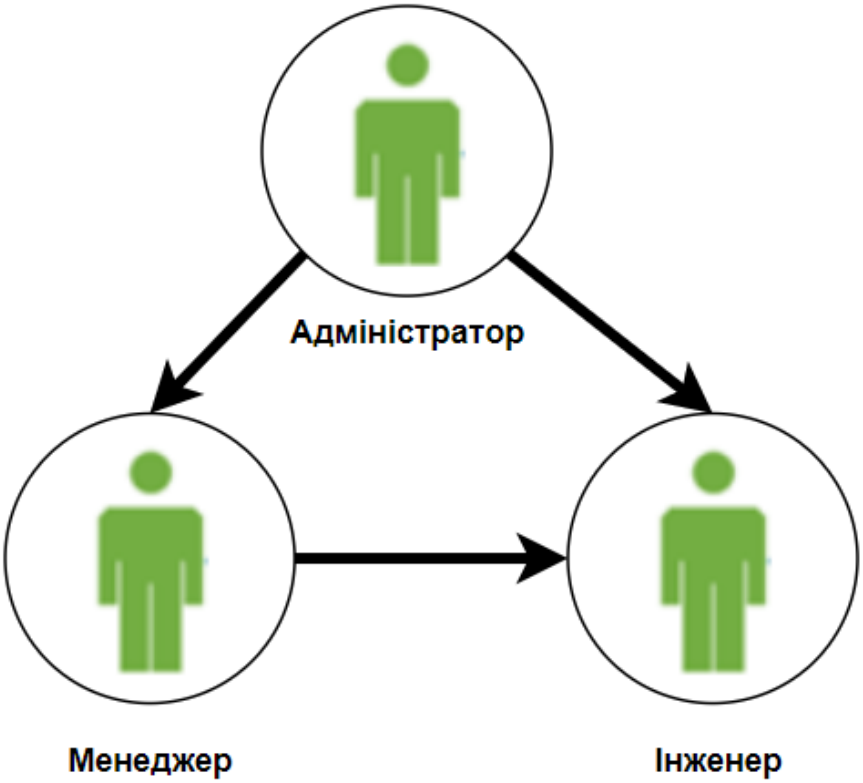


Рис 2.2 Ролі користувачів системи

У цій системі облікові записи користувачів реєструються в системі, а це означає, що адміністратор або інший користувач з певними привілеями може створювати додаткових учасників у системі. Тільки зареєстровані користувачі отримають електронний лист про успішний процес. Після підключення користувач повинен створити власний пароль для цієї системи.

Оскільки система може містити найрізноманітніших користувачів і проектів, у цьому випадку ви зможете швидко шукати для зручності ваших користувачів. Для цього в системі передбачений механізм «Пошук профілю». Адміністратор може шукати та організовувати миттєвий доступ до будь-якої сутності в системі.

Для зручності пошуку та організації користувачів у системі ввімкнено перехресний пошук у всіх особистих профілях у системі. Наприклад, система може дозволити менеджеру бачити всіх інженерів та їхні профілі, а отже, і всіх інших користувачів.

Для облікових записів користувачів доступні такі функції:

- Переглянути особистий профіль. Усі користувачі можуть бачити свою особисту інформацію.
- Переглянути профіль користувача. Адміністратор може здійснювати пошук за ім’ям, прізвищем, адресою електронної пошти, роллю.
- Редагувати профіль користувача. Усі користувачі системи можуть редагувати свій профіль, тобто змінювати свої персональні дані, паролі та дані для входу. У цій системі тільки адміністратор має право змінювати профіль будь-якого користувача.
- Вимкніть профіль користувача. Доступно лише адміністраторам.

Проект – це унікальний перехідний захід, який використовується для досягнення запланованих цілей, які можна визначити з точки зору результатів, результатів або переваг. За загальним правилом, проект можна вважати успішним, якщо він відповідає своїм цілям протягом узгоджених термінів та бюджету відповідно до умов прийнятності. Час, ціна та якість є ключовими елементами будь-якого проекту.

Проект є тимчасовим у тому сенсі, що він має певний початок і кінець у часі і, таким чином, визначений обсяг і ресурси. Унікальність проекту полягає в тому, що це не звичайна операція, а комплекс операцій, спрямованих на досягнення мети.

Управління проектом зосереджується на виробництві кінцевого продукту, що тягне за собою зміни на користь організації, яка ініціює

проект. Це початок, планування та контроль серії завдань, необхідних для доставки кінцевого продукту.

У цій системі проект діє як архів, який містить ряд тестових випадків, об’єднаних для ресурсу, який буде перевірено в майбутньому.

У цій функціональній системі процеси управління проектами можна розділити на п'ять груп:

- Запуск, ініціалізація
- Планування
- Реалізація
- Моніторинг і контроль
- Закінчити

Початок проекту є першим етапом життєвого циклу управління проектом, оскільки він передбачає запуск нового проекту. На етапі запуску вони визначають бізнес-проблему або можливість, визначають рішення, формують проект і призначають проектну групу, яка створить і доставить рішення клієнту. Бізнес-модель створюється для визначення проблеми або можливості та визначення найкращого рішення для реалізації.

Процес проекту починається, коли інженер вказує назву проекту та посилання на веб-сайт, який потрібно протестувати. Тоді система має механізм створення проекту, який переводить створений проект у активний стан, у якому можна продовжувати роботу та створювати різні системні блоки.

Планування — це функція управління, яка передбачає встановлення цілей і визначення напрямку дій, необхідних для досягнення цих цілей. Планування вимагає, щоб керівники були обізнані про умови навколишнього середовища, з якими стикається їх організація, і передбачати майбутні умови. Це також вимагає від лідерів прийняття правильних рішень.

Проектування - це процес, що складається з кількох етапів. Процес починається з огляду середовища, що просто означає, що дизайнери повинні знати про критичні, непередбачені обставини, з якими зіткнеться їхня система.

Система забезпечує планування тестових випадків, сценаріїв і вхідних параметрів, які використовуються в процесі тестування. Інженер може створювати тестові сценарії, які можна використовувати в інших проектах, додатково підкреслюючи гнучкість системи.

Реалізація (або реалізація) проекту — це фаза, на якій реалізується план, розроблений раніше в проекті. Фаза впровадження зазвичай є найдовшою фазою проекту. На етапі реалізації проекту інженерам необхідно провести візуальні тести продуктів або ресурсів.

Етап впровадження має такі характеристики:

Відстеження та перевірка тестів

Управління проблемами

Звіт про випробування та оцінку

Метою етапу моніторингу та контролю проекту є контроль за всіма аспектами реалізації проекту для забезпечення його успіху. На цьому етапі затверджується план проекту та визначаються результати роботи з управління проектом. На цьому етапі реалізується проект, а прогрес відстежується відповідно до плану, встановленого під час планування проекту.

Фаза закриття – це процес завершення всіх заходів проекту. Функція цієї системи – архівувати проект. Як правило, великі дизайнерські фірми не мають передового досвіду видалення об’єктів клієнта, у тому числі проекту. Відповідно, після завершення тестування ресурсів інженер може архівувати проект або створити інший тестовий приклад і скрипт у проекті.

Для проектів доступні такі функції:

- Створити проект.

- Редагувати конфігурацію проекту. Користувач може змінити властивості або інформацію створеного проекту.
- Архівування проекту.

Система використовує функцію архівування замість того, щоб його видалити, тому користувач може закрити проект..

2.1.2.1. Аналіз та процес тестових прикладів

Регуляція розробки тестів – це процес управління діяльністю тестування для забезпечення високої якості та високого рівня тестування веб-додатків. Метод складається з організації, контролю, відстеження та прозорості процесу тестування для забезпечення високої якості веб-сайту. Це гарантує, що процес тестування ресурсів працює правильно.

Процес управління тестуванням надає інструменти для планування, тестування, моніторингу та контролю протягом усього життєвого циклу проекту. Цей курс охоплює ряд заходів, таких як планування, проектування та проведення тестів. Це забезпечує початковий дизайн і дисципліну для процесу тестування програмного забезпечення.

Організація тестування — це процес визначення ролей у тестових програмах. Ви визначаєте, хто за які дії відповідає під час тестування. У цьому ж процесі ми також описуємо функції, інструменти та процедури тестування. Також визначаються навички та знання залучених людей.

стратегія тестування:

Мета тесту

Умови виходу/видалення

Планування ресурсів

Результати тесту

На малюнку «Рис. 2,3" показує всі компоненти системи.

Тестовий приклад — це набір операцій, які виконуються для контролю певної функції або функціональності програмного забезпечення.

Випадок перевірки тесту включає етапи тестування, тестові дані, передумови та пост-вимоги, які використовуються для перевірки вимог для конкретного сценарію тестування. Тестовий приклад містить певні змінні або умови, які дозволяють інженеру з тестування порівнювати очікувані та фактичні результати, щоб визначити, чи відповідає програмне забезпечення вимогам замовника.

Рис 2.3 Компоненти тестувальної системи

Адміністрування тестового пакету:

- Див. тестовий приклад
- Створіть тестовий приклад.
- Редагувати тестовий приклад.
- Зніміть тестовий футляр.

Тестовий сценарій застосовується до всіх функцій, які можна відстежувати. Це загальний тестовий приклад, який може допомогти тестовій групі визначити позитивні та негативні сторони плану. Тестовий сценарій може зрозуміти, наскільки високим має бути рівень тесту.

Сценарій тестування обробляється наступним чином:

- Переглянути тестовий приклад
- Створіть тестовий сценарій.
- Редагувати тестовий сценарій.

- Видалити тестовий сценарій.

Є кілька причин створити тест:

- Тестові випадки допомагають перевірити відповідність застосовним стандартам, політикам та умовам клієнтів, які відповідають вимогам клієнтів.
- Допомагає посилити очікування та вимоги клієнтів
- Покращений контроль потоку, логіка та діапазон
- Моделювання «реальних» сценаріїв кінцевого користувача
- Визначає помилки або пропуски
- Тестові випадки краще організовані та легші, коли тестові випадки необхідні для тестування.

Вам також потрібні причини для створення тестового сценарію:

- Під час написання тестового сценарію головною проблемою є перевірка абсолютної функціональності програмного веб-додатка.
- Це також допомагає забезпечити відповідність бізнес-процесів і процесів функціональним вимогам.
- Сценарії тестування можуть бути схвалені різними зацікавленими сторонами, такими як бізнес-аналітики, розробники та замовники, щоб забезпечити ретельну перевірку тестового центру. Це гарантує, що програмне забезпечення працює в найпоширеніших прикладних ситуаціях.
- Вони служать як швидкий інструмент для визначення обсягу досліджень і надання рекомендацій замовнику або відповідальному відділу кадрів.
- Допомагає ідентифікувати транзакції або програми між ключовими кінцевими точками.

- Після того, як ці сценарії були перевірені та виправлені, тестові приклади можна легко видалити зі сценаріїв тестування.

Тестові дані під час тестування програмного забезпечення – це вхідні дані, надані програмному забезпеченню протягом періоду тестування. Замовник надасть дані, що впливають на продуктивність програмного забезпечення протягом періоду оцінки. Тестові дані також використовуються для позитивних тестів і контролю, а функції надають результати прогнозування для конкретних вхідних даних, а також негативні тести для перевірки здатності програмного забезпечення обробляти незвичайні, надзвичайні або раптові введення.

Набір даних використовується для налаштування полів для використання динамічних значень під час фази стрес-тестування. Зазвичай це ті самі дані, які імітований користувач повинен ввести в поля форми на сторінці.

Тестування – це процес, який виробляє та споживає великі обсяги даних. Тестові дані описують початкові умови тестування та представляють середовище, в якому тестувальник працюватиме з програмним забезпеченням.

Залежно від середовища тестування вам може знадобитися створити тестові дані або принаймні налаштувати відповідні тестові дані для власного тестування.

Дані тесту можуть бути згенеровані:

- Менеджментом.
- Масове копіювання виробничих даних у тестове середовище.
- Пакетна копія тестових даних із застарілих клієнтських систем
- Автоматизовані засоби тестування даних

Загалом, основні дані повинні бути зібрані до початку тесту, інакше керування даними тесту буде складним. Оскільки багато тестових середовищ вимагають кількох попередніх кроків для створення тестових даних або довгострокових конфігурацій тестового середовища. Запис — це запис, необхідний для запуску тестового прикладу. Ви також можете виконувати дії з набором даних:

- Переглянути запис
- Щоб зробити знімок.
- Щоб відредагувати запис
- Видалити набір даних

Ви можете створювати складні кроки дій, щоб організувати менші етапи тестування за допомогою сценаріїв, які потім можна зареєструвати як один завершений тест. Якщо ви хочете об’єднати різні етапи тестування в єдиний робочий процес або наскрізний сценарій, ви можете організувати етапи тестування в складні етапи тестування.

Кожен крок може реалізувати частину сценарію. Типовим прикладом скомпільованого крок є робочий процес онлайн-покупок. Наприклад, ви, можливо, зробили невеликі кроки в деяких частинах своєї онлайн-транзакції. В. Увійдіть, вийдіть, перегляньте, додайте в кошик і надрукуйте. Ви можете об’єднати ці кроки в складний потік кроків тестування. Коли виконується складний крок, кожен із його кроків виконується послідовно..

2.1.2.2. Початкова сторінка системи

Панель інструментів дуже важлива і може бути цікавою для певної ролі користувача. Інформаційна панель надає графічний інтерфейс користувача, який часто надає агреговані KPI, пов’язані з конкретною метою або бізнес-процесом. Наприклад, менеджер повинен побачити графік, що показує частоту тестів і збоїв. Зробіть панель інструментів домашньою сторінкою для всіх користувачів.

Головне вікно містить основні кнопки для певної ролі. Наприклад, відобразити кнопку Показати успішні тести під діаграмою зі значком успіху. Після того, як користувач натисне кнопку, система переспрямує на сторінку з попередньо вибраними фільтрами.

2.2. Огляд альтернативних систем

WebLOAD – це інструмент стрес-тесту з потужними можливостями сценаріїв, які зараз тестують складні сценарії. Машина щедро аналізує вашу веб-розробку та визначає проблеми та можливості, які слід розглянути з точки зору поточних і використаних відповідей. Включає вбудовану інтеграцію для Jenkins, Selenium та багатьох інших континуальних пристроїв DevOps.

Acunetik — це повністю автоматизований сканер безпеки веб-додатків, який виявляє та повідомляє про 4500 уразливостей веб-додатків, включаючи всі типи ін’єкцій SKL та KSSS. Він повністю сумісний з HTML5, JavaScript та односторінковими програмами, які тепер можна оновлювати та перевіряти. Інтегровані функції керування живленням вбудовані безпосередньо в ядро, пріоритетні операції засновані на консолідованій, індивідуальній підтримці та інтегрованому запиті системних даних з іншими платформами.

IO Test – це платформа для масового тестування веб-додатків і веб-сайтів у реальних пошукових системах на реальних пристроях і в реальних умовах.

TestComplete — це комплексний автоматизований механізм тестування взаємовідносин з користувачами, який дозволяє тестувати тести на настільному або мобільному веб-сайті, включаючи програми з користувацькими елементами керування та параметрами. Перевірте зручність використання веб-маяка за допомогою потужного тесту ідентифікації об’єктів. Закінчив тест.

Experitest — це платформа для тестування веб-сайтів і веб-додатків у настільних і мобільних браузерах у хмарі, що забезпечує автоматичні або ручні тестові файли на основі браузера. Система пропонує тести Selena та Appium у різних браузерах, версіях та концепціях обслуговування. У нього є функція, з додатком ми можемо з ним спілкуватися ввечері. Платформа на місці, підтверджена візуальними тестами, досвід користувачів у просторі користувача різноманітний. Після процесу ви отримаєте візуальну інформацію про тест зі скріншотами, відео та даними журналу.

2.3. Порівняння технологій взаємодії з механізмом тестування

Selenium є однією з найпопулярніших концепцій з відкритим кодом для автоматизації тестування. За допомогою Selenium ви можете тестувати веб-додатки або автоматизувати веб-сайти в різних браузерах та операційних системах. Selenium пропонує сумісність з різними мовами програмування, такими як Java, JavaScript, Python, C# тощо, що дозволяє тестувальнику автоматизувати тестування веб-сторінок будь-якою мовою програмування, яка йому підходить.

Selenium працює з дизайном клієнт-сервер. Клієнт-серверний дизайн – це модель архітектури програмного забезпечення, яка складається з двох частин, а саме клієнтської системи та серверної системи, які спілкуються через комп’ютерну мережу або на одному комп’ютері.

Під час написання коду сценарію в середовищі IDE ви повинні спочатку надіслати сценарій до Json Vire за допомогою API.

Json Wire також діє як інтерфейс, щоб клієнти та сервери могли легко розуміти один одного. Провідний протокол Json дозволяє легко передавати дані між клієнтами та серверами через Інтернет. Сервер не розуміє жодної мови програмування, JSON Vire використовує процес серіалізації (перетворення даних об’єкта у формат JSON) та десеріалізації

(перетворення формату JSON в об’єкт). Протокол JSON Wire Protocol має REST API, які працюють через HTTP.

Усі тригери Selenium контролюються відповідним API REST у протоколі JSON Vire. Selenium працює з підтримкою таких команд API, як GET і POST, а сценарії Selenium базуються на отриманих запитах. Потім пропозиції надсилаються на HTTP-сервер драйвера браузера та до браузера через HTTP. Це взаємодіє з програмними компонентами, і відповідь надсилається назад до IDE через той самий канал. Драйвер браузера працює в режимі реального часу з комерційним браузером і надсилає інструкції сценарію автоматизації у відповідний браузер для перетворення.

WebDriver — це інструмент автоматизації браузера, який працює з відкритим вихідним кодом API. Фреймворк працює, одержуючи команди, надсилаючи їх у браузер і працюючи з комерційними програмами. WebDriver є найбільш широко використовуваним механізмом для автоматизації веб-додатків і підтримує всі основні браузери, включаючи Chrome, Firefox, Internet Explorer і Edge.

Функціонально WebDriver є загальнодоступним інтерфейсом, який визначає опорну змінну (драйвер) для типів інтерфейсу. Кожен об’єкт, призначений цим змінним драйвера, має бути екземпляром класу, який реалізує інтерфейс (ChromeDriver або інший клас драйвера браузера).

На картині «Рис. 2.3" визначає мову клієнтського каталогу Selenium. У цьому випадку ChromeDriver є драйвером браузера, а Chrome — браузером. Структура та функції залишаються точно такими ж у всіх інших клієнтських каталогах, драйверах браузера.

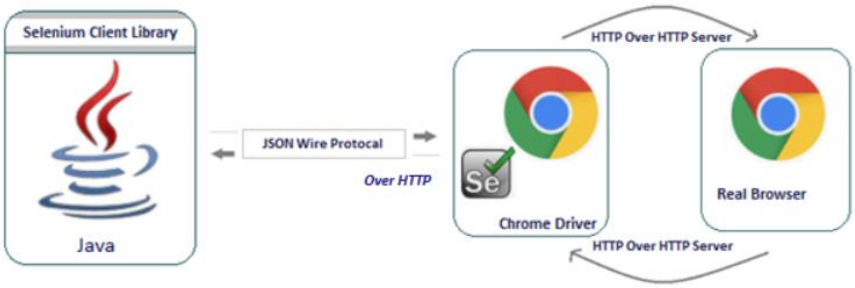


Рис 2.3 Компоненти тестувальної системи

Тестери та розробники вручну зв’язуються з елементами програми, щоб перевірити або завершити транзакції, щоб забезпечити якість програми. Весь процес можна автоматизувати завдяки підтримці архітектури Selenium WebDriver. У цій архітектурі структура драйвера браузера може отримувати команди і HTTP-запити до HTTP-сервера і взаємодіяти з реальними браузерами.

Справжні пошукові системи отримують запити від драйвера свого браузера і працюють з елементами програми відповідно до запиту. В результаті операції браузер відповідає на драйвер браузера. Потім драйвер браузера повертає вихідні дані клієнта через той самий канал.

2.4. Вибір мови програмування

2.4.1. Бек-енд функціональна частина

Функціональна Функціональна частина (бекенд) проекту написана мовою програмування Java з використанням Spring Framework. Для інтерфейсу користувача використовувався JavaScript Angular Framework.

Spring Framework — це фреймворк Java з відкритим кодом для розробки бізнес-додатків. Фреймворк Spring є модульним з різними модулями для різних завдань, а Spring Core є основним модулем, на якому працюють інші системні модулі. Основною відмінністю від Spring Framework є його модульність, що полегшує використання при розробці додатків. Spring дозволяє використовувати одну або кілька рамок на серцевині Spring.

Spring вирішує складні проблеми розробки додатків, надаючи Spring Core, Spring IoC і Spring AOP для інтеграції різних частин бізнес-додатків. Підтримуйте розробку бізнес-додатків через POJO: Spring підтримує розробку бізнес-додатків у відділі POJO, позбавляючи компанії від

необхідності імпортувати важкі контейнери під час розробки. Це значно спрощує тестування програм. Елементарна інтеграція інших фреймворків:

Spring розроблено для роботи з усіма іншими фреймворками Java і може спільно використовувати ORM, Hibernate та інші фреймворки Java. Spring Framework не обмежує фрейми, які можна використовувати разом.

Spring Container можна використовувати для розробки та запуску тестових випадків за межами корпоративного контейнера, що значно спрощує тестування.

Spring Framework є модульним і містить ряд модулів, таких як Spring JDBC, Spring ORM, Spring Transactions, Spring MVC та інші, які мають усі шанси бути модульними для задоволення потреб програми. зустрічає. .Spring Transaction Management - гнучкий і може бути налаштований на використання локальних транзакцій у невеликій програмі, яка адаптується до JTA для глобальних транзакцій.

2.4.2. Фронт-енд частина взаємодії системи та користувача

Для інтерфейсу був обраний фреймворк Angular JS, фреймворк з відкритим кодом, який підтримується Google. Це частина багатої екосистеми JavaScript, яка використовується для створення мобільних і настільних веб-додатків. Однак у новіших версіях Angular розробники не працюють з JavaScript, а пишуть за допомогою HTML і TypeScript, які є надмножинами JavaScript.

Angular використовується в багатьох веб- та мобільних додатках, але особливо ефективний при торгівлі комерційним програмним забезпеченням.

Серед програм, які можна створити за допомогою Angular, є великі бізнес-програми. Angular зазвичай використовується для створення великих систем, які відповідають потребам підприємств або організацій і не потрібні конкретному користувачеві. Такі програми допомагають компаніям організувати свої робочі процеси та керувати певними операціями.

Прикладами таких систем є CRM, ERP-системи, платформи обробки платежів, електронний маркетинг, управління контентом.

Односторінкові програми (SPA) — це динамічні програми або веб-сайти, які взаємодіють з користувачем, щоб замінити поточні веб-сторінки новими даними, а не перезавантажувати нову сторінку із сервера щоразу, коли користувач виконує операцію в програмі. Основна мета таких додатків — забезпечити швидкі переходи та підвищити продуктивність.

Angular PVA Progressive Web Applications — це програма, заснована на веб-технологіях HTML, CSS та JavaScript, а також має функції додатків. PVA працює у браузері, але працює як рідна програма. Прогресивні веб-програми – це корисні шаблони проектування, але не формалізовані стандарти. PVA вважається схожим на AJAX або інші подібні шаблони, що охоплюють атрибути програмного забезпечення, включаючи використання певних веб-технологій і методів. Цей набір документів розповідає вам все, що вам потрібно про нього знати.

2.4.3. Засоби для збереження інформації. База даних

Оскільки цей проект реалізується заощаджуючим способом, система використовує лише надані бази даних для зберігання та обробки даних. База даних PostgreSQL використовується виключно для цього проекту.

PostgreSQL використовує SQL, найпопулярнішу мову програмування для взаємодії з базами даних на сервері. Це стандартна серверна мова, яка використовується для створення та керування реляційними базами даних. SQL інтегрується з низкою систем керування базами даних, такими як SQL Server, Oracle Database і dBase. SQL ілюструє кілька важливих принципів взаємодії з базами даних. Використовуючи розділ сервера SQL, розробники створюють правила для зберігання, отримання та зміни даних сервера.

Особливості PostgreSQL:

- Міжплатформна сумісність з усіма основними мовами та проміжним програмним забезпеченням.

- Підтримка обробки кількох одночасних версій.
- Розширені функції програмування на стороні сервера
- Відповідає стандарту ANSI SQL.
- Повна підтримка архітектури мережі клієнт-сервер.
- Резервний сервер і висока доступність

Підтримка JSON дозволяє підключатися до інших сховищ даних, таких як NoSQL, який діє як центральний центр для багатомовних баз даних.

PostgreSQL підтримує географічні об'єкти, які можна використовувати для служб на основі розташування та географічних інформаційних систем.

Низькі витрати на обслуговування та адміністрування для вбудованого та комерційного використання PostgreSQL.

Вихідний код PostgreSQL є безкоштовним і доступним з відкритим вихідним кодом.

Написання та запис записів PostgreSQL робить базу даних надзвичайно безпомилковою.

PostgreSQL можна розглядати як об’єктно-реляційну базу даних, яка пропонує деякі переваги перед іншими базами даних SQL з відкритим кодом, як-от MiSQL.

Основною особливістю PostgreSQL є підтримка користувацьких об’єктів та їх поведінки, включаючи типи даних, функції, оператори, діапазони та індекси. Це робить PostgreSQL дуже гнучким і надійним. Крім усього іншого, ви можете створювати, зберігати та отримувати складні структури даних.

Великий список типів даних, які підтримує PostgreSQL, див.:

- Числові типи
- Типи дати та часу
- Типи персонажів

- Булеві типи
- Типи мережевих адрес – використовується для хостів IPv4 та IPv6.
- Тип даних MACADDR – використовується для зберігання апаратних адрес для ідентифікації MAC-адрес.

Використовуйте команду CREATE TIPE для створення нових типів даних, таких як складні, перераховані, діапазонні та бази даних.

PostgreSQL все ще дуже популярний сьогодні, оскільки пропонує багато функцій. PostgreSQL може допомогти вам створювати нові програми та керувати даними, незалежно від розміру вашої бази даних.

PostgreSQL працює на таких популярних операційних системах, як Windows, Mac OS X і майже на всіх дистрибутивах Linux і Unique. Природа відкритого вихідного коду полегшує оновлення та розширення. PostgreSQL може визначати власні типи даних, створювати функції та писати код іншою мовою програмування без перекомпіляції бази даних.

Тому PostgreSQL має багато функцій, які використовують об’єктно-реляційні моделі, які підтримують складні структури та широкий спектр вбудованих і визначених користувачем типів даних. Він забезпечує велику кількість даних і є надійним з точки зору цілісності даних.

2.4.4. Огляд веб-хостінгу

Для Щоб розробити проект, вам потрібно завантажити систему на веб-хост, щоб усі користувачі могли користуватися нашою системою в Інтернеті. Серед безкоштовних веб-бібліотек Heroku був обраний сервером, необхідним для реалізації веб-додатків.

Heroku дозволяє розробникам швидко та легко розгортати програму на веб-сервері. Він також пропонує ряд доповнень, які можна інтегрувати в програму.

Heroku — це хмарна платформа, яка дозволяє створювати, транспортувати, контролювати та масштабувати програми. Це розширена

платформа як послуга, яка дозволяє розробникам легко налаштовувати, масштабувати та керувати додатками. Ця платформа пропонує підтримку різноманітних мов програмування, таких як PHP, Java, Ruby, Python, Node.js, Scala тощо. Платформа Heroku і користувацькі програми використовують Amazon Web Services як основну інфраструктуру. Розробники можуть використовувати його для швидкого доступу до розробки додатків, оскільки це дуже зручно.

Програми, запущені на сервері Hero, використовують DNS-сервер для переспрямування на домен своєї програми. Кожен контейнер програми, названий Dinos, розподіляється через "Dino Network" з кількох серверів. Таким чином, всі динозаври надійно ізольовані від усіх.

Dino — це контейнер на платформі Heroku, який використовується для запуску та масштабування програм Heroku. По суті, це віртуальні репозиторії Linux, які використовуються для виконання коду на основі команд користувача. Програми можуть бути оновлені до певної кількості динозаврів за бажанням програміста. Heroku надає можливості керування контейнерами, які дозволяють користувачам легко масштабувати й керувати розміром, типом і кількістю динозаврів відповідно до потреб програми.

У App Store заробітком для авторизованих користувачів керує сервер Heroku Git. Таким чином, ви зможете відразу ввійти в систему, натиснувши Git, і Heroku створить програму за допомогою скриптів.

Щоб використовувати програму на веб-хостингу Herok, потрібно виконати 4 кроки:

1. Створіть проект. Перший крок — створити просту структуру для нашого проекту, використовуючи деякі основні файли.

2. Виберіть систему контролю версій. Наступним кроком є вибір системи контролю версій і розміщення коду в репозиторії на платформі

розробника. Найпопулярнішою системою контролю версій є Git і Github як платформа розробки, тому ми будемо її використовувати.

3. Підключіть контейнер до Heroku. На цьому етапі ми можемо підключити репозиторій Github до нашої програми Heroku.

4. Налаштуйте Heroku, щоб програма працювала належним чином.

ВИСНОВОК ДО РОЗДІЛУ 2

Цей розділ визначає мету проекту, описує функціональність та архітектуру системи, а також описує всі початкові завдання для програми. Ми розробили структуру майбутнього проекту та описали взаємодію клієнта з цією програмою.

Ми розглянули системи, які використовуються для тестування різних веб-додатків, і прийшли до висновку, що існує необхідність розробки іншого типу системи, якої раніше не було, але потреба в цій системі величезна, тому ми вирішили розробити цю програму .

РОЗДІЛ 3

3. РОЗРОБКА ПРОДУКТУ

3.1. Спілкування серверної і клієнтської частин програм

3.1.1. Обмін інформацією між серверами

Як я описав в попередніх розділах, система базується на двох основних структурах:

• Spring – для створення серверної частини веб-програми

• AngularJS - Написання клієнтської сторони веб-додатка

Зовнішня програма зв’язується з внутрішнім сервером за допомогою технології інтерфейсу HttpRequest або HttpClient, що використовується Angular для отримання зовнішніх даних. HttpClient доступний у пакеті Angular/common/http і його потрібно імпортувати для використання служби http.

```
import {HttpClient} from '@angular/common/http';
 constructor(private http: HttpClient) {
 }
 getUserById(id: number): Observable<User> {
   const url = `${this.url}users/${id}`;
   return this.http.get<User>(url);
 }
```

Рис. 3.1. Приклад коду з викристанням HttpClient

На бек-енді будуть використані дві основні анотації щодо надсилання та отримання даних на стороні сервера:

- @CrossOrigin
- @RequestMapping

Реалізований веб-додаток працює на порту 4200, а за допомогою Angular JS ми намагаємося використовувати веб-сервіси RESTful з порту 8080. У цій ситуації зв’язок між серверами стикається з безпекою спільного використання ресурсів між джерелами в їхній мережі. Пошукові системи.

Для вирішення цієї проблеми необхідні 2 умови:

- Веб-служби RESTful необхідні для підтримки спільного використання ресурсів між ресурсами.
- Для доступу до API на порту 8080 потрібна програма веб-служби RESTful.

Використовуючи анотацію @CrossOrigin, ми встановлюємо ресурси для методу керування веб-сервісом RESTful. У цьому посібнику визначено анотований метод або тип, дозволений для запитів із різних джерел. За замовчуванням @CrossOrigin включає всі джерела HTTP, заголовки та методи, перераховані в коментарі @RequestMapping. Примітка. @RequestMapping використовується для зіставлення параметра веб-запиту з параметром методу обробки.. Детальний приклад зображено на «рис. 3.2».

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/users")
@Slf4j
public class UserController {
@RequestMapping(value = "/{id}",method = RequestMethod.GET)
public User getUserById(@PathVariable("id") Integer id){
    return userService.getUserById(id);
}
```

Рис. 3.2. Приклад коду для отримання даних

CrossOrigin * приймає запити з будь-якого джерела і має налаштувати це налаштування для реального середовища. JSON, стандартний текстовий формат для відображення структурованих даних на основі синтаксису JavaScript, використовується для передачі інформації у веб-додатках. У цій системі деякі дані потрібно надіслати від сервера до клієнта, щоб їх можна було відобразити на веб-сторінці або навпаки.

У таблиці 3.1 наведено деякі посилання для обміну інформацією між серверами:

Таблиця 3.1 – Посилання на сервери

Клієнт	**Сервер**
http://localhost:4200/login	http://localhost:8080/login
http://localhost:4200/user/edit/id	http://localhost:8080/user/edit/id
http://localhost:4200/test-case	http://localhost:8080/test-case

3.2. Імплементація алгоритмів автентифікації в системі

Компонент сервера був створений за допомогою Java та Spring, тому ви можете використовувати Spring Security для автентифікації та авторизації користувачів у системі та налаштування всіх функцій зв'язку без стану. Щоб реалізувати Spring Security та всі його компоненти, нам потрібно додати фреймворк Spring Security до нашого проекту, додавши залежність до файлу pom.xml. Приклад залежності наведено на «рис. 3,3».

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Рис. 3.3. Необхідні бібліотеки в pom.xml

Згодом, як я додав ці залежності і спробував отримати доступ до однієї з вищевказаних URL-адрес, система перенаправляє нас на http://localhost:8080/login замість того, щоб побачити результат. Це дії за замовчуванням, оскільки Spring Security вимагає базової аутентифікації для всіх URL-адрес.

Ця програма, написана навесні, з’являється на стороні клієнта і може бути ідентифікована JVT без збереження стану. У цьому випадку JVT повинен зашифрувати інформацію, яку можна безпечно передати як об’єкт JSON.

Надає відкритий POST API для автентифікації та створює JVT після передачі дійсних облікових даних. Якщо користувач спробує отримати доступ до захищеного API, доступ буде надано, лише якщо запит має робочий JVT.

Програма має фільтр, який зареєстровано та перевірено в ланцюжку фільтрів Spring Security. Основна мета цього фільтра — видалити веб-токен із заголовка авторизації. Якщо маркер не порожній, його потрібно перевірити та підготувати до SecurityContext. Приклад детального фільтра зображено на «рис. 3.4».

```
@Override
protected void doFilterInternal(HttpServletRequest httpServletRequest,
                                HttpServletResponse httpServletResponse,
                                FilterChain filterChain)
        throws ServletException, IOException {
    String token = jwtProvider.resolveToken(httpServletRequest);
    if (token != null && jwtProvider.validateJwtToken(token)){
        String email = jwtProvider.getUserNameFromJwtToken(token);
        UserDetails userDetails = userDetailsService.loadUserByUsername(email);
        UsernamePasswordAuthenticationToken auth =
                new UsernamePasswordAuthenticationToken(userDetails, credentials: null,
                        userDetails.getAuthorities());
        auth.setDetails(new WebAuthenticationDetailsSource().buildDetails(httpServletRequest));
        SecurityContextHolder.getContext().setAuthentication(auth);
    }
    filterChain.doFilter(httpServletRequest,httpServletResponse);
}
```

Рис. 3.4. Приклад коду фільтру

Даний Цей алгоритм реалізується наступним чином:

- Користувачі входять у систему, надсилаючи свої особисті облікові дані на сервер.
- Сервер перевіряє облікові дані, отримує дані користувача та створює JVT

- Сервер підписує та шифрує JVT і надсилає його користувачеві у відповідь на початковий запит на акредитив.
- Користувач має JVT протягом певного періоду часу залежно від часу, встановленого сервером.
- Користувач надсилає JVT, збережений у заголовку авторизації, на сервер із кожним запитом.
- Для кожного запиту сервер витягує JVT із заголовка авторизації, розшифровує його, перевіряє підпис і витягує дані користувача та дозволи..

Детальний алгоритм зображено на «рис. 3.5».

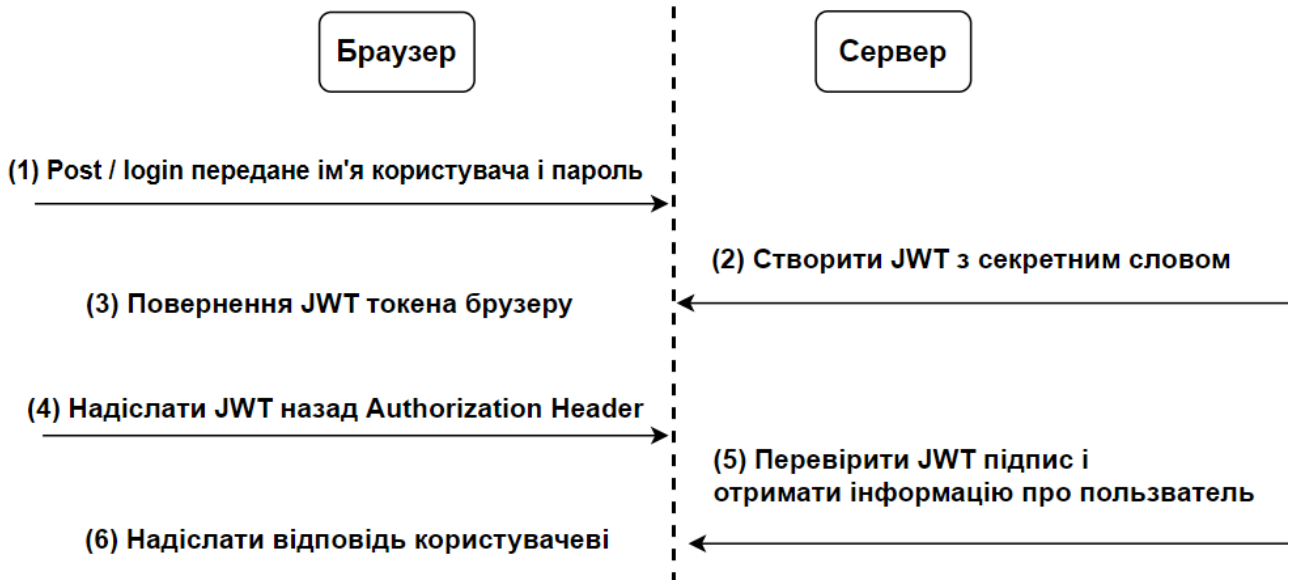


Рис. 3.5. Зображення взаємодії серверов

3.2.1. Порядок дій при реєстрації користувача

У системі є реєстрація нового користувача, але лише користувач з роллю адміністратора може додати нового користувача. Користувач реєструється в системі, після чого на електронну адресу користувача надходить повідомлення з проханням активувати обліковий запис шляхом зміни пароля. Цей алгоритм показаний на рис. 3.6.

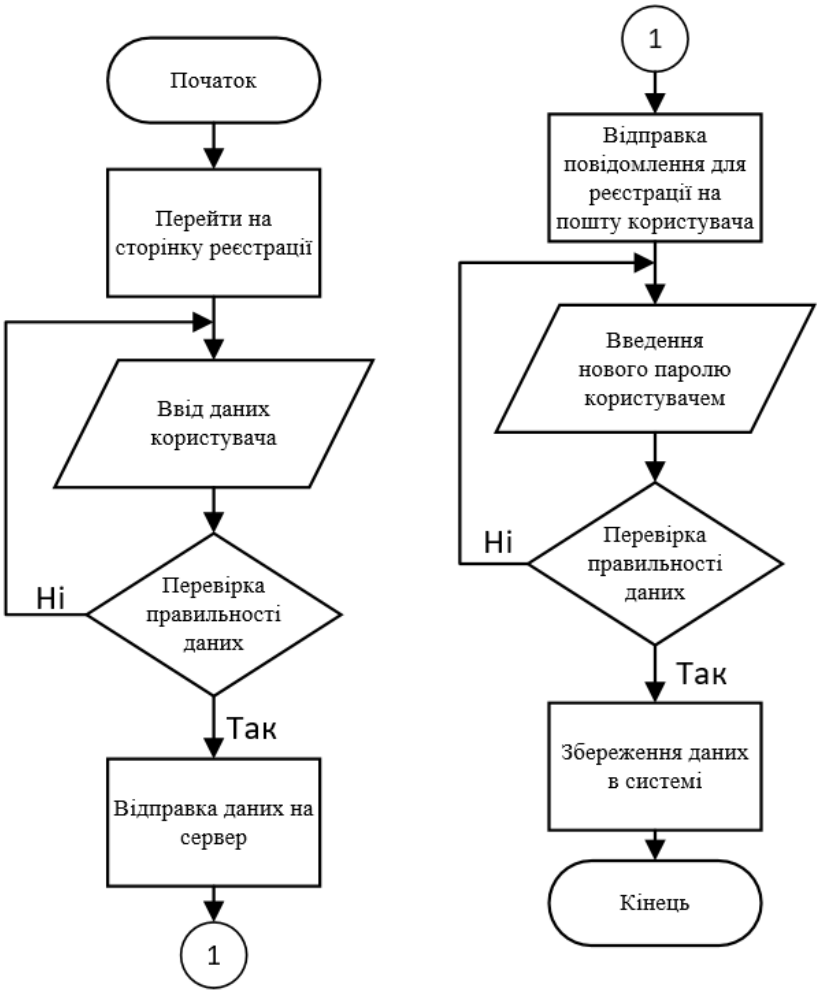


Рис. 3.6. Алгоритм реєстрації користувача

Щоб надіслати повідомлення користувачеві, використовуйте клас SimpleMailMessage, щоб створити просте повідомлення електронної пошти, що містить адресу відправника, адресу одержувача та текстові повідомлення. Приклад коду SimpleMailMessage показано на «Рис. 3,7»

```
public void sendCredentialsByEmail(User user){
    SimpleMailMessage message = new SimpleMailMessage();

    message.setTo(user.getEmail());
    String token = new PasswordToken().generatePasswordResetToken(user);

    message.setSubject(MailConstant.EMAIL_THEME);
    message.setText("To activate your account click on the following link: " +
            RESET_PASSWORD_LINK + token);
    mailSender.send(message);
}
```

Рис. 3.7. Приклад коду відправки повідомлення

Ключі лежать в зашифрованому вигляді в системі. Вона використовує метод шифрування BCrypt, який генерує випадкові байти та об’єднує їх із паролем, перш ніж хеш створює унікальні витяги для кожного пароля користувача. Приклад ключа шифрування показаний на рис. 3.8».

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Override
public void updateUserPassword(User user) {
    userDao.updateUserPassword(user.getEmail(), passwordEncoder.encode(user.getPassword()));
}
```

Рис.3.8. Приклад коду для шифрування паролів

3.3. Можливості в роботі з Selenium

Кожен інтернет-браузер має власний спеціальний драйвер браузера. Для коректної роботи необхідно вказати шлях до драйвера створення об’єкта. Наступним кроком є запуск тесту, який запускає взаємодію Selenium WebDriver між клієнтською бібліотекою Selenium і реальним браузером. Драйвер браузера вважається ключовим для отримання HTTP-запитів, і в будь-якому випадку під час виконання тесту запит на відкриття сторінки входу в програму надсилається через Selena до Vires JSON, який перетворює запит у формат JSON. JSON Vire надсилає запит на HTTP-сервер ChromeDriver за допомогою REST API через HTTP. Тобто виконуються такі операції:

- ChromeDriver надсилає отриманий запит на сервер веб-переглядача.
- Браузер Chrome відкриває сторінку входу до програми за вказаною вами URL-адресою.
- Веб-браузер Chrome надсилає відповідь клієнту після виконання необхідних дій у програмі.
- Ця відповідь надсилається клієнту по тому самому каналу.

На рисунку 3.9 показано кроки для налаштування веб-драйвера вашого браузера:

- Headless: відкриває окреме вікно веб-переглядача Chrome.
- lang = en Надає англомовну версію для браузера.
- -disable-gpu - вимикає прискорення графічного процесора.
- --- без пісочниці – вимикає безпечне виконання програми для всіх типів процесів.
- --disable-dev-shm-usage – вимикає розділ розробника для підвищення продуктивності.продуктивності

```
System.setProperty("webdriver.chrome.driver", "E:\\chromedriver.exe");
ChromeOptions options = new ChromeOptions();
options.addArguments("--disable-gpu");
options.addArguments("--no-sandbox");
options.addArguments("--disable-dev-shm-usage");
options.addArguments("--headless");
options.addArguments("--lang=en");
WebDriver driver = new ChromeDriver(options);
```

Рис. 3.9. Аргументи драйвера браузера Chrome

3.4. Проектування інтерфейсу програми

3.4.1. Маршрутизація Router

Клієнтська частина програми використовує технології, пов’язані з Angular, тобто вихід на маршрутизатор. До цієї директиви можна легко отримати доступ з каталогу маршрутизатора, куди маршрутизатор вставляє компонент, який відповідає URL-адресі поточного браузера. Найпростішою технологією є компонент Angular, який є класом TypeScript під назвою @Compent. Усі компоненти, що відповідають маршрутизатору, відображаються на виходах маршрутизатора. Приклад компонента наведено на «рис. 3,10».

```
@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css']
})
export class UsersComponent implements OnInit {
```

Рис. 3.10. Реалізування компоненту

Програма реалізує маршрутоподібні технології, що складаються з атрибутів маршруту "redirectTo". Шлях посилається на частину URL-адреси, яка визначає відображення користувацького подання, а компонент посилається на компонент Angular для поєднання зі шляхом. На основі визначення маршруту маршрутизатор може направити користувача на певний дисплей. Кожен шлях показує URL-адресу певного компонента. Детальний приклад маршруту наведено на «рис. 3.11».

```
const routes: Routes = [
  { path: 'login', component: AuthenticationComponent},
  { path: 'projects/:project_id', component: ProjectViewComponent},
  { path: 'create-action', component: CreateActionComponent },
  { path: 'menu', component: MenuComponent},
  { path: 'users', component: UsersComponent},
  { path: 'users', component: UsersComponent},
  { path: 'projects', component: ProjectComponent},
```

Рис. 3.11. Приклад реалізації маршрутів

Система реалізує шляхи, які можуть бути порожніми, що є шляхом програми за замовчуванням, зазвичай у будь-якому місці на початку програми. Шлях також може містити символ підстановки (**). Маршрутизатор вибирає цей маршрут, якщо запитувана URL-адреса не відповідає жодному з маршрутів. У системі це відповідає шляху до сторінки помилки, якщо користувач переходить до запиту, який не існує.

3.4.2. Служби в Angular

У програмі сервіси Angular призначені для інтеграції бізнес-логіки та даних у різні компоненти. Зазвичай це клас з чіткою метою щось зробити.

У цій розробці служби є написаними методами, які надсилають запити до бекенда проекту.

Angular інтерпретує клас як службу ін’єкції на основі анотацій @Injectable, необхідних для поточного базового модуля app.module.ts. Служба зареєстрована в модулі інжектора, тому ви можете реалізувати цю послугу в будь-якому з його підкомпонентів. Прикладом сервісів Angular є «Рис. 3,12".

```
@Injectable({
  providedIn: 'root'
})
export class UserService {

  private url = `${environment.url}`;
  private managerUrl = this.url + 'manager';
  private adminUrl = this.url + 'admin';
  private getUsersListUrl = `${this.url}users`;
  private countPagesUrl = `${this.url}users/pages/count`;
  private countSearchPagesUrl = `${this.url}users/pages/count-search`;

  constructor(private http: HttpClient) {
  }

  getPage(params: Params) {
    return this.http.get<UserDto[]>(this.getUsersListUrl, options: {params});
  }
```

Рис. 3.12. Приклад реалізації сервісу

3.4.3. З’єднання та обмін інформацією з базою даних

Розвинена система має величезну кількість інформації, яку потрібно десь зберігати та обробляти. Для цього використовуйте базу даних Postgreskl як сховище об’єктів і технологію JdbcTemplate для запису та отримання необхідних даних у програму. JDBC відстежує обробку будь-яких подій і з'єднань з базою даних, запитів або оновлення SKL і ініціює ітерації в ResultSet. Приклад JdbcTemplate показаний на «рис. 3,13».

```java
@Override
public void updateUserPassword(String email, String password) {
    jdbcTemplate.update(UPDATE_USER_PASS, password, email);
}

@Override
public void saveActionInstancesWithoutCompoundInstanceId(List<TestScenarioItemDto> actions,
                                                         Integer testScenarioId) {
    jdbcTemplate.batchUpdate(
            INSERT_ALL_WITHOUT_COMPOUND_INSTANCE_ID,
            actions,
            actions.size(),
            (preparedStatement, action) -> {
                preparedStatement.setLong( parameterIndex: 1, testScenarioId);
                preparedStatement.setLong( parameterIndex: 2, action.getId());
                preparedStatement.setLong( parameterIndex: 3, action.getPriority());
                preparedStatement.setString( parameterIndex: 4, action.getContextInstanceName());
            });
}
```

Рис. 3.13. Методи JdbcTemplate

На зображенні вище показано деякі приклади того, як JdbcTemplate працює з даними. Програма має методи, вхідними параметрами яких є вся необхідна інформація для введення даних в систему. Найбільш часто використовувані методи впровадження системи:

- оновлення - щоб вставити, оновити та видалити записи.
- query – Виконує запит із зворотним викликом PreparedStatement.
- batchupdate - для оновлення кількох полів одночасно

Як згадувалося вище, цей проект працює на Heroku, що означає, що системна база даних знаходиться на віддалених серверах. Дані, надані Heroku, показані на зображенні нижче. 3,14».

Database Credentials

Get credentials for manual connections to this database.

Please note that **these credentials are not permanent**.

Heroku rotates credentials periodically and updates applications where this database is attached.

Host	ec2-52-22-216-69.compute-1.amazonaws.com
Database	deeb7gg799uir9
User	rqxvgjwerbmlhe
Port	5432
Password	03728ce4f43725c1b4991032b28e15fdc88aecb086b3ad3ad4099578c01e10c3
URI	postgres://rqxvgjwerbmlhe:03728ce4f43725c1b4991032b28e15fdc88aecb086b
Heroku CLI	heroku pg:psql postgresql-clean-24876 --app control-testing-hub-database

Рис. 3.14. Облікові дані бази даних

Підключення до бази даних здійснюється шляхом запису констант у файли програми. Heroku робить усі необхідні дані доступними на своєму сервері після створення бази даних. Правильне заповнення констант бази даних показано на «рис. 3,15».

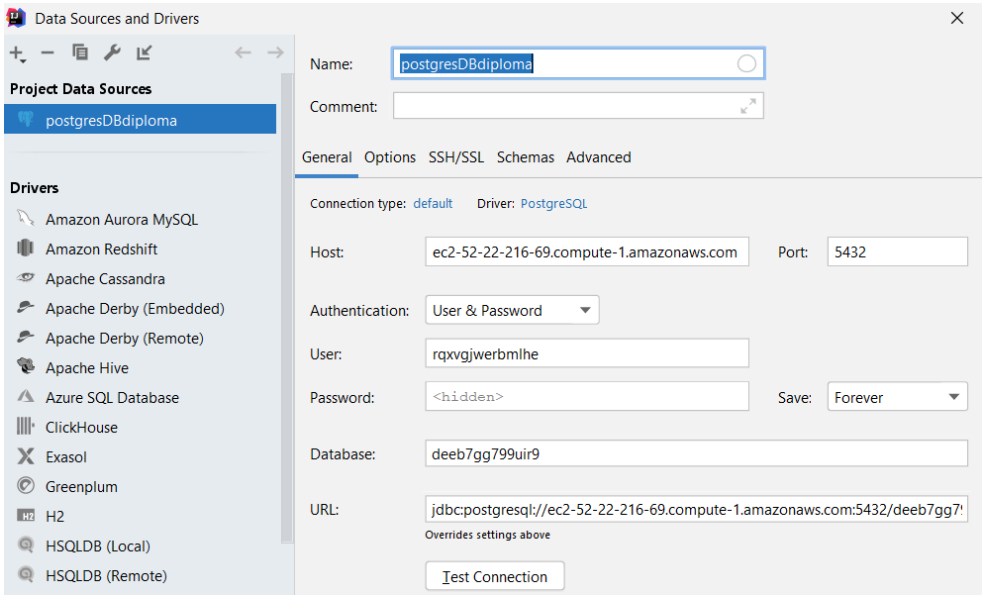


Рис. 3.15. Підключення бази до Intellij Idea

Для локальної роботи з базою даних потрібні налаштування конфігурації, а всі деталі підключення до бази даних повинні бути записані в середовищі розробки Intellij Idea на вкладці «База даних».

3.5. Розробка схем для зберігання даних в системі

3.5.1. Основні дані програми

Ця система використовує величезну кількість інформації, і відповідно її потрібно десь зберігати та обробляти. Для цього програма використовує базу даних, яка дозволяє працювати з наборами інформації. У цій системі є 18 таблиць даних. Найважливіші системні таблиці:

- User – таблиця, яка зберігає всіх користувачів в системі, а так само всю необхідну інформацію для коректної роботи з ним.
- Action – для збереження найменшої тестової одиниці – action.
- Compound – складається зі списку дій, згрупованих за функціоналом.
- Compound_action – для зв’язку компаунда з екшнами.
- Subscription – таблиця підписок необхідна, щоб будь-який користувач зміг підписатися на тест-кейс і незабаром після його виконання, йому прийде повідомлення на пошту про виконане результаті.
- Variable – таблиця в якій зберігаються змінні, які будуть використані при створенні тестового сценарію.
- Variable_value – таблиця зі значеннями для таблиці variable.
- Data_entry – зберігає ключ-значення – де, ключ: елемент для вводу даних;.
- Project – таблиця необхідна для зберігання всіх створених проектів в системі.
- Action_instance – таблиця, в якій створюється об'єкт модельної суті action. У ній відбувається необхідна прив'язка конкретного action до деякого compound.

- Test_case – одна з основних таблиць, яка зберігає всі дані, необхідні для роботи з тестуванням веб-додатків.
- Data_set – таблиця, яка зберігає назви згрупованої суті з таблиці data_entry
- Test_case_execution – таблиця, в якій зберігається інформація про виконані тестів.
- Test_scenario – таблиця яка зберігає всі тест сценарії в даній системі.
- Compound_instance – таблиця, в якій створюється об'єкт модельної суті compound. Необхідна для прив'язки конкретного compound до деякого test_scenario.
- Context_data_item – таблиця, в якій зберігаються значення, що повертаються з тест-кейсів необхідно важливо для продовження виконання тестів.
- Notification – таблиця, зберігає всю інформацію про повідомлення для користувачів.
- Action_execution – в даній таблиці зберігаються необхідна інформація про виконання кожної дії під час тестів.

3.6. Розроблені модулі програми

Повна ієрархія модулів в проекті зображена на рисунку 3.16.

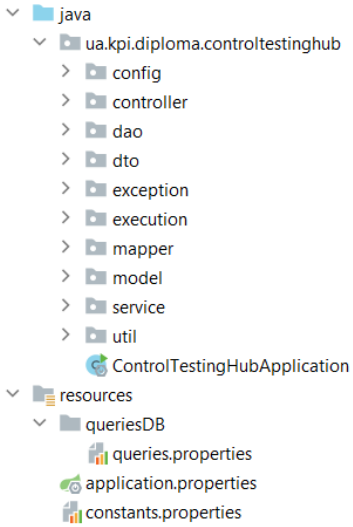


Рис. 3.16. Повна структура проекту

Головною точкою входу в проект є файл, який зображено на «рис. 3.17», є ControlTestingHubApplication.java.

```
@SpringBootApplication
public class ControlTestingHubApplication {

    public static void main(String[] args) {
        SpringApplication.run(ControlTestingHubApplication.class, args);
    }

}
```

Рис. 3.17. Реалізація головного файлу проекту

Анотація @SpringBootApplication використовується для позначення конфігурації класу, яка оголошує один або кілька методів @Bean, але також ініціює автоматичну конфігурацію і тестування bean-компонента. Зазвичай це те саме, що оголошення класу з анотаціями @ComponentScan, @EnableAutoConfiguration та @Configuration.

Файли в модулі ресурсів відповідають за зберігання констант, необхідних для життєвого циклу програми. Файл query.properties містить усі запити Postgreskl, які використовуються для доступу до бази даних.

Цей проект заснований на моделі розробки MVC, яка включає три архітектурні шари:

- Модель;
- Представлення;
- Контролери;

Механізм містить класи, відповідальні за підготовку моделі з даними для відображення в поданні, і може використовувати коментар @ResponseBody, щоб безпосередньо записати запис у процес відповіді та виконати запит. Детальний огляд модуля можна знайти в розділі «Рис. 3,18»

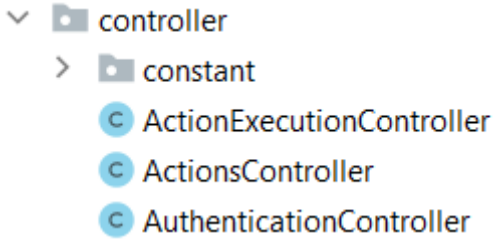


Рис 3.18. Структура модулів контролеру

Сервісний модуль має структуру інтерфейсу та реалізацію інтерфейсу в папці serviceImpl. Це служить для подальшого абстрагування методів. Рівні сервісу забезпечують модульність програмного коду, бізнес-логіку, принципи та правила вказуються на рівні сервісу, який називається рівнем DAO. У цьому випадку рівень DAO відповідає лише за взаємодію з базою даних, у нашому випадку за доступ до бази даних. База даних з JdbcTemplate. Приклад сервісного модуля показаний на рис. 3.19. показано на малюнку.

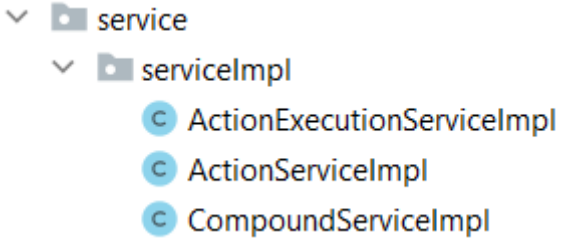


Рис. 3.19 Структру модулів сервісу

У даній реалізації DAO – це об’єкт, який виражає абстрактне представлення інтерфейсу для структур, пов’язаних із реалізацією JDBC. Загальна концепція полягає в наданні доступу до JDBC через окремі допоміжні класи. DAO забезпечує загальну ієрархію винятків, вказуючи коментар @Repository. Приклад DAO показаний на «рис.3.20».

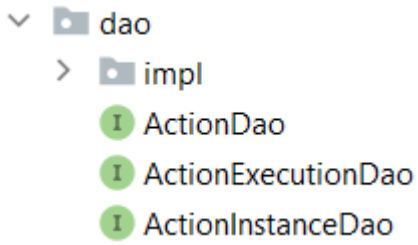


Рис. 3.20 Структура модулів DAO

ВИСНОВОК ДО РОЗДІЛУ 3

Цей розділ описує технологію, яка використовується для успішної розробки системи. Описано серверну та клієнтську частини програми, а також детальні алгоритми створення кожної структури програми.

Результатом цього розділу є алгоритми обміну інформацією між серверами, зв’язку з базою даних та створення модулів проекту. Дано огляд модулів і структур папок, які використовуються в програмі. Ця система дуже гнучка і проста у використанні для користувачів.

4. ПЕРЕВІРКА РОЗРОБЛЕНОЇ ПРОГРАМИ

4.1. Перевірка програмного функціоналу

4.1.1. Властивості серверів веб-хостінгу

Цей додаток поставлено на веб-хостинг Heroku. На жаль, для безкоштовного використання цього ресурсу надано не найпотужніші сервери. У Heroku образ операційної системи, більш відомий як «стек», заснований на Ubuntu, дистрибутиві Linux. Нижче в таблиці 4.1 перераховані основні системні характеристики системи.

Таблиця 4.1 Опис характеристик Heroku

Название характеристики	**Значение характеристики**
Memory(RAM)	512 мб
Количество виртуальных процессоров	1-4 шт. (зависит от расположения сервера)
Средняя скорость интернет загрузки	25.6 кб/с
Время загрузки 30 кб	1.22 сек

4.1.2. Веб-хостінг Heroku

Як і будь-який веб-додаток, перш ніж працювати з ним на хостингу, його потрібно правильно задеплоіть. Heroku тісно пов'язаний з системою контролю версій GitHub, де спочатку лежать наші реалізовані програми для серверної і клієнтської частин. Після кожного commit і push на GitHub, Heroku автоматично робить деплой на свій сервер. На рисунку 4.1 зображено детальний приклад збірки системи.

```
       [INFO] Installing /tmp/build_5c091fea/target/control-testing-hub-0.0.1-SNAPSHOT.jar to /t
SNAPSHOT/control-testing-hub-0.0.1-SNAPSHOT.jar
       [INFO] Installing /tmp/build_5c091fea/pom.xml to /tmp/codon/tmp/cache/.m2/repository/ua/k
SNAPSHOT.pom
       [INFO] ------------------------------------------------------------------------
       [INFO] BUILD SUCCESS
       [INFO] ------------------------------------------------------------------------
       [INFO] Total time:  16.870 s
       [INFO] Finished at: 2021-05-24T11:33:33Z
       [INFO] ------------------------------------------------------------------------
-----> Discovering process types
       Procfile declares types     -> (none)
       Default types for buildpack -> web
-----> Compressing...
       Done: 79M
-----> Launching...
       Released v10
       https://control-testing-hub-java.herokuapp.com/ deployed to Heroku
```

Рис. 4.1. Збірка системи на Heroku

Таким чином, як показано вище на малюнку, відбувається вдала збірка додатки на сервер Heroku. Можна помітити що на сходинці «Done: 79M» пишеться пам'ять яка використана сервером для складання цього додатка.

Application Logs

```
2021-05-24T11:40:01.000000+00:00 app[api]: Build succeeded
2021-05-24T12:12:31.443186+00:00 heroku[web.1]: Idling
2021-05-24T12:12:31.456963+00:00 heroku[web.1]: State changed from up to down
2021-05-24T12:12:39.561033+00:00 heroku[web.1]: Stopping all processes with SIGTERM
2021-05-24T12:12:39.833611+00:00 heroku[web.1]: Process exited with status 143
2021-05-24T15:29:52.000310+00:00 heroku[web.1]: Unidling
2021-05-24T15:29:52.008144+00:00 heroku[web.1]: State changed from down to starting
2021-05-24T15:30:03.381961+00:00 heroku[web.1]: Starting process with command `npm start`
2021-05-24T15:30:06.858643+00:00 app[web.1]:
2021-05-24T15:30:06.858711+00:00 app[web.1]: > control-testing-hub-angular@0.0.0 start /app
2021-05-24T15:30:06.858712+00:00 app[web.1]: > node server.js
2021-05-24T15:30:06.858712+00:00 app[web.1]:
2021-05-24T15:30:07.576914+00:00 heroku[web.1]: State changed from starting to up
2021-05-24T15:30:10.128057+00:00 heroku[router]: at=info method=GET path="/" host=control-test
fwd="77.47.190.22" dyno=web.1 connect=1ms service=40ms status=200 bytes=1246 protocol=https
```

Рис. 4.2. Запуск додатку на Heroku

При першому запуску програми Heroku надає ім'я нашого веб-сайту https://control-testing-hub-angular.herokuapp.com. На малюнку 4.2 показаний під час запуску програми, в першому рядку показано що наше додаток вдало зібралося на сервері і воно готове до роботи. У рядках «State changed from starting to up», говорить про те що додаток вдало розпочало свою роботу і готові до використання.

4.1.3. Перевірка JW токена під час автентифікації користувача

Як описувалося раніше в попередніх розділах, для більшої безпеки додатки використовується авторизація за допомогою технології Jwt token.

Рис. 4.3. Jwt токен

З малюнка видно як виглядає Jwt token, зазвичай він складається з трьох частин:

- Заголовка - містить дані про тип токена і криптографічних алгоритмах, необхідних для захисту інформації.
- Корисного навантаження - містить всю необхідну ідентифікаційну інформацію про користувача для перевірки
- Підпис - необхідний для перевірки того, що даний токен не є підроблений, тільки після цієї перевірки з ним можна працювати.

В даному токені міститися зашифровані дані користувача, таким чином, щоб була можливість в подальшому з ними працювати.

4.1.4. Тестування авторизації користувача

Авторизація користувача відбувається шляхом введення електронної пошти користувача і персонального пароля. Після шляхом обробки даних пароль в зашифрованому вигляді передається з клієнтської частини на серверну. Це зроблено для того щоб ніхто не зміг перехопити трафік і скористатися персональними даними.

На серверної частини пароль шифрується вдруге за допомогою технології BCryptPasswordEncoder, в результаті чого виходить ще більш безпечний пароль «$2a$10$/O5NKVCLz.2VqLtQzVL leuTRL7Pykb WZFMjB.Oh / ArhoY.vse». Після авторизації йде перевірка користувача на конкретні права в цій системі.

Рис. 4.4. Приклад реєстрації користувача

4.1.5. Перевірка драйвера Selenium для автоматизованого тестування

Для автоматичного тестування був обраний Selenium webDriver. При першому запуску автоматичного тест-кейса спрацьовує код і викликається драйвер браузера, який автоматично відкриває веб-браузер на сторінці, яка прописана в засланні проекту.

```
Starting ChromeDriver 90.0.4430.24 (4c6d850f087da467d926e8eddb76550aed655991-refs/branch-heads/4430@{#429}) on port 11617
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
```

Рис. 4.5. Запуск Selenium Chrome драйвера

Виконання кроків тест-кейса виписуються в логи, за допомогою технології логування @ Slf4j. На малюнку 4.6. описані статуси виконання дій тесту.

```
2021-05-24 21:37:08.622  INFO 9352 --- [ null to remote] o.o.selenium.remote.ProtocolHandshake    : Detected dialect: W3C
2021-05-24 21:37:08.649  INFO 9352 --- [io-8080-exec-10] k.d.c.e.TestCaseExecutionServiceSelenium : Test case execution started
Click {button xpath=/html/body/div[1]/header/div/div[2]/div[2]/a}
2021-05-24 21:37:29.638  INFO 9352 --- [io-8080-exec-10] k.d.c.e.TestCaseExecutionServiceSelenium : Action STATUS of click is PASSED
```

Рис. 4.6. Лог кроків тест-кейсів

Дану статистику можна відправити користувачеві на пошту як таблицю результатів.

#	Action Name	Variable Name	Variable Key	Variable Value	Status
1	click	button xpath	click sig	/html/body/div[1]/header/div/div[2]/div[2]/a	PASSED
2	click	button xpath	clcik inp	/html/body/div[1]/header/div/div[2]/div[2]/a	FAILED
3	input	input xpath	text	vadsc	NOTSTARTED

Рис.4.7. Результат тестів

4.2. Перевірка інтерфейсу програми

4.2.1. Робота з інтерфейсом користувача

Щоб система працювала належним чином, потрібно запустити правильний скрипт від початку до кінця. Роботи будуть проводитися в наступні етапи:

1. Зареєструватися в системі

2. Створіть проект із посиланням на сайт для перевірки.

3. Створіть набір даних, у який потрібно записати всі вхідні дані, що використовуються в системі

4. Створіть складну операцію, яка визначає пріоритет набору операцій.

5. Створіть сценарій для запуску тесту

6. У рамках попередньо створеного проекту створіть тестовий кейс з конкретним скриптом і записом, заповніть усі поля операції

7. Запустіть тестовий приклад

8. Переглянути результати, надіслати звіт користувачеві

4.2.2. Створення проекту

Для створення проекту перейдіть на вкладку «Додати» і зареєструйте в системі назву свого проекту та посилання на сайт. Він перевіряє посилання при виклику сторінки і, якщо є, створює проект у системі та вносить всю необхідну інформацію в базу даних, тобто ім’я, посилання та користувача, який створив проект.

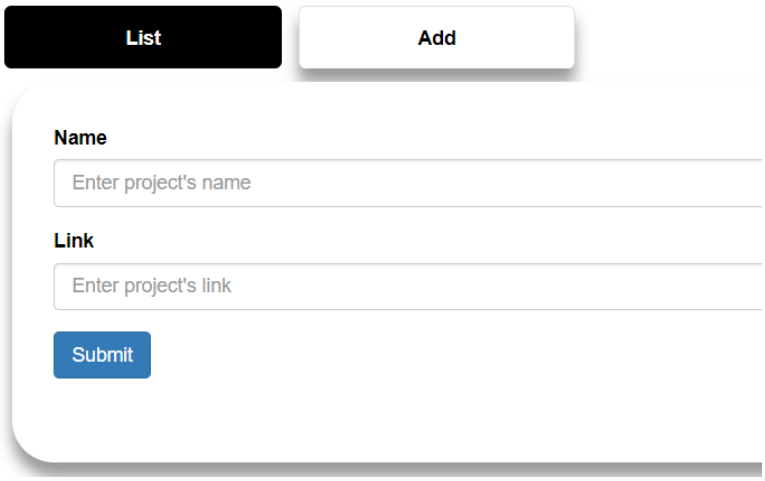


Рис.4.8. Створення проекту в системі

4.2.3. Введення необхідних даних для тестування

При створенні запису необхідно заповнити ключове поле, яке відповідає за відображення значення ключа при створенні тестового прикладу. Поле значення діє як вхідне значення, яке є частиною веб-сайту. Це зроблено для того, щоб користувач міг краще зрозуміти, як працює інтерфейс системи. Приклад ключового поля, значення:

- "Натисніть, щоб увійти" - "/html/body/div[1]/header/div/div[2]/div[2]/a"
- «Введіть у поле входу» - «ім'я користувача01»

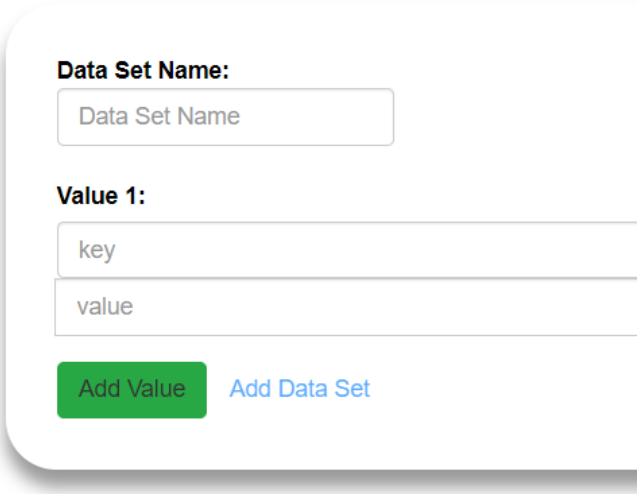


Рис. 4.9. Створення Data Set в системі

4.2.4. Створення сценарію

Щоб тест працював поетапно, потрібно написати скрипт, в якому потрібно вибрати порядок, в якому будуть виконуватися операції. На малюнку KSKS показано сценарій, де вибирається складна дія та дія. Як уже згадувалося, складна операція може містити кілька операцій. Складні операції містять операції, кожна з яких має певний пріоритет.

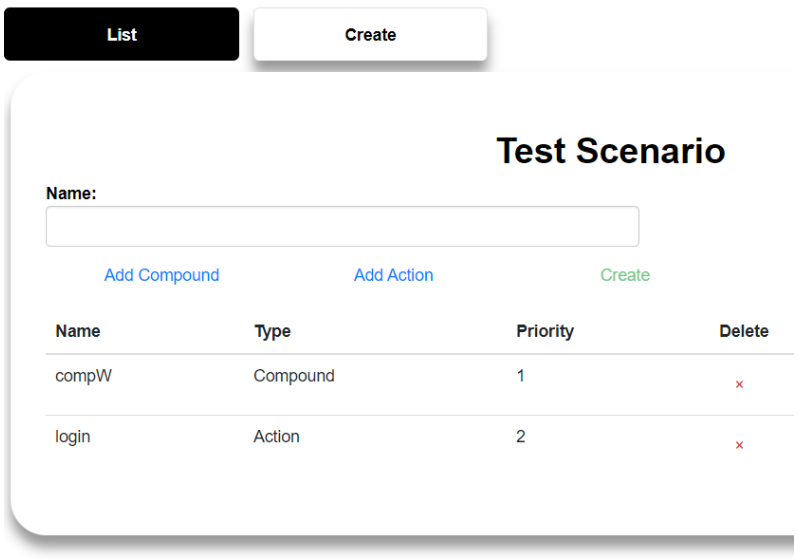


Рис. 4.10. Створення сценарію для тест-кейса

4.2.5. Створення тест-кейсу

Створення тестового випадку в системі є важливим кроком. У спеціально вибраному проекті перейдіть на вкладку «Додати» та виберіть попередньо створений сценарій для запуску тесту та набір даних, щоб вибрати значення для заповнення в полі «action».

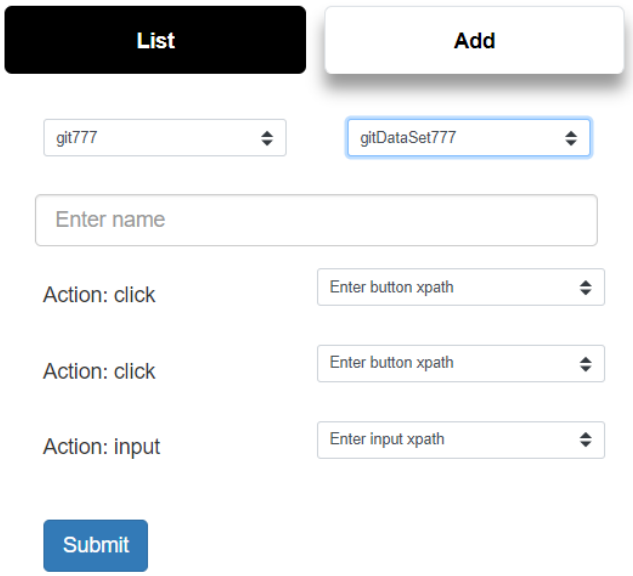


Рис. 4.11. Приклад створення тест-кейсу

4.2.6. Випробування виконання тестів та відправка звітів роботи

Після того, як користувач запустить тестовий приклад, натиснувши кнопку «Виконати», драйвер Selenium виконує кроки, описані в сценарії, вибраному для тестування. На екрані відображається автоматична перевірка веб-сторінки, зазначеної в проекті, користувач може побачити від створеної кінцевої точки до кінцевої точки. Після проходження тесту в меню «Історія тестів» можна знайти результати цього тесту, проаналізувати їх та надіслати користувачеві звіт про виконану роботу.

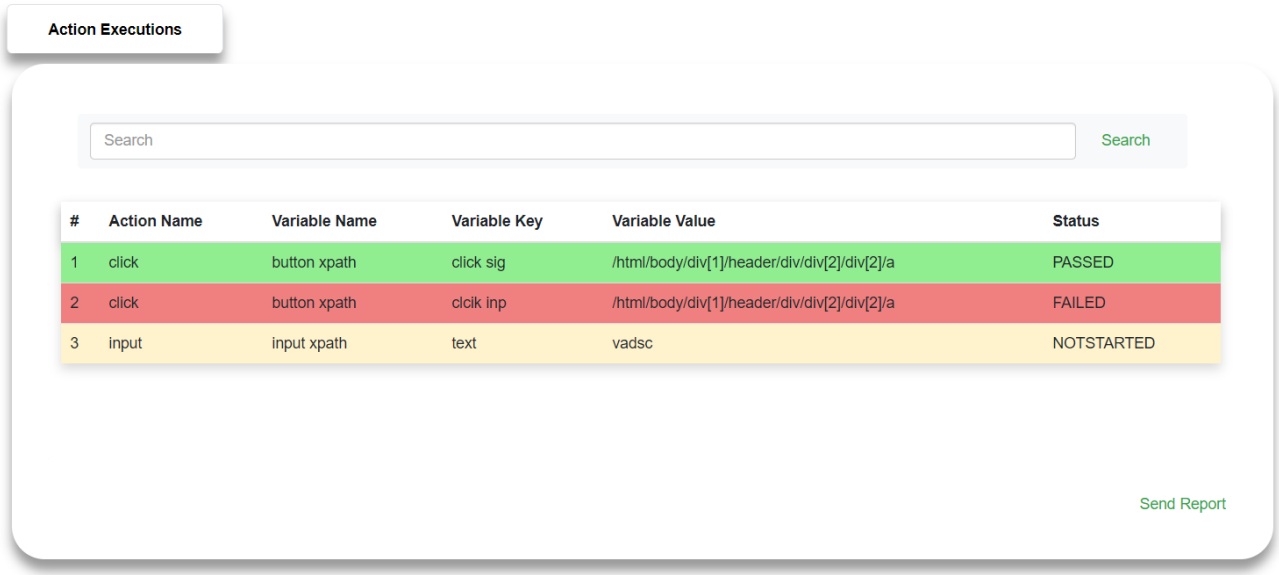


Рис. 4.12. Результати автоматичного тестування

ВИСНОВОК ДО РОЗДІЛУ 4

В останній частині тестувалися автоматизовані системи тестування сайту. Вони описують можливості веб-хосту, на якому працює ця система, а також детальне тестування кожної частини програми.

У цьому розділі наведено детальний опис сценарію, який вимагає від користувача роботи в системі. Ми прийшли до висновку, що ця система досить гнучка для заміни тестових компонентів, що дозволяє користувачам працювати в системі дуже комфортно.

ЗАГАЛЬНІ ВИСНОВКИ

Мета дипломної роботи - створення системи, яка дозволяє автоматично перевіряти помилки сервісу та контролювати доступ до веб-сайтів. Чітко описані завдання для успішної реалізації проекту.

Для успішної реалізації проекту ми надаємо детальну теоретичну інформацію з досліджуваної тематики, а саме: тестування та функціонування комп'ютерних та веб-додатків, безпека та контроль веб-сайтів. Ми розглядаємо різні альтернативи програмному забезпеченню, яке використовується для автоматизованого тестування. Ми також точно розуміємо, що потрібно вбудувати в нашу систему, щоб зробити її корисною та унікальною.

Під час розробки проекту для досягнення поставлених цілей були розроблені чіткі алгоритми, на основі яких має бути реалізована система. Для того, щоб повністю зрозуміти, як працює проект, кожен алгоритм детально розписаний у розділах.

При розробці проекту були використані новітні технології для швидкого та якісного створення програм. Детально описані всі технології, які використовуються при проектуванні системи, та наведено приклади реалізації програмного коду.

Після проектування системи проект був успішно протестований. Впроваджену систему тестували локально та на віддаленому сервері. Обидва тести показали хороші результати. В останньому розділі ми надаємо приклади, а також результати тестування завершеного проекту.

Проект спрямований на успішну розробку автоматизованої системи..

СПИСОК ЛІТЕРАТУРИ

1. Що таке тестування програмного забезпечення [Електронний ресурс] / https://www.guru99.com/software-testing-introduction-importance.html
2. Типи тестування програмного забезпечення [Електронний ресурс] https://www.softwaretestinghelp.com/types-of-software-testing/
3. Життєвий цикл програмного забезпечення [Електронний ресурс] https://qalearning.com.ua/theory/lectures/material/testing-intro/
4. Що представляє собою безпека веб-сайтів [Електронний ресурс] / https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security
5. Software testing [Електронний ресурс] / https://en.wikipedia.org/wiki/Software_testing
6. Web security as “Cybersecurity” [Електронний ресурс] - https://www.goodfirms.co/glossary/web-security/#:~:text=Web%20security%20is%20also%20known,%2C%20stores%2C%20and%20government%20locations
7. What is selenium for web testing [Електронний ресурс] - https://ru.wikipedia.org/wiki/SeleniumSelenium official documents for using in software testing [Електронний ресурс] - https://www.selenium.dev/documentation/en/
8. Як використовувати Heroku для веб-хостінгу [Електронний ресурс] - https://habr.com/ru/post/59270/
9. Heroku putting website on cloud[Електронний ресурс] / https://medium.com/@blondiebytes/heroku-hosting-put-your-website-on-the-cloud-cdc8729a7e09
10. Алгоритми тестування [Електронний ресурс] / https://www.nist.gov/system/files/documents/calibrations/5366.pdf

11. Walls C. Spring in Action. 5th Edition - Manning Publications Co., 2018 – 700c.
12. Bloch J. Effective Java. 3rd Edition. - Addison-Wesley Professional, 2018. - 392c.
13. Navneesh G. Test automation using selenium webdriver with java: step by step guide - Navneesh Garg – 193c.

ДОДАТКИ

Додаток А. Інструкція для користувача

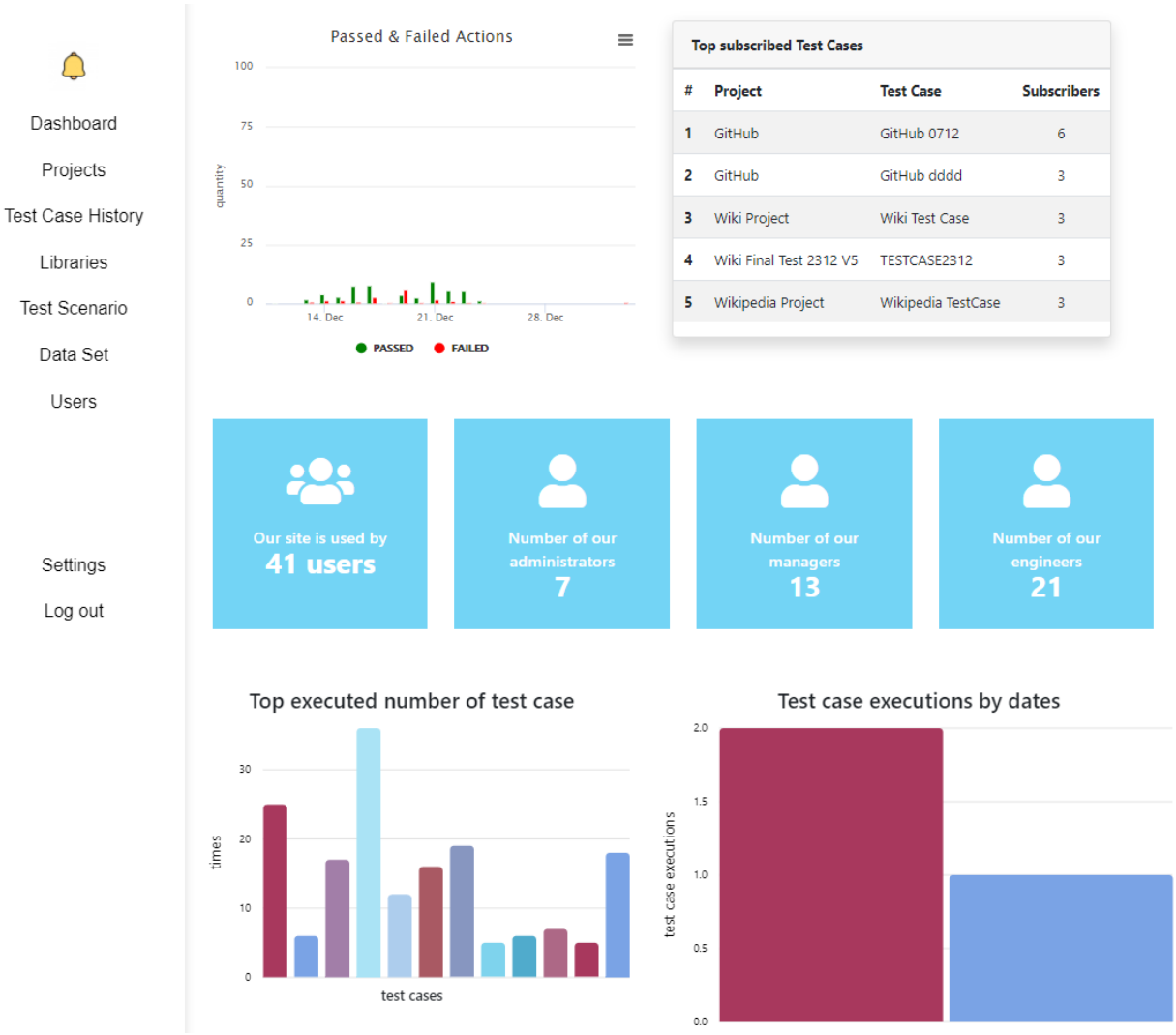


Рис. 5.1. Дашборд

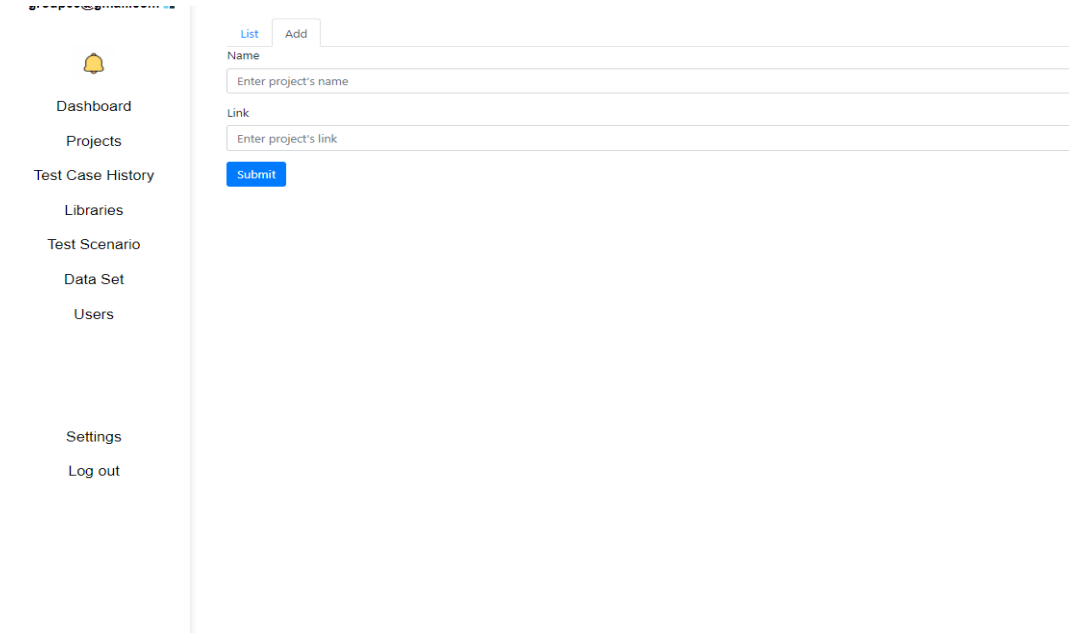


Рисунок 5.2. Створення проекту

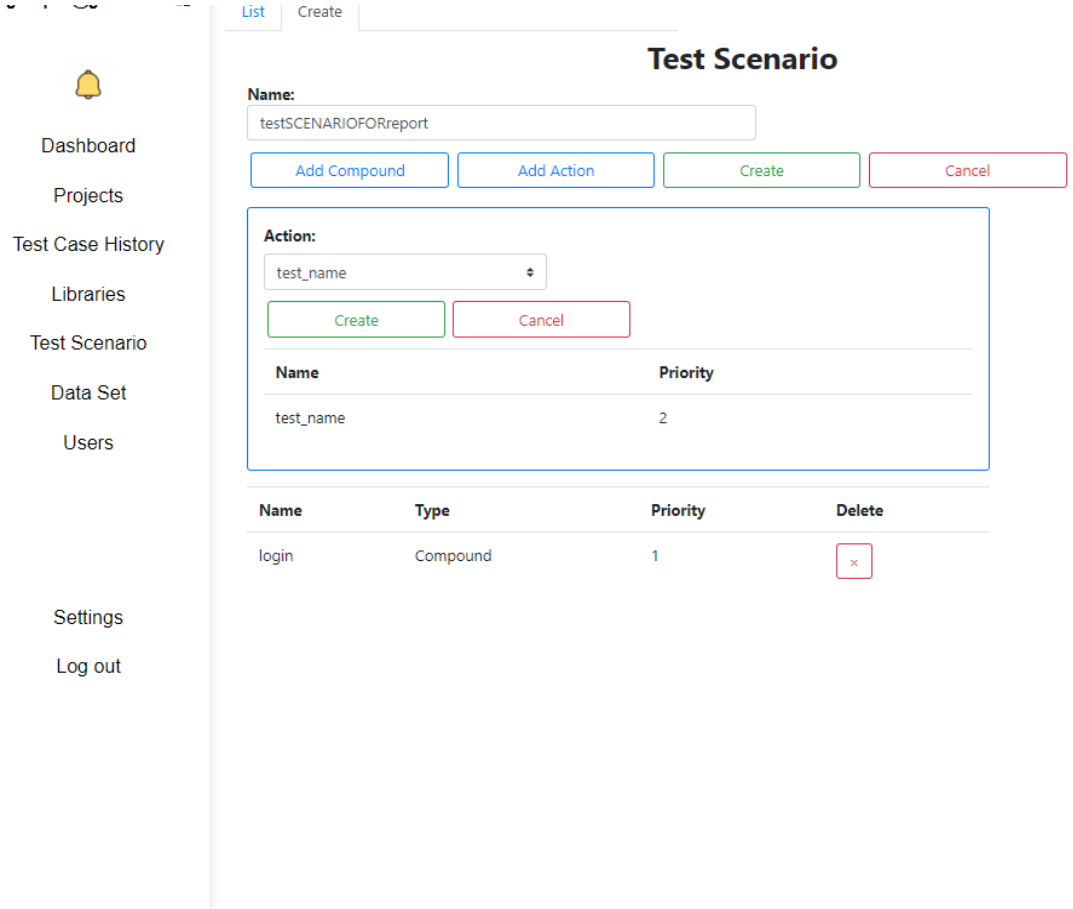


Рис. 5.3. Створення тестового сценарію

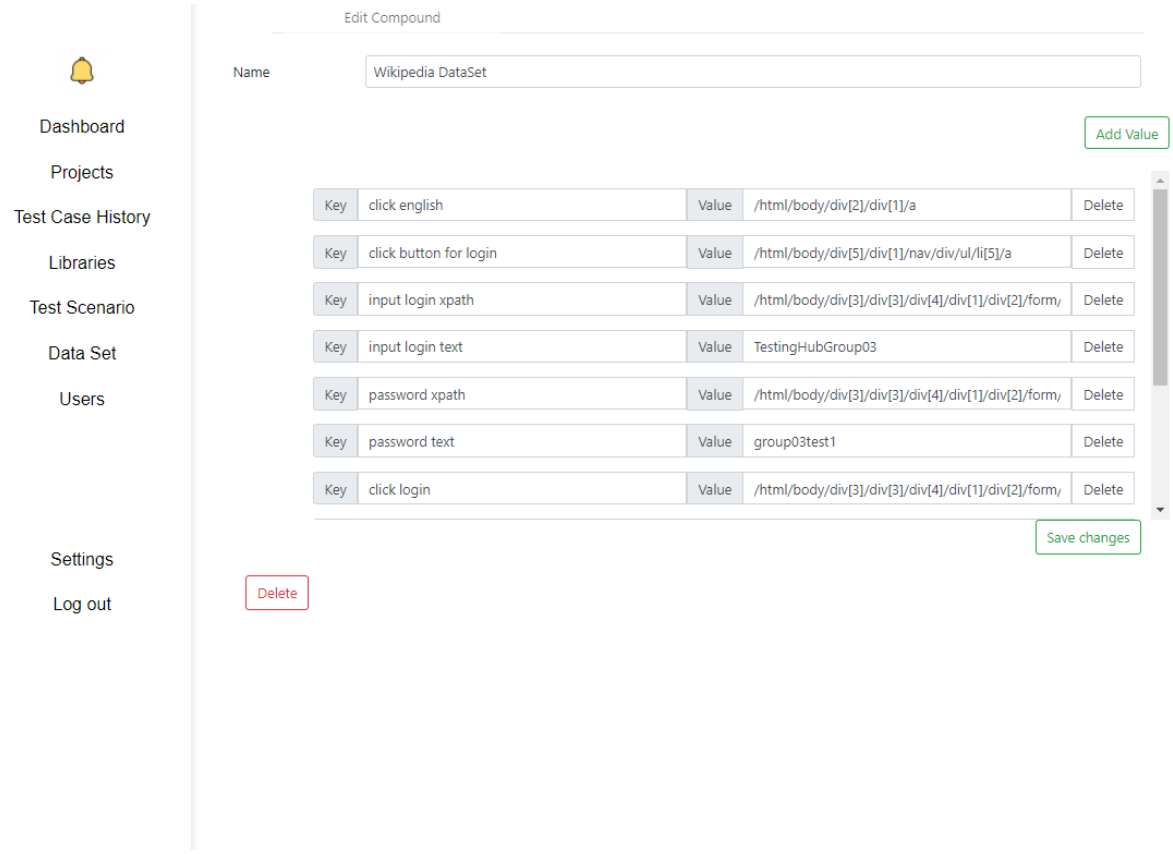


Рис 5.4. Створення дата сету

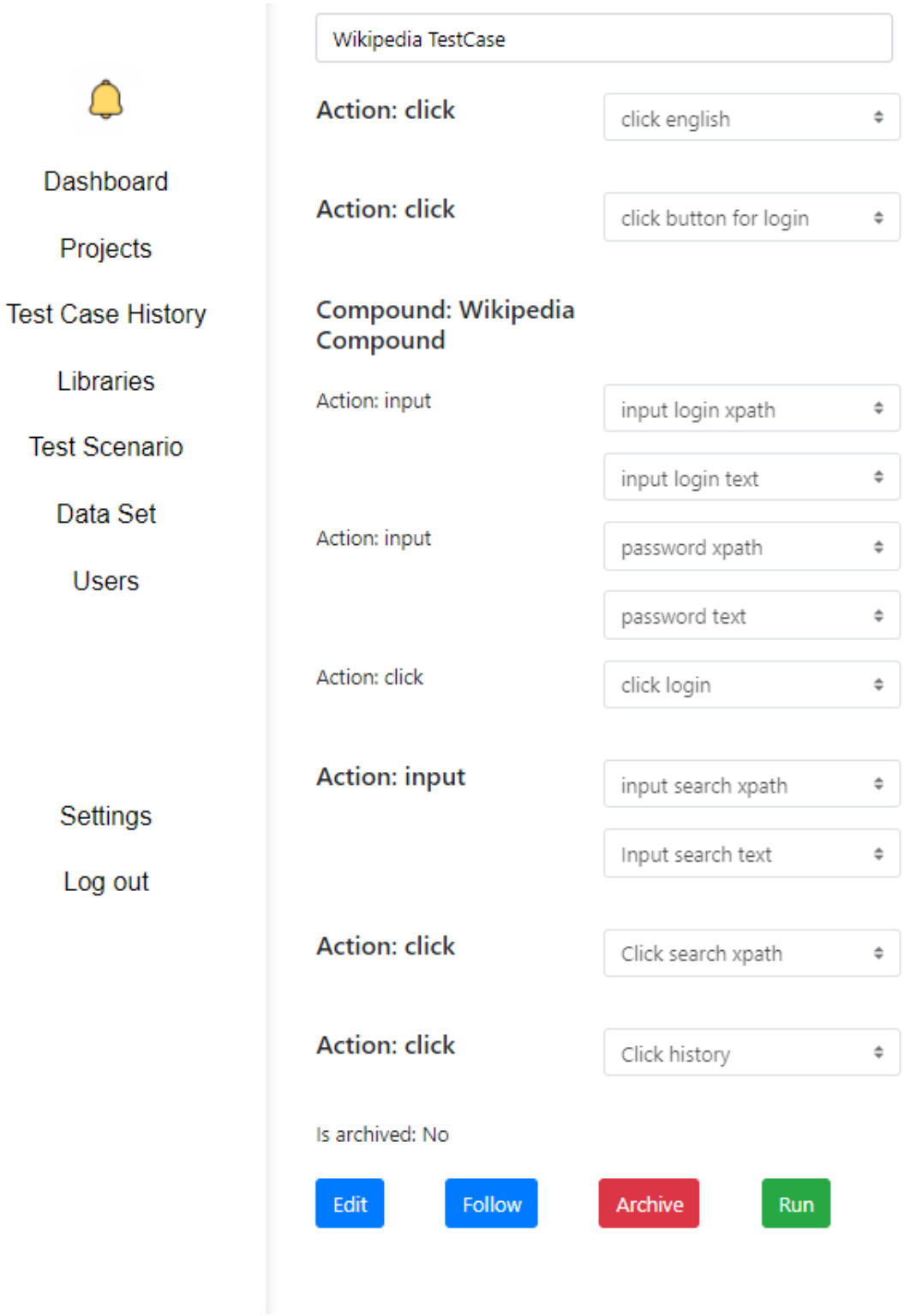


Рис. 5.5. Створення тест-кейсу

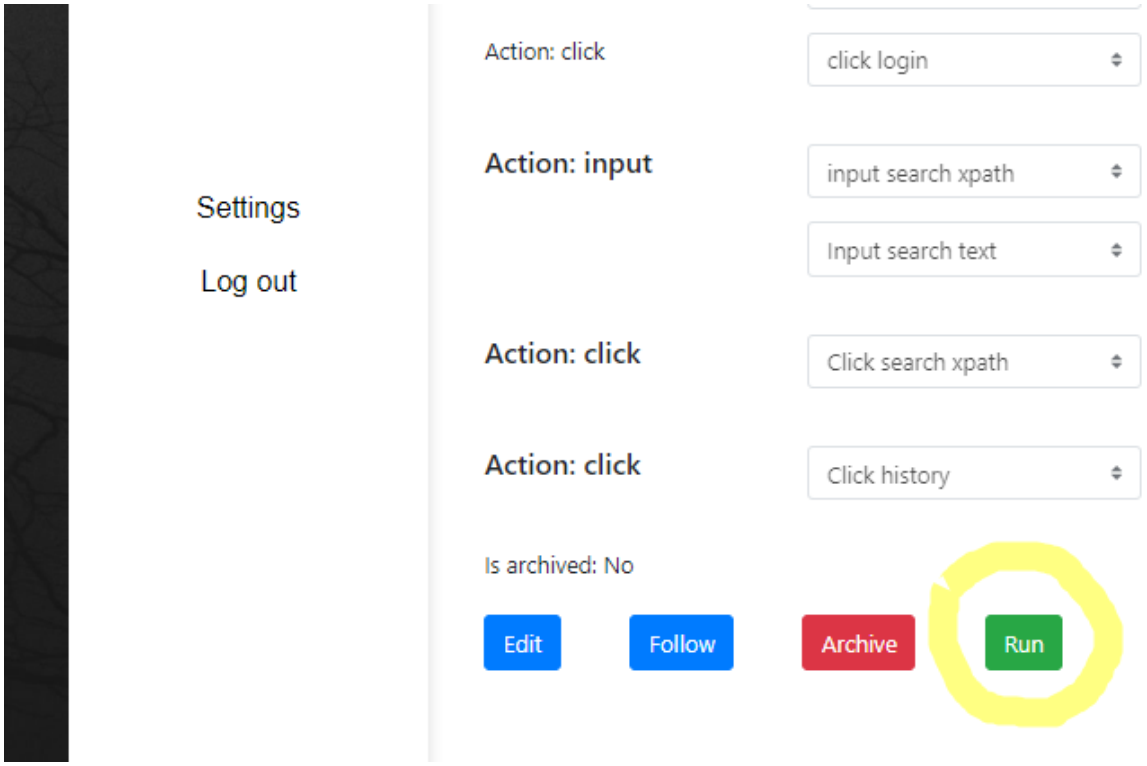


Рис. 5.6. Запуск тест-кейсу

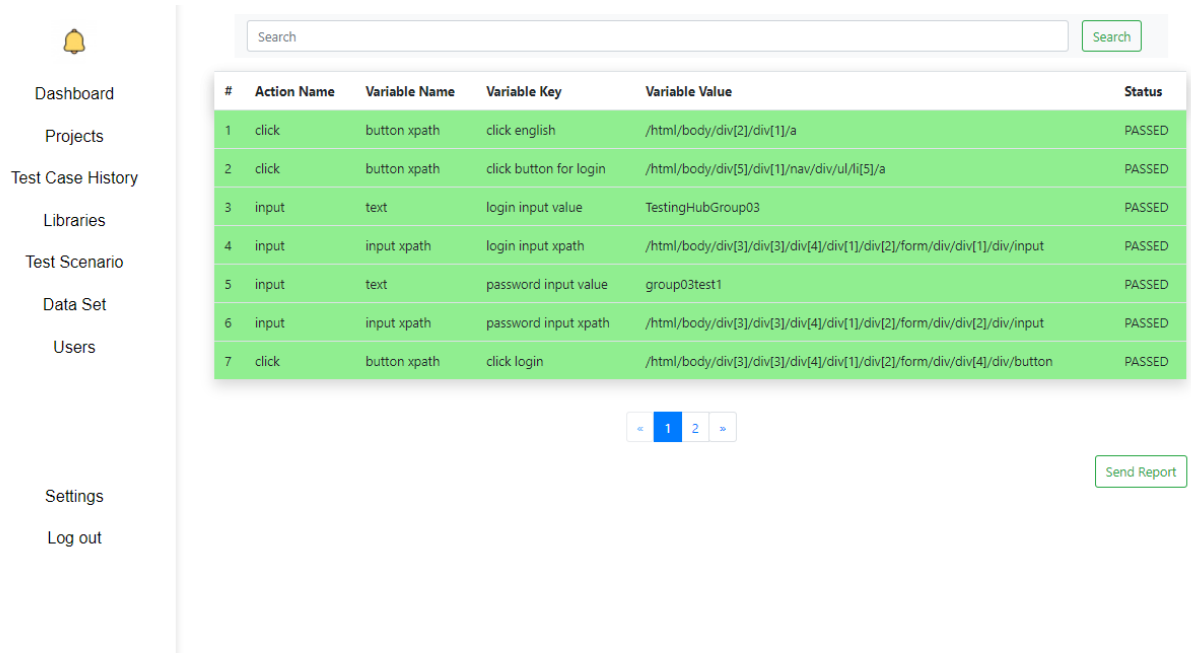


Рис 5.7. Перевірка виконання тест кейсу.