МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

_____ А.С. Савченко

«_ __ _» _____ 2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА
ЗА СПЕЦІАЛЬНІСТЮ
122 «КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ»

Тема: Система управління тест-кейсів

Виконавець: студент групи УС-211м Воскобойник Дмитро Васильович
(студент, група, прізвище, ім'я, по батькові)

Керівник: к.т.н., доцент, Полухін Анатолій Васильович
(науковій ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер:_____ І.Е. Райчев
(підпис) (П.І.Б.)

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Освітній ступінь: Магістр

Спеціальність: 122 «Комп'ютерні науки та інформаційні технології»
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач випускової кафедри

_____ А.С. Савченко

«____»_____2020 р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Воскобойника Дмитра Васильовича
(прізвище, ім'я, по батькові)

1. Тема дипломної роботи: Система управління тест-кейсів
затверджена наказом ректора від 25.09.2019 р. № 2175/ст, № пор. 5

2. Термін виконання дипломної роботи: з 14.10.2019 р. по 09.02.2020 р.

3. Вихідні дані до роботи: Середовище виконання – найбільш популярні на даний час браузери (Google Chrome, Mozilla Firefox, Safari);
необхідна швидкість підключення до мережі Інтернет – не більше 1MB/c

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): Титульний аркуш; Завдання на виконання дипломної роботи; Реферат; Зміст; Вступ; Розділ 1; Розділ 2; Розділ 3; Розділ 4; Висновки; Список використаних джерел; Додаток А.

5. Перелік обов'язкового графічного матеріалу: інформативні пояснювальні рисунки, презентація в *MS PowerPoint*.

6. Календарний план-графік

№ пор.	Етапи виконання дипломної роботи	Термін виконання	Підпис керівника
1.	Розроблення та затвердження календарного плану виконання дипломної роботи	14.10.2019	
2.	Підбір і вивчення літературних та інших джерел	15.10.2019-22.10.2019	
3.	Проведення консультацій з науковим керівником щодо виконання дипломної роботи	14.10.2019-09.02.2020	
4.	Підготовка та оформлення матеріалу за розділами 1, 2	23.10.2019-11.11.2019	
5.	Підготовка та оформлення матеріалу за розділами 3, 4	12.11.2019-30.11.2019	
6.	Проведення досліджень та опрацювання їх результатів	01.12.2019 31.12.2019	
7.	Оформлення пояснювальної записки та ілюстративного матеріалу	02.01.2020-28.01.2020	
8.	Підготовка до захисту та попередній захист дипломної роботи на випусковій кафедрі	29.01.2020-31.01.2020	
9.	Підписання необхідних документів у встановленому порядку та підготовка до захисту дипломної роботи в ЕК	31.01.2020	

7. Дата видачі завдання: 14.10.2019 р.

Керівник дипломної роботи _____ Полухін Анатолій Васильович
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Воскобойник Дмитро Васильович
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Система управління тест-кейсів». Робота містить 26 рисунків та одну таблицю.

Загальний обсяг роботи становить 90 сторінок, основний текст дипломної роботи викладено на 75 сторінках.

TEST CASE, TEST RUN, TEST SUITE, УПРАВЛІННЯ ТЕСТ КЕЙСАМИ, РЕУЛЬТАТ ПРОГОНУ.

Об’єкт дослідження: Інструменти управління тест-кейсів.

Предмет дослідження: Система управління тест-кейсів.

Мета дипломної роботи: Розроблення системи управління тест-кейсів з відкритим системним кодом.

У дипломній роботі висвітлено:

- Особливості роботи систем управління тест-кейсів
- Особливості створення документації тестувальників
- Етап прогону тестів
- Особливості кліент-серверної архітектури

Матеріали даної дипломної роботи можуть бути використані для створення, збереження, та модифікації тест-кейсів; створення тест-планів; виконання прогону тестів. Також вони можуть бути використані у навчальному процесі при підготовці фахівців з тестування, при вивченні артефактів та типів тестування. Таким чином, дані матеріали придатні як для роботи досвідченим фахівцямз тестування, так і для підготовки фахівців-початківців.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

TC	*TEST CASE*
GUI	*Graphical user interface*
HTTP	*HyperText Transfer Protocol*
REST	*Representational State Transfer*
IDE	*Integrated development environment*
TMS	*Test Management System*
OSI	*Open Source Initiative*
ПЗ	*Програмне забезпечення*
БД	*База даних*

ВСТУП

Системи управління тест-кейсів використовуються для зберігання інформації про те, як належним чином проводити тестування, здійснення черговості проведення тестів відповідно до його плану, а також для отримання інформації у вигляді звітів про стадії тестування і про те як тестується продукт. Даний тип ПЗ э досить вузько спеціалізованим і зазвичай розробляється компаніями котрі займаються розробкою чи тестуванням програмного забезпечення. Але відома та широко використовується певна кількість програмних продуктів, котрі розробляються сторонніми компаніями і є самодостатнім продуктом. Усі вони мають власні недоліки і переваги. Хоча подібний софт є досить специфічним, аналіз цих програм показує, що можна визначити певні критерії, по котрим їх можна класифікувати.

- За вартістю
- За розміщенням backend частини
- За швидкодією
- За відкритістю коду
- За додатковим функціоналом

Використовуючи статистику опиту користувачів подібних програм, можна виділити тринадцять найбільш популярних систем управління тест-кейсів:

Test Link, Test IT, Zephyr, qTest, PractiTest, TestLodge, TestRail, Qase, Tematoo, Test Collab, HP ALM, Testuff, XQual.

Система, яка розроблена в даній дипломній роботі, є подібною до вище перелічених і реалізує їх основні функції та нехтує додатковим, вузькоспеціалізованим функціоналом. Додатковий функціонал зазвичай не потрібен більшій кількості користувачів хоча декому він може й знадобитись, але все-таки було прийнято рішення відмовитись від розробки додаткового функціоналу з огляду на вузьке коло його потенціальних користувачів і було прийняте рішення зробити

програмний код системи відкритим щоб користувачі самі могли створити той функціонал який потрібен саме їм.

У дипломній роботі система розробляється як самодостатній продукт, котрий не потребує ніяких додаткових компонентів для функціонування. Система складатиметься з серверної і клієнтських частин жодна з яких не потребуватиме додаткових налаштувань.

Система буде корисною користувачам, котрі займаються тестуванням продуктів, тобто тестувальникам і є досить вузькоспеціалізованою. Програма надає змогу зберігати тест-кейси, об'єднувати їх у логічні групи та виконувати тестування продукту, опираючись на створені кейси. Також система є однією з частин підтримки системної документації продукту котрий тестується так як тест-кейси є однією з форм тестової документації.

Даний продукт може розвиватись далі як набір технологій для тестування, тобто додаючи функції та можливості, котрі полегшують роботу користувача або просто роблять її більш приємною. А політика відкритого програмного коду дозволить розвивати продукт за рахунок користувачів або просто дозволить змінювати систему під себе як на те буде зручно користувачеві.

Система управління тест-кейсів призначена для створення, збереження, структурування тест-кейсів та виконання прогонів тестів. Тобто для планування та виконання тестування продукту що є одним з етапів розробки програмного забезпечення.

Метою роботи є розроблення системи управління тест-кейсів з відкритим системним кодом. Така мета була обрана з огляду на те що продуктів з відкритим програмним кодом у даному сегменті нема або вони вже технологічно застаріли. Тому створення системи з відкритим програмним кодом дозволить користувачам самостійно конфігурувати і розвивати систему під себе не підшукуючи доступний варіант з існуючих на ринку.

У першому розділі пояснювальної записки розглядається класифікація існуючих систем управління тест-кейсів за п'ятьма основними критеріями: вартістю, місцем розміщення серверної частини системи, швидкодією, відкритістю

програмного коду та додатковим функціоналом (тобто тим котрий не є критично важливим для функціонування систем подібного виду).

Далі в ході дослідного вивчення тематичних форумів було відібрано тринадцять найбільш популярних на даний момент систем управління тест-кейсів. Так як не існує певних рейтингів було вибрані ті системи котрі хоча б раз згадувались на тематичних сайтах. Даний підхід має сенс в ході нашої роботи з огляду на специфічність напряму, проте в інших випадках такий підхід не є вдалим.

Після чого проводиться аналіз цих систем за наведеними вище критеріями, будується порівняльна таблиця та виокремлюються головні та найбільш застосовувані технології та рішення.

За матеріалами досліджень у розділі здійснено висновки.

У другому розділі розробляється специфікація вимог до програмного продукту, призначеного для управління тест кейсами. Задля цього проводится загальний опис у котрому описується перспектива продукту, його характеристика, класи користувачів котрі є в системах даного типу та їх характеристики і середовище функціонування.

Окрему увагу приділено обмеженню проектування і реалізації так як саме від цих обмежень залежить зручність використання системи користувача та подальше її супроводження. Далі наводяться характеристики системи -- такі як зручність використання, підтримуваність і швидкодія.

Також описані вимоги до зовнішніх інтерфейсів де описуються як технічні аспекти (програмні та комунікаційні інтерфейси) так і інтерфейси користувача. Додатково описані вимоги до надійності, безпеки та продуктивності проте їм приділено менше уваги з огляду на те створювана програма все ж не є корпоративним продуктом і створюється однією людиною.

За матеріалами досліджень у розділі здійснено висновки.

Третій розділ присвячений безпосередньо розробленню системи управління тест-кейсів. В ньому висвітлюється розроблення клієнтської та серверної частин системи.

Спочатку описується цільове середовище виконання системи, а саме серверної та клієнтських частин.

Далі описується архітектура системи, а саме клієнт-серверна архітектура, також описується мова розробки.

Потім описується клієнтська частина, тобто інтерфейс користувача – технології, котрі при цьому використовувались та чому були обрані саме вони. Описавши інтерфейси користувача надається мапа сайту та опис усіх сторінок клієнтської частини системи з описом фунціоналу.

Окремо описується робота серверної частини. В описі надається інформація щодо технологій використаних при розробці безпосередньо системної логіки, роботи з базою даних та аутентифікації. Також окремо описана робота кожного контролера системи.

В кінці розділу надається UML діаграма системи та висновок з описом виконаних робіт

У четвертому розділі наведено результати тестування розробленого програмного продукту. Тестування системи складається з двох частин – тестування аутентифікації та тестування процесів управління тест-кейсів.

Тестування аутентифікації винесено окремо з огляду на специфічність її реалізації і тому, що це основний фунціонал, котрий відповідає за збереження та безпеку даних.

Тестування процесів управління подано як процес виконання прогону тестів звичайним користувачем. Прогін тестів розбитий на пункти від створення сьюта до безпосереднього прогону. Кожен пункт описаний окремо та підкріплений зображеннями з поясненнями.

За матеріалами досліджень у розділі здійснено висновки.

У заключній частині дипломної роботи опрацьовано та сформульовано підсумкові висновки за результатами досліджень. В них описані результати роботи по кожному розділу, з’ясовано, чи була досягнена мета роботи та які недоліки все ж має розроблена система.

Наведено список використаних джерел в порядку посилань на них у тексті пояснювальної записки дипломної роботи.

У Додатку А наведено програмний код розробленої системи, в котрому описані усі контролери, репозитарії та класи з логікою роботи. Програмний код написаний мовою програмування Java. Також додані файли з розміткою і описи класів у базі даних.

РОЗДІЛ 1

Характеристика існуючих програм управління тест-кейсів

1.1. Класифікація існуючих програм управління тест-кейсів

Відома та широко використовується певна кількість програмних продуктів, призначених для управління тест-кейсів, але їх не дуже багато. Річ у тім, що ці продукти є досить вузькоспеціалізованими і часто великі компанії створюють свої системи для управління тест-кейсів, проте є і зігільнодоступні продукти. Усі вони мають власні недоліки і переваги. Проаналізувавши їх, можна визначити певні критерії, по котрим їх можна класифікувати.

За вартістю: Зважаючи на те, що подібні програми треба підтримувати в процесі експлуатації, логічним є те, що деякі з них небезкоштовні. Безкоштовні програми, або програми в відкритим програмним кодом, не підтримуються і зазвичай потребують завеликих зусиль задля оптимізації під конкретно свій проект

За розміщення backend частини: Одна з головних частин системи – це база данних, в якій зберігаються тест-кейси. Вона може розміщатись на серверах компанії-розробника продукту, що надає ще одну змогу монетизувати систему. Або надає змогу користувачу самостійно разгорнути систему на своїх серверах.

Відкритість коду: Програмний код може бути як закритий, та і відкритий. Проблема з відкритим програмним кодом є те, що його не підтримують. Великі компанії-розробники систем, навпаки, надають підтримку продукту, проте не зацікавлені в його відкритості.

Швидкодія: Швидкодія є одним з головних і одним з найлегших до реалізації факторів системи, тому що виконання прогонів потребує швидкого перемикання кейсів.

На швидкодію впливає час завантаження сторінки, що є наслідком швидкості з’єднання, котра наш час не є проблемою.

Додатковий функціонал: Це є досить сумнівним критерієм, тому що кожному користувачу потрібен свій функціонал або не потребує функцій, що надає система. Зазвичай користувачі відмовляються від систем з багатим функціоналом, котрим не будуть користуватись.

Підсумувавши вище написане, усі критерії можна звести до наступного списку:

- Вартість
    - Безкоштовні
    - Платні
- Розміщення backend частини
    - Сервера компанії розробника
    - Сервера користувача
- Швидкодія
    - Час, за який зантажится сторінка
- Відкритість коду
    - Відкритий початковий код
    - Закритий початковий код
- Додатковий функціонал
    - Наявний (інтеграції з іншими системами, власна система багтрекінгу, таск менеджер і тд)
    - Відсутній (система надає лише базовий функціонал)

1.2. Аналіз існуючих систем управління тест-кейсів

Використовуючи статистику опиту користувачів подібних програм, можна виділити 13 найбільш популярних систем управління тест-кейсів:

- Test Link
- Test IT
- Zephyr

- qTest
- PractiTest
- TestLodge
- TestRail
- Qase
- Tematoo
- Test Collab
- HP ALM
- Testuff
- XQual

Проаналізуємо їх більш детально за вище описаною класифікацією.

Testlink

Один з небагатьох open-source інструментів управління тестуванням, який доступний на ринку. Це – веб-інструмент з типовими функціями, такими як управління вимогами, створення і супровід тест-сьютів, прогін тестів, відстеження помилок, інтеграція з відомими баг-трекінгові системами [1].

Для відстеження прогресу проекту доступні звіти і діаграми, а додаткові функції включають тегірування ключовими словами, вказівка вимог і журнал подій.

Код проекту часто оновлюється і доповнюється.

Можливості:

- Управління вимогами
- Специфікація – визначення тест-кейсів шляхом угруповання в різні набори тестів
- Призначення виконання тест-сьютів на рівні збірки
- Централізоване управління користувачами і ролями
- Кастомізація настроюються користувачем полів

Використовуючи документацію проекту, проаналізуємо дану програму за основними критеріями.

- Вартість: Безкоштовний
- Розміщення backend частини: Сервери користувача
- Швидкодія: Швидка за рахунок відсутності JavaScript елементів
- Відкритість коду: OpenSource
- Додатковий функціонал: створення діаграм результатів прогону

Test IT

Test IT – TMS, яку створюють тестувальники для тестувальників. Цей інструмент відрізняє продуманий і зрозумілий інтерфейс. Усередині системи можна створювати проекти і вести для кожного структуровану бібліотеку тестових випадків і чек-листів, часто повторювані операції виділяються в загальні кроки. Інструмент гнучкий – в кожному проекті створюються додаткові атрибути, розподіляються ролі і права, що спрощує настройку TMS під процеси вашої компанії. Test IT допомагає керівникам груп тестування рівномірно розподіляти навантаження між тестувальниками і контролювати виконання робіт за допомогою призначених для користувача запитів і звітів [2].

Розробники програми приділяють велику увагу автоматизованого тестування, кожен тестовий випадок в бібліотеці тестів можна інтегрувати з Автотест по API. Правильно налаштована інтеграція з Автотест дозволяє стежити за прогонами і їх результатами прямо з TMS в режимі реального часу. Ви зможете бачити якісь автоматичні тести в процесі виконання, аналізувати їх результати і переглядати вихідний код прямо з Test IT.

Додаток активно розвивається, в найближчому майбутньому заявлено багато корисних фіч.

Можливості:

- Управління тест-планами, тест-кейсами і чек-листами
- Загальні кроки для повторюваних дій
- Автоматичний розподіл тестів на QA інженерів
- Інтеграція автоматичних тестів за допомогою API

- Аналітика як по автоматичним, так і по ручним тестів
- Рольова модель і персоналізація
- Гейміфікація
- Двостороння інтеграція з JIRA
- Імпорт з інших систем управління тестуванням

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Швидка за рахунок хорошої оптимізації
- Відкритість коду: Закрита
- Додатковий функціонал: Наявний (описаний вище)

Zephyr

Zephyr – це плагін для всім відомої JIRA, який інтегрує тестування в проектний цикл, дозволяючи вам відстежувати якість програмного забезпечення та приймати рішення в стилі go/no-go. Тест-кейси можуть створюватися, виконуватися і трекаться так само, як і будь-які інші завдання в JIRA. Для більш оптимальної фіксації процесу тестування є інтеграція з інструментами управління якістю, автоматизації, безперервної інтеграції та аналітики [3].

Крім того, у продукту швидко відповідає тих підтримка.

Можливості:

- Посилання на user stories, завдання, вимоги, дефекти
- Конфігурації деплоя: в хмарі, на сервері, в дата-центрі
- Розширена інформація на дашборда аналітики і DevOps
- Інтеграція з JIRA, Confluence, Selenium, Jenkins і Bamboo

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Повільна. Багато користувачів відзначають
- перевантаженість системи
- Відкритість коду: Закрита
- Додатковий функціонал: Відсутній

qTest

Інструмент корисний не тільки тестувальникам, але і всій команді. Інтерфейс qTest нативної зрозумілий, мануали прості в освоєнні. Це дозволяє швидко і ефективно створювати, організовувати і управляти тест кейсами [4].

Розробники нескромно заявляють, що їх інструмент управління тестами - №1.

Згідно з аналізом ринку, qTest є одним з найбільш швидкозростаючих рішень для управління тестуванням серед команд Agile Development.

Можливості:

- планування тестів
- Створення та управління вимогами
- Інтуїтивний drag-n-drop інтерфейс
- Комплексна матриця трасуванню
- Наочні звіти з докладними графіками
- Інтеграція зі сторонніми інструментами баг-трекінгу
- Детальний контроль доступу користувачів
- Хмарний інструмент інтеграції з JIRA

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Повільна. Багато користувачів відзначають перевантаженість системи

- Відкритість коду: Закрита
- Додатковий функціонал: Наявний (описаний вище)

PractiTest

PractiTest – це комплексний засіб управління тестами. Він забезпечує повну наочність процесу тестування і більш глибоке розуміння результатів тестування. Цей інструмент допоможе організувати тест-сьюти відповідно до ваших циклами і спринті. Тестові набори можна формувати за різними критеріями, таким як компоненти, версії або типи. Тул заточений на Agile тестування, регресійні тестування, тестування мікросервісов і DevOps [5].

А в тих підтримку працюють навчені QA співробітники, які можуть швидко зрозуміти вашу проблему.

Можливості:

- Легке додавання тестів нових фіч в регресійне тестування
- Угруповання тестів на основі мікросервісов, які вони охоплюють, навіть крос-сервісні
- Різна відображення інформації для різних груп користувачів
- Інтеграція з JIRA, Redmine, Jenkins, GitLab і Slack

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Наявний (описаний вище)

TestLodge

У ньому є найнеобхідніший мінімум, який потрібен для управління тестуванням: тест-плани, вимоги, тест-кейси та сьюти, і тестові прогони. Тул можна

інтегрувати з існуючими баг-трекера, що дозволяє автоматично створювати звіти про дефекти і заводити завдання [6].

Можливості:

- Базовий набір створення, редагування тест-плану і тест-сьютів
- нативний інтерфейс
- Інтеграція з JIRA, Redmine, Trello, Asana, GitHub і YouTrack

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Інтеграції

TestRail

Це програмне забезпечення зручно як для команд QA, так і для розробки. План тестування можна вибудувати як за сценарієм гнучкою методології, так і для більш традиційного підходу. Інструмент дозволяє отримати уявлення про хід тестування в реальному часі [7].

Можливості:

- Відстеження стану і результатів окремого тесту
- Порівняння результатів декількох тестів, конфігурацій і етапів
- Відстеження робочого навантаження команди для коригування завдань і ресурсів
- Розгорнуті звіти і метрики
- Широкі можливості налаштування, хмарні або локальні варіанти установки
- Інтеграція з JIRA, Redmine, YouTrack, GitHub, Jenkins, Selenium і Visual Studio

- Зручний REST API

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Інтеграції

Qase

Qase – це інструмент, який можна нормально використовувати в роботі. Хмарна TMS, яка допоможе вашій команді підвищити продуктивність і організувати зручний флоу тестування програмного забезпечення [8].

Можливості:

- Тестовий репозиторій: вибудовування тестів в логічні групи
- Складання кроків для кейсів, установка пріоритету і серйозності
- Запуск тестових прогонів з трекингом часу по кожному тест-кейсу
- Зберігання документації по проекту
- Автоматичне заклад дефектів в інтегровані трекери
- Інтеграція з JIRA, Redmine, YouTrack і Slack
- Об'єднання результатів Автотест з REST API

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Інтеграції

Tematoo

Ця хмарна TMS від TestLink не так розрекламована, але не поступається за своєю функціональністю більш дорогим аналогам. Інструмент надає лаконічну інфраструктуру, дозволяючи швидко приступити до тестування продукту. А безкоштовний план підтримки невеликих команд дозволяє проводити пілотні реалізації проекту безкоштовно. Tematoo може бути інтегрований з багатьма баг-трекера, навіть хмарними [9].

Можливості:

- Формування тест-сьютів по білд і типам
- Опис тест-кейса по кроках, з можливістю прикріпити скріншот
- Статус окремих тестів, наборів і загальний статус збірки
- Аналітичні звіти та загальний метрики по тест-плану
- Для платного плану: свій баг-трекер

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Безкоштовна для малих команд
- Розміщення backend частини: Хмара
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: лише на платній версії

Test Collab

Це – сервіс з корисним набором функцій, який необхідний для швидкого старту використання інструменту. Плюс, трохи приємних бонусів: зручний інтерфейс, кастомизация фільтрів і полів, time-трекер для кожного члена команди, і внутрішньосистемне спілкування [10].

Test Collab можна налаштувати в хмарі або в вашій інфраструктурі, тут все досить гнучко.

Можливості:

- Угруповання тест-кейсів по етапах або білд
- управління вимогами
- перевикористання тестів
- Налаштування спринтів
- Звіти за результатами тестів
- Коментування тест-кейсів
- Інтеграція з JIRA, Redmine, Bugzilla, Asana, Trello, YouTrack, GitHub

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Безкоштовна для малих команд
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: відсутній

HP ALM

HP ALM – довгожитель серед систем управління життєвим циклом продукту і його тестуванні в тому числі. Інструмент дозволяє здійснювати планування, створення, тестування і контроль на всіх етапах розробки. Складний в первісному освоєнні, але незамінний для компанії великої руки, де особлива увага приділяється деталям виробництва [11].

Саме тому, що продукт вже обкатаний, в інтернеті є безліч відеогайдов по налаштуванню і використанню.

Можливості:

- Загальний доступ до бібліотек вимог і ресурсів
- Докладні відомості про код, тестуванні, управлінні ризиками та їх оцінкою, а також про відповідність вимогам
- Швидкий доступ до показників, наприклад, до даних про не усунених

дефектах

- Швидка настройка лабораторного середовища для усунення помилок конфігурації в середовищах Agile
- Створення вимог і відстеження їх виконання на всіх етапах життєвого циклу
- Аналітика на будь-який смак і колір: гнучко настроюються звіти
- Інтеграція з 50+ інструментами

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: відсутній

Testuff

Testuff – це потужна веб-платформа управління тестуванням, яка дозволяє легко проектувати, виконувати і управляти необмеженою кількістю тестів. Тул можна налаштувати під будь-який формат тестування: від популярної гнучкою методології і TDD, до White-Box або Black-Box; Top-Down або Bottom-Up [12].

Тут є і оптимізована черга управління завданнями для кожного тестувальника із застосуванням тайм-менеджменту, і організація тестів по проектам, гілках коду і типовим тестовим набором. Крім бонусу експорту в HTML / Excel, є маленькі плюшки у вигляді вбудованого тула створення скріншотів і відео з відображенням покажчика і натискає клавішу.

Можливості:

- Два доступних клієнта - Web і Desktop
- Планування циклу тестування з використанням декількох тестових оточень

- Вбудований захоплення екрану у вигляді скріншоту або відео
- Підключення результатів автоматизованих тестів по API
- Інтеграція з JIRA, Trello, Redmine, Bugzilla, YouTrack, Selenium, GitHub, Visual Studio

Використовуючи документацію проекту, проаналізуємо програму за основними критеріями.

- Вартість: Платна
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Наявний (описаний вище)

XQual

XStudio від XQual ощасливить просунутого тестувальника, який хоче поставити під свій продукт максимально велику кількость випробувань. За допомогою цього інструменту можна управляти не тільки своїми релізами, вимогами, ризиками, специфікаціями, тестами, кампаніями і багами. Він може 99бути інтегрований з усіма платформами безперервної інтеграції і може виконувати будь-який вид тестування [13].

Можливості:

- Розширена настройка кроків тест кейса
- перевикористання тестів
- матриця трасуванню
- Налаштування тестовій лабораторії: hardware, software, тестове оточення
- Просунуте логирование дій (навіть віддалених тестів)
- настроюється аналітика
- Вбудований захоплення екрану
- Інтеграція з JIRA, Redmine, MySQL, Oracle, Apache, Skype
- Інтеграція з 5 різними інтерфейсами для ручного тестування

- Інтеграція з майже 70 туламі автоматизації: Selenium, QTP / UFT, JMeter, Ranorex, TestComplete, JUnit, NUnit, TestPartner, Sahi, NeoLoad, QF-Test, RobotFramework, Sikuli, SoapUi, Squish, TestNg, TestOptimal і багато Інших.

Використовуючи наявну інформацію, проаналізуємо наведені системи управління тест-кейсів за такими основними критеріями:

- Вартість: Безкоштовна для малих команд
- Розміщення backend частини: Сервери користувача
- Швидкодія: Висока
- Відкритість коду: Закрита
- Додатковий функціонал: Наявний (описаний вище)

Зазначені вище характеристики зведені до таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця систем управління тест-кейсів

	Вартість	Розміщення backend частини	Швидкодія	Відкритість коду	Додатковий функціонал
Test Link	Безкоштовна	На сервері користувача	Висока	Відкритий	Наявний
Test IT	Платна	На сервері користувача	Висока	Закритий	Відсутній
Zephyr	Платна	На сервері користувача	Низька	Закритий	Відсутній
PractiTest	Платна	На сервері користувача	Висока	Закритий	Наявний
qTest	Платна	На сервері користувача	Висока	Закритий	Наявний
TestLodge	Платна	На сервері користувача	Висока	Закритий	Наявний
TestRail	Платна	На сервері користувача	Висока	Закритий	Наявний

Test Collab	Платна	На сервері користувача	Висока	Закритий	Відсутній
Qase	Безкоштовна для малих команд	На сервері користувача	Висока	Закритий	Наявний
Tematoo	Безкоштовна для малих команд	На сервері користувача	Висока	Закритий	Наявний
HP ALM	Платна	На сервері користувача	Висока	Закритий	Наявний
Testuff	Платна	На сервері користувача	Висока	Закритий	Наявний
XQual	Безкоштовна для малих команд	На сервері користувача	Висока	Закритий	Наявний

Висновки

З викладеного вище можна зробити висновки, що більшість наведених систем управління тест-кейсів є платними, мають закритий програмний код, високу швидкодію та розташовуються на серверах користувача. Тобто система, що розробляється, повинна відповідати технічним характеристикам і матиме перевагу у вигляді відкритості програмного коду.

Як можна бачити з таблиці, більшість програм мають велику кількість додаткового функціоналу. Найбільш популярні системи відмовляються від власного додаткового функціоналу приділяючи більше уваги інтеграціям, що дозволить користувачеві більш гнучко налаштувати систему. Таку можливість було б слушно додати до власної системи, проте це виходить за межі даної дипломної роботи.

РОЗДІЛ 2

Специфікація вимог до програмного продукту

«Система управління тест-кейсів»

Вступ

Призначення, мета

Система управління тест-кейсів призначена для створення, збереження та об’єднання у сьюти тест-кейсів задля їх подальшого прогону.

Метою роботи є розроблення системи управління тест-кейсів з відкритим системним кодом

Межами програмного продукту є архітектура REST та браузерне середовище використовування.

Посилання

https://spring.io/projects/spring-boot

https://en.wikipedia.org/wiki/Client–server_model

https://ru.wikipedia.org/wiki/Model-View-Controller

https://en.wikipedia.org/wiki/Test_case

https://en.wikipedia.org/wiki/Test_management_tool

2.1. Загальний опис

2.1.1. Перспективи продукту

Дана система є продуктом з відкритим програмним кодом, котрий реалізує базовий функціонал аналогічних систем, що дозволяє користувачу самостійно конфігурувати системи під свої потреби. Система розробляється як самодостатній продукт, котрий не потребує ніяких додаткових компонентів для виконання базових функцій.

Програма буде потрібна користувачам, котрі займаються тестуванням програмних продуктів. Тобто тестувальникам, автоматизаторам, розробникам

програмного забезпечення, менеджерам проекту та іншим. За допомогою цієї системи користувач може оцінити стан покриття тестами програмного продукту, оцінити стан проходження регресійного тестування та використовувати тест-сьюти як проектну документацію

Даний продукт може розвиватись далі як набір модулів з додаванням інтеграцій, які дозволять користувачу підключати необхідні йому функції не створюючи їх самостійно, або навпаки як продукт з відкритим системним кодом котрий дозволить користувачам самостійно створювати необхідні компоненти. Напрямок на систему з відкритим системним кодом є більш пріорітетним з огляду на ширші можливості котрі надає ций підхід. Також це зумовлено низькою кількістю Open Source проектів на даному ринку.

2.1.2. Характеристика продукту

Програма дозволяє:

- Створювати, редагувати і зберігати тест-кейси
- Об’єднувати тест-кейси у сьюти
- Виконувати прогони тестів

У зв’язку з тим, що система має кліент-серверну архітектуру, графічний інтерфейс у неї представлений у вигляді веб-сайту, тому вона націлена на найбільш популярні сучасні браузери – Google Chrome, Mozilla Firefox, Safari. Звісно, до системи можна звернутись і через інші браузери, проте коректність розмітки може не відповідати шаблонам.

З погляду на те, що BackEnd частина розгортається на сервері користувача, необхідно, щоб сервер мав встановлену JVM. З боку технічних характеристик – серверу достатньо мінімальної конфігурації, бо система не ресурсоємка.

Необхідна швидкість підключення до мережі інтернет не менше 0,5 Мб/сек і не більше 1 МВ/сек. Це викликане тим, що підключення через мережу інтернет є ключовою функцією програми і вона повинна бути достатньою, щоб з сервера можна біло отримати чи передати данні. Також графічний інтерфейс не повинен бути перевантаженим графічними елементами, що може сповільнити завантаження сторінок. Задля пришвидшення роботи графічного є сенс використовувати

шаблонізатори замість окремої реалізації клієнтської частини за допомогою JavaScript.

2.1.3. Класи користувачів та їх характеристики

Користувачів даної програми можна поділити на два класи

- Досвідчені користувачі
- Недосвідчені користувачі

Досвідчений користувач – це користувач, котрий вже користувався подібними інструментами, знає, як створювати тест кейси і для чого вони взагалі потрібні. Тобто це може бути досвідчений інженер з тестування програмного забезпечення, менеджер проекту що підтримує документацію проекту або розробник котрий вивчає заплановану поведінку та логіку продукту продукту. Головними ознаками досвідченого користувача є знання термінології, досвід роботи з подібними системами і досвід створення і підтримки такого роду документації.

Недосвідчений користувач – це користувач, котрий стикається з подібним продуктом уперше та/або не розуміє принцип його роботи. Такими користувачами можуть бути початківці в сфері тестування програмного забезпечення, студенти котрі вивчають теорію тестування або розробники котрі не працюють з технічною документацією такого роду. Ознаками недосвідченого користувача є повна відсутність досвіду роботи з системами управління тест-кейсів та мінімальними знаннями тестування програмного забезпечення.

У зв’язку з тим, що програма включає в себе можливість створення, видалення та редагування тест-кейсів, котрі можуть виступати технічною документацією для новачків, необхідно обмежити права користувача на редагування тест-кейсів і дозволити це лише тим, хто їх створював чи тим, хто володіє достатніми правами. Також тест-кейси є одним з видів проектної документації тому на них може поширюватись правила про не розголошення.

Інтерфейс взаємодії робиться максимально зрозумілим, а отже не потребує особливих специфікацій. У разі, якщо користувачу все ж потрібна довідка, він може знайти її у відповідній вкладці меню. Довідку можна реалізувати як текстовий опис дій і відкликів системи, так і в інтерактивній формі у вигляді підказок розкиданих

по всьому сайту. Такий підхід більш зрозумілий для користувача, але також і більш трудомісткий тому він не буде реалізований у ході роботи.

Щоб досвідчені користувачі швидше звикли до нового інструменту, у графічному інтерфейсі використовуються елементи найбільш популярних інструментів (тобто тих інструментів, якими можливо вже користувались). Такими елементами є спискове представлення тест кейсів, блочне представлення списку тест-сьютів, відображення статусу прогону тесту за допомогою шкали прогресу та трьох-секційне представлення сторінки тест-сьюта. Такий підхід, у різних варіаціях, використовується всіма системами котрі були досліджені у першій частині, тому доцільно використати його і в ході даної роботи.

Підсумувавши, можна сказати, що між досвідченим та недосвідченим користувачем не полягає великої різниці, тому що інструмент доволі простий і перейти з другої категорії у першу доволі просто. З цього можна зробити висновок, що не має сенсу створювати дві версії продукту чи надавати якісь складні документації, достатньо невеличкого опису функціоналу. Проте задля успішного використання системи необхідні хоча б мінімальні знання теорії тестування, розуміння для чого необхідні тест-кейси, сьюти та прогони і чому саме їх необхідно створювати і зберігати. Хоча відсутність цих знань не відобразиться на технічних характеристиках системи, вона може вплинути на ефективність її використання.

2.1.4. Середовище функціонування

Frontend

Для роботи з графічною частиною достатньо будь-якого браузера, тому що у системі не буде використовуватись JavaScript, котрий має свої вимоги до виконання. Проте рекомендується використовувати актуальні версії сучасних десктопних браузерів, тому що розробка і оптимізація буде створюватись саме під них. Робити десктопну версію системи не має сенсу так як це зменшить її мобільність так як кожному користувачеві доведеться встановлювати її та оновлювати, а веб версії необхідний лише браузер. Хоча до сайту можна отримати доступ з будь-якого девайса котрий має браузер оріентація буде все ж на компьютерні версії.

Backend

У зв'язку з тим, що система не ресурсоємка, серверу буде достатньо 100МБ фізичної пам'яті для бази даних та стільки ж для безпосередньо системи. Оперативної пам'яті буде достатньо 4 ГБ. Також для роботи системи необхідна встановлена JVM і база даних PostgreSQL. Такі умови викликані певними спрощеннями котрі були використані для зосередженні на меті роботи. На практиці для подальшого поширення продукту від користувача не потрібно буде нічого вимагати, так як усі додаткові елементи будуть іти у встановочному пакеті системи. У межах даної роботи можна знехтувати певними інструментами і встановити необхідні компоненти вручну.

2.1.5. Обмеження проектування і реалізація

Програма пишеться мовою JAVA, тому що ця мова добре підходить для написання кліент-серверних програм. JAVA була обрана мовою проекту ще тому що дозволяє запускати створену програму на будь-якій платформі де може бути встановлена JVM. Також вибір цієї мови дозволить використати Spring Boot котрий спрощує роботу з базами даних і самостійно обробляє багатопоточні запити до системи. І наостанок можна використовувати шаблонізатори для спрощення роботи з графічним інтерфейсом системи. Тому використання Java спрощує нам роботу з базами даних, багатопоточністю і з графічним інтерфейсом дозволяючи уникнути роботи з JavaScript і створення окремого додатку для клієнтської частини системи. Унаслідок цього інструменти і платформи, що будуть використовуватись, обмежуються даною мовою. Також програма повинна відповідати певним вимогам котрі наведені нижче.

2.1.5.1. Апаратні обмеження

Об'єм оперативної пам'яті, необхідний для комфортної роботи сервера – 4 GB. Це мінімальна конфігурація серверів на даний час.

Рекомендований об'єм файлової пам'яті – 250 MB. Програма не важка і не потребує великого об'єму пам'яті. Всі дані, котрі програма зберігає, знаходяться в окремій базі даних, котра не займе більше 100МБ. Такі потреби зумовлені тим що

дані в базі текстові і не займають багато місця, а сама система не важить більше 100МБ за рахунок відсутності важки компонентів.

Необхідна швидкість підключення до мережі інтернет не менше 0,5 Мб/сек і не більше 1 МВ/сек. Це викликано тим, що підключення через мережу інтернет є ключовою функцією програми і вона повинна бути достатньою, щоб до сервера можна було звернутись і отримати результат.

Браузер, що використовується для роботи з графічною частиною повинен відповідати сучасним вимогам до браузерів, проте система до цього не вибаглива.

2.1.5.2. Специфічні технології

Програма потребує встановленої на сервері програмного компоненту JAVA версії 8 чи більше. Це необхідно, тому що Spring Boot працює лише з версією JAVA 8 чи більше. А також це надає додаткові можливості при ьезпосередньій розробці системи.

Для збереження даних програма використовує базу даних PostreSql котру необхідно встановити окремо. Це для спрощення виконання дипломної роботи і зосередженні на меті. На практиці користувачу необхідно буде просто встановити інсталяційний пакет, але в рамках дипломної роботи це не має сенсу і не є метою роботи.

2.1.5.3. Протоколи комунікацій

У зв’язку з тим, що система кліент-серверна і кліентом є веб-сайт, то використовуватись будуть протоколи передачі гіпертексту (https/http). Протоколом FTP можна знехтувати з огляду на те що ніякі додаткові файли у тест кейсах не зберігаються. Також так як більшість браузерів «не довіряють» з’єднанню типу http (зважаючи на його не захищеність) система повинна орієнтуватись саме на https протокол. Підтримка http додана для зворотньої сумісності зі старими браузерами та для корпоративних підсистем де http може використовуватись локально.

2.1.5.4. Мовні вимоги

Для написання і подальшого підтримання необхідно використовувати JAVA версії 8 або більше. Це пов’язано з тим, що для роботи зі Spring Boot необхідна мінімум JAVA 8.

2.1.5.5. Проектні програмні стандарти

Для полегшення модернізації та підтримки продукту програма побудована за концепцією MVC.

Моде́ль–вигляд–контро́лер (*Model-view-controller*, **MVC**) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

У нашому випадку за вигляд і контролер відповідає клієнтська частина системи, а обробкою моделі займається серверна частина.

2.1.5.6. Документація користувача

Керівництво з використання

Повинно включати в себе опис процесу розгортання системи, встановлення та підключення бази даних. Керівництво з використання може бути подане як окремий файл з інструкціям, так і як пояснення на сайті розробника системи. Дане керівництво слід зробити винесеним окремо з огляду на те що у користувача можуть з’явитись проблеми з встановленням системи, тобто до її фактичного використання. Історично склалося що для цього використовується файл з назвою README в котрому коротко описана ключова інформація та посилання на ресурси де можна отримати більш повну довідку.

on-line допомога

Опціональна вимога, котра може бути представлена у наданні контактів розробника (на даний момент) або окремого відділу, котрий займається розробкою та оптимізацією продукту. Онлайн допомога є одним з кращих варіантів підтримки користувача з боку розробника продукту так як має більший охват проблем, дозволяє швидше локалізувати проблему та знайти індивідуальне рішення котре найкраще підійде саме цьому користувачу. Створення служби online-допомоги є окремою і досить великою частиною проекту яка вимагає набір сепцифічних технологій та працівників певної кваліфікації.

Інструкція

Повинна включати в себе приклади створення тест-кейсів, сьютів та прогонів.

Дана документація може бути представлена як веб-сторінка чи окреме вікно програми, в котрій наведено текст усіх трьох розділів. Створювати окремі вікна для кожного розділу немає сенсу.

Блок створення тест-сьюту повинен включати в себе:

- Зображення кнопки чи послідовності дій котрі відкриються дану сторінку
- Зображення чи опис процесу створення тест-сьюту
- Зображення чи опис помилок котрі можуть виникнути при створенні тест-сьюту та методи їх уникнення

Блок створення тест-кейсу повинен включати в себе:

- Зображення кнопки чи послідовності дій котрі відкриються дану сторінку
- Зображення чи опис процесу створення тест-кейсу
- Зображення чи опис помилок котрі можуть виникнути при створенні тест-кейсу та методи їх уникнення

Блок створення прогону тесту повинен включати в себе:

- Зображення кнопки чи послідовності дій котрі відкриються дану сторінку
- Зображення чи опис процесу створення прогону тестів

- Зображення чи опис помилок котрі можуть виникнути при створенні прогону тестів та методи їх уникнення

Блок виконання прогону тесту повинен включати в себе:

- Зображення кнопки чи послідовності дій котрі відкриються дану сторінку
- Зображення чи опис процесу виконання прогону тестів

2.2. Характеристики системи

2.2.1. Швидкодія

Опис і пріоритет

Швидкодія може бути виражена як час від запиту до сторінки до моменту, коли з нею можна буде працювати. У зв'язку з тим, що даний параметр дуже залежить від швидкості з'єднання оцінювати його слід у порівнянні з іншими системами аналогічного призначення. Дане порівняння необхідно проводити на однакових конфігураціях ПК, з однаковою швидкістю з'єднання і однаковими конфігураціями серверів. Створювана система повинна реагувати на рівні систем конкурентів або швидше.

Послідовності дій/відгуків

Головною умовою дій і відгуків є повернення результату менше ніж за секунду. В окремих випадках послідовності дій і відгуків наступні:

- При довгій обробці запиту відображати анімацію завантаження даних
- Виконання запитів збереження у фонових процесах дозволяючи користувачеві працювати з системою не чекаючи завершення запису даних в базу

Функціональні вимоги

Для збільшення швидкодії програми необхідно:

- Зменшити об'єм функціоналу, котрий обробляється кліентом (REQ-1)
- Не використовувати важкі запити до бази даних (REQ-2)
- За неможливості уникнення обробки важких запитів винети їх у фонові процеси (REQ-3)

2.2.2. Зручність використання

Опис і пріоритет

Зручність використання важко описати технічно, тому що ніяку метрику сюди додати неможливо. Тому зручність використання оприділяти необхідно за допомоги контрольної групи або колег.

Дана характеристика має найвищий пріорітет, оскільки головним критерієм при виборі TMS є зручність її використання.

Послідовності дій/відгуків

Головними умовами послідовності дій і відгуків системи з боку зручності є те що система завжди повертає якесь значення на взаємодію з інтерактивним елементом. В окремих випадках послідовності дій і відгуків наступні:

- Клік по полю введення робить дане поле активним (підсвічує поле та встановлює курсор на початку рядку)
- Клік по кнопці збереження відображає попап з повідомленням що операція пройшла успішно
- Клік по кнопці видалення оновлює сторінку і забирає з неї видалений елемент
- Клік по кнопці створення оновлює сторінку і додає на неї новий елемент
- Система не підвисає при взаємодії з нею (під «підвисанням» слід розуміти відсутність відгуків системи на дії користувача впродовж тривалого часу)

Функціональні вимоги

Для збільшення зручності використання слід зробити інтерактивні елементи очевидними до використання (у вигляді кнопок, свічів та інше) (REQ-1)

2.2.3. Підтримуваність та розвиток

Опис і пріоритет

Цю характеристику можна описати як здатність продукту розвиватись (додавати новий функціонал) та виправляти дефекти, котрі дійшли до користувачів.

Ця характеристика має високий пріоритет, тому що будь-яка програма проходить декілька циклів розробки, що дозволяє їй стати кращою та бути орієнтованою на сучасний напрям розвитку технологій.

Функціональні вимоги

- Продукт повинен відповідати сучасним моделям побудови ПЗ
- Продукт повинен відповідати єдиному паттерну проектування

2.3. Вимоги зовнішніх інтерфейсів

2.3.1. Користувацькі інтерфейси

Користувацький інтерфейс взаємодії з базою даних

Головною частиною TMS є змога зберігати тест кейси у базі даних.

Список тест-кейсів підвантажується автоматично при відкритті сьюта.

Для збереження тест-кейсу необхідно заповнити всі поля форми та натиснути кнопку «Save» вікні.

Для відкриття тест-кейсу необхідно натиснути на його назву у списку.

Для видалення необхідно натиснути по кнопці із зображення «Кошика».

2.3.2. Програмні інтерфейси

Програмними інтерфейсами у системі є взаємодія з базой даних, що відбувається за допомоги відповідного драйвера, проте а ні розробник, а ні користувач не взаємодіють з ним напряму, так як цю проблему вирішує Hibernate і все це залишається на відповідальності фреймворку.

2.3.3. Комунікаційні інтерфейси

Програма підтримує наступні комунікаційні протоколи:

- **HTTP** — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від **H**yper **T**ext **T**ransfer **P**rotocol, протокол передачі гіпертекстових документів.

HTTP належить до протоколів моделі OSI сьомого прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб-сторінками (зображення і застосунки) та і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма — веб-браузер — здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування символів веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

- **HTTPS** — схема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет. Використання https: URL вказує, що протокол HTTP має використовуватися, але з іншим портом за замовчуванням і додатковим шаром шифрування/автентифікації між HTTP і TCP.

Ця схема була винайдена у компанії Netscape Communications Corporation для забезпечення автентифікації та шифрування комунікацій і широко використовується в Інтернеті у програмному забезпеченні, в якому важлива безпека комунікацій, наприклад, у платіжних системах та корпоративних логінах.

2.4. Інші нефункціональні вимоги

2.4.1. Вимоги продуктивності

Програма повинна витрачати на відправку і отримання даних небільше одної секунди при стандартній швидкості інтрнету.

Ці вимоги базуються на аналогічних характеристиках програм подібного призначення. Враховуючи те, що продукт, який розроблюється, повинен бути більш досконалим, вимоги продуктивності до нього більші, ніж до існуючих програм.

Програма не виконує паралельні обчислення, а отже вимоги продуктивності стосуються лише одного потоку виконання.

2.4.2. Вимоги надійності

Вимоги до точності збереження даних

Надійність системи можна описати як точність збереження даних, котрі записує користувач. Так як тест-кейси можуть бути об’ємними, слід підібрати правильний тип даних, котрий збереже текст повністю. Також слід використовувати метод і таблицю шифрування котрий зберігає і підтримує усі символи. Це

пояснюється тим що користувачі можуть зберігати у тест кейсах ключі та паролі для тестових систем, тестові дані та інше, а отже вони повинні зберігатись і відображатись без спотворень.

2.4.3. Вимоги безпеки

Так як тест кейси можуть бути частиною документації, слід шифрувати базу даних (що робиться автоматично) та шифрувати паролі користувачів, що необхідно зробити при розробці за допомогою компоненту Spring Security.

Також на етапі тестування системи слід приділити увагу можливості взаємодії з системою не авторизованого користувача. Слід перевірити наступні варіанта звернення до системи неавторизованого користувача:

- Неавторизований GET-запит. Запит на отримання даних. Система повинна відображати помилку або перекидати користувача на сторінку логіну в разі неавторизованої спроби отримання даних
- Авторизований GET-запит. Запит на отримання чужих даних. Система повинна відображати помилку в разі спроби користувача отримати доступ до даних іншого користувача
- Неавторизований POST-запит. Спроба змінити данні неавторизованим користувачем. Система повинна відображати помилку на такі запити
- Авторизований POST -запит. Спроба змінити данні іншого. Система повинна відображати помилку в разі спроби користувача змінити дані іншого користувача

Також слід приділити увагу шифруванню. В створювані системі відсутні персональні данні котрі необхідно шифрувати окрім паролю користувача котрий не повинен отримуватись навіть при прямому зверненню до бази даних. Шифрування створюваних користувачем тест кейсів не є бажаним оскільки при великих об’ємах даних це призведе до суттевого сповільнення системи.

Доступ до самої бази даних повинен бути захищений паролем. Пароль необхідно зробити чисельно-символьнимі (так як це дозволяє система) використати не менш 20 символів.

2.4.4. Атрибути якості програмного продукту

Атрибутами якості продукту є якості, котрі повинна мати програма задля успішного вдоволення потреб користувача та конкурентоспроможності.

Проаналізувавши найпопулярніші програми подібного типу (наведені вище), можна привести наступні атрибути якості продукту:

- Супроводжуваність
- Тестопридатність
- Зручність використання
- Функціональність
- Коректність
- Наявність інтеграцій чи зовнішніх взаємодій
- Можливість переносу даних з однієї TMS в іншу

Супроводжуваність

Супроводжуваність – це показник, котрий ілюструє, наскільки зручно для розробників виконувати супроводження програмного забезпечення (ПЗ), що є процесом покращення, оптимізації і виправлення дефектів програмного забезпечення після здачі в експлуатацію. Супроводження ПЗ – це одна з фаз життєвого циклу ПЗ, котра дозволяє виправляти дефекти, покращувати якість продукту та покращувати зручність використання, а отже є обов’язковим атрибутом.

Супроводжуваність є обов’язковим атрибутом якості продукту не лише подібного виду, а й інших програм, не зважаючи на їх призначення.

Тестопридатність

Тестопридатність – це здатність програми до тестових перевірок після чого її можна повернути до початкового стану. В даній програмі основні вимоги до тестопридатності, які ставляться до бази даних, можна навести наступні вимоги:

- Очищення збережених запитів
- Закриття допуску до видозмінення структури бази даних

Простіше кажучи – під час тестування тестувальник повинен мати змогу видаляти дані, котрі він ввів та не мати доступу до змінення структури бази даних.

Зручність використання

Зручність використання є одним з головних атрибутів програмного забезпечення. Під зручністю розуміється те, наскільки користувачу приємно працювати з програмою, як програма реагує на ті чи інші дії, скільки дій необхідно виконати задля того, щоб почати роботу та отримати результат.

Також під зручністю можна розуміти те, наскільки швидко новий користувач ознайомиться з програмою та зрозуміє, як нею користуватись. Також до зручності відносять те наскільки користувачу приємно користуватись програмою. Це є досить важливим атрибутом оскільки користувач проводитеме с цією системою значну частину робочого часу.

Щоб збільшити зручність використання програми, можна частково повторити функціонал подібних програм, тому що це спростить перехід від одного продукту до іншого або покращувати її, прислухаючись до думки користувачів і покращуючи продукт від однієї здачі в експлуатацію до іншої. Другий варіант більш складний та довгий, проте дає можливість зробити продукт максимально зручним.

Функціональність

Функціональність – це підтримання продуктом функцій, котрі наявні в інших програмах подібного типу. Не обов’язково повинні підтримуватись всі функції достатньо підтримувати основні ключові функції. Це атрибут є дуже важливим, тому що визначає конкурентоспроможність програми та доцільність її використання взагалі. В нашій програмі головними функціями є збереження і відображення тест-кейсів, збереження тест-кейсів у тест-сьюти та прогін тестів по тест-сьютам

Коректність

Коректністю є точність виконання задачі та відображення результату. В даній програмі вимоги коректності відносяться до бази даних (збереження та завантаження даних) та до клієнтської частини (коректність відображення і передачі даних на збереження у БД)

Вимоги до бази даних по збереженню даних є розподілення даних по відповідним стовпчикам у базі даних коректність збереження даних, збереження

форматування (у випадку роботи з текстом) та підтримання безпеки типів даних у стовпцях (система не повинна зберігати в одних і тих же стовпцях різні типи даних).

Вимоги до бази даних по завантаженню даних є правильне заповнення полів даними з таблиці. У зв'язку з тим, що вимоги по збереженню та вимоги до завантаження є високими, модифікація даних покличе за собою виконання дій, котрі не плануються користувачем.

Також до вимог корректносі слід віднести коректність у збереженні даних при модифікації бази даних. У ході розробки та покращенні системи часто буде змінюватись і сама структура база даних, структура таблиць та взаємозв'язків. З боку програмного забезпечення це реалізується за допомогою міграцій котрі при зміні базі даних корректно переносять данні по новим таблицям та заповнюють дефолтними значеннями порожні значення в уже створених рядках.

2.4.5. Вимоги до середовища розробки (IDE)

Вимоги до середовища розробки не є строгими, так як кожен розробник обирає її під себе, тому даний розділ є по більшій частині рекомендаціями котрими розробник може як знехтувати так і взяти до увагу. ПРитримання іх рекомендується оскільки вони підвищують ефективність розробки.

Середовище розробки повинно мати вбудовану систему збірки Maven що дозволить динамічно додавати бібліотеки та модулі до проекту та спростить збірку фінального системного файлу. Також допускається використання системи збірки Gradle що є аналогом системи Maven. Вибір системи збірки покладається на розробника.

Середовище розробки повинно мати вбудовані компілятор та інтерпретатор, що також пришвидшить час розробки. Також буде зручним статичний аналізатор коду котрий дозволить знаходити синтаксичні помилки до запуску системи. Це дозволить не лише зекономити час відладки, але й виключить цілий ряд помилок котрі виникають при роботі з кодом.

Також зручним буде можливість підключення плагінів, котрі спрощують роботу з різним синтаксисом та форматування тексту, з яким при розробці стикнеться розробник. Прикладом таких плагінів є форматери HTML-коду котрі

автоматично закривають тееги, дозволяють зменшувати розмір блоків коду та мають автодоповнення тегів. За замовчування у сучасних IDE подібні плагіни використовуються але лише для для одної-двох мов програмування.

Висновок

Використовуючи аналіз, проведений у першому розділі пояснювальної записки, можна навести ключові фактори, на основі котрих будується вищеописана специфікація.

Специфікація містить загальний опис, котрий включає в себе опис продукту та його перспективи, описує класи користувачів, котрі можуть ним користуватись, наводить обмеження продукту і його середовище функціонування, що є базовими означеннями, за якими буде створюватись система. Дотримання наведеної специфікації необхідно задля створення конкуренто спроможної системи котру можна підтримувати та розвивати.

Вище описана характеристика системи описує вимоги, яких потрібно дотримуватись при розробці ПЗ задля створення якісного продукту та спрощення безпосереднього виконання роботи. Також у специфікації описані поради щодо вибору інструментів розробки котрі повинні спростити розробку та подальше підтримання і модифікацію системи.

Вимоги до зовнішніх інтерфейсів описують інтерфейси, з якими буде працювати програма, з чого можна підібрати платформи та модулі програми для її більшої ефективності. Описані інтерфейси також дозволяють краще зоріентуватись щодо середовища виконання системи. Також опираючись на описані інтерфейси подальші користувачі матимуть змогу більш ефективно покращувати систему і не порушать структуру самої системи і не ускладнить модифікацію для інших користувачів.

Інші нефункціональні вимоги описують вимоги до продуктивності, надійності, безпеки та інших факторів, необхідних для стабільної роботи програми та задоволення потреб користувача. Використання рекомендованих технологій спростить роботу та зробить її більш ефективною для розробника. Також

дотримання нефункціональних вимог робить використання системи більш «приємною» з естетичної точки зору та позиції комфортної роботи.

Вивчивши і проаналізувавши створену специфікацю можна розпочинати розробку програмного продукту дотримуючись наведених вище.

РОЗДІЛ 3

Розроблення системи управління тест-кейсів

3.1. Цільове середовище виконання

Цільовим середовищем виконання системи є десктопні браузери для клієнтської частини та сервери на будь-якій операційні системи для серверної частини. З огляду на це слід дотримуватись певних стандартів при розробці. Також слід взяти до уваги, що користувачі будуть використовувати різні браузери для роботи з системою, а отже під час тестування слід перевірити адаптацію під найбільш популярні браузери. Також є сенс перевірити типічні серверні конфігурації на різних операційних системах і з різними об’ємами фізичної та оперативної пам’яті.

3.1.1. Розгортання backend частини

SpringBoot – додаток, котрий представляє backend частину системи можна зберегти як .war, так і .jar -файли. Розгорнута кожним з цих методів система буде працювати, проте у кожного варіанта є свої недоліки і переваги.

.jar файл є найлегшим для встановлення та зборки проекту. Але з огляду на те, що він не може бути обновлений необхідно буде перезавантажувати додаток кожен раз, як буде необхідно встановити обновлення.

.war складніший у збірці, проте дозволяє полегшити обновлення системи та є більш придатним для таких цілей.

3.1.2. Необхідність встановлення додаткових компонентів

Так як програма написана мовою Java, вона потребуватиме для своєї роботи встановленої но комп’ютері Java версії 8 чи більше. Це неможливо реалізувати програмно, тому про це необхідно інформувати користувача перед встановленням програми у інструкції з користування або у джерелі,

з якого буде поширюватись програма (наприклад на сторінці веб-сайту).

Для роботи системи необхідно окремо встановити PostgreSQL базу даних (хоча це можна за допомогою контейнерів Docker, але у межах дипломної роботи можна не ускладнювати систему зайвий раз). Також необхідно встановити JVM так як використовується мова розробки Java.

3.1.3. IDE

За рекомендаціями у специфікації, IDE буде використана IntelijIDEA.

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition»і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію.

Для розробки нашої програми достатньо буде використати версію «Community Edition» так як вона задовольняє усім необхідним нам умовам.

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNGi JUnit системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.[14]

3.2. Архітектура

3.2.1. Мова розробки

Мовой розробки програми було обрано Java. Ця мова орієнтована на веб розробку, дозволяє працювати з frontend і backend частиною, що робить її

оптимальним вибором для реалізації даної задачі. В програмі використовується версія Java 8 та версію JDK 1.8.2

3.2.2. Модель архітектури

З погляду на те, що система повинна працювати у браузері, при цьому не зберігаючі данні локально, має сенс будувати систему за кліент-серверною архітектурою.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта.

У даній роботі буде використана дворівнева архітектура з тонким клієнтом.

Схематично зображення системи видно на рисунку 3.1:

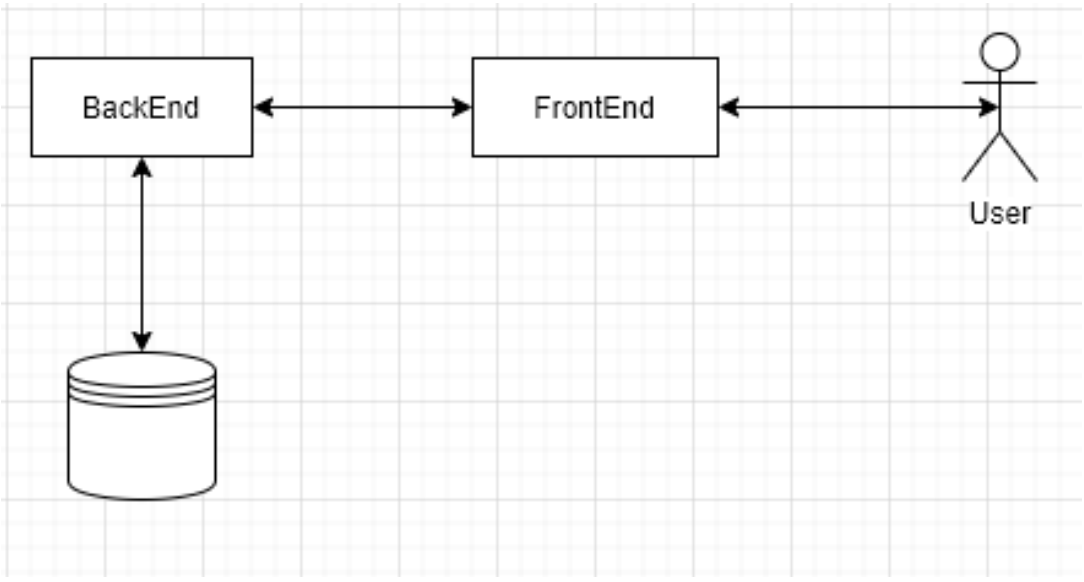

Рис. 3.1. Схематичне зображення системи

3.3. Інтерфейс користувача

3.3.1. Реалізація інерфейсу користувача

Інтерфейс користувача було реалізовано за допомогою темплейтів Feemarker. Feemarker забезпечує простий спосіб генерації текстів (HTML, вихідний код, конфігураційні файли, електронні листи тощо), який залежить від зміни даних. FreeMarker створений щоб відокремити логіку рендеринга/форматування (дизайн, HTML) від підтримки додатків і технічної складності. Він має гнучкий API, так що ви можете вбудовувати його в свої програми.

Загальні цілі:

- Може бути використаний для створення будь-якого виду тексту: HTML, XML, RTF, вихідний код Java, і т. д.
- Простота вбудовування в програму: немає залежностей і не потрібна середовище сервлетів.
- Завантажувач шаблонів: ви можете завантажувати шаблони з будь-яких джерел: локальні файли, бази даних і т. д.
- Ви можете робити все, що ви хочете зі згенерованим текстом: зберігати його в локальному файлі, відправити його по електронній пошті, відправити його назад через веб-браузер веб-додаток і т. д.

Потужна мова шаблонів:

- Звичайні директиви if/elseif/else зациклюються за списками включаючи інші шаблони.
- Створення і зміна змінних в шаблонах.
- Можна використовувати складні вирази для визначення значень майже всюди.
- Користувацькі директиви (макроси) з назвами і позиційними параметрами і з вкладеним змістом (body). Наприклад: <@myMacro color="red" width=2>... Вони можуть бути визначені прямо в шаблонах, або навіть в Java.

- Користувацькі директиви можуть фільтрувати висновок свого вкладеного контенту, так як роблять white-space compression, підсвічування синтаксису тощо.
- Простір імен допомагає будувати і підтримувати часто використовувані макро/функціональні бібліотеки або ділити великі проекти на окремі модулі, не лякаючись зіткнення імен.

Універсальна модель даних:

FreeMarker не працює через пряме відображення на об'єктах Java; об'єкти Java піддаються шаблоном у вигляді дерева змінних через підключаються object wrappers. Таким чином, ви можете показати об'єкти (Java beans, XML documents, результати запиту SQL і т. д.) в абстрактному, настроеннsм для авторів шаблону чином, не потрудившись з технічними деталями.

Це також означає, що об'єкти, які надходять для не-Java JVM мов (таких як Jython) можуть бути доступні як зручно, як "рідні" об'єкти (якщо належний object wrapper реалізований).[15]

3.3.2. Мапа сайту

Основними сторінками сайту є:

- Сторінка логіну та регістрації
- Сторінка зі списком сьютів
- Сторінка додавання тест-кейсів
- Сторінка прогону тестів
- Сторінка виконання прогону тестів
- Головна сторінка

Сторінкою за замовчуванням для не авторизованого користувача є сторінка логіна користувача куди завжди потрапляє неавторизований користувач. Для авторизованого користувача сторінкою за замовчуванням є сторінка «Main»

Навігація по сторінкам (за виключенням сторінки логіна) здійснюється за допомогою панелі навігації. Також на панелі навігації є можливість розлогінитись.

Схематично мапу сайту зображено на рисунку 3.2

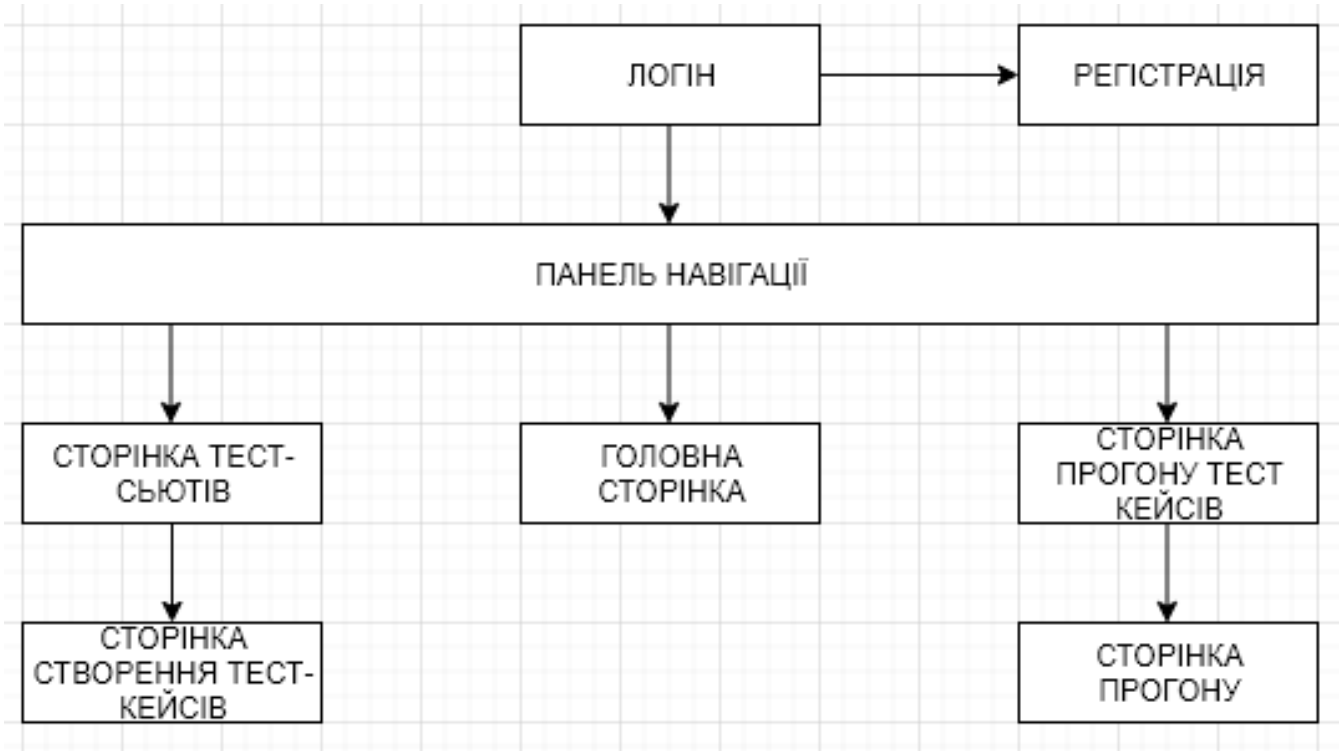

Рис. 3.2. Мапа сайту

Як видно зі схеми першою сторінкою що бачить користувач є сторінка логіну. Якщо користувач ще не зареєструвався він може перейти на сторінку регістрації де створить собі аккаунт. Без створеного облікового запису користувач не матиме доступу до системи.

Навігація між розділами відбувається за допомогою панелі навігації в котрій є посилання на всі сторінки, а також кнопка котра дозволяє вийти з облікового запису. Панель навігації доступна з будь-якої сторінки системи.

Сторінки виконання прогону тестів и додавання тест кейсів є залежними від сторінок на котрих створюються сьюти та прогони тому перейти на них можна лише з відповідної сторінки.

3.3.3. Сторінка логіну та регістрації

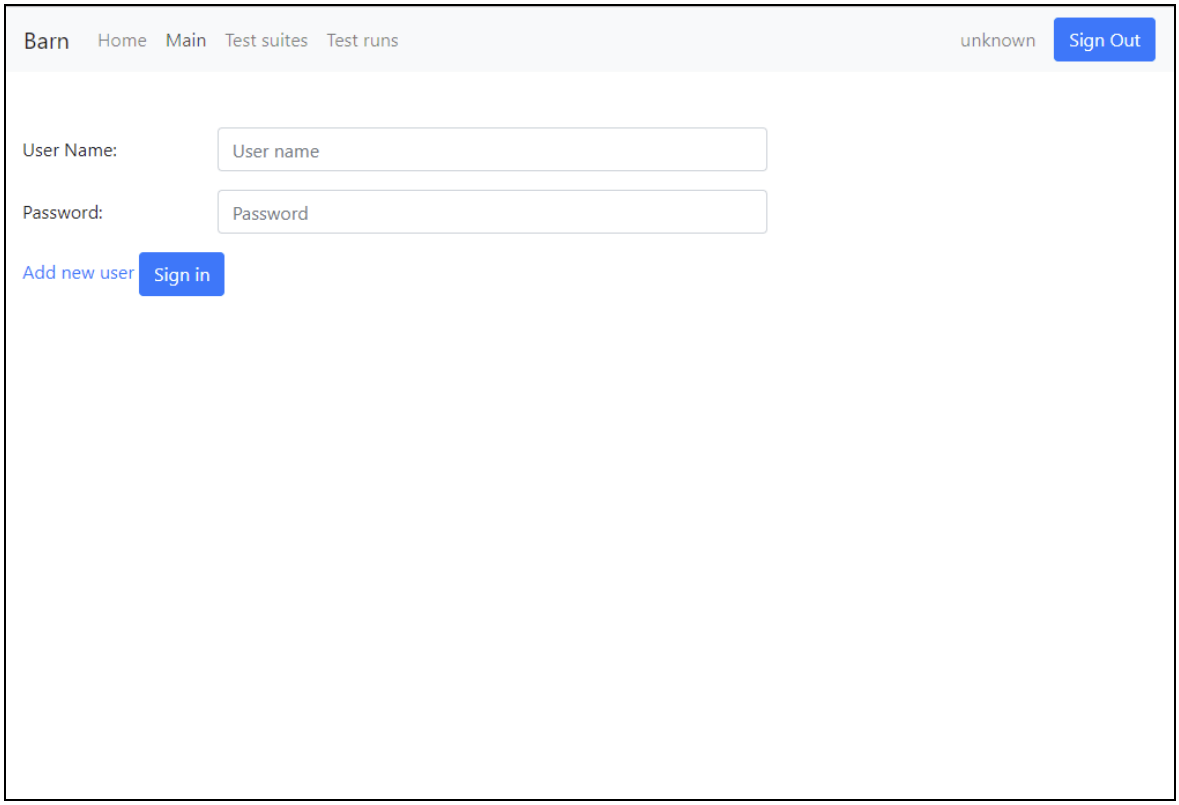

Рис. 3.3. Сторінка логіну та регістрації

Логі́н — цифро-буквенній набір символів, що ідентифікує користувача системи. Логін разом із паролем зберігаються в обліковому записі на сервері у зашифрованому виді та використовуються системою для надання користувачу дозволу на з'єднання з комп'ютерною системою, визначення його прав доступу до ресурсів мережі та створння csrf токенів котрі необхідні для виконання запитів створення та отримання данних. Логін має бути унікальним в межах даної системи. В разі спроби створення дублікату користувачу відобразиться помилка.

Логін — процедура входу (ідентифікації і згодом авторизації) користувача в систему. В разі введення не існуючого логіну чи невалідного паролю користувачу відобразиться відповідна помилка. В разі не авторизованого http запиту користувачу буде повернена помилка 401 котра означає неавторизований доступ.

Сторінка логіну є першою сторінкою що бачить користувач коли відкриває сайт системи. Вона складається з наступних частин:

- Поле вводу логіну котре приймає цифри та букви
- Поле паролю котре приймає цифри та букви

- Кнопка підтвердження
- Посилання на сторінку на сторінку створення аккаунту

Панель навігації присутня і на цій сторінці але кожне посилання буде вести назад на сторінку логіну.

Після підтвердження валідного логіну та паролю користувача перекидує на сторінку Main.

Після підтвердження невалідного логіну та паролю відображається повідомлення помилки і користувач залишається на даній сторінці.

Щоб залогінити юзера необхідно:

1. Ввести існуючий логін
2. Ввести пароль
3. Підтвердити

Щоб створити юзера необхідно:

1. Клікнути по лінці “Add new user”
2. Ввести будь-яке ім’я в поле логіну
3. Ввести будь-який пароль в поле паролю
4. Підтвердити (після цього користувача перекину на сторінку логіну)
5. Ввести логін
6. Ввести пароль
7. Підтвердити

3.3.4. Сторінка Main

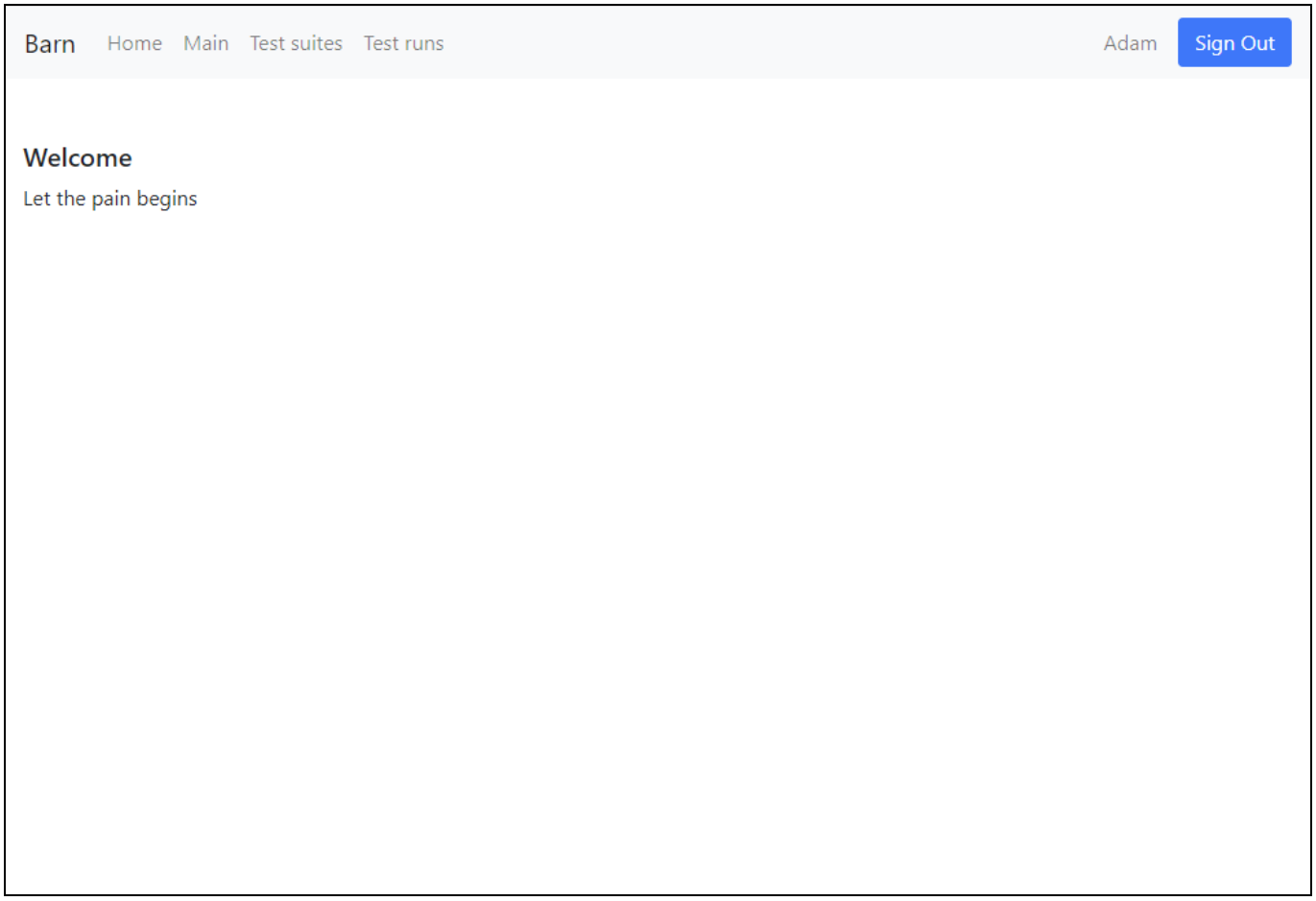

Рис. 3.4. Сторінка Main

Сторінка Main є стартовою сторінкою системи. Її функція лише інформативна і ніяких функцій окрім інформативних не несе.

Призначення подібних сторінок полягає в тому щоб користувач міг опрідилитись з подальшими кроками використання системи.

На даній сторінці зазвичай знаходиться посилання на довідки, короткі інструкції, реклама нових продуктів компанії розробника та інше.

В межах даної роботи на сторінці буде лише повідомлення-привітання.

3.3.5. Сторінка зі списком сьютів

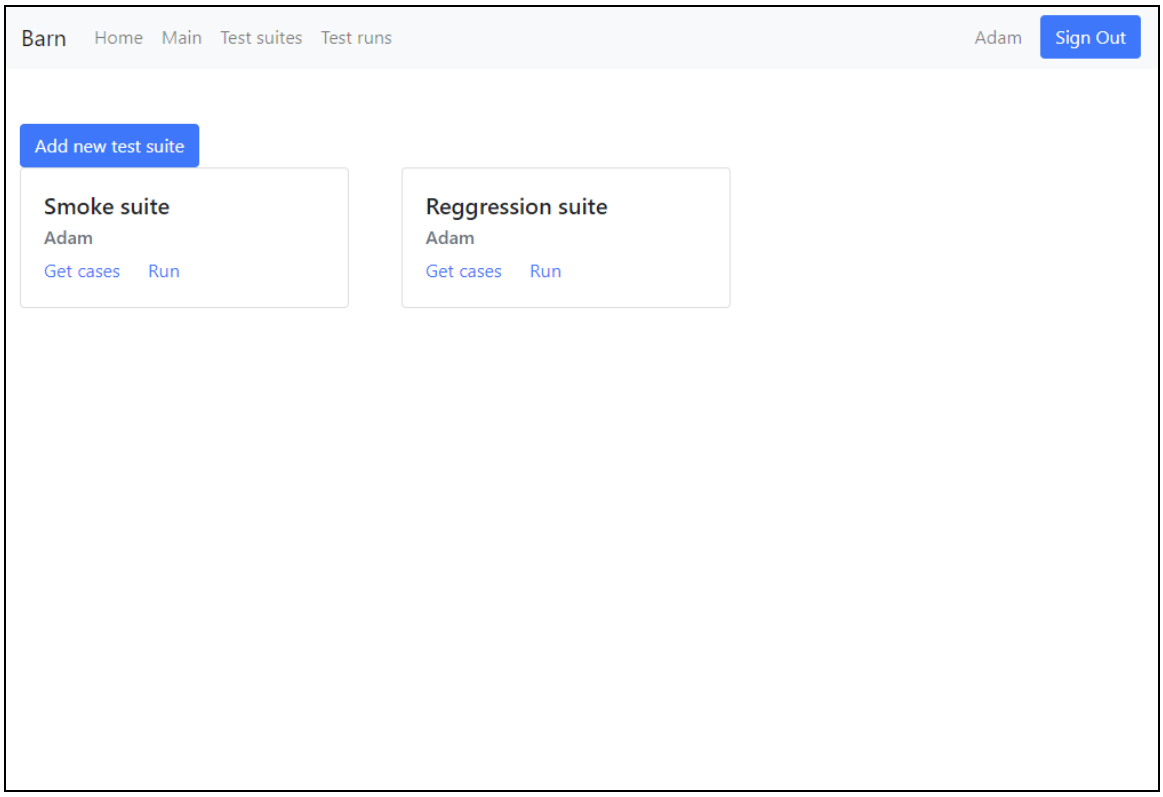

Рис. 3.5. Сторінка тест сьютів

Тест-сьют це набір тест кейсів, які об'єднані тим що відносяться до одного модулю що тестуеться, функціональності, пріоритетністю або одного типу тестування. Кожен тест-сьют складається з більш ніж одного тест кейса і часто виконується всієї «пачкою» в процесі тестування.

Тест кейси об'єднують в тест-сьюти для більшої зручності при проходження тест кейсів, проходячи їх послідовно від модуля до модуля, від одного типу тестування до іншого, а не сумбурно, кидаючись з одного кутка в куток залишивши неперевіреними більшу частину модуля або загальної функціональності.

Сторінка Test Suites призначення для відображення існуючих тест кейсів та створення нових.

Сторінка складається з:

- Розгортаючоїся форми створення тест кейсу
- Блоку тест кейсу котрий складається

Блок тест кейсу складається з:

- Назви
- Імені автора
- Лінки на створення тест кейсів
- Лінки на виконання прогону тестів

Форма створення тест сьюту складається з:

- Кнопки котра розгортає форму
- Поле назви
- Кнопка підтвердження

Щоб додати новий тест кейс необхідно:

1. Залогінити користувача
2. Перейти на сторінку Test suites
3. Клікнути по кнопці Add new test suite
4. Вести назву тест сьюту
5. Підтвердити

3.3.6. Сторінка додавання тест-кейсів до тест сьютів

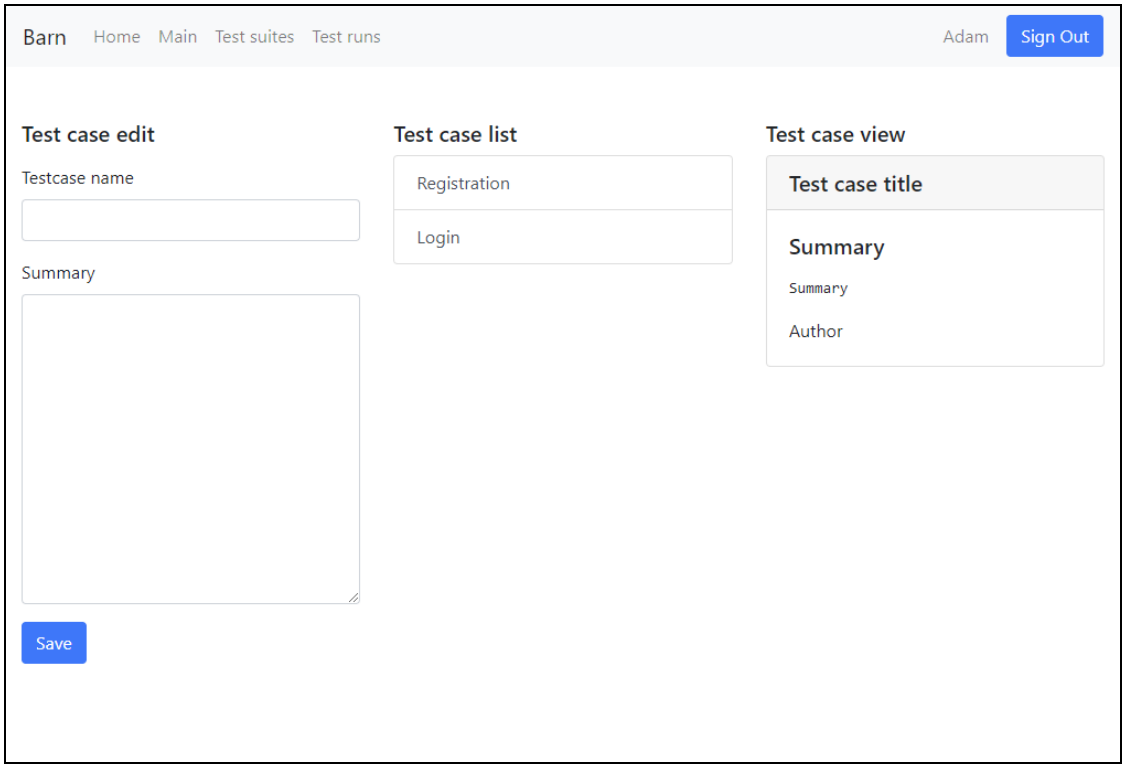

Рис. 3.6. Сторінка тест кейсів

Тестовий випадок (Test Case) - це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції що тестується або її частини.

Будь-тест-кейс обов'язково включає в себе:

Унікальний ідентифікатор тест-кейса - необхідний для зручної організації зберігання і навігації по тест-сьютам. У нашому випадку ідентифікатори скриті і використовуються лише системою.

Назва - основна тема, або ідея тест-кейса. Короткий опис його суті.

Передумови - опис умов, які не мають прямого відношення до перевіряемого функціоналу, але повинні бути виконані.

Наприклад, залишити коментар на вашому порталі може тільки зареєстрований користувач. Значить для тест-кейса «Створення коментаря» буде необхідне виконання передумови «користувач зареєстрований», і «користувач авторизований»

Кроки - опис послідовності дій, яка повинна привести нас до очікуваного результату

Очікуваний результат – результат що ми очікуємо побачити після виконання кроків.

Сторінка додавання тест кейсів до тест-сьюта призначена для додавання, переглядання та редагування тест-сьютів. Сторінка складається з:

- Поля назви тест-кейсу
- Поля Summary
- Списка тест-кейсу
- Області відображення обраного кейсу
- Кнопки збереження

Для додавання тест-кейса до сьюта необхідно:

1. Відкрити сторінку сьютів
2. На обраному сьюті натиснути кнопку «Get cases»
3. Ввести назву кейсу
4. Ввести Summary
5. Клікнути по кнопці Save

Щоб відкрити вже створений тест кейс необхідно:

1. Відкрити сторінку тест-сьютів
2. На обраному сьюті натиснути кнопку «Get cases»
3. Клікнути по обраному тест-кейсу

3.3.7. Сторінка прогонів тестів

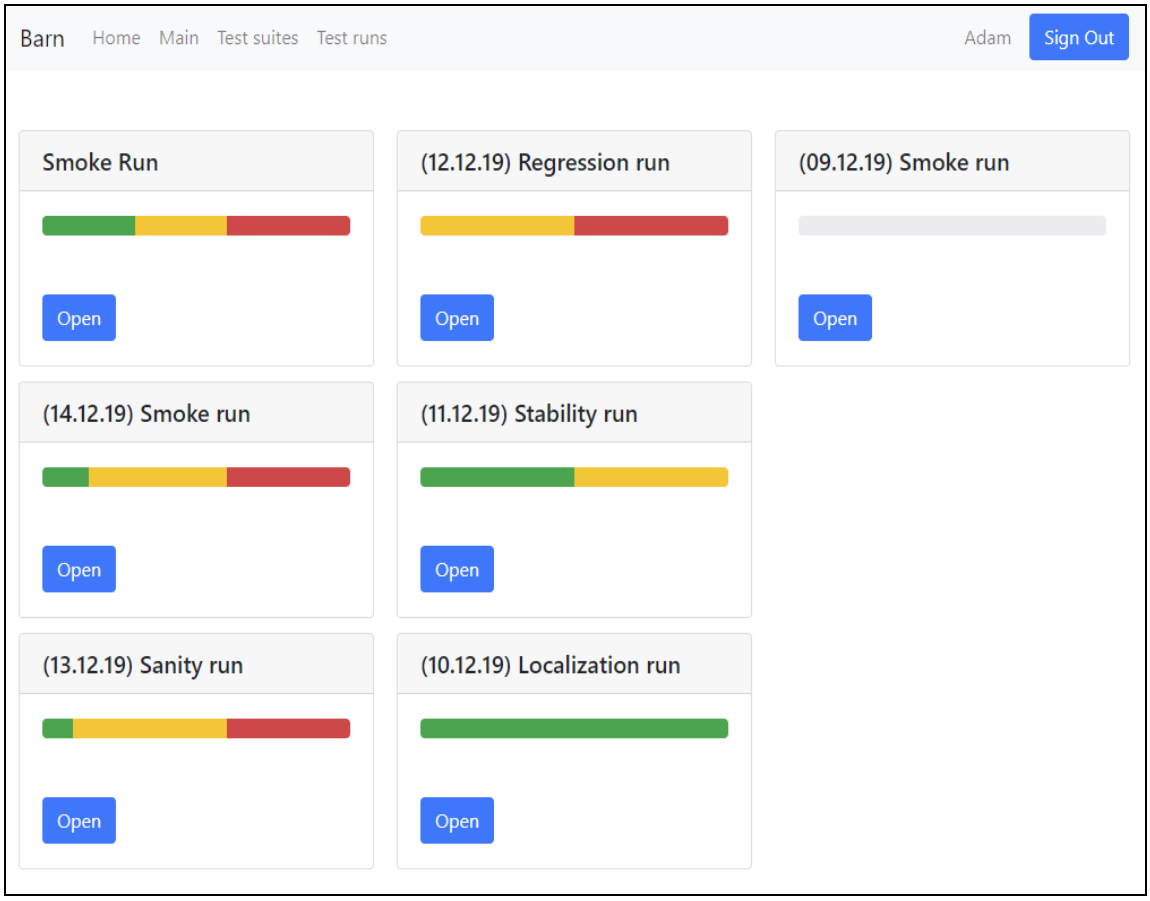

Рис. 3.7. Сторінка прогонів тестів

Сторінка прогонів призначена для переглядання результатів виконаних тест кейсів та закінчення процесу прогону.

Вона складається з блоків прогонів котрі в свою чергу складаються з:

- Назви прогону
- Результату та статусу прогону
- Кнопки відкриття прогону

Щоб створити новий прогін необхідно

1. Відкрити сторінку тест кейсів
2. Клікнути по кнопці “Run tests”
3. Перейти на сторінку прогонів тестів

3.3.8. Сторінка прогону

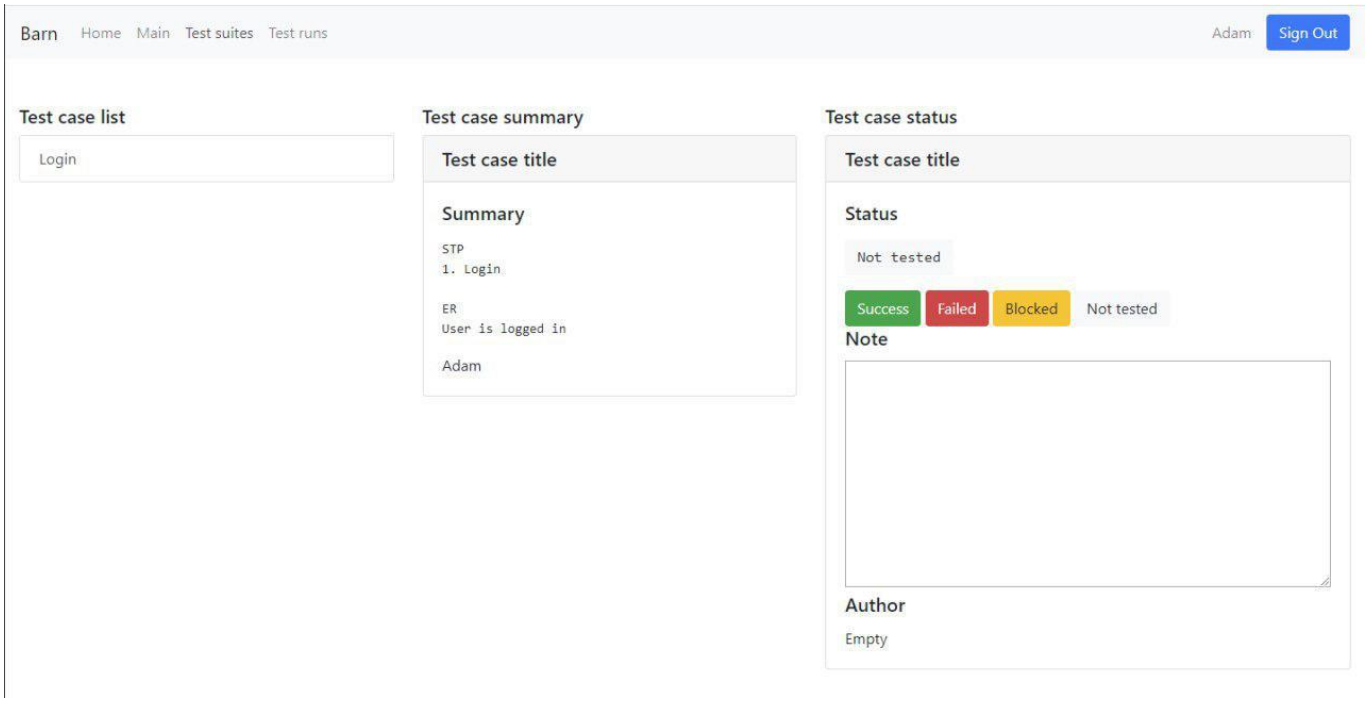

Рис. 3.8. Сторінка прогону тесту

Прогін тесту (виконання) - це процес присвоєння результату («Блокований», «Не запущено», «Помилка», «Пройдено») для обраних тестів. Назви станів тестів означають наступне:

- Блокований – немає можливості виконати даний тест
- Не запущений – тест ще не виконувався
- Помилка – в ході виконання тесту була знайдена помилка
- Пройдено – тест був виконаний успішно

Сторінка прогону призначена для безпосереднього виконання прогону тестів. Вона складається з наступних компонентів:

- Список тест кейсів
- Блок з текстом тест-кейсу
- Статус тест-кейсу
- Нотатки щодо виконання
- Поле автора

Для того щоб прогнати тест необхідно:

1. Створити прогон тестів
2. Відкрити прогон
3. Клікнути по будь-якому тест-кейсу
4. Клікнути по будь-якому статусу

3.4. Серверна частина

3.4.1 SpringBoot

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування. Spring Framework став популярним в спільноті Java як альтернатива або навіть доповнення моделі Enterprise JavaBean (EJB).

Spring Framework складається з кількох модулів, які надають широкий спектр послуг:

- Контейнер Інверсії управління: конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через Інверсію управління
- Аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури
- Доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних
- Управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів

- Модель-Вигляд-Управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful.
- Аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security (колишня система безпеки AcerI для Spring).
- Віддалене керування: конфігураційний вплив і управління Java-об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX
- Тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів [16]

У межах даній дипломній роботі будуть використані модулі аутентифікації, MVC та доступу до бази даних. Структурно система має наступний вигляд:

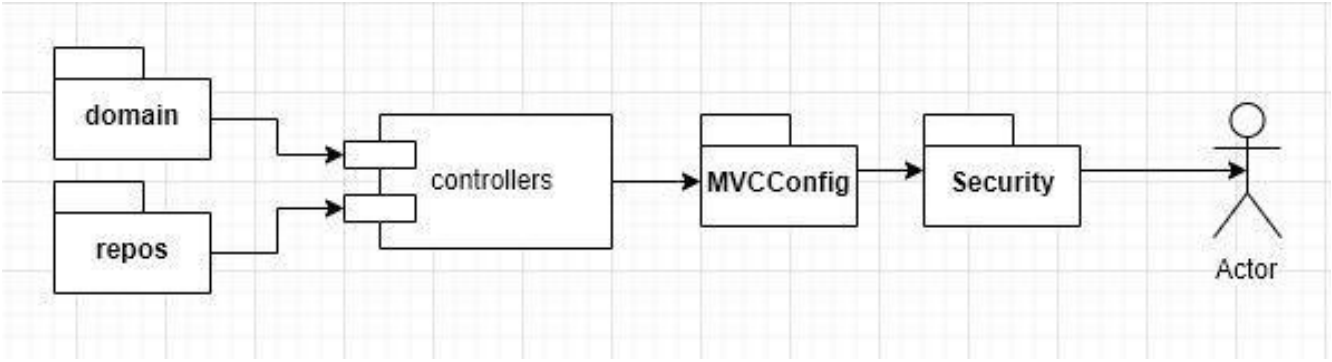

Рис. 3.9. Схематичне зображення системи

Розглянемо систему ближче

- Domain – це програмний опис сутностей бази даних. Являє собою клас з ключами що представляють собой строки у базі даних та методи для їх обробки.
- Repos – інтерфейси для пошуку даних по таблицям. Являє собою інтерфейси з методами пошуку даних у таблиці. Реалізацію цих методів бере на себе SpringBoot

- Controllers – класи з функціями котрі оброблюють дані котрі передає клієнтська частина
- MVC Config – конфігурація Spring Boot котра призначена для ініціалізації об’єктів класів та своєчасного їх виклику
- Security Config – конфігурація Spring Boot котра призначена для захисту запитів користувача за допомогою токенів, ключів, та іншого.
- Actor – клієнтська частина системи. Так як клієнтська частина не є окремим застосунком ми можемо об’єднати в актора и клієнтську частину і безпосередньо користувача.

3.4.2 Аутентифікація

Так як тест-кейси можуть бути частиною проектної документації, слід обмежити до них доступ. В межах даної роботи буде використаний CSRF-токен.

Логіка роботи аутентифікації у системі наступна:

1. Користувач логіниться за допомогою логіна та паролю
2. Система повертає токен браузеру
3. У кукі браузера зберігається отриманий токен
4. Усі запити до системи підтверджуються цим токеном

З цього ми отримуємо, що неавторизований користувач не може отримати доступ до даних. Кожна спроба отримати чи передати дані буде повертати користувача на сторінку логіну.

Даний підхід не є абсолютно безпечним, так як токен створюється лише раз і не змінюється впродовж використання системи, проте в межах даної дипломної роботи він виконує свою задачу. Для більшої надійності слід використовувати Oath2 верифікацію при котрій токен оновлюється з заданою розробником частотою.

3.4.3. Контроллери

Для роботи з ендпоінтами у SpringBoot використовуються класи анотаціями @Controller методи яких відповідають за певний ендпоінт. Структура та назви не

важливі, так як SpringBoot сам контролює структуру. Основними класами конроллерами в системі є:

- TestCaseController – відповідає за створення, отримання та модифікацю тест кейсів
- TestSuiteController – відповідає за створення і отримання тест сьютів
- RunController – відповідає за створення прогонів тестів
- AuthController – відповідає за логін та розлогін користувача
- UserController – відповідаю за створення і змінення користувача

Приклад структури

Розглянемо принцип роботи контролерів на прикладі Контроллера тест-сьюта

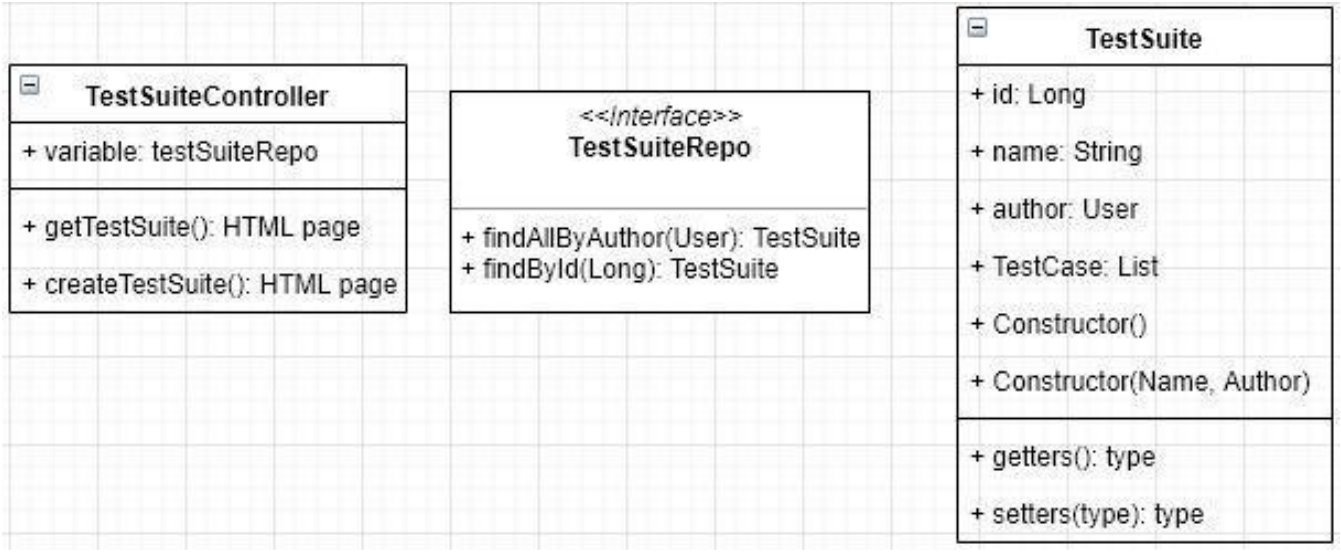

Рис. 3.20. Схематичне структури роботи контроллера

Робота з кожною сутністю бази даних передбачає роботу з трьома основними компонентами (для спрощення пояснень не будемо звертати увагу на роботу контейнерів авторизації, інверсії управлінь та іншого) – програмна реалізація сутності у базі даних, інтерфейс Repo призначений для роботи з базою даних, та безпосередньо сам контролер.

Реалізація сутності бази даних

Реалізація сутності є Java класом котрий позначений анотацією @Entity. Spring Boot створює таблицю базі даних з назвою цього класу, ключами якого є його константи, перемінні та інше. Функції цього класу (геттери і сеттери, конструктори та інше) ігноруються. Отримання даних з таблиць виконуються через цей клас.

Інтерфейс Repo

Ці інтерфейси призначені для виконання пошуку по базі даних. Інтерфейс являє собой інтерфейс Java, методи якого виконують пошук. Цікавою стороною цієї системи є те що реалізація методів не потрібна проте важливі строгі правила по іменуванню цих методів. Реалізацю пошуку даних бере на себе Spring Boot.

Контролер

Контролери виконують певні дії при зверненні до конкретного ендпоінта. Вони представляють собою клас за нотацією @Controller і методів з означенням типу запитів і ендпоінтів які вони оброблюють. Наприклад метод з нотацією @GetMapping("/testase?id=5") означає отримання тест кейсу з id рівним п'яти.

Підсумувавши роботу контроллерів можна описати як послідовність наступних дій:

1. Користувацька частина передає певні данні до якогось ендпоінта
2. Серверна частина знаходячи потрібний ендпоінт виконує відповідний метод
3. Метод оброблює данні звертаючись до відповідної таблиці бази даних через інтерфейс Repo
4. Після обробки даних повертає результат у вигляді оновленої сторінки.

3.4.4. База даних

У даній дипломній роботі базою даних було обрано PostgreSQL, так як вона безкоштовна і підтримується по всьому світу. Так як в роботі використовується SpringBoot, нам не потрібно руками створювати таблиці. Усе це бере на себе SpringBoot. Від розробника необхідно лише помітити класи, котрі повинні бути описані у базі анотацією @Entity та створити саму базу. Підключається база даних до системи за допомогою файлу properties, у котрому описується адреса бази даних, логін та пароль до неї.

Так як кожен класс створений з анотацією @Entity, то кожен об'єкт цього класу є записом цієї таблиці. Щоб створити зв'зки між таблицями типу one-to-mane чи many-to-many, необхідно просто описати необхідну нотацію. База даних, котра створюється після першого запуску системи, має наступний вигляд:

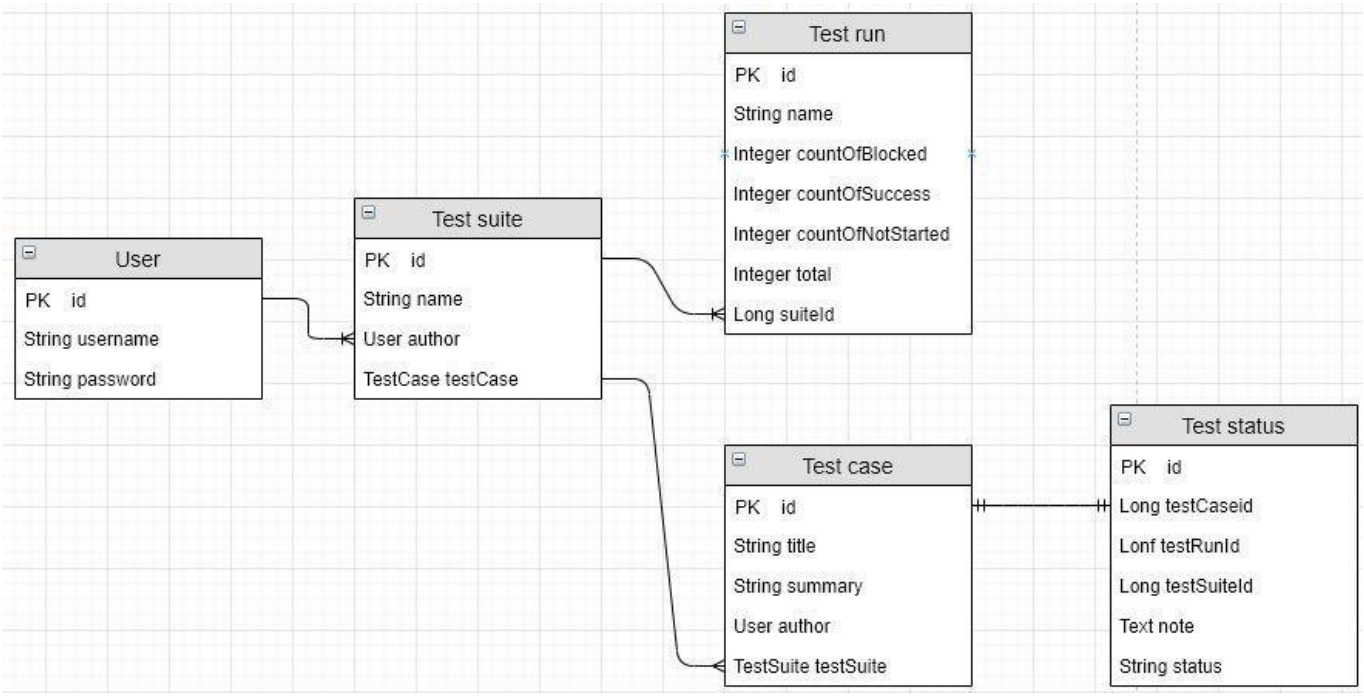

Рис. 3.31. Структура бази даних

Таблиця User

Таблиця User містить собі головну інформацію про користувача, а саме його логін та пароль. Данні з цієї таблиці використовуються для авторизації користувача і для отримання тест-сьютів та прогонів котрі належать користувачу. Данні в цю таблицю заносяться при створенні нового користувача. Id користувача є Primary Key і зв'язаний з Foreign Key author таблиці Test suite. Тип зв'язку є один до багатьох (у користувача може бути багато сьютів, проте у сьюта завжди лише один автор).

Таблиця Test suite

Таблиця Test suite є головною таблицею системи так як об'єднує через себе всі інші таблиці. Дана таблиця містить в собі назву тест-сьюта, Foreign Key з посиланням на автора. Id тест-сьюта є Primary Key і зв'язаний з Foreign Key таблиць Test run і Test case. В обох випадках тип зв'язку один до багатьох, тобто у тест-сьюта може бути безліч прогонів, але прогон виконується лише по одному сьюту, у сьюта може бути безліч тестів, але тести належать лише одному сьюту.

Таблиця Test run

Таблиця Test run призначена для збереження даних щодо виконаних прогонів тестів. Вона містить в собі назву, кількість провалених, блокованих, успішно виконаних та не виконаних тестів і посилання на тест-сьют.

Таблиця Test case

Таблиця Test case призначена для збереження даних тест кейсу. Вона містить в собі назву, текст, автора і посилання на тест-сьют. Id тест-сьюта є Primary Key і зв’язаний з Foreign Key таблиці Test status зв’язком один до одного (у кожного тест кейсу є лише один статус).

Таблиця Test status

Таблиця Test status призначена для збереження даних щодо виконання тест кейсу. Таблиця містить в собі посилання на тест кейс, айді сьюта, айді прогону та безпосереднє значення статусу. Дана таблиця порушує третю нормальну форму, але призначена для спрощення структуру коду. Вона дозволяє працювати з однією таблицею замість трьох що спрощую як написання коду так і безпосереднє навантаження на базу даних.

Висновки

Ураховуючи вимоги, описані у специфікації, було розроблено програмний продукт мовою програмування Java. Графічний інтерфейс створений за допомогою темплйтів Freemarker. За базу даних було взято PostgreSQL, взаємодія з якою відбувається за допомогою Hibernate, котрий є частиною Spring Boot. Уся серверна логіка реалізована на фреймворку Spring Boot. Даний фреймворк бере на себе роботу з базою даних що значно спрощує роботу з нею. Також спрощується робота з ендпоінтами контроллерів.

Створена програма є робочою, проте повинна бути протестована перед початком її експлуатації, щоб впевнитись у цьому. Необхідно протестувати взаємодію з базою даних, зручність використання системи, правильність відображення і збереження даних та інше.

Використання темплейтів спрощує архітектуру системи, так як не потрібно використовувати JavaScript, а також дозволяє обійтись без додаткового обміну даними через JSON чи XML. Проте це не дозволяє використовувати технології Angular (оновлення частини даних замість усієї сторінки) що тробить систему менш гнучкою.

Використання клієнт-серверної архітектури дає змогу працювати з важкими системами з будь-якого комп’ютера на котрому є браузер. Також це дає змогу отримувати доступ до своїх даних з будь-якого комп’ютера що є зручним та безпечним у відношенні до даних.

РОЗДІЛ 4
РЕЗУЛЬТАТИ ДОСЛІДНОГО ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ТЕСТ КЕЙСАМИ

4.1.Тестування аутентифікації

Тестування аутентифікації можна провести на двох рівнях – модульному та системному.

Тестування на модульному рівні – це тестування конкретної функції без її зв’язку з іншими компонентами системи. В нашому випадку модульне тестування аутентифікації – це безпосередній виклик функції аутентифікації без запуску усієї системи в цілому.

Так як модульне тестування досить об’ємне з огляду на кількість функцій, котрі викликається, то доцільним буде протестувати систему системно.

Системне тестування – це тестування компонентів у зв’язку один з одним. Цей рівень надає найбільш повне уявлення про те, як працює система, проте такий підхід значно повільніший за модульний.

Системне тестування аутентифікації проводилося наступним чином:

1. Систему запустили локально запустивши мавен проект через консоль та відкривши локалхост
2. Відкрили сторінку аутентифікації. Ця сторінка відкриється автоматично при відкритті локал хосту
3. Вибрали створення користувача клікнувши по кнопці Add user
4. Ввели бажаний логін та пароль та підтвердили ввід даних клікнувши по кнопці Create

5. Повернулись на сторінку аутентифікації. Користувача автоматично поверне на сторінку логіну після створення користувача
6. Ввели логін та пароль котрі були вибрані при створенні нового користувача
7. Підтвердили клікнувши по кнопці Sign In

Очікуваний результат: відкрита сторінка Main, у правому верхньому куті відображається ім’я користувача, користувач може переходити на сусідні сторінки. В базі даних у відовідній таблиці створений новий запис з даними котрі ввів користувач. В кукі браузера доданий токен користувача котрий і дозволяє користувачам користуватись системою.

Фактичний результат відображений на приведених нижче зображеннях

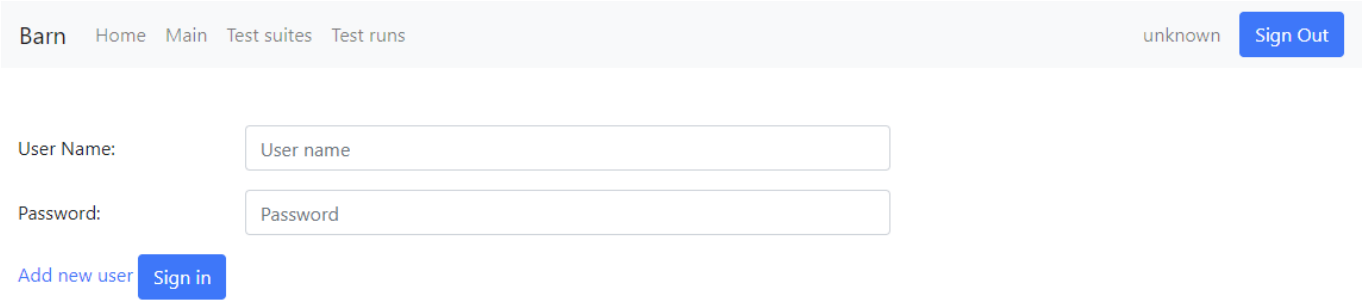

Рис. 4.1. Порожня сторінка логіну

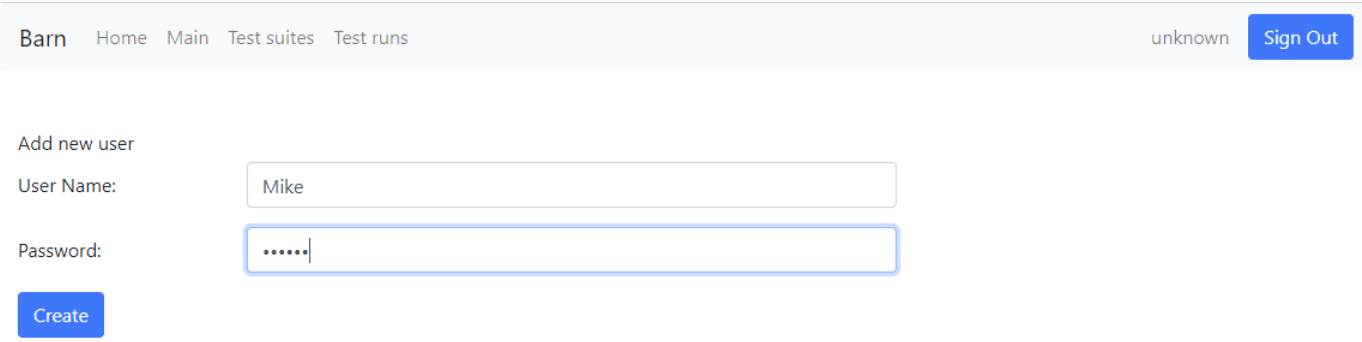

Рис. 4.2. Сторінка логіну з заповненими полями

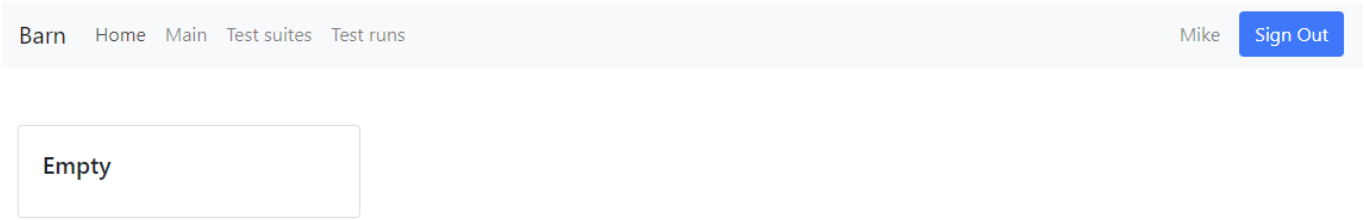

Рис. 4.3. Порожня сторінка тест-сьютів

Як видно зі скріншотів користувач був успішно сторений, йому не відображаються чужі прогони тестів і він має змогу перемикатись між вкладками. Система успішно створила нового користувача, створила токен і дозволяє користуватись системою. В ході виконання тестування функціоналу помилки знайдені не були.

Додатковим тестом до цього є логін користувача з невалідними даними. Дана перевірка була проведена наступним чином:

1. Систему запустили локально запустивши мавен проект та відкривши у браузері сторінку локалхосту
2. Відкрили сторінку аутентифікації. Ця сторінка відкривається автоматично при відкритті будь-якої сторінки системи неавторизованим користувачем
3. Ввели валідний логін та невалідний пароль або навпаки у відповідні поля фми логіну
4. Підтвердили клікнувши по кнопці Log in

Очікуваний результат: відображення помилки неправильного паролю. Користувач не залогінений, в кукі системи не доданий токен токен, користувач не має можливості відкривати сторінки системи

Фактичний результат відображений на приведених нижче зображеннях

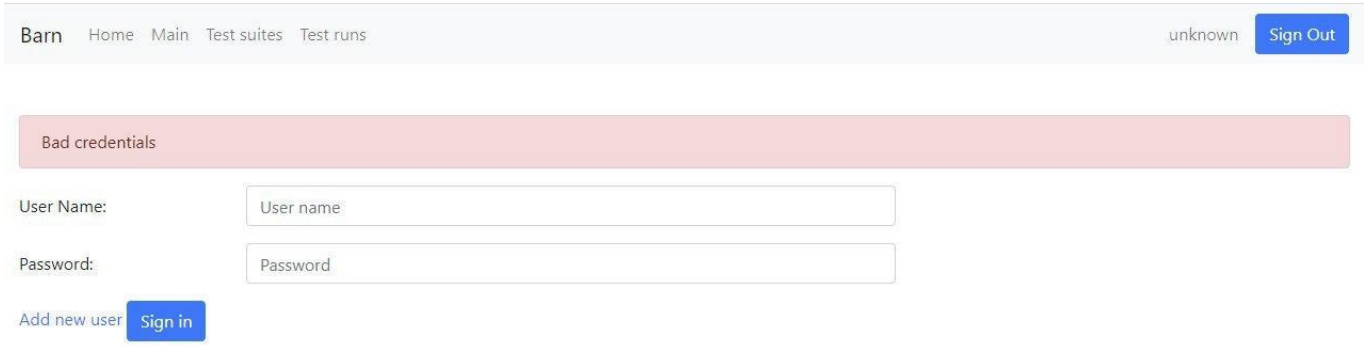

Рис. 4.4. Помилка верифікації на сторінці логіну

Як видно зі скріншоту користувачу повідомляється що при введенні логіну чи паролю були введені невірні данні, але не повідомляється які саме. Це є певним недоліком котрий спричинений обраним стеком технологій розробки. Проте інші системи теж іноді не повідомляють користовачу які саме данні введені неправильно з огляду на питання безпеки. В ході тестування даного функціоналу була знайдена помилка котра була пов’язана з поверненням помилки на порожні поля. Повідомлення помилки що повертає система не було зрозумілим. Дана помилка була виправлена створенням власного обробника помилок.

Останніми та найважливішими тестами аутентифікації є спроба отримати чи змінити закриті данні не залогіненим юзером. Ці тести мають найбільшу сер’йозність, проте не мають сенсу без виконання попередніх двух. Тест на отримання закритих даних можна провести наступним чином:

1. Систему запустили локально. Для цього достатньо запустити мавен проект, відкривати сторінку браузеру не потрібно
2. В адресному рядку браузера прописати шлях то будь якого тест кейсу, наприклад localhost/testcase?id=5
3. Підтвердити

Очікуваний результат: відображення сторінки логіну, дані користувача не відображаються не авторизованому юзеру.

Фактичний результат відображений на приведених нижче зображеннях

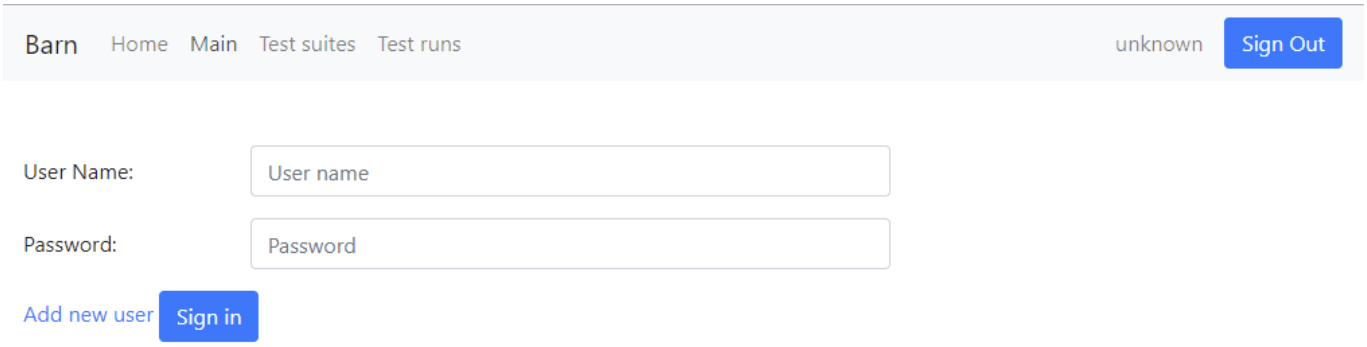

Рис. 4.5. Порожня сторінка логіну

Як видно зі скріншоту користувача перекинуло на сторінку логіну, дані користувачів системи не відображаються не авторизованим юзерам. При подальшому розвитку системи можна додати системне повідомлення проте на даний момент цього достатньо. В ході тестування даного функціоналу помилки знайдені не були.

Тест на зміну закритих даних виконали за допомогою http-кліенту Postman:

1. Систему запустили локально запустивши мавен проект через командний рядок. Відкривати локалхост не потрібно
2. Запустили Postman/cURL або будь-який інший інструмент котрий призначений для виконання API запитів

3. Створили POST-запит на створення тест кейсу у відомому сьюті
4. Виконали запит
5. Перевірили таблицю даного сьюту у базі даних

Очікуваний результат: буде отримано 401-код сервера, котрий означає помилку доступу, новий запис у базі даних не створено.

Фактичний результат відображений на приведених нижче зображеннях

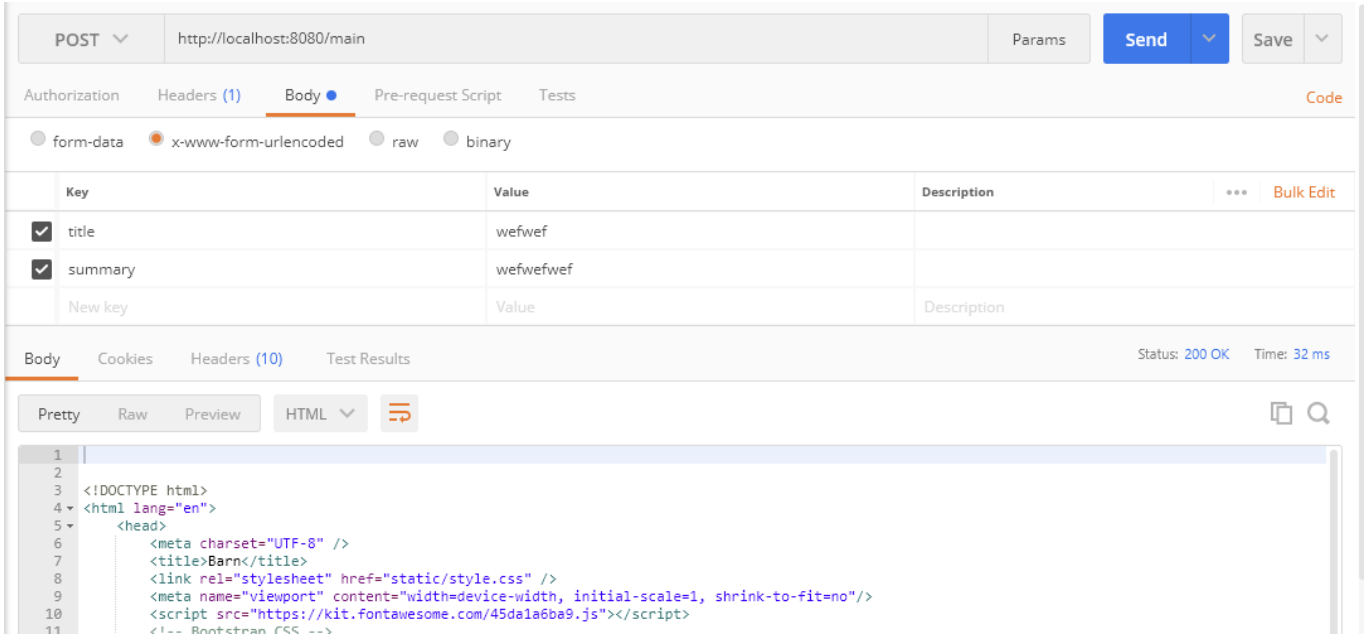

Рис. 4.6. Виконаний запит в Postman

Як видно зі скріншоту 401 помилка не була повернута системою проте і данні не були змінені. Це спричинено підходом котрий був використаний при розробці системи авторизації, а саме використання стендартної системи авторизації Spring Boot. Даний результат є задовільним з огляду на те що він виконує поставлену задачу (захист персональних даних) проте може ускладнити дебагінг системи при подальшій його розробці. В ході виконання тестування цього функціоналу помилки знайдені не були.

Після виконання даних тестів і виправлення знайдених помилок можна зробити висновок, що аутентифікація у системі працює вірно. Вона виконує свої головні функції, а саме скриття і захист персональних даних персональних даних від

неавторизованих користувачів. Єдиним недоліком даної системи на даний момент є використання статичного токену що не суттево в розрізі даного проекту.

4.2. Тестування процесів управління тест-кейсів

4.2.1. Створення тест-сьюту

Для створення тест-сьюту необхідно провести наступні дії:

1. Запустити систему локально запустивши мавен проект через командний рядок та відкривши локалхост
2. Залогінити користувача. Можна як використати старого юзера так і створити нового
3. Відкрити сторінку створення тест кейсів клікнувши по відповідному посиланні на панелі навігації
4. Клацнути по кнопці Add new test suite
5. У випадаючому меню створення що відкрилося ввести бажану назву тест-сьюта
6. Підтвердити клікнувши по кнопці «Додати»

Очікуваний результат: відображення блока тест-сьюта на сторінці тест-сьютів. У базі даних створений відповідний запис. Ім’я тест-сьюта відповідає назві що введе користувач.

Фактичний результат відображений на приведених нижче зображеннях

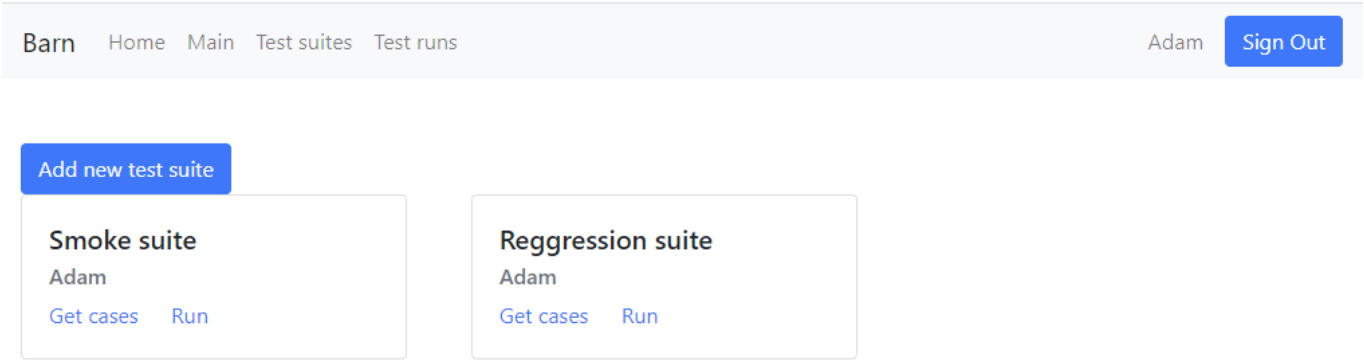

Рис. 4.7. Сторінка тест-сьютів

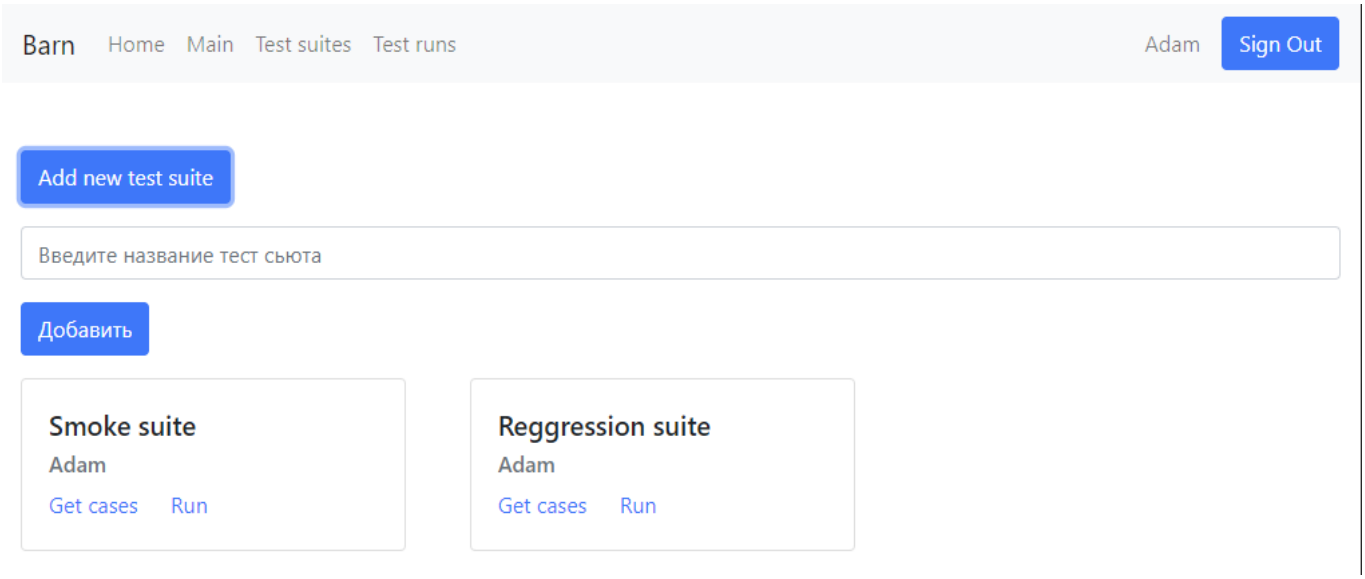

Рис. 4.8. Відкрита форма створення тест-сьюту

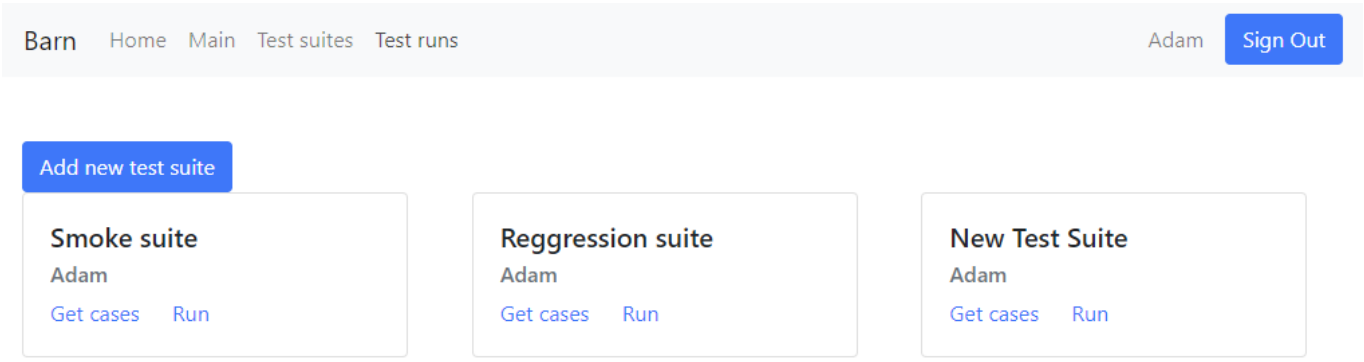

Рис. 4.9. Сторінка тест-сьютів

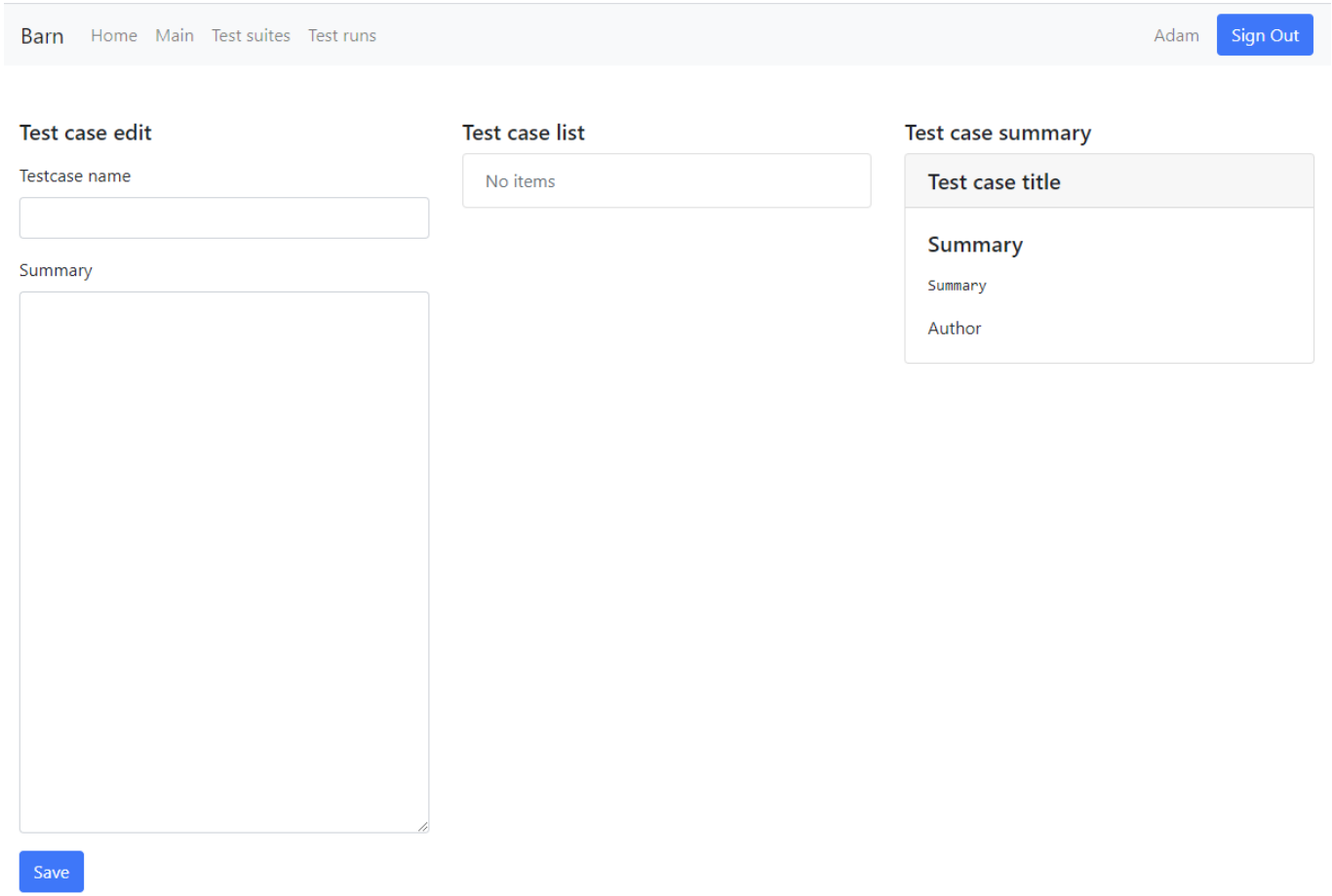

Рис. 4.10. Порожня сторінка кейсів тест-сьюта

На приведених скріншотах видно процесс створення тест кейсу

1. На першому скріншоті ми можемо бачити саму сторінку тест сьютів
2. На наступному зображено додавання нового тест-сьюу за допомогою спеціального випадаючого меню
3. На останньому скріншоті відображається сторінка тест кейсів на яку додався новий створений тест-сьют
4. При відкритті тест сьюту можна побачити що він ще пустий і в нього треба додати тест-кейси

Як видно з ходу виконання тесту користувач може додавати тест-сьюти знаходячись на відповідній сторінці через форму створення тест-сьютів. Після створення сьюту додається запис до бази данних і при подальшому використання системи тест-сьют буде підгружатись з бази даних разом з іншими сьютами користувача. Створений тест-сьют доступний для використання лише юзеру котрий

його створив. В ході виконання тесту було знайдено декілька багів котрі були пов’язані з розміткої сторінки та відкриттям форми створення тест-сьюту. Після локалізації помилки були успішно виправлені.

4.2.2. Створення тест-кейсу

Для створення тест-кейсу необхідно провести наступні дії:

1. Запустити систему локально. Це можна зробити запустивши з консолі мавен проект, після чого перейти по localhost://8080
2. Залогінити користувача. Можна створити як нового так і використати користувача з попереднього тесту
3. Створити тест-сьют. Можна як створити новий так і перевикористати з попереднього тесту
4. Клікнути по кнопці «Get cases». Ця сторінка повинна відкрити список тест-сьютів
5. Заповнити всі поля створення тест кейсу, а саме поля Summary та Description
6. Підтвердити клікнувши по кнопці Save

Очікуваний результат: відображення блока тест-кейса у списку кейсів і відображення його складу при кліку по ньому. Створення відповідного запису у базі даних. Задане форматування при створенні тест кейсу (переноси строки, відступи та табуляція) збереглося та відображається при відкритті тест кейсу. Відображення тест-кейсу лише у відповідному тест-сьюті для відповідного користувача.

Фактичний результат відображений на приведених нижче зображеннях

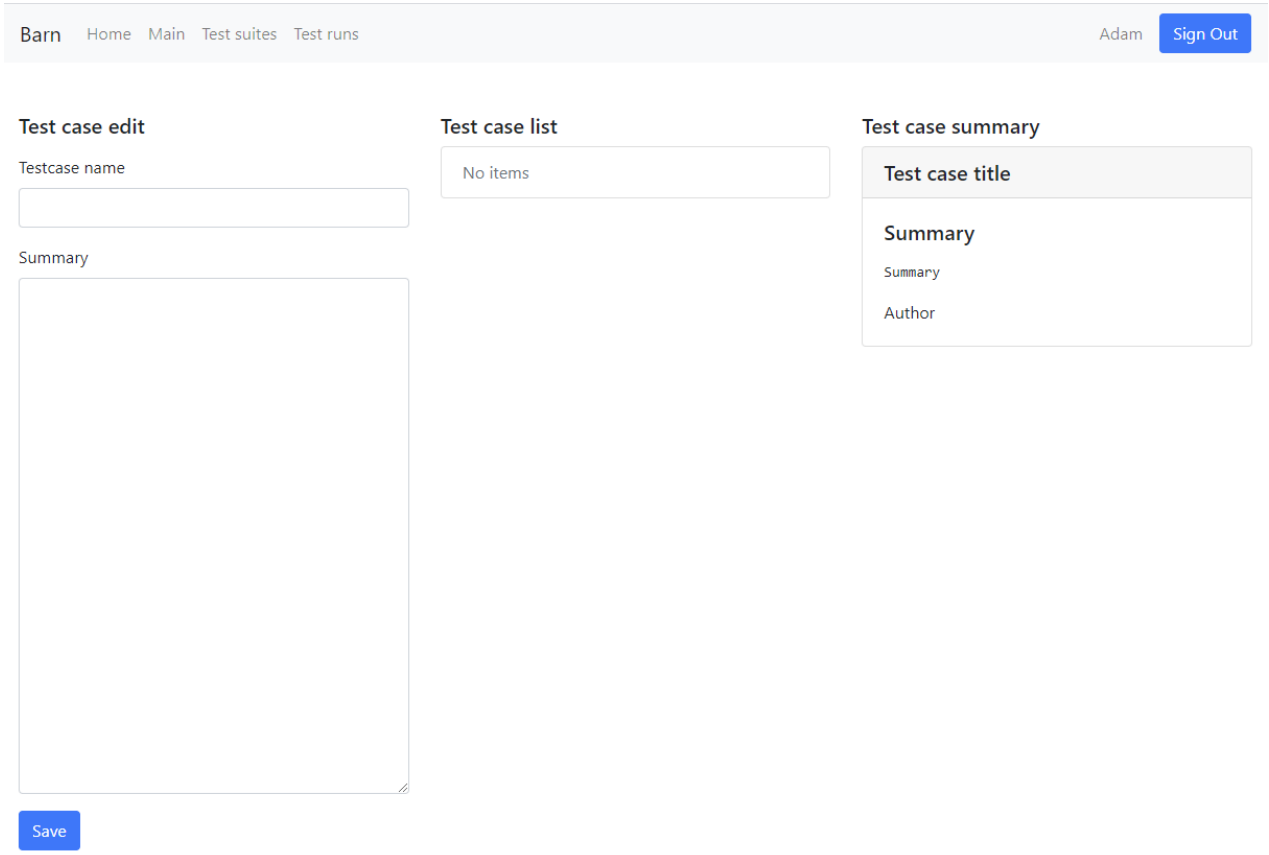

Рис. 4.11. Порожня сторінка кейсів тест-сьюта

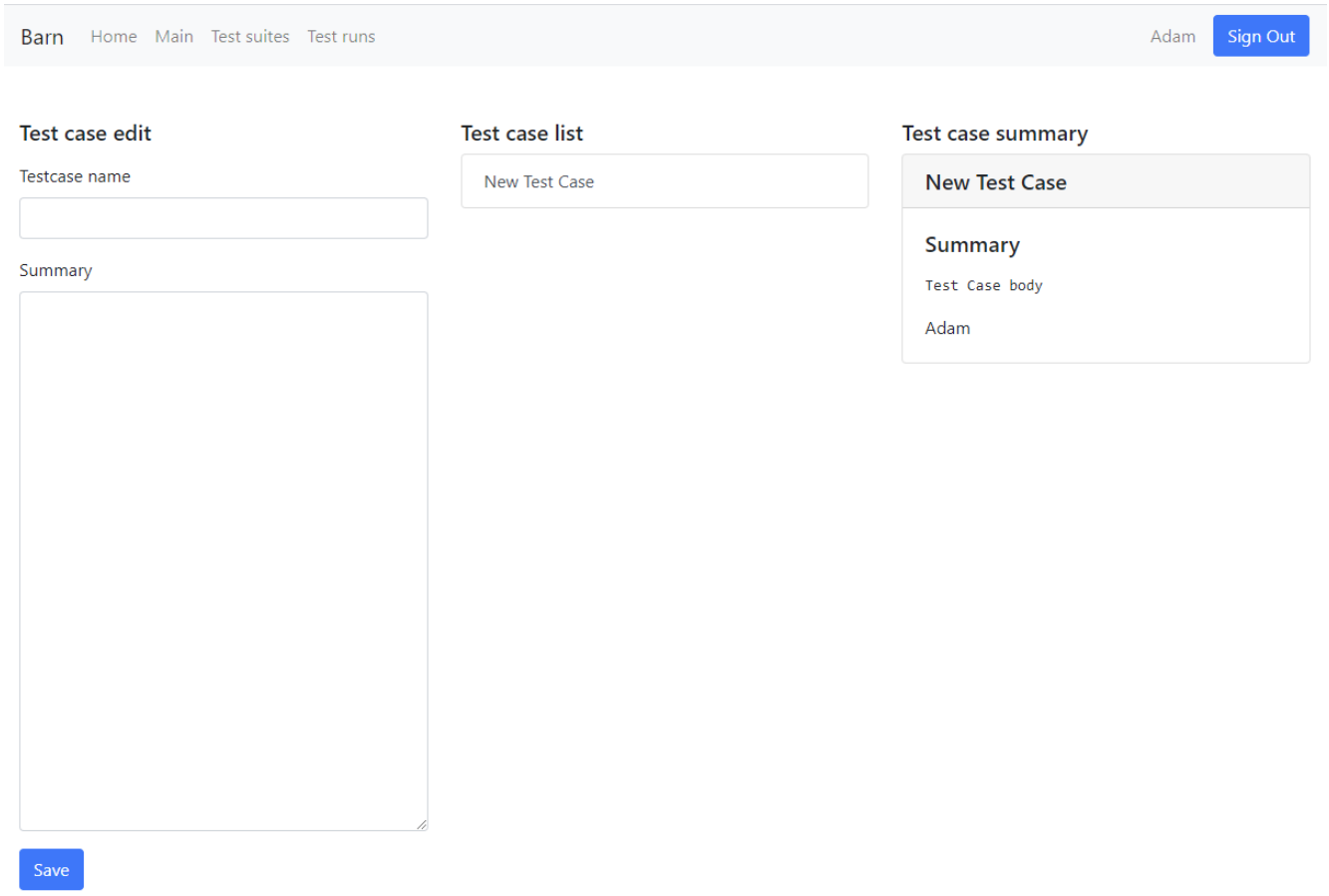

Рис. 4.12. Сторінка тест-сьюта з доданим тест-кейсом

На приведених вище зображеннях ми можемо бачити процес створення тест кейсу

1. На сторінці тест кейсів обирається (або створюється) тест сьют в який будуть додавати тест-кейс
2. У відповідні поля вводиться назва кейсу на його текст і зберігається кліком по кнопці «Save»
3. Після створення тест-кейсу він додається у відповідний список
4. Для перегляду тест кейсу необхідно клікнути по ньому

Як видно з ходу виконання тесту – тест кейс був успішно створений. Його склад можна переглянути клікнувши по ньому і у базу даних доданий відповідний запис. В ході тесту були знайдені помилки відображення тексту тест-кейсу, а саме його форматування та відображення списку тест кейсів. Знайдені помилки були локалізовані та виправлені.

4.2.3. Виконання прогону тесту

Для прогону тестів необхідно провести наступні дії:

1. Запустити систему локально запустивши мавен проект через командний рядок та відкривши локалхост у будь-якому браузері
2. Залогінити користувача. Можна використати як старого користувача з попередніх тестів так і створити нового
3. Обрати будь-який сьют. Якщо тест-сьюти відсутні необхідно створити новий
4. Клацнути по кнопці прогону тесту Run tests після чого користувача перекине на сторінку з тест кейсами
5. Напроти кожного тесту поставити статус будь-який не порожный статус (Failed, Success, Blocked)

Очікуваний результат: відображення прогрессу прогону тестів на сторінці прогонів. У базі даних створюються відповідні записи з результатами прогону і

статусами тестів. Навпроти кожного тесту зберігаються проставлені статуси і їх можна побачити перезайшовши на сторінку прогону.

Фактичний результат відображений на приведених нижче зображеннях

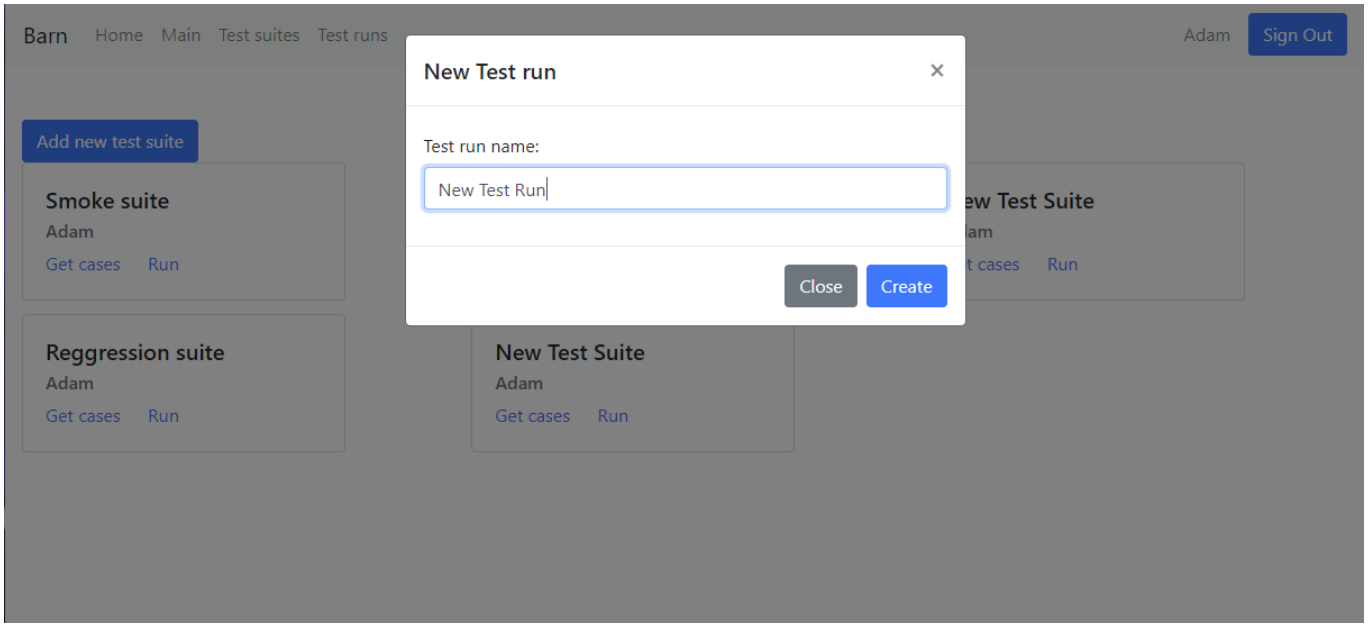

Рис. 4.13. Форма створення прогону тестів

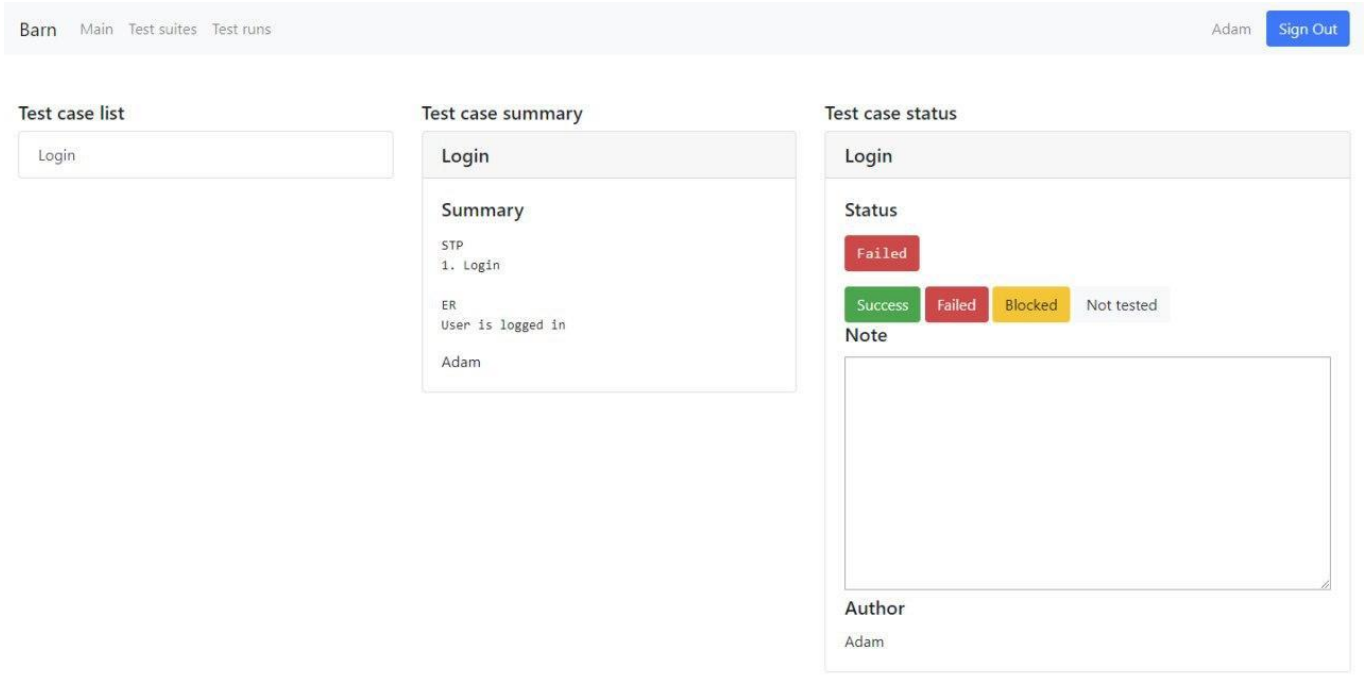

Рис. 4.14. Сторінка прогону тестів

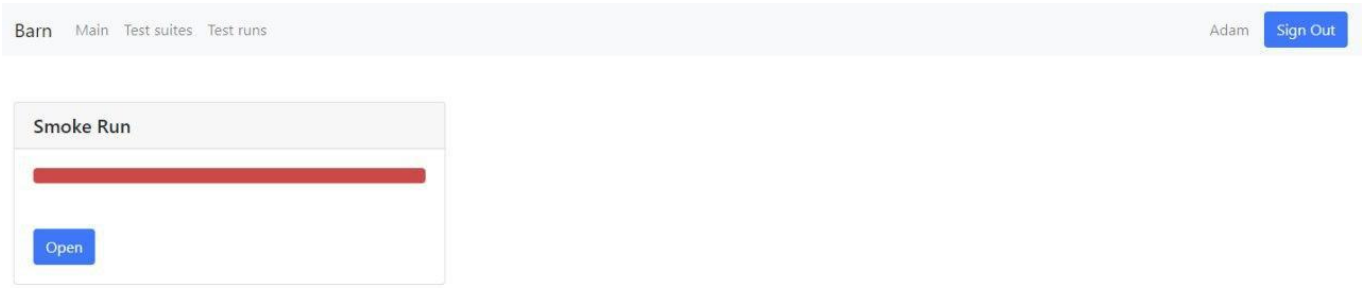

Рис. 4.15. Сторінка прогонів тестів

На приведених вище зображеннях ми можемо бачити процес створення тест кейсу

1. На сторінці тест кейсів обирається тест сьют по котрому буде виконуватись прогін тестів
2. Клікнувши по кнопці «Run» відкриється вікно у котре вводиться назва прогону тестів
3. Після створення прогону додасться відповідний блок на сторінці Test Runs із назвою прогону котру ввів користувач
4. Клікнувши по кнопці Open відповідного прогону відкриється список тест-кейсів котрі необхідно виконати. Список тест кейсів береться із тест сьюта по котрому виконується прогін
5. Клікаючи по тест-кейсам і проставляючи їм відповідні статуси виконується прогін тестів. Тестам проставляється будь-який статус або залишається дефолтний порожній статус Not tested
6. По закінченню прогону і поверненню на сторінку Test Runs на обраному блоці буде відображений прогрес прогону тестів. Прогресс прогону тестыв вираховується з усіх статусів тестів які є у даному сьюті, по замовчуванню статус тест-рану порожній

Як видно з виконання тесту – було створено і виконано прогін тестів за обраним сьютом. У тест ран були успішно перенесені тести з тест-сьюта та були занесені

відповідні данні у базу даних. Тестам проставляються статуси та підраховується статистика виконання прогону. В ході тестування функціоналу були знайдені помилки котрі були виправлені. Причиною помилок було перевикористовуваня шаблонів схожих блоків сторінок тест-сьютів та прогонів тестів.

Висновок

Були протестовані основні функції програми, такі як створення тест-кейсів, сьютів, виконання прогонів та аутентифікація.

Тестування аутентифікації показало що система виконує поставлену перед нею задачу є безпечною і не допускає отримання даних не авторизованими користувачами. Також підтримує створення і збереження персональних даних котрі доступні лише тим користувачам які ці дані створювали.

Тестування створення тест кейсів показало що система коректно записує та відображає дані котрі вводить користувач, підтримує форматування тексту і не спотворює дані введені користувачем, а отже система аутентифікації та авторизації коректно виконує поставлені задачі.

Тестування створення тест сьютів показало що система правильно і без спотворень зберігає і відображає дані котрі вводить користувач. Дані записуються в відповідні таблиці і дозволяються тест-сьютам об’єднувати тест-кейси у групи, а отже функціонал виконує поставлену задачу.

Тестування створення прогонів тестів показало що система правильно підтягує і форматує тест-кейси по обраному тест-сьюту. Прогін дозволяє маркувати тести відповідно до їх статусу після чого створюється шкала прогресу котра у процентному відношенні відображає прогрес виконання прогону, долю успішно та не успішно виконаних тестів, а отже функціонал правильно виконує поставлену задачу.

В ході виконання тестів були знайдені помилки. Більшість помилок були пов’язані з обраним стеком технологій котрий не дозволяє оновлювати контент

сторінки частково, а вимагає повного оновлення сторінки на кожний запит системи. Зважаючи на це були внесені певні зміни у структуру контролерів. Інші помилки були пов’язані як з людським фактором, так і з особливістю браузерів на котрих виконувалось тестування системи. Знайдені помилки були локалізовані та виправлені.

Система дозволяє пройти весь цикл життя тест кейсу (від створення тест сьюта до отримання результатів його прогону), а отже система виконує поставлену перед нею функціональну задачу.

Час, котрий витрачається на виконання будь-яких дій, відповідає описаним у специфікації (1-2 секунди, не враховуючи час з’єднання), тобто цілі, задані у специфікації, досягнуті.

ВИСНОВКИ

У ході виконання даної дипломної роботи було створено систему управління тест-кейсів. Для цього було проаналізовано одинадцять найпопулярніших систем подібного типу. Для порівняння була створена таблиця в котрій були сведенні данні по обраним системам. Були проаналізовані їх недоліки, переваги та особливості. З них було виокремлено основні важливі функції, властиві для кожної системи, та проаналізовано недоліки, котрі властиві цим системам.

За результатами аналізу було визначено основні характеристики, якими повинна володіти створювана система. Була розроблена специфікація до програмного продукту, що створюється. Система була задумана подібною до систем які були проаналізовані і котра реалізує їх основні функції та нехтує додатковим, вузькоспеціалізованим функціоналом. Таке рішення було прийняте з огляду на те що додатковий функціонал не завжди потрібен користувачам з огляду на вузьку спеціалізацію продукту. Додатково було прийнято рішення зробити систему з відкритим системним кодом. У майбутньому це не тільки спростить підтримку продукту, а й дозволить користувачам самостійно підтримувати, розвивати і покращувати продукт в тому напрямку котрий необхідний саме їхньому проекту. Також, як показує практика, до проектів з відкритим програмним кодом більш позитивне ставлення. Це пов’язано з тим що користувачі можуть самостійно зконфігурувати систему під себе.

У специфікації було описано необхідний функціонал, інтерфейси, протоколи та інші технологічні нюанси, котрим повинен задовольняти програмний продукт.

Система складається з серверної і клієнтських частин жодна з яких не потребує додаткових налаштувань. Такий результат досягається тим що в роботі був використаний Spring Boot. Цей фреймворк дав змогу відмовитись від мануального налаштування сервера – головної частини BackEnd сторони системи. Spring Boot використовує власний вбудований сервер Tomcat котрий запускається при старті системи. Також Spring Boot за допомогою dependency inversion системи збирає систему що також спростило розробку системи і подалший її запуск. Результатом

роботи стало те що для запуску системи достатньо розмістити створену системи на сервері та запустити її як maven застосунок.

Незалежність кліентської частини була досягнута використанням шаблонізаторів та підходу кліент-серверної архітектури з «тонким» кліентом. Використання шаблонізаторів дало змогу відмовитись від окремої розробки кліентського додатку, а розробляти його одночасно з серверною частиною. Такий підхід зекономив час та сили котрі повинні були піти на розробку кліенту, проте це також зробило кліент менш гнучким у плані модернізації та у майбутньому не дозволить перейти на використання JavaScript технологій котрі хоч і потребують окремого докладу зусиль проте всеж роблять інтерфейс користувача більш гнучким і зручним. Використання підходу з тонким кліентом дозволило зробити кліентську частину досить швидкою, але знов таки, як і з шаблонізаторами це робить графічну частину повільшой бо на кожне оновленне контенту, навіть найменше, необхідно перезавантажувати всю сторінку повністю.

У пояснювальній записці висвітлюється аспекти розроблення клієнтської та серверної частин системи.

При описанні створення клієнтської частини, тобто інтерфейс користувача – висвітлюються технології, котрі були вибрані та чому саме вони були обрані. Графічна частина була створена на основі вимог котрі були описані у специфікації.

В записці також додана мапа сайту, скріншоти сторінок та мапа створеного сайту на основі яких можна мати уявлення про роботу системи.

Окремо був задокументований процес створення серверної частини. В описі надається інформація щодо технологій використаних при розробці безпосередньо системної логіки, роботи з базою даних та аутентифікації. Також окремо описана робота кожного контролера системи. Були описані нюанси роботи в використаним стеком технологій, пояснено причини за якими були обрані саме ці технології і їх технічні особливості.

В кінці розділу надається UML діаграма системи та опис використаних контролерів. Описаний узагальнений процес роботи контролерів, взаємодії з базою

даних та роботу системи в цілому. Даний опис дозволяє отримати представлення о роботі серверної частини системи. Опис контролерів показує як працює система н

Після завершення розробки системи було проведено її тестування. З огляду на те що захищеність даних є дуже важливим компонентом подібних систем тестування аутентифікації було винесено окремо.

Тестування аутентифікації показало що система закриває дані користувачів від неавторизованих осіб, не дозволяє змінювати їх чи переглядати, а також підтверджує кожну дію що виконує користувач токенами. Проте система котр використовується на даний момент не є ідеальною так як використовує статичний ключ. Якщо система будет використовуватись у великих корпоративних системах має сенс вибрати інший метод шифрування або регулярно змінювати ключі що використовуються системою.

Для тестування безпосередньо логіки роботи з тест-кейсами було виконано прогін тестів. В ході цього тестування були створені тест сьюти, в них були додані тест кейси, після цього були створені і виконані тест кейси. В ході виконання тестування були знайдені графічні та функціональні дефекти котрі були виправлені після чого тестування було повторене ще раз. Причинами помилок більшою частиною був людський фактор та візуальне відображення на різних браузерах з чого можна зробити висновок що система стійка при розробці і придатна для подальшої модернізації. Після досягнення бажаного результату і впевневшись що помилок більше немає тестування було закінчене.

Підсумувавши, можна стверджувати що мета дипломної роботи досягнута. Розроблена система відповідає поставленим вимогам і придатна до користування. Вона виконує всі задачі, такі як створення тест-кейсів, об’єднання їх у сьюти та виконання прогонів котрі виконують аналогічні існуючі системи. Система є відкритою, а отже користувачі зможуть підстроювати її під себе по мірі необхідності. Система є безпечною і тому може використовуватися для збереження текстової документації. Проте перед наданням загального доступу до системи її необхідно більш прискіпливо протестувати на різних платформах і конфігураціях та більше уваги приділити обробленню помилок. Увагу слід приділити насамперед

різним конфігураціям серверів так як основна частина розрахунків та оброблення даних виконуються саме на сервері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація TMS Testlink. Електронний доступ. URL: http://testlink.org/
2. Документація TMS TestIT. Електронний доступ. URL: https://testit.software/ Документація TMS Zephyr. Електронний доступ. URL: https://www.getzephyr.com/
3. Документація TMS qTest. Електронний доступ. URL: https://www.qasymphony.com/software-testing-tools/qtest-manager/test-case-management/
4. Документація TMS PractiTest. Електронний доступ. URL: https://www.practitest.com/
5. Документація TMS TestLodge. Електронний доступ. URL: https://www.testlodge.com/
6. Документація TMS TestRail. Електронний доступ. URL: https://www.gurock.com/testrail
7. Документація TMS Qase. Електронний доступ. URL: https://qase.io/ /
8. Документація TMS Tematoo. Електронний доступ. URL: http://tematoo.com/
9. Документація TMS Test Collab. Електронний доступ. URL: https://testcollab.com/
10. Документація TMS HP ALM. Електронний доступ. URL: https://www.performance-lab.ru/alm
11. Документація TMS Testuff. Електронний доступ. URL: https://www.testuff.com/
12. Документація TMS XQual. Електронний доступ. URL: https://www.xqual.com/
13. Документація IDE IntelijIDEA. Електронний доступ. URL: https://www.jetbrains.com/idea/
14. Документація шаблонізатору Freemarker. Електронний доступ. URL: https://freemarker.apache.org/
15. Документація Spring Boot. Електронний доступ. URL: https://spring.io/projects/spring-boot
16. Документація серверу Tomcat. Електронний доступ. URL: http://tomcat.apache.org/

17. Документація Hibernate. Електронний доступ. URL: https://hibernate.org/
18. Документація JPA. Електронний доступ. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

ДОДАТОК А

Configs

MvcCongif

```
@Configuration
public class MvcConfig implements
WebMvcConfigurer {

  public void
addViewControllers(ViewControllerRegi
stry registry){
    registry.addViewController("/login").
setViewName("login");
  }

  @Override
  public void
addResourceHandlers(ResourceHandlerR
egistry registry){
    registry.addResourceHandler("/static/
**")
        .addResourceLocations("classpa
th:/static/");
  }
}
```

Web Security Config

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostE
nabled = true)
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {

  @Autowired
  private UserService userService;

  @Override
  protected void configure(HttpSecurity
http) throws Exception {
    http
        .authorizeRequests()
          .antMatchers( "/registration",
"/static/**").permitAll()
          .anyRequest().authenticated()
          .and()
            .formLogin()
            .loginPage("/login")
            .permitAll()
          .and()
            .logout()
            .permitAll();
  }

  @Override
  protected void
configure(AuthenticationManagerBuilder
auth) throws Exception {
    auth.userDetailsService(userService)
        .passwordEncoder(NoOpPasswo
rdEncoder.getInstance());

  }
}
```

Controllers

GreetingController

```
@Controller
public class GreetingController {

  @Autowired
  private UserRepo userRepo;

  @GetMapping("/")
  public String greeting(Model model){
    return "greeting";
  }

  @GetMapping("/registration")
  public String registration(){
    return "registration";
  }

  @PostMapping("/registration")
  public String addUser(@Valid User
user, BindingResult bindingResult, Model
model){
```

```
        User userFromDB =
userRepo.findByUsername(user.getUsern
ame());

        if(userFromDB!=null){
            model.addAttribute("message",
"User exists!");
            return "registration";
        }

        user.setActive(true);
        user.setRoles(Collections.singleton(R
ole.USER));
        userRepo.save(user);

        return "redirect:/login";
    }



}
```

TestCase Controller

```
@Controller
public class TestCaseController {

    @Autowired
    private TestCaseRepo testCaseRepo;

    @Autowired
    private UserRepo userRepo;

    @Autowired
    private TestSuiteRepo testSuiteRepo;

    @PostMapping("/main")
    public String add(
            @AuthenticationPrincipal User
user,
            @RequestParam String title,
            @RequestParam String summary,
            Model model){
        TestCase testCase = new
TestCase(title, summary, user);

        testCaseRepo.save(testCase);

        Iterable<TestCase> testCases =
testCaseRepo.findAll();

        model.addAttribute("testCases",
testCases);

        return "main";
    }



    @GetMapping("/getTC")
    public String
getTestCase(@AuthenticationPrincipal
User user,
                        @RequestParam Long
id,
                        @RequestParam Long
tcId,
                        Model model){


        Optional<TestCase> testCases =
testCaseRepo.findById(id);
        TestCase testCase = testCases.get();

        model.addAttribute("title",
testCase.getTitle());
        model.addAttribute("summary",
testCase.getSummary());
        model.addAttribute("author",
(testCase.getAuthor()!=null)?testCase.get
Author().getUsername():"");


        Iterable<TestCase> testCases1 =
testCaseRepo.findAll();

        model.addAttribute("testCases",
testCases1);
```

```
        return getTCListByTS(user, tcId,
model);
    }

    @GetMapping("/tcOfTestSuite")
    public String
getTCListByTS(@AuthenticationPrincipa
l User user, @RequestParam Long id,
Model model){
        Iterable<TestCase> testCases =
testCaseRepo.findAllByTestSuiteId(id);

        model.addAttribute("testCases",
testCases);
        return "main";
    }

    @PostMapping("/tcOfTestSuite")
    public String
addTCToTS(@AuthenticationPrincipal
User user,
                    @RequestParam Long
id,
                    @RequestParam String
title,
                    @RequestParam String
summary, Model model){

        TestSuite testSuite =
testSuiteRepo.findAllById(id);

        TestCase testCase = new
TestCase(testSuite, title, summary, user);
        testCaseRepo.save(testCase);

        Iterable<TestCase> testCases =
testCaseRepo.findAllByTestSuiteId(id);

        model.addAttribute("testCases",
testCases);
        return "main";
    }


}
```

TestRun controller

```
@Controller
@RequestMapping("/testRun")
public class TestRunController {

    @Autowired
    private TestRunRepo testRunRepo;

    @GetMapping
    public String
getTRList(@AuthenticationPrincipal
User user, Model model){
        Iterable<TestRun> testRuns =
testRunRepo.findAllByAuthor(user);
        model.addAttribute("testRuns",
testRuns);
        return "testRun";
    }

    @PostMapping
    public String
createTR(@AuthenticationPrincipal User
user,
                    @RequestParam String
testRunName,
                    @RequestParam String
testSuiteId,  Model model){

        TestRun testRun = new
TestRun(testRunName,
Long.valueOf(testSuiteId), user);
        testRunRepo.save(testRun);

        return getTRList(user, model);
    }



}
```

TestRunPageController

```
@Controller
@RequestMapping("/testRunPage")
public class TestRunPageController {

  @Autowired
  private TestRunRepo testRunRepo;

  @Autowired
  private TestCaseRepo testCaseRepo;

  @GetMapping
  public String
getTRList(@AuthenticationPrincipal
User user, @RequestParam Long id,
Model model){

      Long suiteId =
testRunRepo.findById(id).get().getSuiteI
d();
      Iterable<TestCase> testCases =
testCaseRepo.findAllByTestSuiteId(suiteI
d);

      model.addAttribute("testCases",
testCases);

      return "run";
  }


  @GetMapping("/getTC")
  public String
getTRTC(@AuthenticationPrincipal User
user, @RequestParam Long id,
@RequestParam Long tsId, Model
model){


      TestCase testCase =
testCaseRepo.findById(id).get();


      model.addAttribute("title",
testCase.getTitle());
      model.addAttribute("summary",
testCase.getSummary());
      model.addAttribute("author",
(testCase.getAuthor()!=null)?testCase.get
Author().getUsername():"");

      model.addAttribute("status",
"danger");
      model.addAttribute("status_text",
"Failed");

      Iterable<TestCase> testCases =
testCaseRepo.findAllByTestSuiteId(tsId);

      model.addAttribute("testCases",
testCases);



      return getTRList(user,
testRunRepo.findAllBySuiteId(tsId).getId
(), model);
  }



}
```

TestSuite controller

```
@Controller
@RequestMapping("/testSuite")
public class TestSuiteController {

  @Autowired
  private TestSuiteRepo testSuiteRepo;


  @GetMapping
  public String
getTestSuitesPage(@AuthenticationPrinci
pal User user, Model model){
      Iterable<TestSuite> testSuites =
testSuiteRepo.findAllByAuthor(user);
      model.addAttribute("testSuites",
testSuites);
```

```
        return "testSuite";
    }

    @PostMapping
    public String
addNewTestSuite(@AuthenticationPrinci
pal User user, @RequestParam String
name, Model model){

        TestSuite testSuite = new
TestSuite(name, user);
        testSuiteRepo.save(testSuite);

        return getTestSuitesPage(user,
model);
    }}
```

UserController

```
@Controller
@RequestMapping("/user")
@PreAuthorize("hasAuthority('ADMIN')
")
public class UserController {

    @Autowired
    private UserRepo userRepo;

    @GetMapping
    public String userList(Model model){
        model.addAttribute("users",
userRepo.findAll());
        return "userList";
    }

    @GetMapping("{user}")
    public String
userEditForm(@PathVariable User user,
Model model){
        model.addAttribute("user", user);
        model.addAttribute("roles",
Role.values());
        return "userEdit";
    }

    @PostMapping
    public String userSave(
            @RequestParam String username,
            @RequestParam Map<String,
String> form,
            @RequestParam("userId") User
user
    ) {
        user.setUsername(username);

        Set<String> roles =
Arrays.stream(Role.values()).map(Role::
name).collect(Collectors.toSet());

        user.getRoles().clear();

        for (String key : form.keySet()){
            if(roles.contains(key))
                user.getRoles().add(Role.valueO
f(key));
        }

        userRepo.save(user);
        return "redirect:/user";
    }



}
```

Domain

TestCase

```
@Entity
public class TestCase {

    @Id
    @GeneratedValue(strategy =
GenerationType.AUTO)
    private Long id;
    private String title;

    @Type(type =
"org.hibernate.type.TextType")
    private String summary;
```

```
    @ManyToOne(fetch =
FetchType.EAGER)
    @JoinColumn(name = "user_id")
    private User author;

    @ManyToOne(fetch =
FetchType.EAGER)
    @JoinColumn(name = "testSuite_id")
    private TestSuite testSuite;

    public TestCase() {}

    public TestCase(String title, String
summary, User user) {
        this.author = user;
        this.title = title;
        this.summary = summary;
    }

    public TestCase(TestSuite testSuite,
String title, String summary, User user) {
        this.testSuite = testSuite;
        this.author = user;
        this.title = title;
        this.summary = summary;
    }

    public Long getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getSummary() {
        return summary;
    }

    public void setSummary(String
summary) {
        this.summary = summary;
    }

    public User getAuthor() {
        return author;
    }

    public void setAuthor(User author) {
        this.author = author;
    }

    public TestSuite getTestSuite() {
        return testSuite;
    }

    public void setTestSuite(TestSuite
testSuite) {
        this.testSuite = testSuite;
    }
}
```

TestCaseStatus

```
public class TestCaseStatus {

    @Id
    @GeneratedValue(strategy =
GenerationType.AUTO)
    private Long id;

    private Long tcId;

    private Long testRunId;

    private Long testSuiteId;

    private String note;

    @Enumerated(EnumType.STRING)
    private Status status;



}
```

TestRun

```
@Entity
public class TestRun {

  @Id
  @GeneratedValue(strategy =
GenerationType.AUTO)
  private Long id;

  private String name;

  private Integer countOfFailed;

  private Integer countOfBlocked;

  private Integer countOfSuccess;

  private Integer countOfNotStarted;

  private Long suiteId;

  @ManyToOne(fetch =
FetchType.EAGER)
  @JoinColumn(name = "user_id")
  private User author;

  @ManyToOne(fetch =
FetchType.EAGER)
  private TestSuite testSuite;

  public TestRun(){}

  public TestRun(String name,  Long
suiteId, User author){
     this.name = name;
     this.author = author;
     this.suiteId = suiteId;
  }

  public Long getId() {
     return id;
  }

  public String getName() {
     return name;
  }

  public void setName(String name) {
     this.name = name;
  }

  public User getAuthor() {
     return author;
  }

  public void setAuthor(User author) {
     this.author = author;
  }

  public TestSuite getTestSuite() {
     return testSuite;
  }

  public void setTestSuite(TestSuite
testSuite) {
     this.testSuite = testSuite;
  }

  public Integer getCountOfFailed() {
     return countOfFailed;
  }

  public void setCountOfFailed(Integer
countOfFailed) {
     this.countOfFailed = countOfFailed;
  }

  public Integer getCountOfBlocked() {
     return countOfBlocked;
  }

  public void setCountOfBlocked(Integer
countOfBlocked) {
     this.countOfBlocked =
countOfBlocked;
  }

  public Integer getCountOfSuccess() {
     return countOfSuccess;
```

```
  }

  public void setCountOfSuccess(Integer
countOfSuccess) {
      this.countOfSuccess =
countOfSuccess;
  }

  public Integer getCountOfNotStarted()
{
      return countOfNotStarted;
  }

  public void
setCountOfNotStarted(Integer
countOfNotStarted) {
      this.countOfNotStarted =
countOfNotStarted;
  }

  public Long getSuiteId() {
      return suiteId;
  }

  public void setSuiteId(Long suiteId) {
      this.suiteId = suiteId;
  }
}
```

Test suite

```
@Entity
public class TestSuite {

  @Id
  @GeneratedValue(strategy =
GenerationType.AUTO)
  private Long id;

  private String name;

  @ManyToOne(fetch =
FetchType.EAGER)
  @JoinColumn(name = "user_id")
  private User author;

  @ManyToMany(mappedBy =
"testSuites")
  private Set<User> testers;

  @OneToMany(fetch =
FetchType.EAGER)
  private Set<TestCase> testCases;

  public TestSuite(){}

  public TestSuite(String name, User
author){
      this.name = name;
      this.author = author;
  }
  public Long getId() {
      return id;
  }

  public String getName() {
      return name;
  }

  public void setName(String name) {
      this.name = name;
  }

  public User getAuthor() {
      return author;
  }

  public void setAuthor(User author) {
      this.author = author;
  }

  public Set<TestCase> getTestCases() {
      return testCases;
  }

  public void
setTestCases(Set<TestCase> testCases) {
      this.testCases = testCases;
  }
}
```

TestSuite

```
@Entity
@Table(name = "usr")
public class User implements UserDetails
{

  @Id
  @GeneratedValue(strategy =
GenerationType.AUTO)
  private Long id;
  private String username;
  private String password;
  private boolean active;

  @ElementCollection(targetClass =
Role.class, fetch = FetchType.EAGER)
  @CollectionTable(name = "user_role",
joinColumns = @JoinColumn(name =
"user_id"))
  @Enumerated(EnumType.STRING)
  private Set<Role> roles;

  @ManyToMany(fetch =
FetchType.EAGER)
  @JoinTable(
       name = "test_suites",
       joinColumns =
{@JoinColumn(name = "user_id")},
       inverseJoinColumns =
{@JoinColumn(name = "testSuite_id")}
  )
  private Set<TestSuite> testSuites;

  public boolean isAdmin(){
    return roles.contains(Role.ADMIN);
  }

  public Long getId() {
    return id;
  }

  public String getUsername() {
    return username;
  }

  @Override
  public boolean isAccountNonExpired()
{
    return true;
  }

  @Override
  public boolean isAccountNonLocked()
{
    return true;
  }

  @Override
  public boolean
isCredentialsNonExpired() {
    return true;
  }

  @Override
  public boolean isEnabled() {
    return isActive();
  }

  public void setUsername(String name) {
    this.username = name;
  }

  @Override
  public Collection<? extends
GrantedAuthority> getAuthorities() {
    return getRoles();
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String
password) {
    this.password = password;
  }

  public boolean isActive() {
    return active;
  }
```

```
  public void setActive(boolean active) {
    this.active = active;
  }

  public Set<Role> getRoles() {
    return roles;
  }

  public void setRoles(Set<Role> roles) {
    this.roles = roles;
  }

  public Set<TestSuite> getTestSuites() {
    return testSuites;
  }

  public void saveTestSuite(TestSuite
testSuite) {
    testSuites.add(testSuite);
  }
}
```

Repos

TestCase repo

```
public interface TestCaseRepo extends
CrudRepository<TestCase, Long> {

  Iterable<TestCase>
findAllByTestSuiteId(Long id);


}
```

TestRun repo

```
public interface TestRunRepo  extends
CrudRepository<TestRun, Long> {

  Iterable<TestRun>
findAllByAuthor(User author);

  TestRun findAllBySuiteId(Long id);


}
```

Test suite repo

```
public interface TestSuiteRepo extends
CrudRepository<TestSuite, Long> {

  Iterable<TestSuite>
findAllByAuthor(User author);

  TestSuite findAllById(Long id);}
```

UserRepo

```
public interface UserRepo extends
JpaRepository<User, Long> {

  User findByUsername(String
username);

}
```

Service

Userservice

```
@Service
public class UserService implements
UserDetailsService {

  @Autowired
  private UserRepo userRepo;

  @Override
  public UserDetails
loadUserByUsername(String username)
throws UsernameNotFoundException {
      User user =
userRepo.findByUsername(username);

      if(user == null){
        throw new
UsernameNotFoundException("User not
found");
      }

      return user;
  }
}
```

Application

```
@SpringBootApplication
public class Application {

   public static void main(String[] args){
      SpringApplication.run(Application.cl
ass, args);
   }}
```