Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія для організації управління тест-кейсами»

Виконала: студентка 2-го курсу, групи 1КН-23м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Єпіфанова А.О.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
Яровий А.А.
(прізвище та ініціали)
«05» 12 2024 р.

Опонент: д.т.н., проф. каф. КСУ
Юхимчук М.С.
(прізвище та ініціали)
«05» 12 2024 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«06» 12 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
11. 10. **2024 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Єпіфановій Аліні Олегівні
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія для організації управління тест-кейсами
   керівник роботи д.т.н., проф. кафедри КН Яровий А.А.
   затверджені наказом вищого навчального закладу від "17" 09 2024 року № 310
2. Строк подання студентом роботи 11. 11. 2024р.
3. Вихідні дані до роботи:
   Мова програмування – web-орієнтовна, об'єктно-орієнтована; середовище розробки, що підтримує віртуалізацію сервера; мінімальна кількість тест-кейсів у наборах – 2; тип архітектури – клієнт-серверна.
4. Зміст текстової частини:
   Вступ, аналіз методів і засобів організації управління тест-кейсами, аналіз процесів організації управління тест-кейсами та проектування інформаційної технології, програмна реалізація інформаційної технології організації управління тест-кейсами, економічна частина, висновки, перелік використаних джерел, додатки
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
   Приклад організації тест-кейсів; узагальнений алгоритм роботи експертної системи для пріоритезації, інтегрована у систему для організації управління тест-кейсами; Узагальнений алгоритм роботи програмного продукту; робочі вікна програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконанн прийняв
1-3	Яровий А.А., д.т.н., проф. каф. КН		
4	Лесько О.Й., к.е.н., проф. каф. ЕПВМ		

7. Дата видачі завдання 02.09. 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Пр мі
1	Аналіз методів і засобів організації управління тест-кейсами. Постановка задач дослідження	02.09.24 – 17.09.24	
2	Програмна реалізація інформаційної технології	18.09.24 – 30.09.24	
3	Програмна реалізація інформаційної технології організації управління тест-кейсами та тестування	01.10.24 – 16.10.24	
4	Підготовка економічної частини	17.10.24 – 28.10.24	
5	Апробація та/або впровадження результатів дослідження	29.10.24 – 04.11.24	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	05.11.24 – 11.11.24	

Студент _____ Аліна Єпіфанова

Керівник роботи _____ Андрій Яровий

АНОТАЦІЯ

УДК 004.05

Єпіфанова А.О. Інформаційна технологія для організації управління тест-кейсами. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма – системи штучного інтелекту. Вінниця: ВНТУ, 2024. 93 с.

На укр. мові. Бібліогр.: 25 назв; рис.: 35; табл. 11.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для організації управління тест-кейсами. Проведено аналіз предметної області. Розглянуто сучасні підходи до організації управління тест-кейсами, оглянуто їх переваги та недоліки. Здійснено класифікацію підходів до тестування програмного забезпечення, розглянуто структуру тест-кейсів та їх життєвий цикл, проведено аналіз існуючих підходів до організації та управління наборами тест-кейсів. Досліджено сучасні технології експертних систем для автоматизації управління тест-кейсами. Розроблено модель експертної системи для пріоритезації тест-кейсів та алгоритм її функціонування, а також структуру функціонування інформаційної технології. Обґрунтовано вибір вибір технологічних рішень для розробки інформаційної технології управління тест-кейсами. Розглянуто процес інтеграції системи управління тест-кейсами з інструментом Jira та розроблено алгоритм роботи інформаційної технології. Проведено тестування розробленої інформаційної та її порівняння із програмами-аналогами. У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення.

Графічна частина складається з 4 плакатів.

Ключові слова: тестування ПЗ, тест-кейси, експертні системи, управління тест-кейсами.

ABSTRACT

Yepifanova A.O. Information technology for the organization of test case management. Master's thesis in the specialty 122 - artificial intelligence systems, educational program - computer science. Vinnytsia: VNTU, 2024. 93 p.

In Ukrainian language. Bibliographer: 25 titles; fig.: 35; table 11.

This master's thesis is devoted to the development of software for the organization of test case management. The subject area was analyzed. Modern approaches to the organization of test case management were considered, their advantages and disadvantages were reviewed. Approaches to software testing were classified, the structure of test cases and their life cycle were considered, and existing approaches to the organization and management of test case sets were analyzed. Modern technologies of expert systems for automating test case management were studied. An expert system model for prioritizing test cases and an algorithm for its functioning were developed, as well as the structure of the functioning of information technology. The choice of technological solutions for the development of information technology for test case management was justified. The process of integrating the test case management system with the Jira tool was considered and an algorithm for the operation of information technology was developed. The developed information technology was tested and compared with similar programs. In the economic section, the amount of costs for the development and production of a new technical solution was calculated.

The graphic part consists of 4 posters.

Keywords: software testing, test cases, expert systems, test case management.

ЗМІСТ

ВСТУП

Актуальність. У сучасному процесі розробки програмного забезпечення тестування відіграє важливу роль у забезпеченні якості продукту. Тест-кейси є основним інструментом, що дозволяє перевіряти функціональність та відповідність вимогам. Ефективне управління тест-кейсами є ключовим елементом успішного тестування, оскільки воно забезпечує контрольоване та структуроване виконання тестових сценаріїв.

Зважаючи на зростаючу складність програмного забезпечення, традиційні методи управління тест-кейсами можуть виявитися недостатньо ефективними. Відсутність централізованого зберігання тест-кейсів, нерегулярна або суб'єктивна пріоритезація та обмежені можливості автоматизації часто призводять до затримок та збільшення кількості помилок. Тому впровадження сучасних систем управління тестуванням, які дозволяють централізовано зберігати, керувати та пріоритезувати тест-кейси, є необхідністю для підтримки високої якості продукту. Одним із найбільш важливих аспектів ефективного управління є автоматизація пріоритезації тест-кейсів на основі об'єктивних критеріїв. Використання експертних систем для автоматизованого визначення пріоритетів допомагає зосередити увагу тестувальників на найбільш важливих аспектах продукту, що значно підвищує ефективність тестування та мінімізує ризики. Крім того, забезпечення доступу всіх членів команди до актуальної інформації про стан тест-кейсів через централізовані платформи сприяє покращенню співпраці, дозволяє краще контролювати процес тестування та швидше реагувати на зміни у вимогах або у продукті. Важливість цього аспекту зростає в умовах дистанційної роботи та великих команд.

Таким чином, актуальність дослідження полягає у необхідності розробки і впровадження систем, що дозволяють не лише ефективно організовувати і пріоритезувати тест-кейси, але й автоматизувати процес управління ними, що в кінцевому результаті забезпечує більш якісний і швидкий цикл тестування та розробки програмного забезпечення.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп'ютерних наук Вінницького національного технічного університету 22 К1 «Розробка прикладних інтелектуальних інформаційних технологій та систем» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання досліджень. Метою даної магістерської роботи є розширення функціональних можливостей інформаційної технології для організації управління тест-кейсами.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- здійснити аналіз сучасних технологій для організації тест-кейсів;
- здійснити порівняльний аналіз програм аналогів для організації тест-кейсів;
- здійснити проектування інформаційної технології для організації тест-кейсів;
- розробити алгоритм функціонування інформаційної технології для організації тест-кейсів;
- здійснити програмну реалізацію запропонованої інформаційної технології;
- провести тестування програмного засобу та проаналізувати отримані результати.

Об'єкт дослідження − процес тестування програмного забезпечення.

Предмет дослідження – програмні засоби для організації управління тест-кейсами.

Методи дослідження. У роботі використані такі методи наукових досліджень: методи та моделі подання знань, методи математичного моделювання, методи автоматизованого тестування програмних засобів, зокрема, функціональне тестування та тестування методом «чорного ящика», методи та техніки тест-дизайну, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів.

Запропоновано інформаційну технологію для організації управління тест-кейсами, яка відрізняється від існуючих впровадженням експертної системи для пріоритезації тест-кейсів та інтеграцією із системою управління проектами Jira для автоматичного змінювання статусу тест-кейсів, що забезпечило розширення функціональних можливостей інформаційної технології при тестуванні ПЗ.

Практичне значення одержаних результатів полягає у створенні програмного забезпечення для організації та управління тест-кейсами, яке дозволяє скоротити час і ресурси, потрібні для забезпечення якості ПЗ.

Запропонована інформаційна технологія сприяє підвищенню ефективності процесу організації управління тест-кейсами:

- розроблено експертну систему для автоматизації пріоритезації тест-кейсів;
- розроблено алгоритм функціонування інформаційної технології для організації управління тест-кейсами;
- розроблено програмні засоби для організації управління тест-кейсами.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології для організації управління тест-кейсами.

Особистий внесок здобувача. Усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація результатів роботи. Основні результати досліджень апробовано на міжнародній науково-практичній конференції "Молодь в науці: дослідження, проблеми, перспективи" (МН-2025) [1].

Публікації. За результатами досліджень опубліковано тези доповідей на науково-технічній конференції [1].

1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ ОРГАНІЗАЦІЇ УПРАВЛІННЯ ТЕСТ-КЕЙСАМИ

1.1 Аналіз сучасних методів організації управління тест-кейсами

Організація та управління тест-кейсами – це складний і багатогранний процес, який потребує впровадження сучасних методів та інструментів для забезпечення максимальної ефективності тестування. Зважаючи на зростаючу складність програмних продуктів, ефективне управління тестуванням стає невід'ємною частиною розробки. Існує кілька ключових підходів, які дозволяють оптимізувати цей процес та забезпечити надійність результатів [2].

Перший підхід – це використання систем для організації управління тест-кейсами. Одним із найбільш важливих та продуктивних підходів до організації тест-кейсів є впровадження спеціалізованих систем управління тестовими сценаріями. Ці системи забезпечують централізоване зберігання, управління та моніторинг тест-кейсів, що дозволяє мінімізувати можливі розбіжності в роботі різних членів команди. Такі інструменти також полегшують контроль над виконанням тестів та допомагають уникнути втрати критичної інформації, пов'язаної з тестуванням.

Однією з основних переваг таких систем є можливість інтеграції з іншими інструментами розробки ПЗ, такими як Jira. Це значно спрощує процес управління дефектами, відстеження прогресу тестування та комунікацію між тестувальниками і розробниками. Інші переваги включають автоматизацію рутинних операцій, таких як створення звітів і відстеження статусу виконання тест-кейсів. Усі ці функції підвищують прозорість і контрольованість процесу тестування, знижуючи операційні ризики та підвищуючи ефективність усієї команди.

Ще одним фундаментальним елементом організації тест-кейсів є їх грамотна пріоритезація. У реальних умовах обмеженого часу та ресурсів не завжди можливо виконати всі тести, тому визначення пріоритетів дозволяє

зосередитися на найбільш важливих аспектах системи, що мають критичне значення для користувачів або бізнесу.

Пріоритезація тест-кейсів базується на різних факторах, зокрема [3]:

- критичність функціоналу для основної роботи системи;
- імовірність появи дефектів у конкретному компоненті.
- рівень впливу помилок на бізнес або кінцевого користувача.

Наприклад, тест-кейси, що перевіряють критичні бізнес-функції, отримують найвищий пріоритет і виконуються в першу чергу. Це дозволяє знизити ризики критичних помилок на пізніх етапах розробки та уникнути потенційних затримок у випуску продукту через недопрацьовані важливі компоненти. Пріоритезація також дозволяє більш раціонально розподіляти ресурси команди, зосереджуючи зусилля на найважливіших частинах продукту.

Модульний підхід до тестування. Цей метод передбачає поділ системи на окремі функціональні модулі та створення для кожного з них відповідного набору тест-кейсів. Модульний підхід значно спрощує управління тест-кейсами, оскільки дозволяє легко локалізувати і відстежувати тести для конкретних функціональних частин системи. Це особливо корисно у великих проектах з високим рівнем складності, де модульність забезпечує гнучкість та масштабованість процесу тестування.

Крім того, модульний підхід полегшує підтримку тестової документації: у разі змін у певному модулі потрібно оновлювати лише відповідні тести, що зменшує загальні витрати на підтримку тестових сценаріїв.

Використання автоматизованих тестів для рутинних або повторюваних сценаріїв є одним із ключових методів підвищення продуктивності команди та зменшення навантаження на тестувальників. Автоматизоване тестування є особливо ефективним для регресійного тестування, коли зміни в коді мають бути перевірені на відповідність попереднім вимогам без необхідності повторного ручного виконання всіх тестів.

Завдяки автоматизації можна швидко виконувати великий обсяг тестів, зокрема нічні або стрес-тести, що дозволяє команді зосередитися на складніших

завданнях, таких як проектування нових тест-кейсів або аналіз результатів. Крім того, автоматизація дозволяє підвищити повторюваність тестів, зменшуючи ймовірність помилок, які можуть виникати під час ручного тестування.

1.2 Огляд існуючих програмних рішень для організації управління тест-кейсами

Сьогодні на ринку доступні безліч програмних рішень для організації управління тест-кейсами, кожне з яких має свої переваги та недоліки. Найпопулярніші інструменти в цій сфері – це TestRail, qTest, Zephyr та TestLink. Розглянемо дані програми більш детально та проведемо їх порівняння.

TestRail – це інструмент для управління тест-кейсами, який сприяє організації та виконанню тестування програмного забезпечення [4]. Основною метою TestRail є підвищення якості продукту та ефективності процесу тестування. Програма надає можливість створювати тест-плани та тест-кейси, відстежувати їх виконання, отримувати результати та генерувати детальні звіти. Крім того, TestRail дозволяє візуалізувати результати у вигляді діаграм та графіків, що полегшує аналіз тестування. Інтеграція з іншими інструментами тестування, такими як Jira, значно спрощує управління дефектами та забезпечує зручний обмін інформацією між тестувальниками і розробниками.

Попри свої переваги, TestRail має такі недоліки:

- Вартість. Це комерційний продукт, що може стати перепоною для невеликих команд або компаній з обмеженим бюджетом.
- Складність налаштування. Для ефективного розгортання потрібні певні технічні знання.
- Обмежена масштабованість. При великій кількості тест-кейсів продуктивність системи може знижуватись.

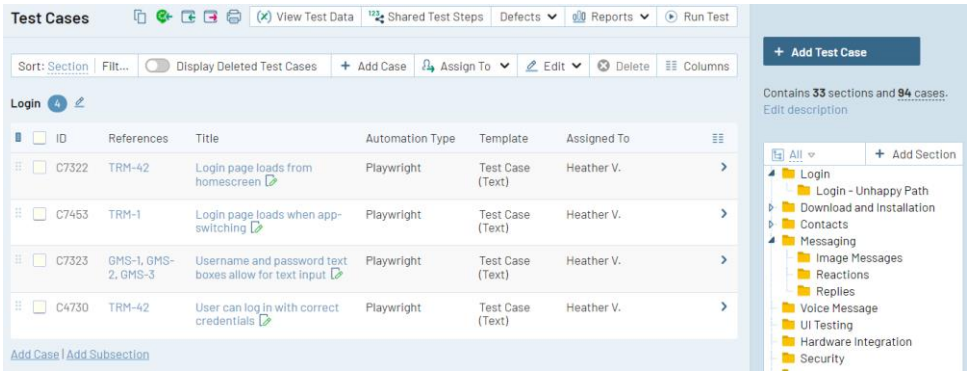


Рисунок 1.2 – Вигляд програми для управління тест-кейсами «TestRail»

qTest – це інший популярний інструмент для керування тест-кейсами, який дозволяє створювати тестові сценарії з детальним описом кожного етапу тестування та пов'язаними з ними даними [5]. Він також підтримує керування версіями тест-кейсів, що дозволяє відстежувати зміни та зберігати їх історію. qTest надає інструменти для організації, фільтрування та пошуку тест-кейсів, а також інтеграцію з іншими інструментами автоматизації тестування, що дозволяє переносити результати на інші платформи для аналізу та звітності.

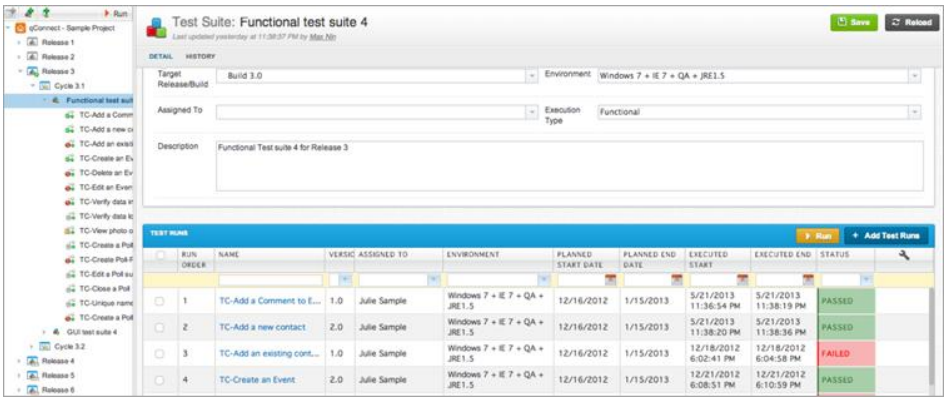


Рисунок 1.3 – Вигляд програми для управління тест-кейсами «qTest»

Основні недоліки qTest:

– Вартість. Цей інструмент також є комерційним, що може створювати фінансові труднощі для невеликих команд.

– Складність налаштування. Як і у випадку з TestRail, встановлення та налаштування даного програмного продукту може бути складним для

користувачів з обмеженим досвідом в сфері програмного забезпечення. Потрібні певні технічні знання для ефективного розгортання та налаштування системи.

– Обмежена масштабованість. Велика кількість користувачів може викликати проблеми з продуктивністю.

Zephyr – ще один популярний інструмент для управління тест-кейсами, що надає широкий функціонал для ефективного тестування ПЗ [6]. Основні функції Zephyr включають створення та управління тест-кейсами, планами тестування, наборами тестів та звітами. Програма дозволяє виконувати автоматизоване тестування та інтегрувати результати з іншими інструментами автоматизації.

Попри свої переваги, Zephyr має схожі до TestRail та qTest недоліки:

– Вартість. Це комерційний продукт з відповідними фінансовими витратами.

– Складність налаштування. Система вимагає технічних знань для налаштування.

– Інтерфейс користувача. Деякі користувачі вважають інтерфейс менш інтуїтивним порівняно з іншими системами. Для ефективного використання програмним продуктом знадобиться досвід.

– Обмежена масштабованість. Великий обсяг даних може впливати на продуктивність.

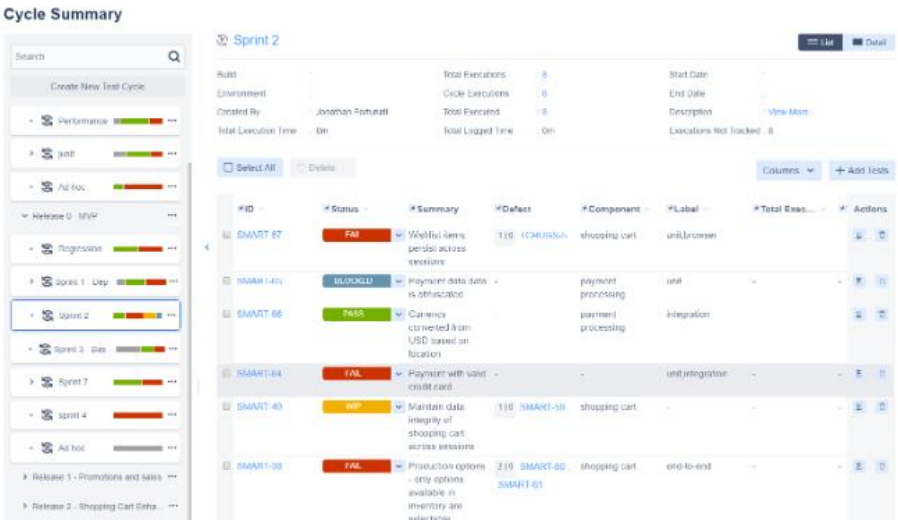


Рисунок 1.4 – Вигляд програми для управління тест-кейсами «Zephyr»

TestLink – це безкоштовна система управління тест-кейсами, яка надає базові можливості для організації тестування [7]. Вона підтримує інтеграцію з такими інструментами, як Bugzilla та Jira, що дозволяє створювати зв'язки між тест-кейсами та дефектами. TestLink також надає звіти про результати тестування, включаючи кількість успішних та невдалих тестів.

Основні недоліки TestLink:

– Складність налаштування. Для налаштування та використання системи потрібні технічні знання.

– Обмежений інтерфейс. Інтерфейс виглядає застарілим і не таким сучасним, що може ускладнити роботу з ним.

– Відсутність автоматизації. TestLink не надає вбудованих можливостей для автоматизованого виконання тестів

– Недостатня підтримка. Оскільки це відкрите програмне забезпечення, користувачам може бути складно отримати підтримку.

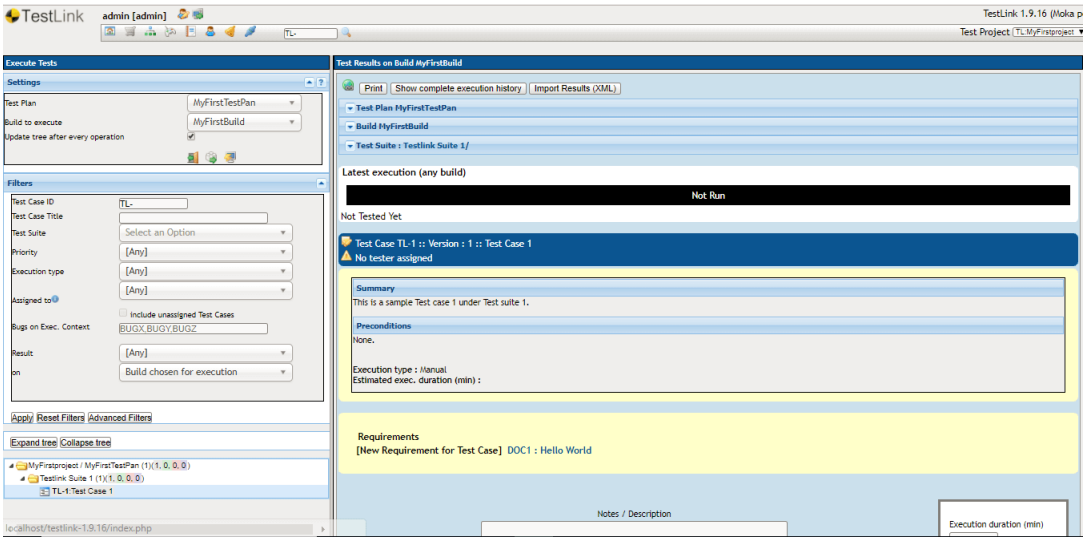


Рисунок 1.5 – Вигляд програми для управління тест-кейсами «TestLink»

Щоб обрати найбільш підходящий засіб для моєї предметної області, на якому буде базуватись інформаційна технологія для організації управління тест-кейсами, створимо таблицю, де будуть оцінені певні характеристики засобів, де «+» – це найкращий показник, «-» – найгірший, «+/-» – середній.

Таблиця 1.1 – Порівняльна характеристика програм-аналогів

	Зручність інтерфейсу	Ціна	Функціональність	Швидкість роботи
TestRail	+	-	-	+/-
qTest	+	-	+	+/-
Zephyr	+/-	+/-	+	+
TestLink	-	+	+/-	-

Отже, підсумовуючи результати, можна дійти до висновку, що TestRail є найменш ефективним інструментом через високу вартість, складність налаштування та обмежений функціонал у порівнянні з іншими інструментами. Zephyr забезпечує найкращий баланс між функціональністю та зручністю використання, незважаючи на зазначені недоліки. Він є доволі зручним, має більш обширний функціонал у порівнянні із іншими системами.

1.3 Перспективи використання методів і засобів штучного інтелекту в тестуванні програмного забезпечення

В умовах стрімкого розвитку інформаційних технологій зростають вимоги до швидкості та якості розробки програмного забезпечення. Традиційні методи тестування, що базуються переважно на ручній праці та класичних підходах, часто стають перешкодою для швидкого випуску продукту. Застосування штучного інтелекту (ШІ) у тестуванні програмного забезпечення стає ключовим елементом для підвищення ефективності та надійності тестування [8].

Однією з найбільших переваг використання ШІ є його здатність до автоматизації рутинних і повторюваних завдань, що вимагають значних людських ресурсів. Завдяки алгоритмам машинного навчання тестувальники можуть зосередитися на більш важливих завданнях, таких як стратегічне планування та аналіз результатів. Наприклад, генерація тест-кейсів або створення тестової документації, що зазвичай займає значний час, може бути

виконана ШІ за лічені хвилини, враховуючи всі можливі комбінації вхідних даних та умов.

Крім того, ШІ здатний виявляти приховані закономірності в даних, що дозволяє тестувальникам передбачати потенційні проблеми ще до того, як вони виникнуть. Це забезпечує не лише економію часу, але й підвищує рівень безпеки та надійності програмного забезпечення. Сучасні системи тестування на базі ШІ можуть самостійно адаптуватися до змін у коді, що є важливим у світі швидкої розробки, де зміни часто вносяться в останню хвилину.

З іншого боку, впровадження ШІ не позбавлене викликів. По-перше, необхідність у високоякісних даних для навчання моделей є ключовим фактором, від якого залежить ефективність роботи ШІ. Якщо навчальні дані містять помилки або є недостатньо репрезентативними, результати тестування можуть бути недостовірними. По-друге, процес впровадження ШІ в тестування може бути складним і дорогим, особливо для організацій, що не мають достатнього досвіду в роботі з новітніми технологіями.

Ще одним аспектом, який варто розглянути, є проблема прозорості роботи ШІ. Багато алгоритмів діють як «чорні скриньки», що означає, що користувачі часто не розуміють, як саме ШІ приймає рішення. Це може створити певний рівень недовіри та ускладнити процес перевірки правильності роботи системи.

Незважаючи на ці виклики, перспективи використання ШІ у тестуванні виглядають багатообіцяючими. З розвитком технологій машинного навчання і обробки природної мови ШІ поступово стає більш адаптивним та інтегрованим з існуючими процесами розробки. Використання ШІ у поєднанні з хмарними технологіями дозволяє створювати масштабовані рішення, які можна легко адаптувати під специфічні потреби проекту.

Погляд на майбутнє свідчить про те, що впровадження ШІ стане нормою у тестуванні програмного забезпечення. Технології, що використовують аналітику великих даних, будуть здатні аналізувати мільйони рядків коду, прогнозувати потенційні ризики та самостійно оновлювати тестові сценарії у відповідь на зміни у вимогах або архітектурі. Це створить умови для значної економії ресурсів

та часу, підвищуючи при цьому рівень задоволеності кінцевих користувачів і забезпечуючи конкурентні переваги на ринку.

Зокрема, перспективним вважається застосування експертних систем як напрямку штучного інтелекту для вирішення задачі організації управління тест-кейсами. Експертні системи показують найбільшу ефективність у сферах або предметних областях, де більшість задач є слабоформалізованими, і важливим є застосування досвіду експертів. Особливо у процесах, які потребують обробки великої кількості даних або прийняття рішень на основі складних критеріїв.

1.4 Постановка задачі організації управління тест-кейсами

Задача організації управління тест-кейсами є важливим та невід'ємним аспектом розробки програмних продуктів, забезпечення якості програмного забезпечення (ПЗ) та має велике практичне значення, оскільки тестування та правильна організація тестової документації надають змогу виявляти помилки, перевіряти функціональність та надійність системи ще до її впровадження.

Тестування в сфері розробки ПЗ – це процес перевірки відповідності поведінки програмного продукту встановленим вимогам [9]. Воно дає змогу виявити можливі помилки або невідповідності на ранніх етапах, що дозволяє оптимізувати витрати на виправлення дефектів і підвищити надійність продукту. Належним чином організований процес тестування забезпечує стабільну роботу продукту в умовах експлуатації, задоволення потреб користувачів та економію часу та ресурсів.

Незважаючи на значні витрати на тестування, необхідно враховувати, що дефекти можуть виникати на будь-якому етапі розробки. Важливим є те, що ціна виправлення помилок зростає експоненційно на пізніших етапах життєвого циклу ПЗ [10]. Тому тестування повинно бути організоване так, щоб виявляти максимальну кількість дефектів на ранніх стадіях розробки .

На рисунку 1.6 показано, як змінюється вартість виправлення помилки на різних етапах розробки програмного продукту, а також відсоток дефектів, що

виникають на кожному етапі та відсоток дефектів, які зазвичай знаходяться на кожному етапі.

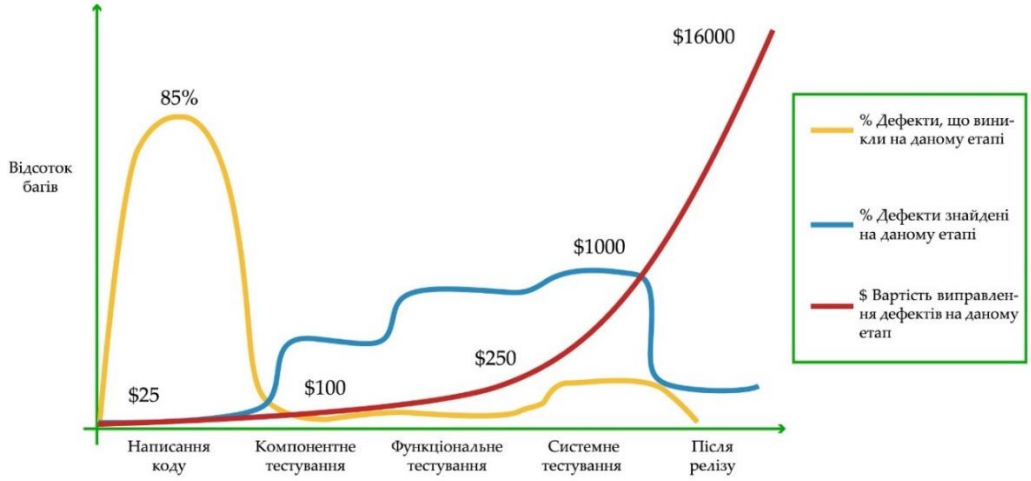


Рисунок 1.6 – Зміна вартості виправлення дефектів та кількість дефектів на різних етапах розробки ПЗ

Таким чином, виявлення дефектів на ранніх стадіях є значно економічнішим, ніж їх виявлення на фінальних етапах розробки.

Однак, однією з основних проблем, з якими стикаються команди тестувальників, є правильна організація процесу тестування. Тест-кейси дозволяють структурувати цей процес, забезпечуючи повторюваність та систематичність перевірок, що особливо важливо для великих програмних проектів. Проте, для досягнення максимальної ефективності важливо не лише створювати тест-кейси, а й організовувати їх оптимальним чином, враховуючи обмежені ресурси та час. Тому виникає необхідність у розробці системи, яка допоможе організувати цей процес.

У даній роботі необхідно реалізувати інформаційну технологію для організації управління тест-кейсами, що дозволить здійснити експертну пріоритезацію тест-кейсів, автоматизацію процесу управління тест-кейсами, розширення функціональних можливостей. Така система повинна забезпечувати:

- Централізоване зберігання тест-кейсів. Усі тест-кейси повинні бути зібрані в одному місці, щоб забезпечити доступ для всіх учасників процесу тестування.
- Гнучке планування та виконання тестів. Система повинна дозволяти створювати тестові плани, розподіляти тест-кейси по пріоритетах, встановлювати послідовність їх виконання.
- Пріоритезацію тест-кейсів. У реальних умовах обмеженого часу і ресурсів важливо визначати, які тест-кейси є найбільш критичними для перевірки у першу чергу. Впровадження експертної системи для автоматизації цього процесу є необхідним кроком до підвищення ефективності тестування.
- Відслідковування виконання та результатів тестів. Система повинна дозволяти фіксувати статуси виконання тест-кейсів, зберігати їх результати для подальшого аналізу та звітування.
- Інтеграцію з сторонніми інструментами. Ефективне управління тест-кейсами вимагає інтеграції з системами відстеження дефектів (наприклад, Jira), що дозволяє зберігати зв'язок між тест-кейсами та виявленими дефектами.

Таким чином, організація управління тест-кейсами має на меті створення такої системи, яка дозволить планувати, виконувати та контролювати процес тестування програмного забезпечення. Особливий акцент робиться на автоматизації пріоритезації тест-кейсів, що сприятиме підвищенню якості кінцевого продукту та ефективності роботи тестувальної команди.

1.5 Висновок до розділу 1

У першому розділі проведено аналіз предметної області організації управління тест-кейсами. Розглянуто сучасні підходи до організації управління тест-кейсами та сформульовано постановку задачі.

Проведено огляд сучасних методів організації управління тест-кейсами, зокрема використання спеціалізованих систем для організації тестових сценаріїв та пріоритезацію тест-кейсів, таких як TestRail, qTest, Zephyr та TestLink. Для кожного з інструментів описано основні переваги та недоліки, що дозволило оцінити їх придатність для різних потреб команд тестування. Проведено порівняння даних інструментів на основі їх зручності, функціональності, вартості та швидкодії.

Крім того, розглянуто можливості використання експертних систем для оптимізації процесу управління тест-кейсами. Експертні системи виявилися перспективним інструментом, що може полегшити пріоритизацію тест-кейсів на основі ризиків, аналізу історичних даних, оптимального використання ресурсів команди та інших даних.

2 АНАЛІЗ ПРОЦЕСІВ ОРГАНІЗАЦІЇ УПРАВЛІННЯ ТЕСТ-КЕЙСАМИ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

2.1 Класифікація підходів до тестування програмного забезпечення

Тестування програмного забезпечення можна класифікувати за різними критеріями, що дозволяє адаптувати його до цілей та особливостей проекту. Загалом тестування поділяють на функціональне та нефункціональне.

Функціональне тестування – це один із найважливіших видів тестування, метою якого є перевірка того, чи відповідає програмне забезпечення функціональним вимогам [9]. Функціональне тестування проводиться на основі специфікацій та вимог користувача та охоплює такі аспекти, як перевірка коректності обробки введених даних, взаємодія з іншими системами та коректність виведення даних.

Нефункціональне тестування перевіряє аспекти якості програмного забезпечення, які не пов'язані з його основною функціональністю. Основна мета цього тестування – оцінити продуктивність, безпеку, зручність використання та інші нефункціональні характеристики. Воно зазвичай проводиться для того, щоб забезпечити відповідність програмного забезпечення стандартам якості, а також визначити його поведінку за різних умов.

Основні перевірки нефункціонального тестування включають [9]:

- Тестування продуктивності. Оцінюється швидкість реагування системи та її стабільність при великому навантаженні.
- Тестування зручності використання. Перевіряє, наскільки зручним і зрозумілим є інтерфейс для кінцевих користувачів, дозволяючи покращити користувацький досвід.
- Тестування безпеки. Перевіряє, наскільки добре система захищена від можливих атак, перевіряє механізми автентифікації, авторизації та безпеку даних.

– Сумісність. Оцінює здатність програмного забезпечення працювати в різних середовищах, операційних системах, браузерах і на різних пристроях.

– Надійність. Перевіряє стабільність та надійність системи при тривалому використанні.

Основні підходи до класифікації функціонального тестування представлені на рисунку 2.1 [11].

Рисунок 2.1 – Спрощена класифікація тестування

Розглянемо детальніше кожен із даних типів [11]:

За запуском коду на виконання тестування поділяють на статичне та динамічне. Статичне тестування – це процес перевірки програмного забезпечення без запуску коду на виконання. Основні методи включають перевірку коду вручну або за допомогою спеціальних інструментів, огляд та аналіз документації. Динамічне тестування – це процес тестування програмного забезпечення з виконанням коду для перевірки функціональної поведінки.

За рівнем деталізації виділяють модульне, системне, інтеграційне та приймальне тестування. Модульне або компонентне тестування передбачає перевірку окремих частин програмного забезпечення – модулів, функцій чи методів. Інтеграційне тестування проводиться після модульного та перевіряє

взаємодію між кількома модулями та компонентами системи. Системне тестування перевіряє всю систему як єдине ціле. Приймальне тестування проводиться на завершальному етапі перед випуском продукту, щоб переконатися, що програмне забезпечення відповідає вимогам замовника та готове до використання. Цей вид тестування часто виконується кінцевими користувачами або представниками замовника.

За техніками автоматизації тестування поділяється на ручне та автоматичне. Ручне тестування передбачає виконання тест-кейсів тестувальником вручну, а в автоматизованому використовуються спеціалізовані інструменти та скрипти для автоматизації виконання тестів.

За доступом до коду та архітектури програмного продукту виділяють метод білої скриньки, коли тестувальник має доступ до внутрішньої структури коду та використовує ці знання для розробки тестів; метод чорної скриньки, при якому тестувальник не має доступу до внутрішньої структури програмного забезпечення і перевіряє тільки його зовнішню функціональність; та метод сірої скриньки, що являє собою комбінацію методів білої та чорної скриньки, де тестувальник має часткове уявлення про внутрішню структуру системи, але переважно працює із зовнішнім тестуванням.

За цілями та задачами виділяють димове, регресійне та санітарне тестування. Димове тестування – це поверхнева перевірка основних функціональних можливостей програмного забезпечення для підтвердження його стабільності перед проведенням більш детального тестування. Його мета – переконатися, що програмне забезпечення не ламається відразу після запуску, і його основні функції працюють. Регресійне тестування – це тестування програмного забезпечення після внесення змін у код для того, щоб переконатися, що нові зміни не порушили існуючу функціональність. Воно виконується після виправлення дефектів або додавання нових функцій для підтвердження стабільності системи. Санітарне тестування – це вузько спрямоване тестування, яке проводиться для перевірки коректності роботи конкретної функціональності після внесення змін у програмний код. Воно є частиною регресійного тестування

та виконується для швидкої перевірки того, що певна функція працює належним чином після виправлення або оновлення.

За принципом роботи з програмним продуктом тестування поділяють на позитивне та негативне. Позитивне тестування – це процес перевірки програмного забезпечення з використанням коректних даних і очікуваних сценаріїв для того, щоб переконатися, що система працює відповідно до специфікацій. Мета полягає у перевірці того, що система функціонує так, як передбачалося, за нормальних умов. Негативне тестування – це процес перевірки системи з використанням некоректних, несподіваних або неприпустимих вхідних даних для виявлення того, як програма обробляє помилки та несподівані ситуації. Метою є перевірка надійності системи в умовах, коли користувачі вводять дані, що виходять за межі очікуваних сценаріїв.

Класифікація тестування за наведеними ознаками дозволяє створити комплексний підхід до перевірки програмного забезпечення, що враховує всі аспекти його функціональності та якості.

2.2 Аналіз процесу організації управління тест-кейсами

Управління тест-кейсами є одним із найважливіших аспектів процесу розробки програмного забезпечення та ключовою складовою діяльності тестувальників та інженерів із забезпечення якості. Ефективне керування тест-кейсами забезпечує низку переваг для процесу тестування та всього життєвого циклу розробки програмного забезпечення. Серед основних причин, чому управління тестовими випадками є необхідним можна виділити такі [12]:

– Забезпечення повноти тестування. Створення тест-кейсів, а також їх правильна організації дозволяють переконатися, що всі важливі аспекти програмного забезпечення перевірено, а потенційні проблеми виявлено та вирішено ще до виходу продукту на ринок. Це також гарантує, що всі тестові сценарії виконані та задокументовані, що підвищує загальну якість тестування.

– Підвищення ефективності. Організація та зберігання тест-кейсів сприяють підвищенню ефективності процесу тестування. Це дозволяє тестувальникам швидко знайти необхідні тестові дані та пройти відповідні тест-кейси без потреби витрачати час на створення нових кейсів або пошук попередніх сценаріїв.

– Сприяння співпраці. Ефективне керування тест-кейсами полегшує співпрацю між усіма членами команди розробки. Окрім інженерів із забезпечення якості, тест-кейси можуть бути використані розробниками для перевірки, що написаний програмний код відповідає встановленим вимогам і стандартам.

– Полегшення регресійного тестування. Наявність готових та структурованих тест-кейсів спрощує процес регресійного тестування, забезпечуючи стабільність продукту при внесенні змін.

– Документування результатів тестування. Управління тест-кейсами надає чітку та структуровану документацію про виконані тести та їх результати. Ця документація є корисною для подальшого аналізу та може використовуватися для підготовки звітів, а також як база знань для майбутніх ітерацій розробки.

Організація тест-кейсів часто передбачає їх структурування у вигляді наборів, кожен з яких відповідає певній функціональності програмного продукту. Для великих проектів, де є багато окремих функціональних блоків, кожен з них може бути розділений за цілями тестування (рис. 2.2).

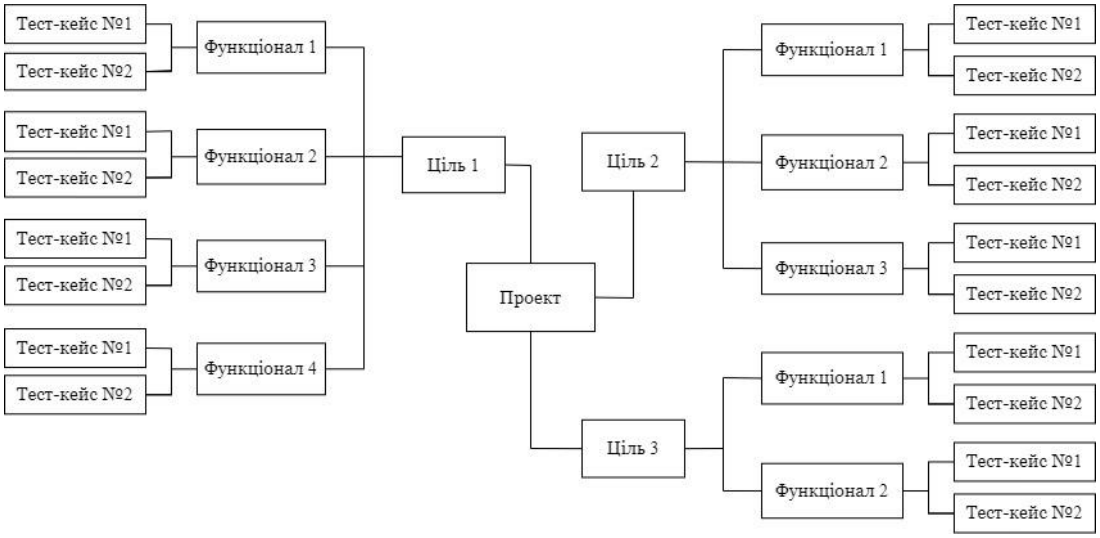


Рисунок 2.2 – Приклад організації тест-кейсів

Подібна організація виглядає як ієрархічна структура тек, у кожній з яких містяться підтеки, що відповідають конкретному функціоналу. Наприклад, для тестування сторінки входу на сайт можна створити окремі тест-кейси для різних функцій, таких як процес логіну, реєстрація нового користувача та відновлення паролю. Цей підхід дозволяє чітко розділити тестування за завданнями та підтримувати логічну структуру перевірок.

Загалом, правильна організація управління тест-кейсами забезпечує ефективність та контрольованість процесу тестування, підвищуючи якість програмного забезпечення та сприяючи своєчасному виявленню дефектів на різних етапах розробки.

2.2.1 Аналіз особливостей тест-кейсу

Тест-кейс – це тестовий артефакт, який описує чітку послідовність кроків, умови виконання та вхідні параметри, необхідні для перевірки певної функціональності програмного продукту [3]. Основна мета тест-кейсу – переконатися, що програмний продукт працює коректно, та підтвердити його відповідність вимогам замовника.

Тест-кейси відіграють важливу роль у забезпеченні якості програмного продукту. Вони використовуються не лише для виявлення помилок, але й для запобігання їх повторному виникненню шляхом ретельного документування процесу тестування та його результатів. Завдяки чітким інструкціям та опису очікуваних результатів, тестувальники можуть легко відтворювати тести, що дозволяє значно скоротити час на пошук та виправлення дефектів у майбутньому. Регулярне оновлення та адаптація тест-кейсів до нових версій програмного продукту допомагають підтримувати актуальність процесу тестування та уникати застарілих перевірок.

Тест-кейси поділяються на високорівневі та низькорівневі. Високорівневі сценарії характеризуються меншою деталізацією, що означає відсутність конкретних вхідних даних та очікуваних результатів. Вони застосовуються для загального огляду та можуть використовуватися на ранніх етапах тестування, коли важлива загальна перевірка функціональності. Низькорівневі тест-кейси, навпаки, мають детальний опис із зазначенням конкретних вхідних даних та очікуваних результатів, що забезпечує максимальну точність та послідовність тестування. Такий підхід особливо важливий для регресійного тестування, де необхідно гарантувати, що навіть незначні зміни у програмному коді не призвели до непередбачуваних результатів.

Основні поля, які включає тест-кейс, такі [13]:

- ідентифікатор тест-кейсу (унікальний код або номер для ідентифікації);
- назва тест-кейсу (коротка, але інформативна назва, що описує зміст перевірки);
- передумови (умови або дії, які потрібно виконати перед початком тестування);
- кроки відтворення (докладний опис дій для виконання перевірки);
- тестові дані (інформація, необхідна для виконання тесту);
- очікуваний результат (опис результату, який повинен бути отриманий у разі успішного виконання тесту);

− статус (результат тестування, отриманий після проходження сценарію);

− дата проходження (час і дата виконання тесту).

Крім того, тест-кейс може містити додаткову інформацію, таку як:

− ідентифікатор набору тест-кейсів (вказує на групу, до якої він належить);

− автор тест-кейсу (ім’я особи, що створила тест-кейс);

− ім’я виконавця (ім’я тестувальника, який виконував тестування);

− коментарі (примітки для фіксації важливих деталей або уточнень);

− посилання на дефекти (дані про дефекти, знайдені під час тестування);

− інформація про версію програмного продукту, а також про пристрої та операційні системи, що використовуються.

Точна структура та поля тест-кейсів зазвичай відрізняються у різних компаніях та командах, адже потрібно враховувати різні аспекти конкретного проекту та склад команди. На приклад, якщо програмний продукт перевіряє лише один тестувальник, то не має сенсу додавати автора тест-кейсу та ім’я виконавця, проте у більших командах це може бути важливою інформацією. Проте основні поля залишаються незмінними.

2.2.2 Аналіз життєвого циклу тест-кейсу

Життєвий цикл тест-кейсу представляє собою послідовність станів, через які тест-кейс проходить протягом свого існування [14]. Цей процес відображає етапи використання та обробки тест-кейсу в межах тестування програмного забезпечення. Слід зазначити, що немає єдиного стандартизованого варіанту життєвого циклу тест-кейсу, оскільки його структура може значно відрізнятися в різних компаніях та проектах. Однак можна виділити загальну схему, яка є типовою для більшості процесів тестування (рис. 2.3).

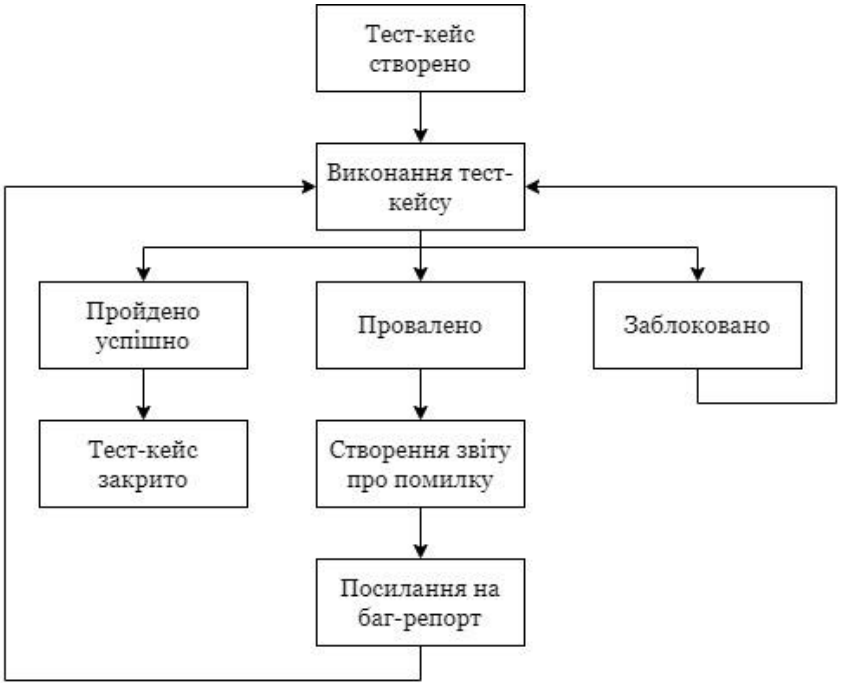


Рисунок 2.3 – Життєвий цикл тест-кейсу

Розглянемо детальніше кожний з етапів [14]:

– Тест-кейс створено. Це початковий етап життєвого циклу тест-кейсу, коли його щойно створено та ніякі подальші дії не проводились. На даному етапі визначаються всі його поля та основні параметри.

– Виконання тест-кейсу. На цьому етапі тестувальник здійснює перевірку програмного забезпечення відповідно до зазначених кроків і тестових даних. Тест-кейс переходить у стан активного виконання, де здійснюється тестування описаної функціональності.

– Пройдено успішно. Цей стан означає, що виконання тест-кейсу завершилося відповідно до очікуваного результату без виявлення дефектів. У такому випадку тестування функціональності підтверджує, що програмний продукт працює коректно.

– Провалено. У випадку, коли під час виконання тест-кейсу були виявлені дефекти або невідповідності вимогам, тест-кейс переходить у стан «Провалено». Варто зазначити, що якщо під час виконання тестування було знайдено дефекти, не пов’язані з цим конкретним тест-кейсом, статус

тест-кейсу може залишатися «Пройдено успішно», а відповідні дефекти виправляються окремо.

– Заблоковано. Цей стан свідчить про те, що виконання тест-кейсу на даний момент неможливе через зовнішні фактори, на приклад, відсутність необхідної сторінки, де вповинен розміщуватись функціонал.

– Створення звіту про помилку. Після виявлення дефекту під час виконання тест-кейсу тестувальник обов'язково створює звіт про помилку, де фіксує всі подробиці про знайдену проблему для подальшого її аналізу та виправлення.

– Посилання на баг-репорт. Після створення звіту про помилку тест-кейс отримує посилання на відповідний баг-репорт, що забезпечує зручність у відстеженні знайдених дефектів та їхнього статусу.

– Тест-кейс закрито. Цей стан позначає завершення роботи з тест-кейсом в межах даної ітерації тестування. Тест-кейс вважається закритим, коли всі дії завершені або у випадках, коли зміни у вимогах роблять тест-кейс неактуальним. Іноді цей стан може не використовуватися, якщо тест-кейс вважається частиною циклічних перевірок.

Життєвий цикл тест-кейсу дозволяє систематизувати процес тестування, забезпечуючи його прозорість та ефективність. Відстеження статусів тест-кейсів допомагає керівникам проектів та тестувальникам ефективно планувати та контролювати процес тестування, а також своєчасно виявляти й усувати проблеми.

2.2.3 Аналіз особливостей наборів тест-кейсів

Набір тест-кейсів – це логічна група тест-кейсів, об'єднаних за спільною ознакою або призначених для виконання одного завдання [15].

Структуровані набори тест-кейсів допомагають оптимізувати процес тестування, забезпечуючи легке управління та контроль за виконанням тестування.

Існують два основних типи наборів: довільні та послідовні. У довільних наборах порядок виконання тест-кейсів не має значення, що дозволяє виконувати їх у будь-якій послідовності. Такі набори використовуються для перевірки окремих функцій без необхідності дотримання певної черговості. Послідовні набори, навпаки, вимагають виконання тестів у заданому порядку, де успішне завершення одного тесту є передумовою для початку наступного. Такий підхід дозволяє перевірити залежності між функціональностями та виявити прогалини у тестуванні, якщо одна функція впливає на іншу.

Для відображення важливості кожного тесту в наборі можна використовувати систему пріоритетів, що дозволяє визначити черговість виконання тестів. Це особливо важливо для послідовних наборів, де правильний порядок проходження тест-кейсів впливає на точність результатів тестування.

Набір тест-кейсів може містити різні типи тестів, такі як функціональні, регресійні, інтеграційні та інші, що дозволяє охопити всі необхідні аспекти тестування програмного забезпечення. Завдяки цьому тестувальники можуть створювати комплексні плани тестування, що включають різноманітні перевірки для повного охоплення системи.

Програмна реалізація наборів тест-кейсів часто відображається у вигляді ієрархічних структур, подібних до тек, що містять у собі окремі тест-кейси. Це спрощує організацію та навігацію по наборах, а також дозволяє зберігати логічну структуру тестування навіть при роботі з великою кількістю тестів. Зручний доступ до наборів тест-кейсів дає змогу швидко знаходити необхідні тести, редагувати їх та слідкувати за їх виконанням.

Загалом, набори тест-кейсів є ефективним інструментом для координації процесу тестування, оскільки вони забезпечують можливість групування тестів за певними критеріями, підвищуючи керованість процесу та його адаптивність до змін у вимогах програмного продукту.

2.3 Аналіз технологій експертних систем для автоматизації процесу управління тест-кейсами

Експертні системи – це потужний інструмент, що використовує знання та алгоритми для вирішення складних завдань на основі принципів штучного інтелекту (ШІ). Вони моделюють процес міркування експертів у певній галузі, що дозволяє приймати обґрунтовані рішення без прямої участі людини. Експертні системи активно застосовуються у різних сферах, таких як медицина, інженерія, фінанси та управління бізнесом [16].

Експертні системи базуються на знаннях, зібраних від фахівців та документації, і використовують алгоритми для прийняття рішень або рекомендацій. Їхня структура включає дві основні частини: базу знань та механізм виведення.

База знань представляє собою зібрану інформацію та правила, що описують певну предметну область. У тестуванні ПЗ вона може містити інформацію про типи дефектів, критичність функціональних модулів, результати попередніх тестів тощо.

Механізм виведення – це набір алгоритмів, які обробляють дані з бази знань і приймають рішення на основі наявної інформації [17]. Використовуючи логіку й методи штучного інтелекту, механізм виведення аналізує проблеми й надає рекомендації.

Одним із найважливіших завдань під час тестування є визначення пріоритетів виконання тест-кейсів. Пріоритезація тестів допомагає раціонально використовувати час і ресурси, зосереджуючи увагу на найбільш критичних аспектах продукту. Однак кількість тест-кейсів у сучасних проектах може сягати сотень і навіть тисяч, тому визначення їхнього пріоритету вручну є не лише трудомістким і повільним, але й схильним до суб'єктивних рішень. Експертні системи можуть значно покращити цей процес, забезпечуючи об’єктивні й обґрунтовані рекомендації.

Основною перевагою експертних систем є їхня здатність обробляти великі обсяги даних та аналізувати різноманітні фактори з мінімальною участю людини. Завдяки цьому такі системи можуть виконувати оцінку тест-кейсів на основі комплексних правил і надавати об'єктивні результати, що допомагають зосередити ресурси тестування на найкритичніших елементах продукту.

Експертні системи можуть аналізувати ризики, пов'язані з різними частинами ПЗ, та історичні дані. На основі цієї інформації система присвоює пріоритет тестам залежно від рівня ризику в різних компонентах та тенденцій виникнення помилок у різних компонентах ПЗ, використовуючи інформацію про попередні тестування. Проте експертні системи можуть враховувати не лише технічні аспекти, а й поточні ресурси команди: кількість тестувальників, доступний час і технічні засоби. На основі цієї інформації система може допомогти тест-менеджерам приймати рішення щодо того, як найкраще розподілити ресурси для досягнення максимальних результатів тестування.

Встановлені правила й алгоритми дозволяють експертним системам мінімізувати суб'єктивні рішення, застосовуючи однакові критерії для всіх тест-кейсів. Це допомагає уникнути ситуацій, коли менш важливі аспекти тестуються з більшою ретельністю, ніж критичні.

Експертні системи надають низку переваг при тестуванні програмного забезпечення. Автоматична пріоритезація тест-кейсів на основі відповідей на попередньо визначені питання дозволяє знизити навантаження на тестувальників і мінімізувати людський фактор. Використання єдиних правил для всіх тест-кейсів знижує суб’єктивність підходів і допомагає уникнути помилок через індивідуальні інтерпретації членів команди.

Автоматизація також дозволяє швидше приймати рішення в умовах високої динаміки змін у розробці. Завдяки автоматизації, рішення щодо пріоритету тест-кейсів приймаються значно швидше, що особливо важливо в умовах стиснутих термінів розробки. Експертна система може обробляти великі обсяги інформації за короткий час, що економить ресурси команди та дозволяє зосередитись на виконанні тестів замість організаційних завдань.

Крім того, експертні системи забезпечують гнучкість і адаптивність. Критерії для визначення пріоритетів можуть бути налаштовані під конкретні проекти або вимоги бізнесу. Це дозволяє експертним системам враховувати нові запити та оновлювати правила відповідно до історичних даних.

Системи також можуть генерувати аналітичні звіти щодо пріоритетів тест-кейсів, що допомагає менеджерам та тест-лідерам оцінити стан тестування і своєчасно коригувати планування та розподіл ресурсів.

Ще однією перевагою є можливість інтеграції експертних систем в існуючі тест-менеджмент системами. Це забезпечує зручність у використанні та дозволяє отримувати автоматизовані рекомендації безпосередньо в рамках щоденних робочих процесів. Така інтеграція сприяє ефективнішому управлінню тестовою документацією та зменшує можливість помилок у координації тестування.

2.4 Розробка моделі та алгоритму експертної пріоритезації тест-кейсів

Процес розробки експертної системи для пріоритезації тест-кейсів передбачає створення чіткої структури, яка дозволяє автоматично аналізувати тест-кейси на основі визначених критеріїв та встановлювати для них пріоритети. Для цього потрібно розробити кілька ключових компонентів: базу знань, механізм виведення, інтерфейс користувача та алгоритм оцінки. Взаємодія цих компонентів забезпечує об'єктивну й ефективну роботу системи.

База знань є основою експертної системи, оскільки вона містить усі правила та критерії, за якими система приймає рішення. Для пріоритезації тест-кейсів база знань включає такі частини:

- Критерії пріоритезації. Визначаються на основі бізнес-вимог, історичних даних і ризиків, пов'язаних з кожною функцією продукту. Це можуть бути критичність функціональності, вплив дефекту на користувача, частота використання функції, історія дефектів і регуляторні вимоги.

- Правила оцінки. Кожен критерій має свої правила, що використовуються для присвоєння тест-кейсам балів. Наприклад, функції, критичні для бізнесу, отримують максимальні бали, тоді як функції, що рідко використовуються або мають низький вплив, оцінюються нижче. Ці правила можна налаштувати відповідно до потреб проекту.

Механізм виведення відповідає за прийняття рішень на основі бази знань. Він обробляє введені користувачем дані про тест-кейс і присвоює пріоритет, базуючись на сукупній оцінці за всіма критеріями. Основні етапи його роботи:

1. Користувач відповідає на серію питань щодо кожного тест-кейсу.
2. Механізм виведення присвоює кожному критерію бали на основі відповідей користувача.
3. Після підсумовування балів система визначає пріоритет тест-кейсу: високий, середній або низький.

Механізм виведення також може включати алгоритми для вирішення конфліктних ситуацій, коли кілька тест-кейсів мають однаковий пріоритет. У таких випадках система застосовує додаткові критерії для остаточного рішення.

Інтерфейс користувача є важливою частиною системи, оскільки саме через нього тестувальники взаємодіють з експертною системою. Інтерфейс має бути зручним і інтуїтивно зрозумілим, що дозволяє швидко вводити інформацію про тест-кейси і отримувати результати.

Основні функції інтерфейсу:

- Введення даних. Користувачі відповідають на запитання, що стосуються тест-кейсів (так/ні, вибір із кількох варіантів, числові значення).
- Виведення результатів. Після обробки даних система надає інформацію про пріоритет кожного тест-кейсу і пояснення, які критерії вплинули на кінцевий результат.

Щоб підвищити ефективність і зручність використання, експертна система інтегрована з системою для організації управління тест-кейсами.

Алгоритм експертної системи для пріоритезації тест-кейсів базується на послідовній оцінці ключових критеріїв, які впливають на важливість кожного тест-кейсу. Основною функцією алгоритму є присвоєння балів на основі відповідей на запитання, що охоплюють такі аспекти, як критичність функціональності, ризики й впливи, історичні дані, регуляторні вимоги і додаткові критерії.

Алгоритм оцінює критичність функціональності тест-кейсу за наступними параметрами:

1. Яка критичність цієї функціональності для бізнесу?
   - Висока – 5 балів
   - Середня – 3 бали
   - Низька – 1 бал
2. Чи є ця функціональність фундаментальною для основних бізнес-процесів?
   - Так – 5 балів
   - Ні – 1 бал
3. Як часто використовується ця функціональність користувачами?
   - Щодня – 5 балів
   - Щотижня – 3 бали
   - Щомісяця – 2 бали
   - Рідко – 1 бал

Оцінка ризиків та впливів вимірює потенційний ризик і вплив дефектів у даній функціональності:

1. Який потенційний вплив дефекту в цій функціональності на користувачів?
   - Критичний (втрата даних, неможливість використання продукту) – 5 балів
   - Високий (значні проблеми в роботі, збої) – 4 бали
   - Середній (незначні незручності) – 2 бали
   - Низький (майже непомітно) – 1 бал

2. Чи є ця функціональність частиною критичних шляхів у системі (наприклад, обробка платежів)?

- Так – 5 балів
- Ні – 1 бал

3. Чи може дефект у цій функціональності спричинити фінансові втрати або пошкодження репутації компанії?

- Так – 5 балів
- Ні – 1 бал

Критерій «Історичні дані» враховує інформацію про попередні релізи та зміни функціональності:

1. Чи були в цій функціональності виявлені дефекти у минулому?

- Так – 5 балів
- Ні – 1 бал

2. Яка була частота дефектів у цій області в минулих релізах?

- Висока – 5 балів
- Середня – 3 бали
- Низька – 1 бал

3. Чи була ця функціональність змінена або оновлена недавно?

- Так – 5 балів
- Ні – 1 бал

4. Чи є в цій функціональності інтеграція з іншими системами або компонентами?

- Так – 5 балів
- Ні – 1 бал

Також оцінюється підпадання під регуляторні вимоги

1. Чи підпадає ця функціональність під регуляторні вимоги?

- Так – 5 балів
- Ні – 1 бал

Алгоритм також враховує додаткові критерії, які можуть вплинути на важливість функціональності:

1. Чи очікуються майбутні зміни в найближчому часі?
   - Так – 5 балів
   - Ні – 1 бал
2. Чи є ця функціональність видимою для кінцевих користувачів?
   - Так – 5 балів
   - Ні – 1 бал
3. Наскільки важлива ця функціональність для користувацького досвіду?
   - Важлива – 5 балів
   - Середньо важлива – 3 бали
   - Неважлива – 1 бал
4. Який вплив має ця функціональність на інші частини системи?
   - Високий вплив – 5 балів
   - Середній вплив – 3 бали
   - Низький вплив – 1 бал
5. Чи можуть дефекти в цій функціональності спричинити проблеми в інших компонентах?
   - Так – 5 балів
   - Ні – 1 бал

Після обробки всіх критеріїв система підсумовує бали та визначає пріоритет:

- Високий пріоритет. Тест-кейси, які набрали максимальну кількість балів (висока критичність функції, великі ризики для бізнесу, часті дефекти).
- Середній пріоритет. Тест-кейси із середньою кількістю балів (функціональність важлива, але з меншими ризиками або рідшим використанням).
- Низький пріоритет. Тест-кейси, які набрали мінімальні бали (функціональність менш критична, рідко використовується, має незначний вплив).

Пріоритет тест-кейсу (P) можна визначити за такою формулою:

$$P = w_1 \times C + w_2 \times I + w_3 \times H + w_4 \times R + w_5 \times A, \quad (2.1)$$

де P – загальний пріоритет тест-кейсу;

w_1, w_2, w_3, w_4, w_5 – вагові коефіцієнти для кожного критерію;

C – критичність функціональності;

I – ризики та впливи;

H – історичні дані (дефекти, частота змін) ;

R – регуляторні вимоги;

A – додаткові критерії (наприклад, вплив на користувацький досвід).

Експертна система, розроблена для автоматизації пріоритезації тест-кейсів, функціонує як окремий модуль, інтегрований у загальну інформаційну технологію для управління тест-кейсами. Її основне завдання – автоматично оцінювати кожен тест-кейс на основі визначених критеріїв і встановлювати пріоритети для тестування, що дозволяє раціональніше використовувати ресурси та підвищити ефективність процесу тестування.

На рисунку 2.4 наведено схему роботи експертної системи для пріоритезації, що інтегрована у систему для організації управління тест-кейсами.

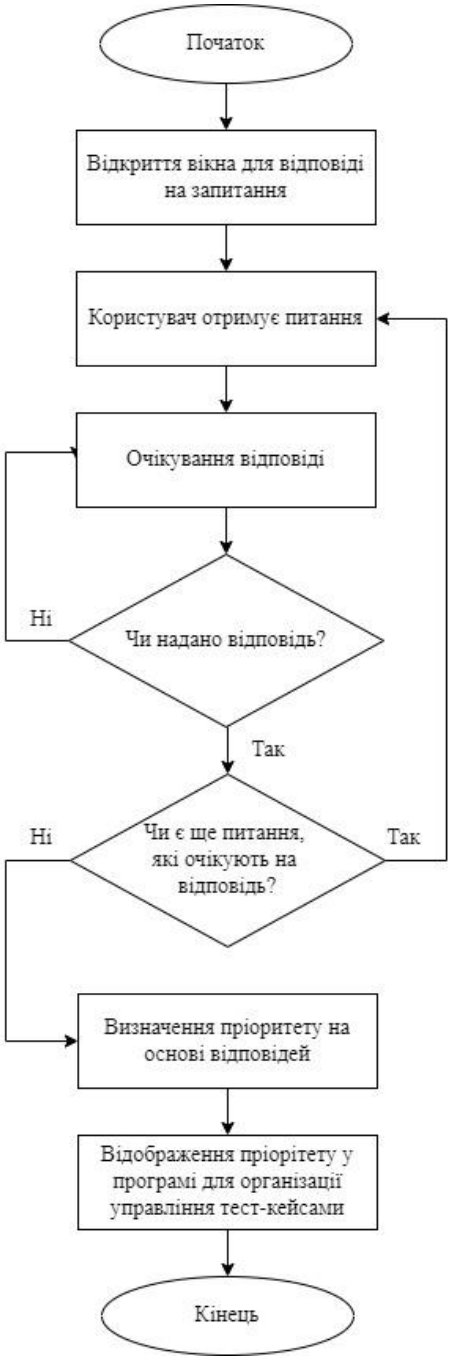


Рисунок 2.4 – Узагальнений алгоритм роботи експертної системи для пріоритезації, інтегрована у систему для організації управління тест-кейсами

Таким чином, після початку роботи модуля відкривається вікно для надання відповіді користувача на запитання для визначення пріоритету. Після цього користувач отримує перше питання. Програма очікує відповідь. Якщо відповідь надано, і ще є питання, які очікують відповідь та не були показані користувачу, на екран виводиться наступне питання.

Якщо відповідь на питання не було надане, очікування продовжується. Якщо питань, які очікують відповідь та не були показані користувачу більше немає, це означає, що користувач дав відповідь на усі питання і починається визначення пріоритету на основі його відповідей. Після цього пріоритет відображається у програмі для організації управління тест-кейсами і роботу програмного модуля завершено.

2.5 Проектування інформаційної технології для організації управління тест-кейсами

Проектування є важливим етапом у створенні системи, яка дозволяє ефективно керувати процесом тестування та забезпечує зручність роботи з тестовими сценаріями. Для початку визначимо структуру інформаційної технології для організації управління тест-кейсами. Вона включає такі основні компоненти:

- базу даних;
- інтерфейс користувача;
- сервер;
- веб-клієнт;
- модуль інтеграції з системою відстеження дефектів;
- експертну систему пріоритезації тест-кейсів.

База даних є основою системи, що виконує функцію центрального сховища усіх даних системи, включаючи тест-кейси, шаблони, набори тестів, пріоритети, статуси, а також посилання на задачі в системі відстеження дефектів. Це дозволяє усім учасникам тестування отримувати доступ до тестової документації, зокрема до історії змін, що підвищує прозорість процесу. Кожен тест-кейс має бути унікально ідентифікований для спрощення пошуку та відстеження.

Інтерфейс користувача є важливим елементом структури, що забезпечує взаємодію користувачів із системою, відображення усіх необхідних елементів та

зручність доступу до системи організації управління тест-кейсами. Інтерфейс повинен бути інтуїтивно зрозумілим, дозволяючи користувачам швидко додавати нові сутності та модифікувати їх.

Сервер є центральною обчислювальною одиницею, яка обробляє запити від клієнтів і забезпечує доступ до бази даних та експертної системи пріоритезації. Він відповідає за обробку даних, валідацію запитів та забезпечення можливості змін записів у базі даних.

Веб-клієнт є основою для доступу користувачів до системи через веб-браузер. Він надсилає запити до сервера, отримує відповіді, відображає дані в інтерфейсі користувача та забезпечує зручну навігацію між різними модулями системи.

Одним із ключових елементів системи є модуль інтеграції з системою відстежування дефектів. Він інтегрується з зовнішньою системою відстеження дефектів Jira та забезпечує можливість прив'язувати тест-кейси до дефектів, переглядати пов'язану інформацію про задачі та автоматично оновлювати статуси тест-кейсів залежно від стану пов'язаних задач.

Експертна система для пріоритезації тест-кейсів, інтегрована в структуру інформаційної технології, автоматизує визначення пріоритету тест-кейсів, ґрунтуючись на певних критеріях і правилах. Вона інтегрована в інтерфейс користувача, де надає послідовні питання для визначення пріоритету, а потім обчислює значення пріоритету для кожного тест-кейса. Це дозволяє системі автоматично виставляти пріоритети для кожного тест-кейсу, допомагаючи команді ефективно розподіляти ресурси та зосереджувати увагу на найбільш критичних функціях продукту.

Структурну схему взаємодії компонентів програми наведено на рисунку 2.5. Інтерфейс користувача отримує дані від сервера та передає їх у зручному для користувача форматі. Користувач через веб-клієнт взаємодіє з інтерфейсом користувача, надсилаючи запити для додавання, редагування або перегляду тест-кейсів. Веб-клієнт передає дані користувача на сервер, обробляє їх і взаємодіє з

базою даних для отримання чи збереження потрібної інформації. Також він забезпечує отримання відповідей від сервера для подальшого відображення.

База даних працює разом із сервером, зберігає всю інформацію та надає її за запитами, забезпечуючи узгодженість даних для всіх компонентів системи. Модуль інтеграції з системою відстеження дефектів взаємодіє із сервером, надсилаючи та отримуючи інформацію про дефекти, і синхронізується з базою даних. Сервер при необхідності звертається до експертної системи для визначення пріоритету або до модуля інтеграції для синхронізації із системою відстеження дефектів.

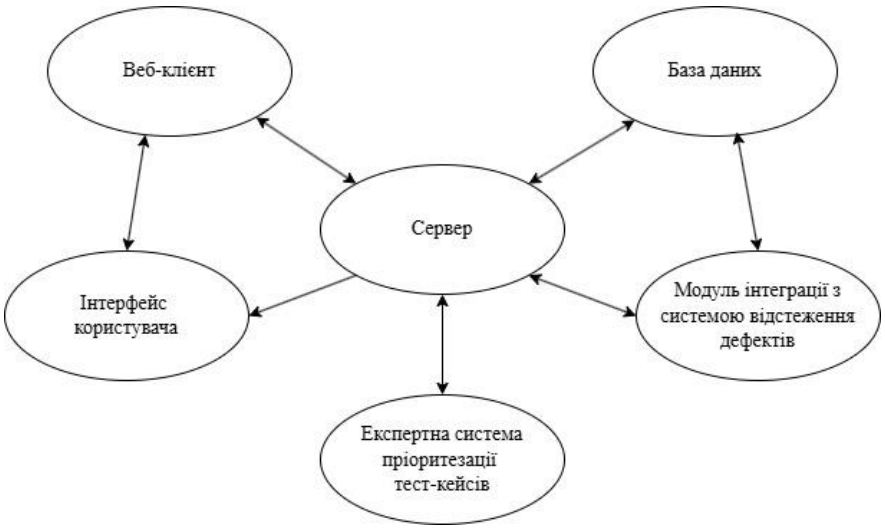


Рисунок 2.5 – Структурна схема взаємодії компонентів програми

Основна мета цієї структури – забезпечити прозорість і централізацію управління тестовими сценаріями, що сприяє підвищенню ефективності тестування та поліпшенню якості програмного забезпечення.

Для проектування інформаційної технології скористаємось UML діаграмою класів. UML (Unified Modeling Language) – це стандартизована мова моделювання, що зображає структуру системи, показуючи класи, їх атрибути, методи та відношення між ними. Вона є одним з основних засобів візуалізації об’єктно-орієнтованого дизайну системи [18].

UML діаграми класів надають чітке візуальне уявлення про структуру системи, що допомагає розробникам і зацікавленим сторонам зрозуміти взаємодії між об'єктами. Крім того, вони сприяють ефективному плануванню та документуванню архітектури, що полегшує підтримку, розширення та внесення змін до системи в майбутньому.

Діаграма класів інформаційної технології для організації управління тест-кейсами представлено на рисунку 2.6.

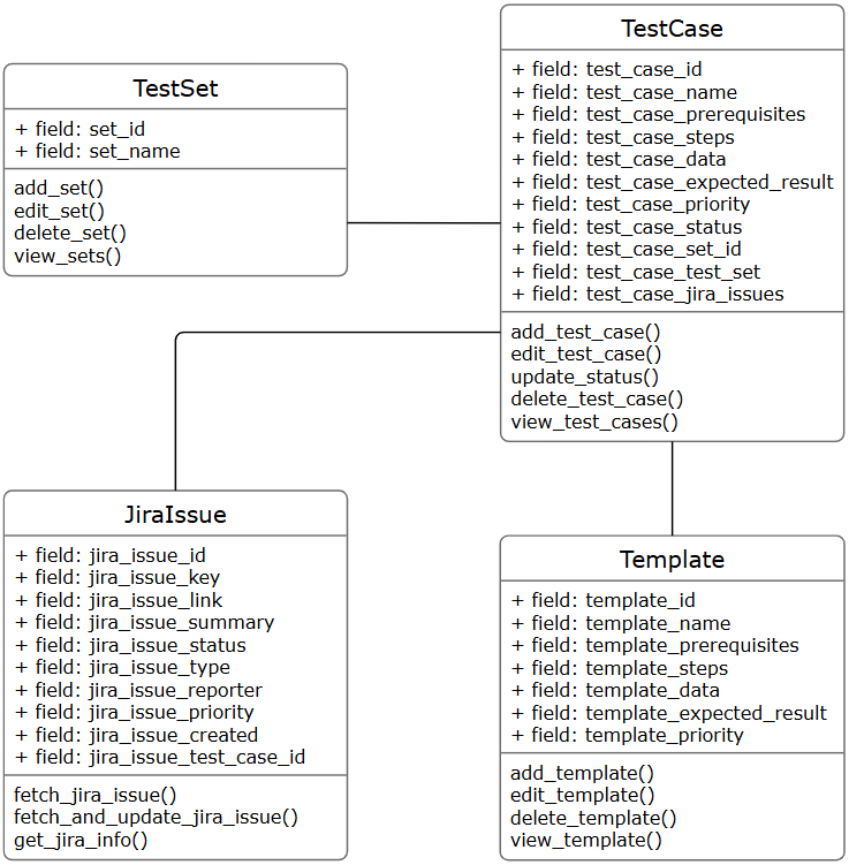


Рисунок 2.6 – UML діаграма класів інформаційної технології для організації управління тест-кейсами

Програмний продукт містить основні класи TestSet, TestCase, Template та JiraIssue.

У класі TestSet зберігається поле зі значенням назви набору та його ідентифікатора. До нього входять такі функції:

– add_set() відповідає за створення нового набору;

- edit_set() відповідає за редагування набору;
- delete_set() відповідає за видалення набору.
- view_sets() відповідає за перегляд усіх існуючих наборів.

Класі TestCase зберігає поля, що описують атрибути тест-кейсу: ідентифікатор, назву, передумови, кроки відтворення, тестові дані, очікуваний результат, пріоритет, ідентифікатори завдань Jira, статус, поля для вибору шаблону та пріоритету. До класу входять такі функції:

- add_test_case() відповідає за створення нового тест-кейсу;
- edit_test_case() відповідає за редагування тест-кейсу;
- update_status() відповідає за встановлення та зміну статусу тест-кейсу;
- delete_test_case() відповідає за видалення тест-кейсу;
- view_test_cases() відповідає за перегляд усіх існуючих тест-кейсів.

У класі Template зберігаються поля, що описують атрибути шаблону: ідентифікатор, назву, передумови, кроки відтворення, тестові дані, очікуваний результат та пріоритет. Даний клас містить такі функції:

- add_template() відповідає за створення нового шаблону;
- edit_template() відповідає за редагування шаблону;
- delete_template() відповідає за видалення шаблону;
- view_templates() відповідає за перегляд усіх існуючих шаблонів.

Клас JiraIssue зберігає поля для реалізації інтеграції з Jira. Даний клас містить такі функції:

- fetch_jira_issue() відповідає за отримання даних завдання з Jira;
- fetch_and_update_jira_issue() відповідає за отримання статусу завдання з Jira і його оновлення його в базі даних;
- get_jira_info() відповідає за отримання інформації про Jira завдання на основі списку ідентифікаторів.

Таким чином, розроблена структура інформаційної технології дозволяє оптимізувати процес управління тест-кейсами, забезпечуючи централізоване

зберігання тестової документації, інтеграцію з іншими системами та ефективне управління пріоритезацією тест-кейсів.

2.6 Висновок до розділу 2

У другому розділі виконано аналіз процесів організації управління тест-кейсами. Здійснено класифікацію підходів до тестування програмного забезпечення, розглянуто структуру тест-кейсів та їх життєвий цикл, що дозволило виокремити основні етапи та особливості використання тест-кейсів у процесі тестування.

Проведено аналіз існуючих підходів до організації та управління наборами тест-кейсів, що включає їхнє групування та пріоритезацію. Визначено ключові фактори, що впливають на ефективність управління, такі як модульність, послідовність виконання та оптимізація використання ресурсів. Також досліджено сучасні технології експертних систем для автоматизації управління тест-кейсами, зокрема для автоматизованої пріоритезації тестів на основі об’єктивних критеріїв, що включають ризики, історичні дані та критичність функціональних елементів програмного забезпечення.

Розроблено модель експертної системи для пріоритезації тест-кейсів та алгоритм її функціонування, що дозволяють інтегрувати ці підходи в систему для організації управління тест-кейсами. Розроблено структуру функціонування інформаційної технології та створено UML діаграму класів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОРГАНІЗАЦІЇ УПРАВЛІННЯ ТЕСТ-КЕЙСАМИ

3.1 Обґрунтування вибору мови та середовища програмування

Для розробки інформаційної технології для організації управління тест-кейсами обрано Python, HTML, CSS та JavaScript, оскільки кожна з цих технологій забезпечує необхідну функціональність для створення надійного та зручного у використанні веб-додатка.

Python використовується для розробки бекенду, що відповідає за обробку запитів, виконання бізнес-логіки, взаємодію з базою даних, а також передачу даних на фронтенд. Бекенд є важливим компонентом веб-додатка, оскільки він забезпечує функціональність, що не відображається безпосередньо на інтерфейсі користувача, але є необхідною для стабільної роботи системи. Завдяки Python можна реалізувати складні логічні процеси, а також забезпечити швидкість та гнучкість роботи додатка.

Python – це високорівнева мова програмування, що є однією з найпопулярніших у світі завдяки своїм широким можливостям і простоті використання. Вибір Python для даної розробки обумовлений такими перевагами [19]:

- Простота і зручність. Python має зрозумілий і легкий для читання синтаксис, що прискорює розробку і спрощує процес тестування. Це особливо корисно для швидкої реалізації бізнес-логіки.
- Широкий набір бібліотек. Python має потужні бібліотеки та фреймворки, такі як Flask або Django, які дозволяють легко створювати бекенд-додатки з чітко визначеною структурою.
- Розширюваність та інтеграція. Python добре інтегрується з іншими мовами програмування і підтримує різноманітні формати даних, що забезпечує гнучкість системи.

- Активна спільнота. Python має значну спільноту розробників і великий обсяг документації, що забезпечує доступ до широкого спектра ресурсів і підтримки.

JavaScript використовується для фронтенду, зокрема для додавання динамічної взаємодії на веб-сторінці. Фронтенд – це частина веб-додатка, яка відповідає за відображення інтерфейсу користувача та взаємодію з ним. JavaScript дозволяє додавати анімацію, обробляти події, а також динамічно змінювати вміст сторінки, що покращує користувацький досвід. Переваги JavaScript для цього проекту включають [20]:

- Інтерактивність. JavaScript дозволяє реалізувати елементи, що реагують на дії користувача, наприклад, відкриття модальних вікон і перевірку валідності введених даних.
- Можливості асинхронного обміну даними. Використання JavaScript дозволяє надсилати запити до серверу без перезавантаження сторінки, що робить веб-додаток більш інтерактивним і швидким.
- Широка підтримка у браузерах. JavaScript підтримується всіма сучасними браузерами, що забезпечує кросбраузерну сумісність додатка.

HTML використовується для визначення структури веб-сторінки, де відображатимуться тест-кейси, набори та шаблони, сторінки їх редагування та додавання. HTML є основою для створення елементів інтерфейсу, таких як форми, таблиці та кнопки, що забезпечує легкість у створенні та впорядкуванні інформаційного вмісту [21].

CSS використовується для стилізації веб-сторінок, включаючи налаштування кольорів, шрифтів, розмірів і розташування елементів. Основна роль CSS полягає у забезпеченні привабливого вигляду додатка та відповідності вимогам до дизайну. Використання CSS дозволяє відокремити структуру від стилю, що полегшує процес редизайну без змін у коді HTML [21].

Щоб переконатися, що Python добре підходить для програмної реалізації, створимо порівняльну таблицю мов програмування (Таблиця 3.1).

Характеристики оцінені за шкалою, де «+» – це найкращий показник, «-» – найгірший, «+/-» – середній.

Таблиця 3.1 – Порівняльна характеристика мов програмування

	Швидкодія	Простота використання	Функціональність	Оновлюваність	Підтримка спільноти та ресурсів
Python	+/-	+	+/-	+	+
Java	+	-	+	+	+/-
C#	+/-	+/-	+	+	+/-
PHP	-	-	+/-	+/-	+/-
TypeScript	+/-	+/-	+/-	+/-	+/-

Обрані мови програмування, зокрема Python для бекенду та JavaScript, HTML, CSS для фронтенду, забезпечують повноцінний функціонал веб-додатка з управління тест-кейсами, роблячи його зручним, швидким та легким у підтримці й оновленні.

Для розробки програмного модуля управління тест-кейсами обрано середовище програмування PyCharm – інтегроване середовище розробки, оптимізоване для мови Python [22]. PyCharm пропонує широкий набір функцій, що полегшують процес написання, тестування та підтримки Python-коду, а також підтримує додаткові мови та технології, такі як HTML, CSS і JavaScript, що є важливими для веб-розробки. Основні переваги використання PyCharm такі [22]:

- Зручність і простота в налаштуванні. PyCharm має інтуїтивно зрозумілий інтерфейс, який спрощує налаштування та роботу з проєктом. Середовище дозволяє легко організовувати файли, бібліотеки та залежності, що особливо важливо при розробці комплексного веб-додатка.
- Потужний редактор коду. PyCharm надає розширені функції автодоповнення та перевірки синтаксису, що дозволяє розробникам швидше писати та тестувати код, зменшуючи ймовірність синтаксичних

помилок. Крім того, PyCharm підтримує рефакторинг коду, що дозволяє швидко вносити зміни без ризику порушення структури проєкту.

− Інтеграція з системами контролю версій. PyCharm інтегрується з Git, що дозволяє зручно керувати версіями коду, відслідковувати зміни та співпрацювати з іншими розробниками. Це забезпечує надійність і можливість відмінити зміни, якщо в процесі розробки виникнуть помилки.

− Інтеграція з веб-фреймворками та бібліотеками. PyCharm має вбудовану підтримку для веб-фреймворків, таких як Flask і Django, що спрощує розробку та тестування бекенду. IDE також надає вбудовані засоби для роботи з HTML, CSS та JavaScript, що є необхідними для фронтенд-частини проєкту.

− Вбудований тестувальний модуль. PyCharm дозволяє легко інтегрувати модульні тести та здійснювати автоматичне тестування компонентів додатка. Це особливо важливо для контролю якості при розробці модуля управління тест-кейсами, забезпечуючи стабільність роботи всіх компонентів.

− Налаштовувані інструменти для відлагодження. PyCharm містить потужні засоби для відлагодження, що дозволяють розробникам контролювати виконання коду, відстежувати значення змінних і знаходити помилки в реальному часі. Завдяки цьому процес відлагодження коду стає простішим та швидшим.

− Підтримка роботи з базами даних. PyCharm надає вбудовані інструменти для роботи з базами даних, зокрема SQLite, яка використовується в цьому проєкті. Це дозволяє безпосередньо у середовищі розробки виконувати запити до бази даних, аналізувати результати та тестувати взаємодію між бекендом і базою даних.

− Підтримка багатьох плагінів. PyCharm підтримує велику кількість плагінів, що розширюють функціонал IDE. Це дозволяє адаптувати середовище під специфічні потреби проєкту, додаючи необхідні

інструменти або додаткові функції, наприклад, плагіни для інтеграції з системами відслідковування помилок.

– Активна підтримка та документація. PyCharm має значну підтримку серед розробників і детальну документацію, що дозволяє швидко знаходити відповіді на питання або вирішувати проблеми, що виникають в процесі розробки. Це забезпечує надійність та стабільну підтримку інструменту.

Отже, вибір PyCharm як середовища розробки для інформаційної технології організації управління тест-кейсами обумовлений його функціональністю та зручністю для розробки веб-додатків на Python. PyCharm надає все необхідне для продуктивної роботи – від автодоповнення і перевірки синтаксису до інструментів для тестування, відлагодження та роботи з базами даних. Це дозволяє забезпечити високу якість коду, ефективність у розробці та зручність у підтримці додатка.

3.2 Обґрунтування вибору фреймворку

Для реалізації програмного модуля управління тест-кейсами обрано Flask – мінімалістичний веб-фреймворк для Python [23]. Flask забезпечує легкість і гнучкість у розробці веб-додатків, дозволяючи розробникам самостійно обирати потрібні компоненти і створювати структуру додатка з урахуванням конкретних потреб. Flask є популярним вибором для розробки невеликих і середніх проектів, де важливі швидкість і простота розгортання.

Переваги використання Flask [23]:

– Мінімалістичний дизайн. Flask не нав'язує додаткових модулів чи бібліотек, що дає можливість зосередитися на розробці необхідної функціональності без перевантаження програми зайвими компонентами. Це дозволяє підтримувати додаток легким і забезпечує швидку його роботу.

− Модульна структура. Flask побудований з невеликих, багаторазових компонентів, що забезпечує зручність у розширенні та налаштуванні. Розробники можуть додавати або видаляти функції за потреби, підтримуючи гнучкість і чистоту коду, а також полегшуючи подальше обслуговування додатка.

− Простота використання та інтуїтивний API. Flask має простий у використанні API, що дозволяє легко розпочати роботу навіть розробникам-початківцям. Завдяки цьому додаток можна швидко налаштувати та розгорнути, а простота синтаксису сприяє ефективному розумінню коду.

− Швидке налаштування. Flask легко встановлюється за допомогою менеджера пакетів Python, що дозволяє швидко розпочати розробку. Невелика кількість шаблонного коду робить процес розробки ще простішим, дозволяючи запустити базовий додаток буквально за кілька рядків.

− Вбудований сервер розробки. Flask надає вбудований сервер для тестування, який дозволяє запускати додаток локально без необхідності налаштовувати зовнішній сервер. Це спрощує процес налагодження і тестування програми в режимі розробки, допомагаючи швидко виявляти помилки.

− Документація та підтримка спільноти. Flask має детальну документацію з описами функцій і прикладами, а також активну спільноту розробників. Це дозволяє швидко знайти рішення поширених проблем і отримати підтримку в процесі розробки.

− Легкість інтеграції з іншими інструментами. Flask можна легко інтегрувати з іншими інструментами та бібліотеками Python, такими як SQLAlchemy для роботи з базами даних або Jinja2 для шаблонізації HTML-сторінок. Це підвищує гнучкість системи і дозволяє налаштувати додаток відповідно до специфічних потреб проєкту.

Тамким чиномм, вибір Flask для розробки веб-додатка управління тест-кейсами обумовлений його простотою, гнучкістю та зручністю налаштування. Flask дозволяє швидко розпочати роботу над проектом, залишаючи можливість додавати потрібні компоненти за потреби. Його мінімалістичний підхід, а також зручні інструменти для налагодження та тестування роблять його оптимальним вибором для проєктів середньої складності, забезпечуючи ефективну та гнучку розробку.

3.3 Інтеграція системи управління тест-кейсами з інструментом Jira

Інтеграція системи управління тест-кейсами з інструментом Jira забезпечує ефективну координацію між процесом тестування та управлінням задачами. Jira є потужним інструментом для управління проєктами, широко використовуваним для відстеження задач, дефектів та підтримки гнучних процесів [24]. Завдяки інтеграції з Jira, система управління тест-кейсами дозволяє тестувальникам зручно пов'язувати тест-кейси із задачами, створеними у Jira, а також автоматизувати оновлення статусів на основі змін у завданнях.

У системі управління тест-кейсами завдяки інтеграції з Jira можна виконувати такі завдання:

- прив’язку завдань Jira до тест-кейсів;
- перегляд прив'язаних завдань на сторінці тест-кейса;
- перехід до завдань у Jira;
- перегляд основних полів Jira у системі тест-кейсів;
- автоматичне оновлення статусу тест-кейса.

Під час створення або редагування тест-кейса користувач може вводити ідентифікатор задачі з Jira, що дозволяє прив’язати завдання до конкретного тест-кейса. Це дає можливість відстежувати, які завдання мають відношення до певного тест-кейса, і полегшує доступ до відповідної інформації.

На сторінці перегляду тест-кейсів користувач може бачити всі завдання з Jira, прив'язані до конкретного кейса. Це спрощує процес відстеження статусів пов’язаних задач і дозволяє оперативно отримувати доступ до їх деталей.

Кожне прив’язане завдання має активне посилання, що дозволяє користувачам переходити безпосередньо до Jira. Це забезпечує зручний перехід між двома системами, дозволяючи швидко отримувати додаткову інформацію про завдання або оновлювати його статус.

Крім того, система управління тест-кейсами відображає основні поля з прив’язаних завдань Jira безпосередньо в інтерфейсі тест-кейса. Це дозволяє отримати ключову інформацію про завдання (наприклад, статус, назву, пріоритет, людину, що створила завдання) без необхідності переходу до Jira.

У разі, коли всі завдання, прив'язані до тест-кейса, закриті в Jira, тест-кейсу автоматично виставляється статус "Passed". Це дозволяє автоматизувати процес оновлення статусів тест-кейсів і знижує необхідність ручного контролю з боку тестувальників.

Для реалізації інтеграції з Jira у систему управління тест-кейсами використовується Jira API, який забезпечує програмний доступ до даних у Jira, включаючи створення та оновлення задач, отримання їхнього статусу та іншої інформації. Таким чином, щоб інтегрувати систему з Jira, перш за все необхідно налаштувати доступ до API. Для цього використовують токени авторизації, які забезпечують безпечний обмін даними між додатком і Jira. Встановлення авторизації може здійснюватися через особистий токен API, який генерується користувачем у Jira.

Для полегшення роботи з API у Python використовується бібліотека jira-python, яка надає доступ до функцій API Jira, таких як отримання інформації про задачі або їхнє оновлення. Після встановлення бібліотеки налаштовується підключення до Jira, зокрема передаються параметри авторизації та адреса сервера.

Для отримання доступу до задач у Jira у програмному модулі створюється об’єкт клієнта, що працює з Jira API. Це дозволяє отримувати ідентифікатори,

статуси та інші поля задач Jira, пов’язані з тест-кейсами. За допомогою API користувачі можуть отримувати інформацію про задачу за її ідентифікатором і відображати її у системі управління тест-кейсами.

Таким чином, при введенні ідентифікатора Jira на формі створення або редагування тест-кейсу задачі зберігаються як прив’язані елементи.

API також дозволяє відстежувати статуси задач і автоматично оновлювати статус тест-кейсу, коли всі пов’язані задачі завершені. Це зменшує потребу в ручному оновленні статусів і забезпечує точність даних.

Інтеграція системи управління тест-кейсами з Jira за допомогою API дозволяє автоматизувати значну частину процесів управління тестуванням, зокрема прив'язку задач, оновлення статусів і створення нових задач. Це підвищує ефективність та точність процесу тестування, забезпечуючи прозорість і контроль над станом завдань у реальному часі.

3.4 Розробка узагальненого алгоритму роботи програмного продукту

Покроковий опис алгоритму роботи інформаційної технології виглядає таким чином:

1. Відображається головна сторінка з трьома основними блоками.
2. Якщо натиснута кнопка «Усі Набори», відображається сторінка «Усі Набори». Якщо ні, переходимо до пункту 11.
3. Якщо натиснута кнопка «Додати Набір», відображається сторінка «Додати Набір». Якщо ні, переходимо до пункту 7.
4. Користувач заповнює усі необхідні поля.
5. Відбувається валідація даних. Якщо вони валідні, набір зберігається до БД. Якщо ні, повертаємось до пункту 4.
6. Виводиться сторінка «Усі Набори».
7. Якщо натиснута кнопка «Редагувати Набір», виводиться сторінка «Редагувати Набір».
8. Користувач заповнює усі необхідні поля.

9. Відбувається валідація даних. Якщо вони валідні, набір зберігається до БД. Якщо ні, повертаємось до пункту 8.
10. Виводиться сторінка «Усі Набори».
11. Якщо натиснута кнопка «Усі Шаблони», відображається сторінка «Усі Шаблони». Якщо ні, переходимо до пункту 20.
12. Якщо натиснута кнопка «Додати Шаблон», відображається сторінка «Додати Шаблон». Якщо ні, переходимо до пункту 16.
13. Користувач заповнює усі необхідні поля.
14. Відбувається валідація даних. Якщо вони валідні, шаблон зберігається до БД. Якщо ні, повертаємось до пункту 13.
15. Виводиться сторінка «Усі Шаблони».
16. Якщо натиснута кнопка «Редагувати Шаблон», виводиться сторінка «Редагувати Шаблон».
17. Користувач заповнює усі необхідні поля.
18. Відбувається валідація даних. Якщо вони валідні, шаблон зберігається до БД. Якщо ні, повертаємось до пункту 17.
19. Виводиться сторінка «Усі Шаблони».
20. Якщо натиснута кнопка «Усі Тест-кейси», відображається сторінка «Усі Тест-кейси». Якщо ні, переходимо до пункту 1.
21. Якщо натиснута кнопка «Додати Тест-кейс», відображається сторінка «Додати Тест-кейс». Якщо ні, переходимо до пункту 28.
22. Якщо кнопку «Шаблон» натиснуто, відбувається вибір шаблону. Якщо ні, переходимо до пункту 24.
23. Усі поля тест-кейсу заповняються полями шаблону.
24. Якщо необхідно визначити пріоритет, визначаємо його за допомогою експертної системи. Якщо ні, переходимо до пункту 25.
25. Користувач заповнює усі необхідні поля.
26. Відбувається валідація даних. Якщо вони валідні, набір зберігається до БД. Якщо ні, повертаємось до пункту 25.
27. Виводиться сторінка «Усі Тест-кейси».

28. Якщо кнопка «Редагувати Тест-кейс» натиснута, виводиться сторінка «Редагувати Тест-кейс».
29. Користувач заповнює усі необхідні поля.
30. Відбувається валідація даних. Якщо вони валідні, тест-кейс зберігається до БД. Якщо ні, повертаємось до пункту 29.
31. Виводиться сторінка «Усі Тест-кейси».

Візуалізований узагальнений алгоритм роботи програмного продукту наведено на рисунку 3.1.

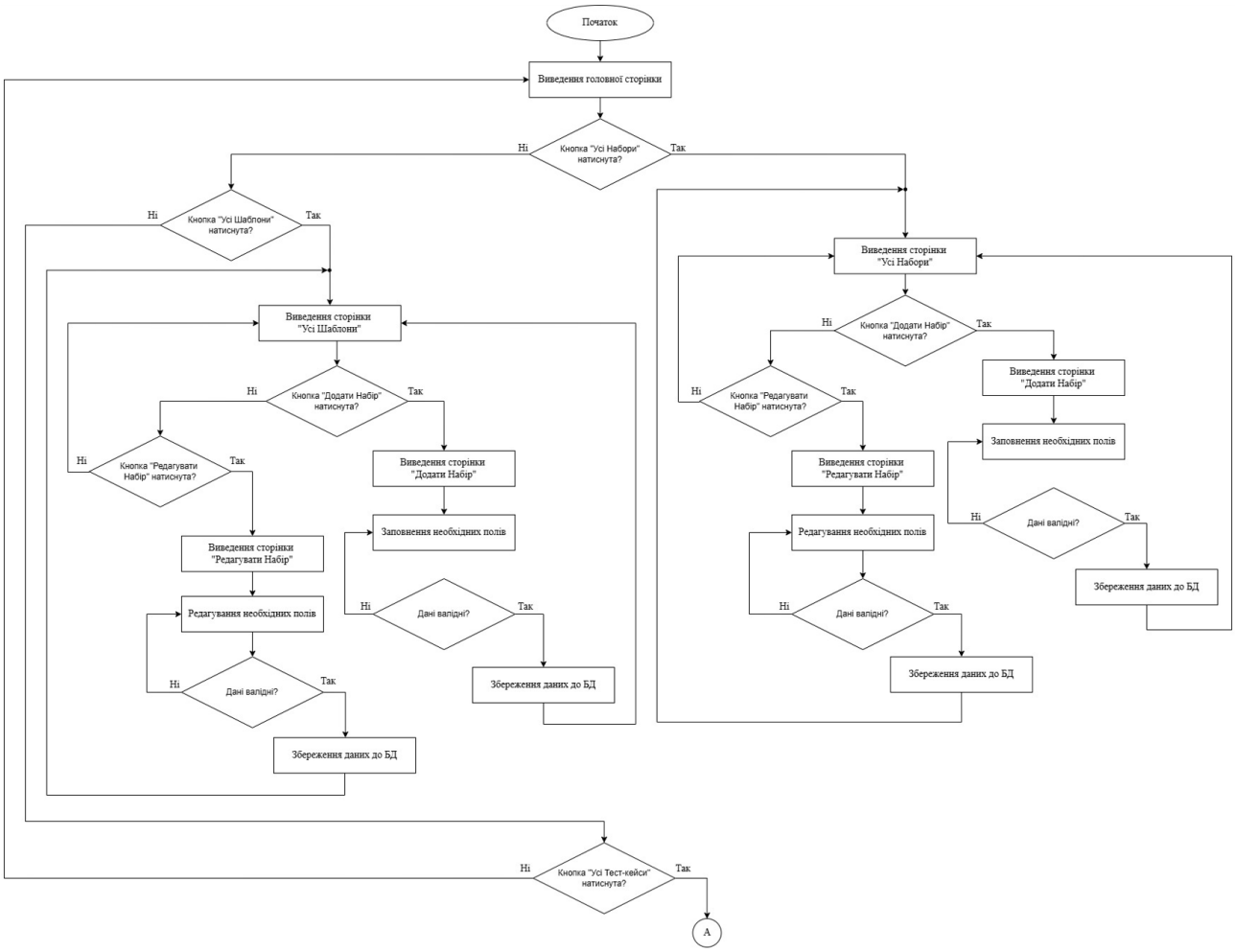


Рисунок 3.1 – Узагальнений алгоритм роботи програмного продукту

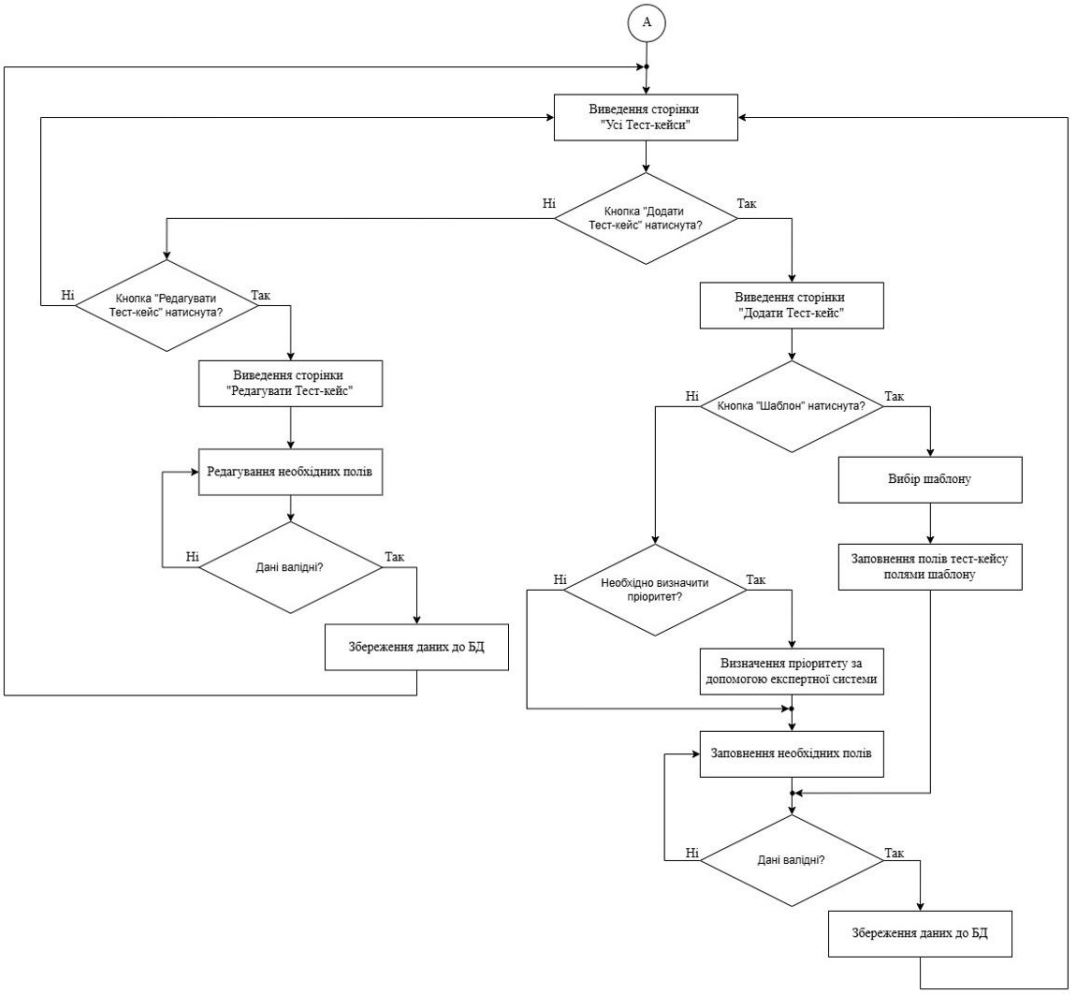


Рисунок 3.1, аркуш 2

Таким чином, у алгоритмі розглянуто узагальнений процес роботи з тест-кейсами, наборами та шаблонами.

3.5 Тестовий приклад роботи програмного продукту і аналіз результатів

З метою тестування функціоналу інформаційної технології для організації управління тест-кейсами протестовано різні функції враховуючи вихідні дані. Для перевірки змоделюємо ситуацію тестування інтернет-магазину.

Після запуску програмного продукту, на екран виводиться головна сторінка (Рис. 3.2).

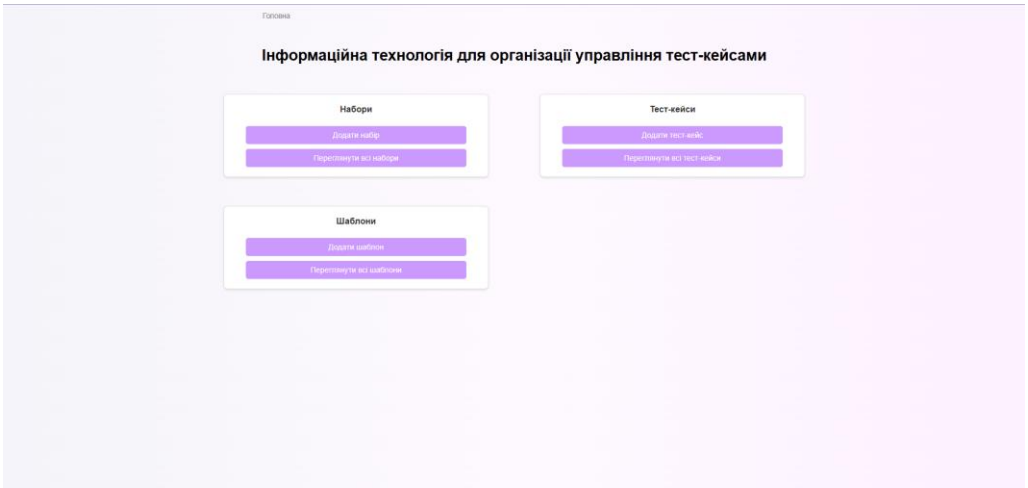


Рисунок 3.2 – Головна сторінка

Створимо набори текст-кейсів з метою подальшого додавання тестових сценаріїв. Після натиснення на кнопку «Додати Набір» відкривається сторінка для додавання нового набору із полем для введення його назви (Рис. 3.3).

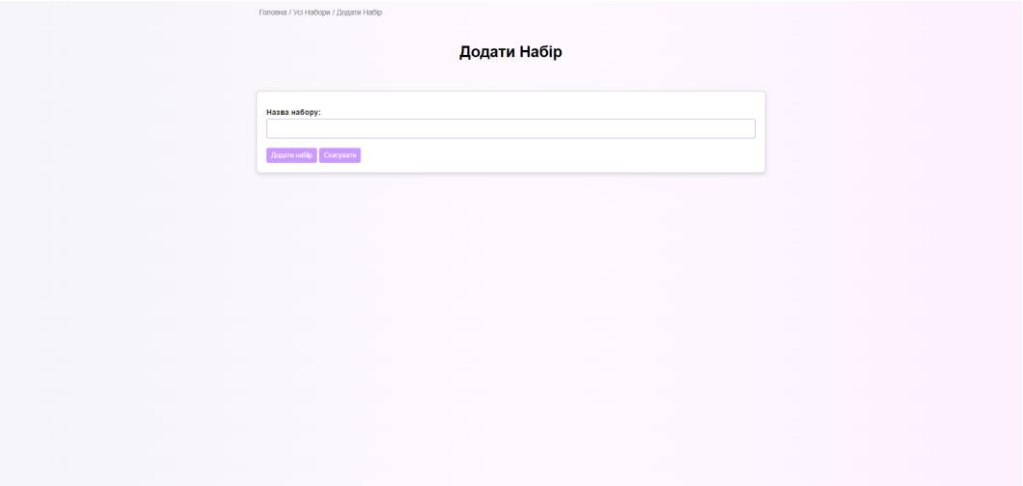


Рисунок 3.3 – Сторінка додавання нового набору

Створимо шість наборів: «Логін», «Реєстрація», «Особистий кабінет», «Кошик», «Список товарів» та «Відновити пароль» (Рис. 3.4).

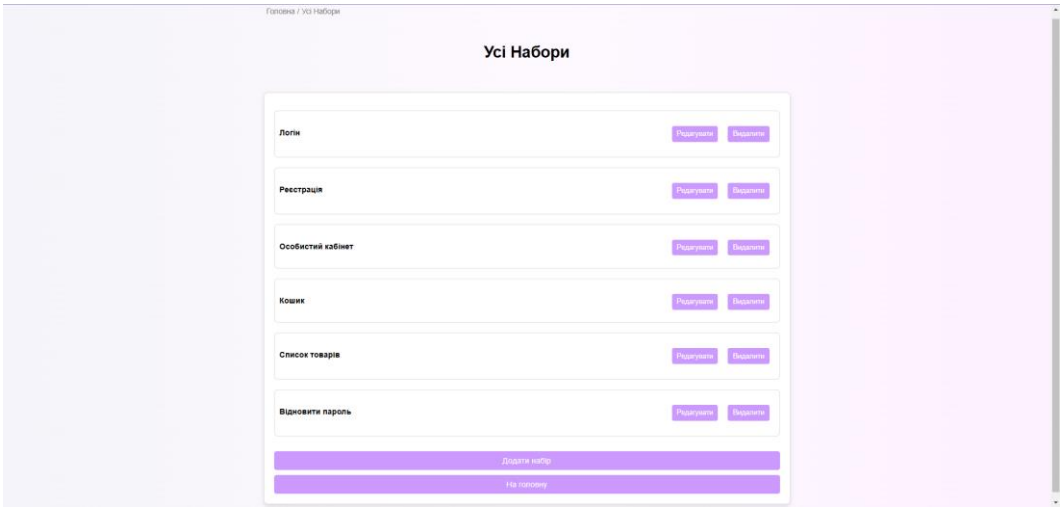


Рисунок 3.4 – Сторінка з усіма наборами

Переконаємось, що при натисненні на кнопку «Редагувати» біля відповідного набору відкривається сторінка редагування набору (Рис. 3.5).

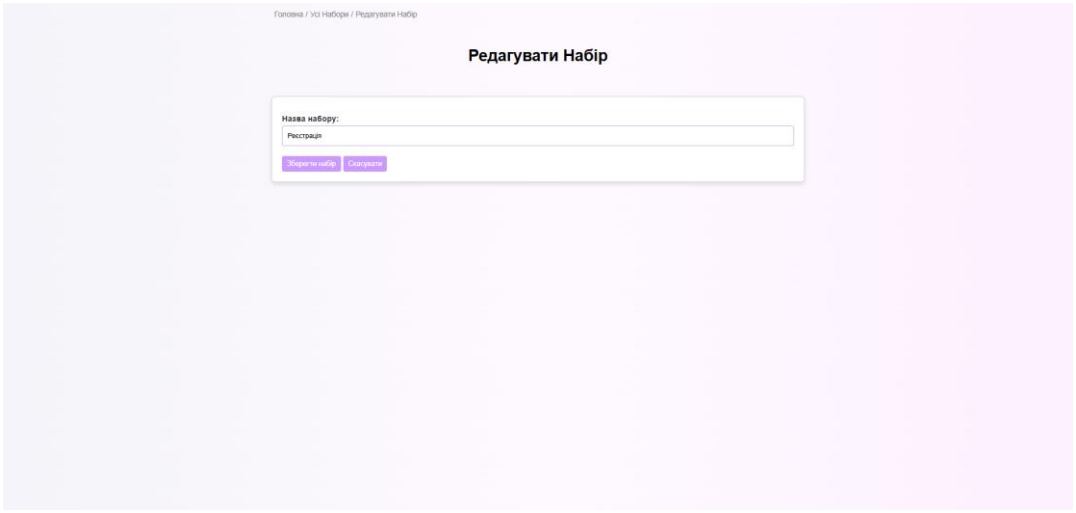


Рисунок 3.5 – Сторінка з усіма наборами

Перевіримо сторінки «Усі Шаблони», «Додати Шаблон» та «Редагувати Шаблон» (Рис. 3.6-3.8).

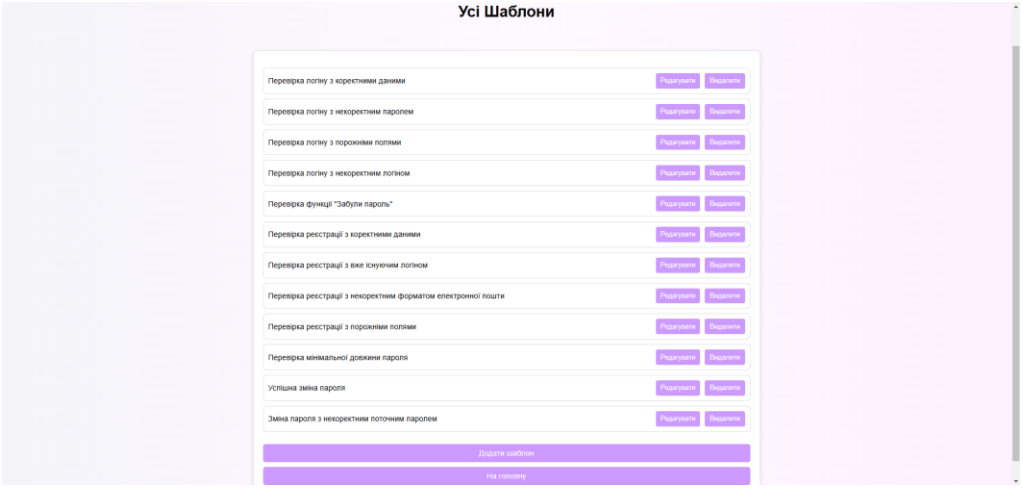


Рисунок 3.6 – Сторінка з усіма шаблонами

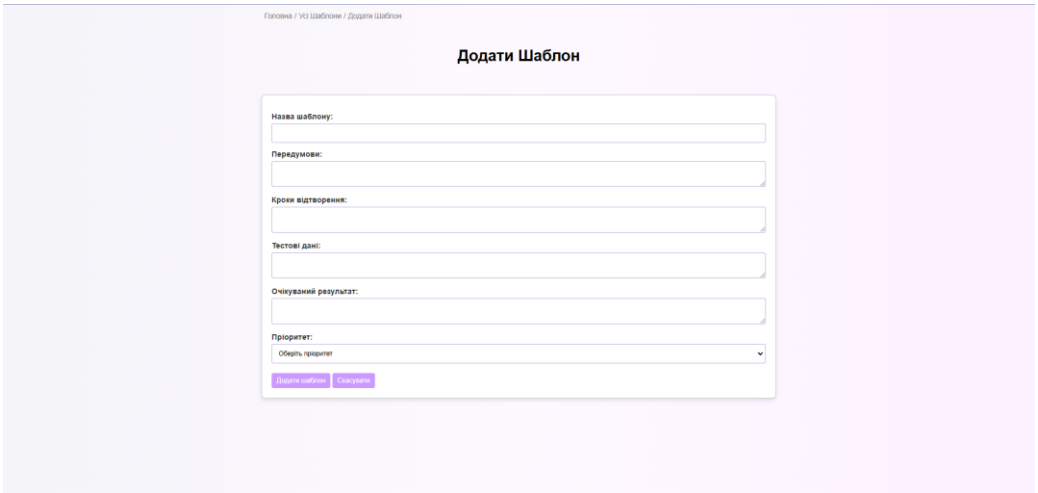


Рисунок 3.7 – Сторінка з додаванням шаблону

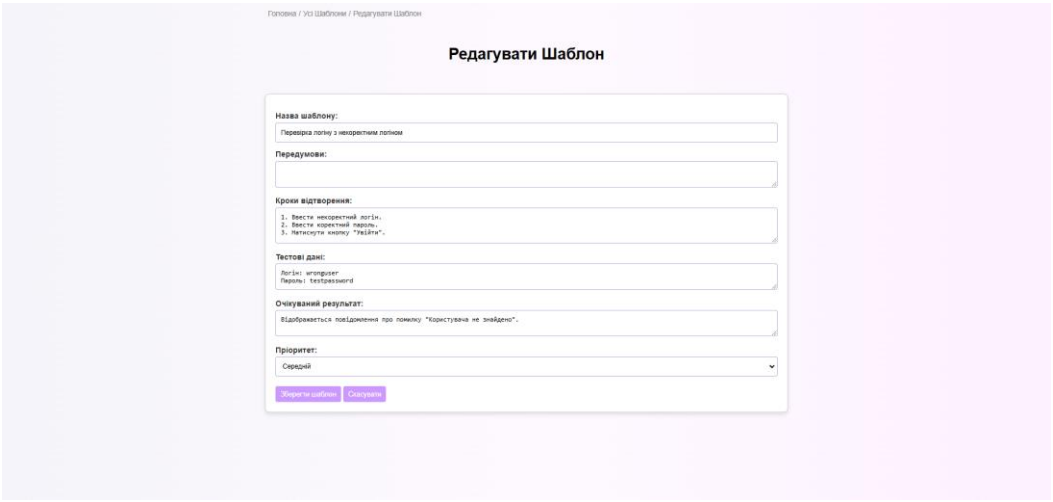


Рисунок 3.8 – Сторінка з редагуванням шаблону

Перейдемо до сторінки «Додати тест-кейс» (Рис. 3.9).

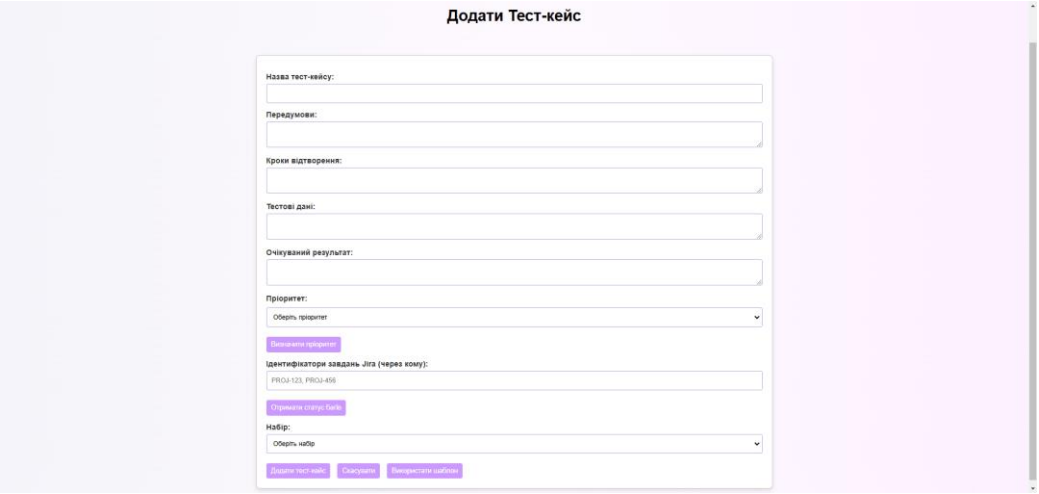


Рисунок 3.9 – Сторінка додавання нового тест-кейсу

Поля для тест-кейсу можна заповнити вручну, або натиснути на кнопку «Вибрати шаблон» та у модальному вікні обрати потрібний шаблон (Рис. 3.10).

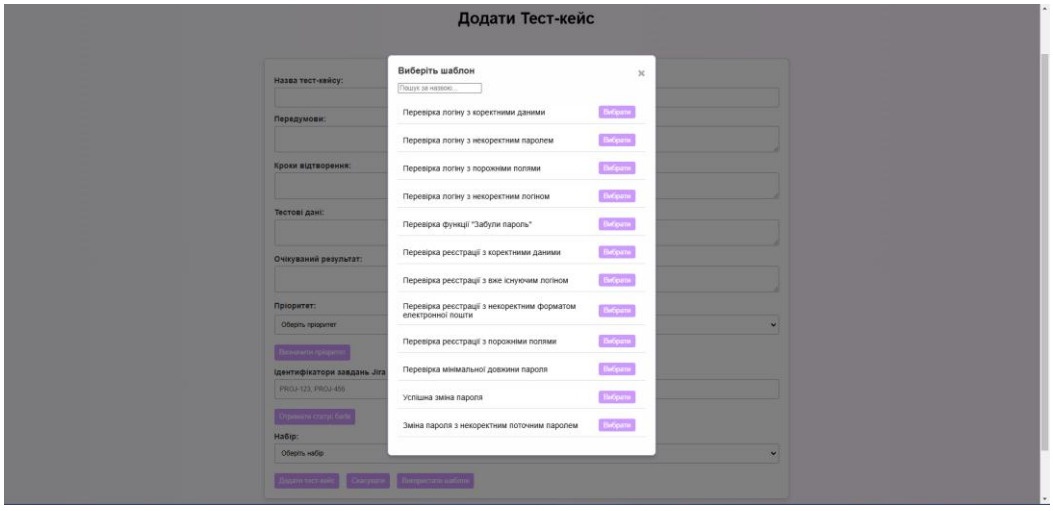


Рисунок 3.10 – Вибір шаблону

Після вибору шаблону усі поля заповняються полями шаблону (Рис. 3.11).

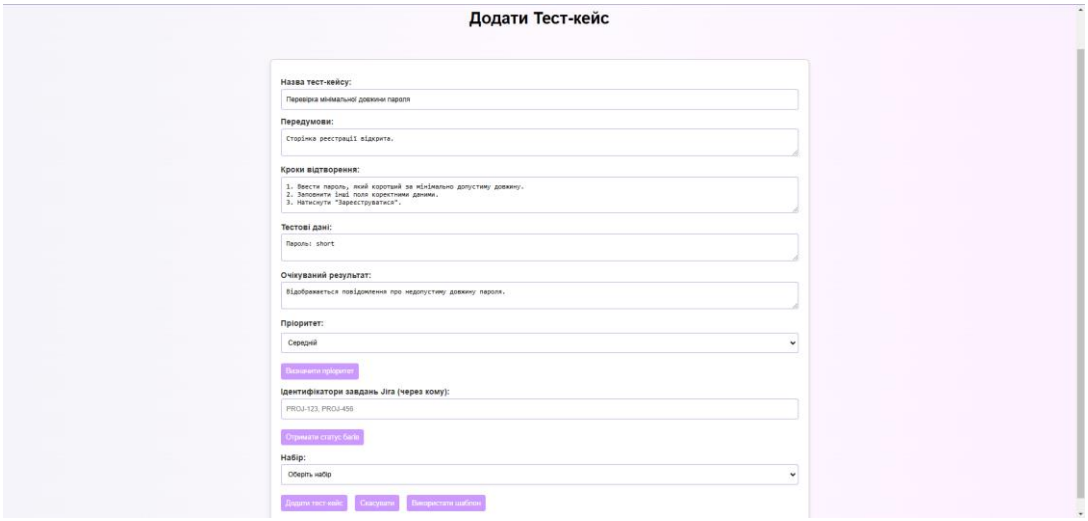


Рисунок 3.11 – Заповнені поля тест-кейсу

Перевіримо можливість додавання задачі Jira до тест-кейса. Введемо кілька ідентифікаторів завдання у відповідне поле та натиснемо на кнопку «Отримати статус багів» (Рис. 3.12).

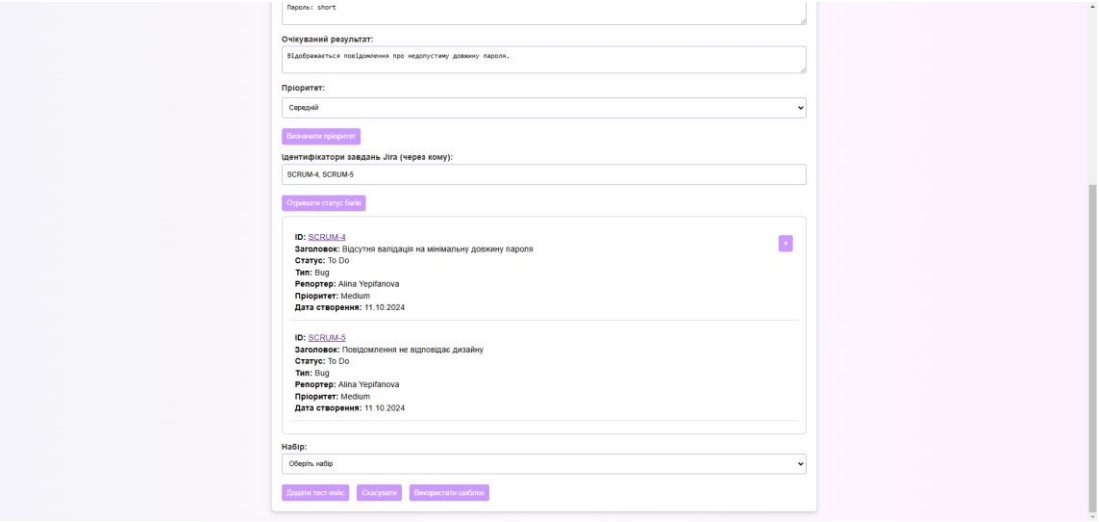


Рисунок 3.12 – Заповнені поля тест-кейсу

Натиснемо на ідентифікатор першого завдання «SCRUM-4» та перевіримо, що у новій вкладці відкривається відповідне завдання у Jira (Рис. 3.13).

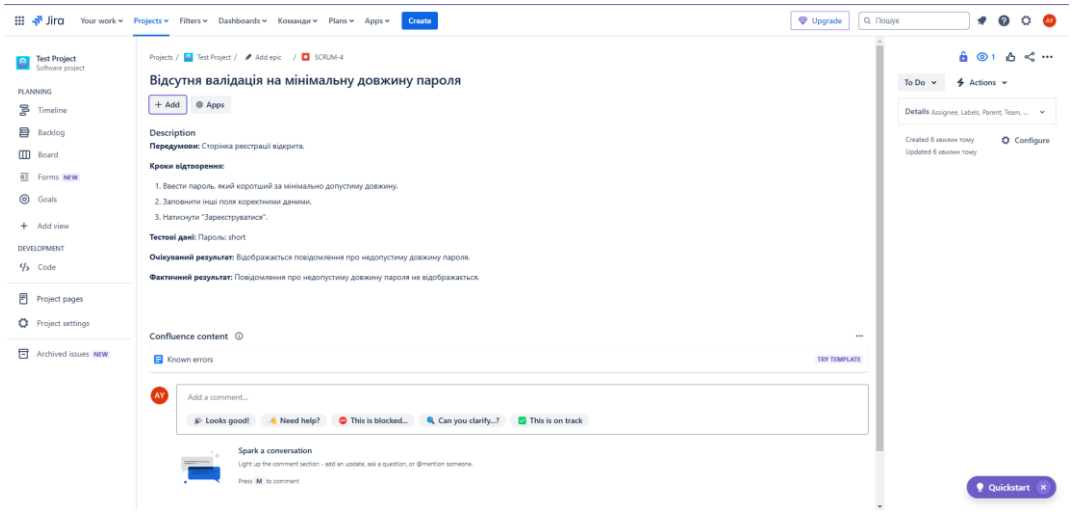


Рисунок 3.13 – Заповнені поля тест-кейсу

Далі натиснемо на кнопку «Визначити пріоритет» (Рис. 3.14).

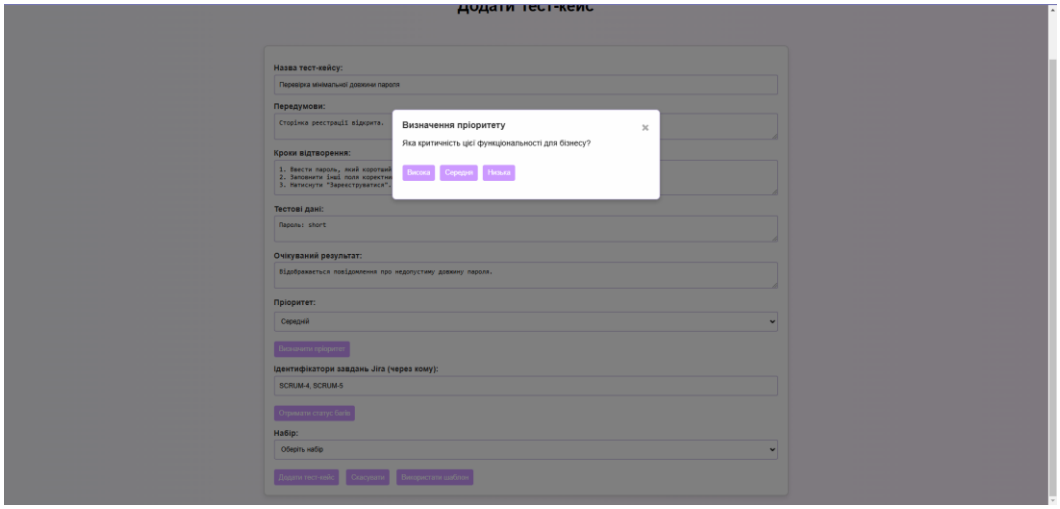


Рисунок 3.14 – Заповнені поля тест-кейсу

Для того, щоб перевірити коректність роботи експертної системи, спробуємо обрахувати пріоритет для кількох тест-кейсів вручну, а після цього порівняємо результати з автоматичним визначенням пріоритету.

Для першого прикладу візьмемо тест-кейс для перевірки відображення логотипу на сторінці логіну. Очікуваний результат: логотип компанії відображається коректно, має правильний розмір і є чітким. У даного тест-кейсу очікуваний пріоритет є низьким.

Дана функціональність може вважатися критичною для бізнесу, адже компанія дбає про коректність логотипу та його відображення. Тому питанню

«Яка критичність цієї функціональності для бізнесу?» поставимо оцінку «Висока».

«Чи є ця функціональність фундаментальною для основних бізнес-процесів?» - «Ні».

На питання «Як часто використовується ця функціональність користувачами?» відповідаємо «Щодня», адже кожного дня принаймні один користувач при користуванні сайту логіниться.

На питання «Який потенційний вплив дефекту в цій функціональності на користувачів?» відповідаємо «Низький», адже скоріше за все користувачі не звертають увагу на відображення логотипу, це на них ніяк не впливає.

«Чи є ця функціональність частиною критичних шляхів у системі?» - «Ні».

«Чи може дефект у цій функціональності спричинити фінансові втрати або пошкодження репутації компанії?» - «Ні».

«Чи були в цій функціональності виявлені дефекти у минулому?» - у даному випадку такої інформації немає, тому можемо відповісти «Ні».

«Яка була частота дефектів у цій області в минулих релізах?» - «Низька». У даному випадку частота дефектів у функціональності не може бути високою, адже логотип рідко змінюється».

«Чи була ця функціональність змінена або оновлена недавно?» - Відповімо «Так».

«Чи є в цій функціональності інтеграція з іншими системами або компонентами?» - «Ні».

«Чи підпадає ця функціональність під регуляторні вимоги?» - «Ні».

«Чи очікуються майбутні зміни в найближчому часі?» - «Ні», адже логотип змінюється рідко.

«Чи є ця функціональність видимою для кінцевих користувачів?» - «Так».

«Наскільки важлива ця функціональність для користувацького досвіду?» - «Неважлива».

«Який вплив має ця функціональність на інші частини системи?» - «Низький вплив».

«Чи можуть дефекти в цій функціональності спричинити проблеми в інших компонентах?» - «Ні».

Як можна побачити на рисунку 3.15, після відповіді на усі питання виставлено пріоритет «Низький».

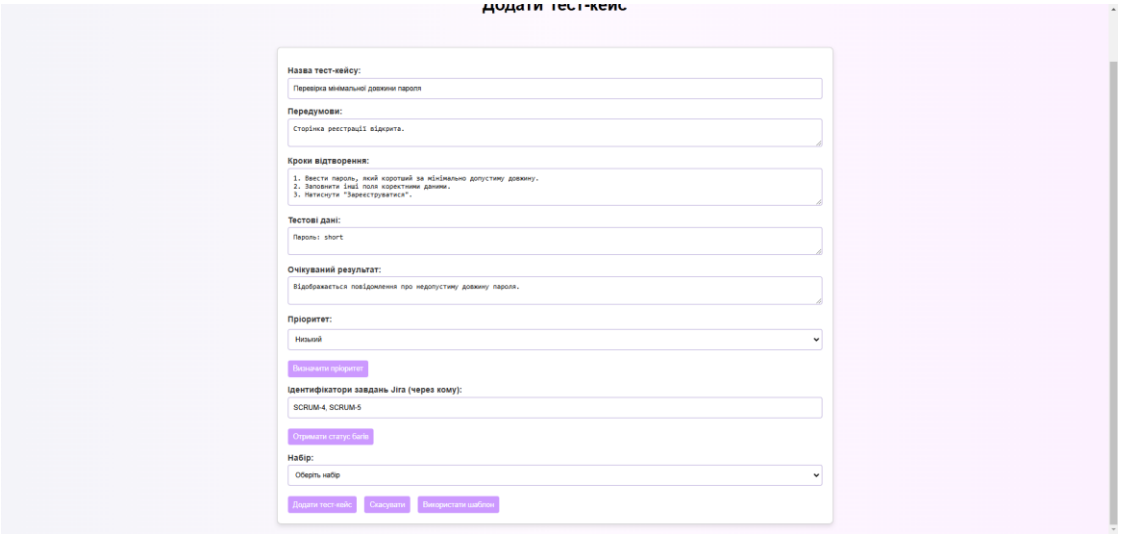


Рисунок 3.15 – Заповнені поля тест-кейсу

Тепер розглянемо тест-кейс із високим пріоритетом - перевірка логіну з коректними даними**.** Очікуваний результат: користувач успішно авторизується і перенаправляється на головну сторінку.

«Яка критичність цієї функціональності для бізнесу?» - «Висока».

«Чи є ця функціональність фундаментальною для основних бізнес-процесів?» - «Так».

«Як часто використовується ця функціональність користувачами?» - «Щодня».

«Який потенційний вплив дефекту в цій функціональності на користувачів?» - «Високий».

«Чи є ця функціональність частиною критичних шляхів у системі?» - «Ні».

«Чи може дефект у цій функціональності спричинити фінансові втрати або пошкодження репутації компанії?» - «Ні».

«Чи були в цій функціональності виявлені дефекти у минулому?» - у даному випадку такої інформації знову немає, тому можемо відповісти «Ні».

«Яка була частота дефектів у цій області в минулих релізах?» - «Низька».

«Чи була ця функціональність змінена або оновлена недавно?» - Відповімо «Ні».

«Чи є в цій функціональності інтеграція з іншими системами або компонентами?» - «Так», адже при логіні часто може використовуватись логін через сторонні сервіси.

«Чи підпадає ця функціональність під регуляторні вимоги?» - «Ні».

«Чи очікуються майбутні зміни в найближчому часі?» - «Ні».

«Чи є ця функціональність видимою для кінцевих користувачів?» - «Так».

«Наскільки важлива ця функціональність для користувацького досвіду?» - «Важлива».

«Який вплив має ця функціональність на інші частини системи?» - «Середній вплив».

«Чи можуть дефекти в цій функціональності спричинити проблеми в інших компонентах?» - «Так».

Після відповіді на усі питання експертною системою виставлено пріоритет «Високий» (Рис. 3.16).

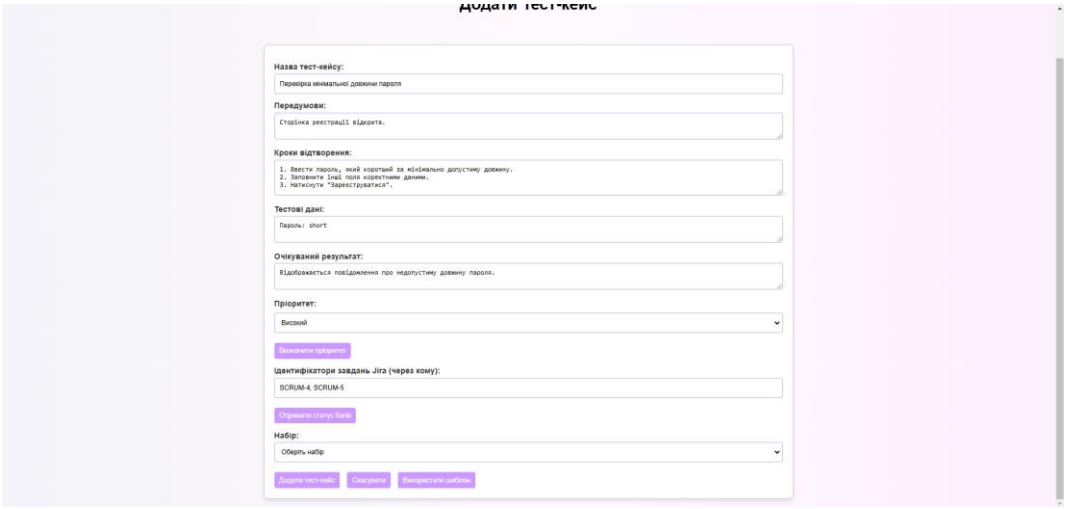


Рисунок 3.16 – Заповнені поля тест-кейсу

Таким чином, можна прийти до висновку, що експертна система працює коректно.

Після заповнення всіх полів відповідною інформацією, натиснемо на кнопку «Додати тест-кейс» та перейдемо на сторінку з усіма тест-кейсами (Рис. 3.17).

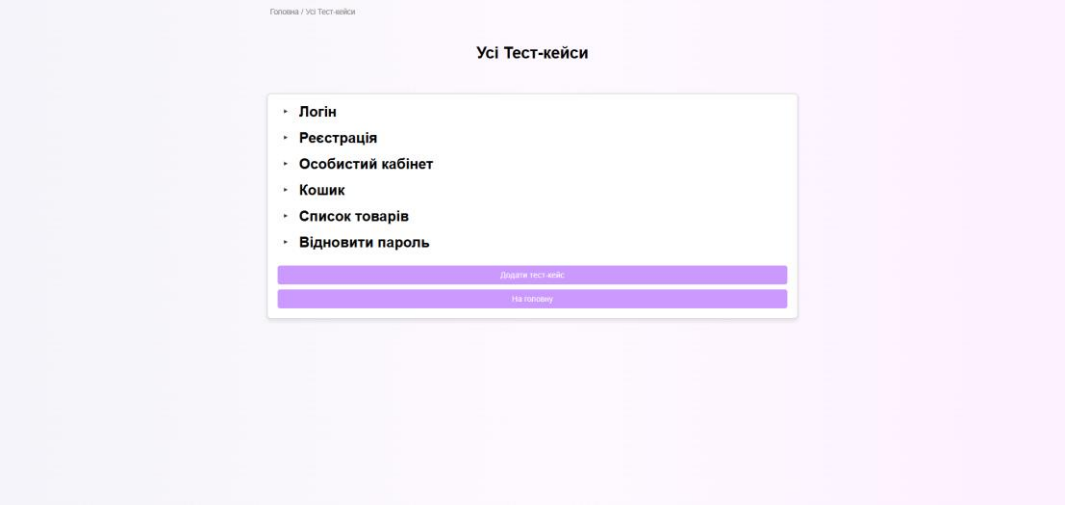


Рисунок 3.17 – Заповнені поля тест-кейсу

На сторінці представлені усі набори. Розгорнемо набір «Реєстрація» та переглянемо, що щойно створений тест-кейс входить у даний набір (Рис. 3.18).

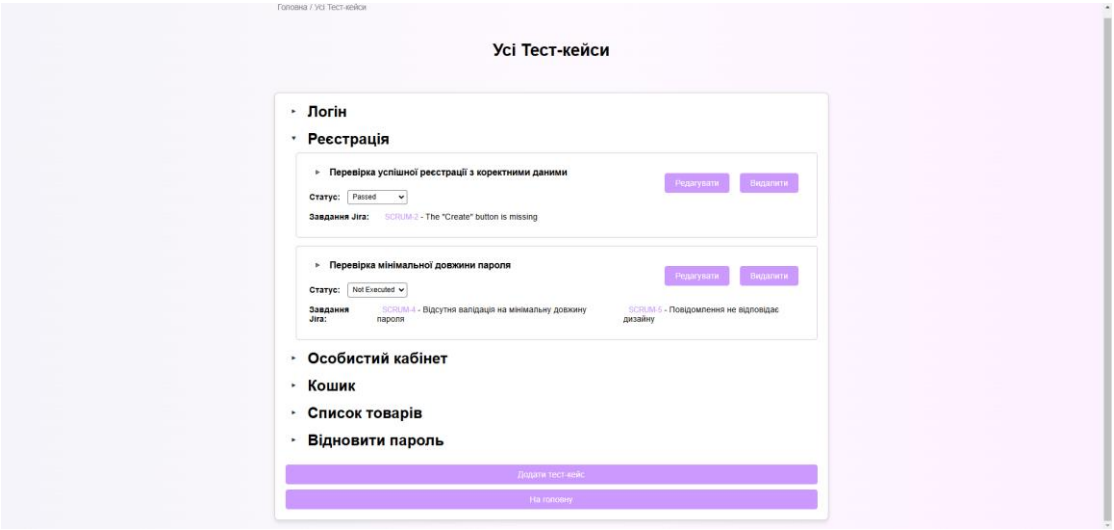


Рисунок 3.18 – Заповнені поля тест-кейсу

Наведемо мишку на додане завдання Jira та переконаємось, що відображається відповідна інформація про нього (Рис. 3.19).

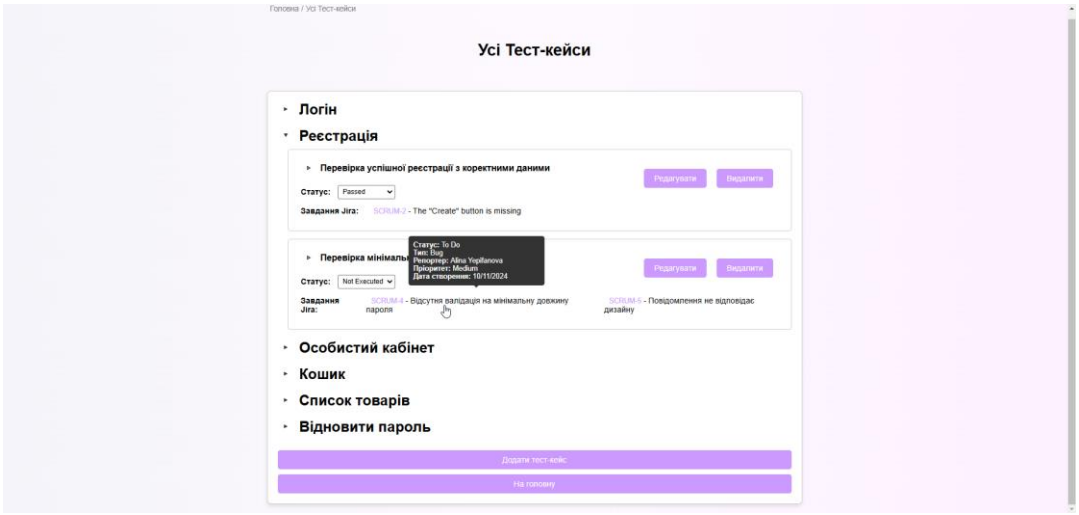


Рисунок 3.19 – Заповнені поля тест-кейсу

Переведемо усі додані завдання у статус «Done» та переконаємось, що статус тест-кейса змінено на «Passed» (Рис. 3.20).

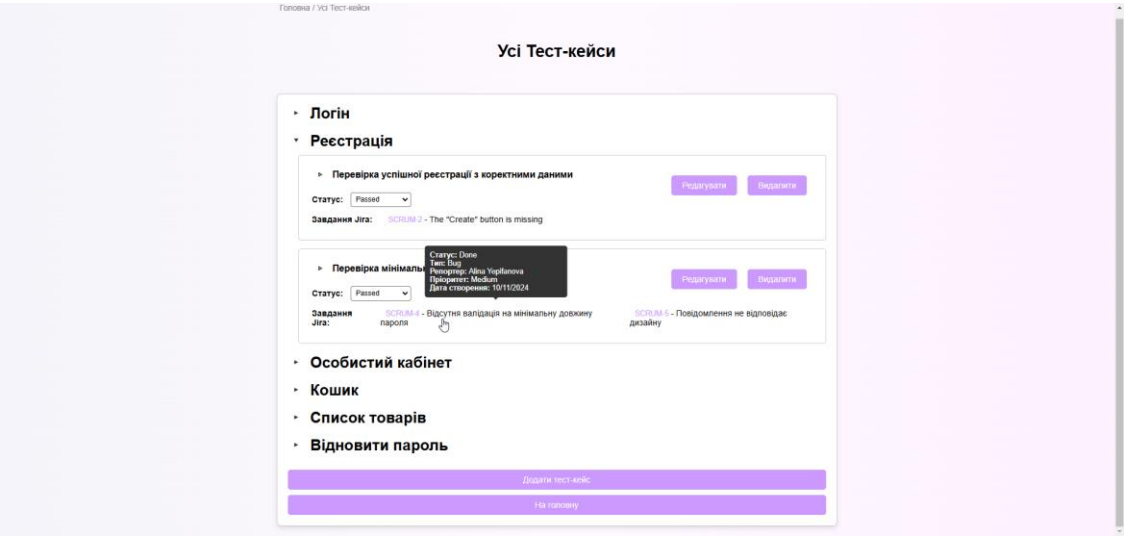


Рисунок 3.20 – Заповнені поля тест-кейсу

Переконаємося, що можемо розгорнути детальну інформацію про тест-кейс (Рис. 3.21).

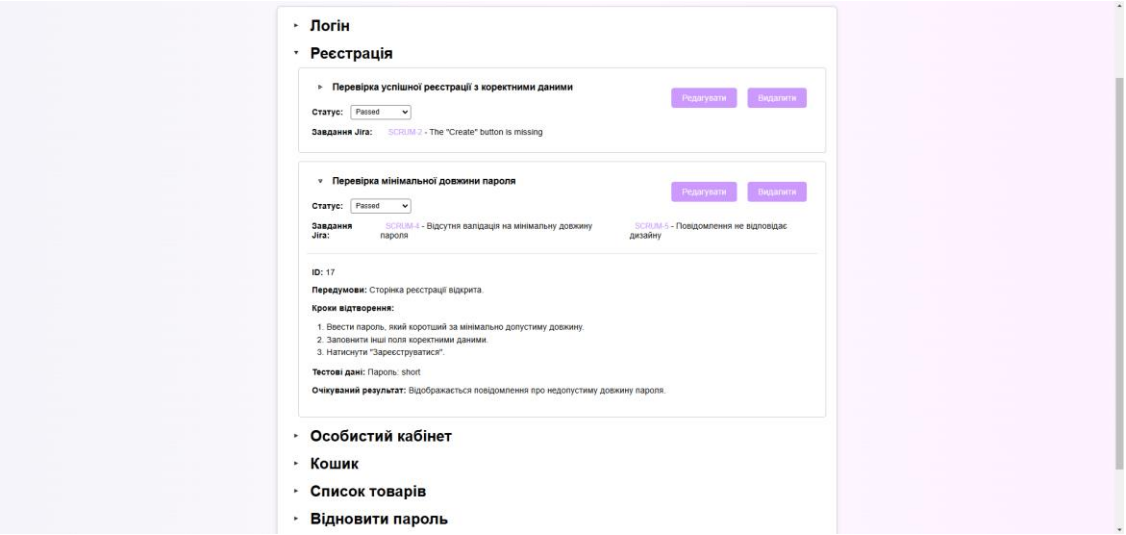


Рисунок 3.21 – Заповнені поля тест-кейсу

Перевіримо можливість зміни статусу тест-кейсів вручну (Рис. 3.22).

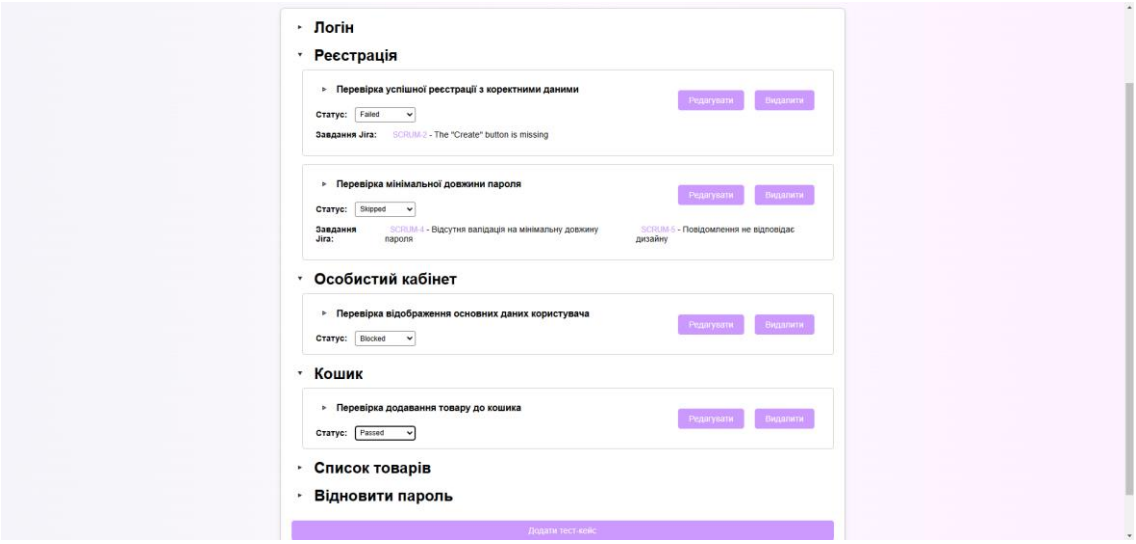


Рисунок 3.22 – Заповнені поля тест-кейсу

Перевіримо, що на сторінці «Редагувати тест-кейс» відображається уся відповідна інформація (Рис. 3.23).

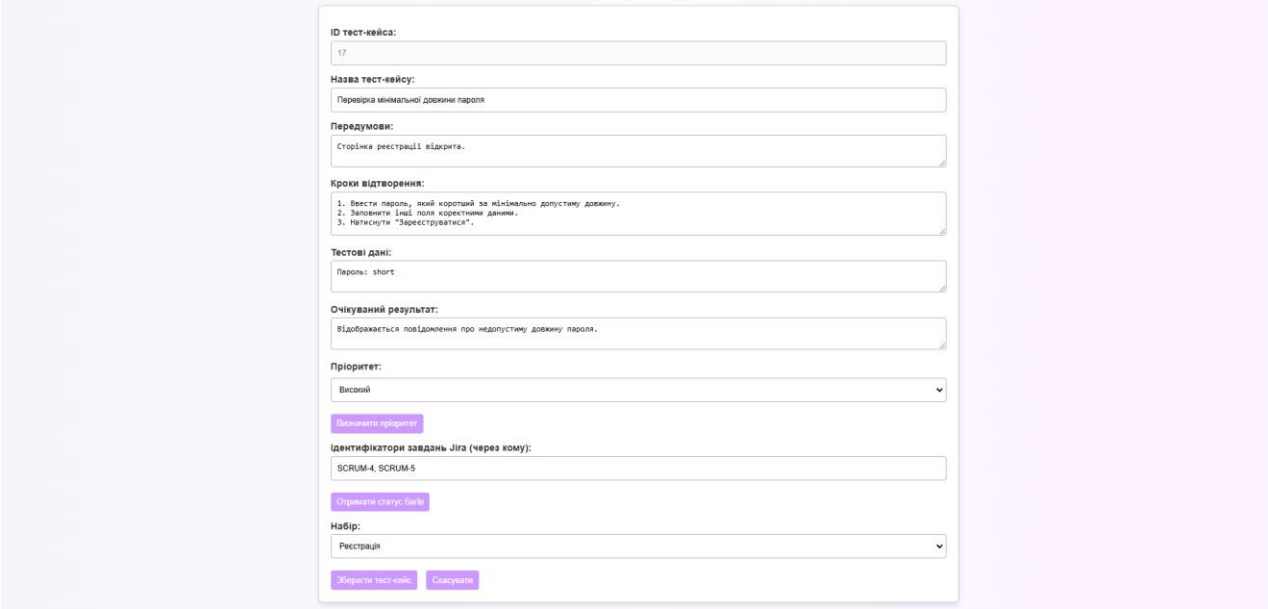


Рисунок 3.23 – Заповнені поля тест-кейсу

Переконаємось, що всі тест-кейси відображаються на сторінці з наборами (Рис. 3.24).

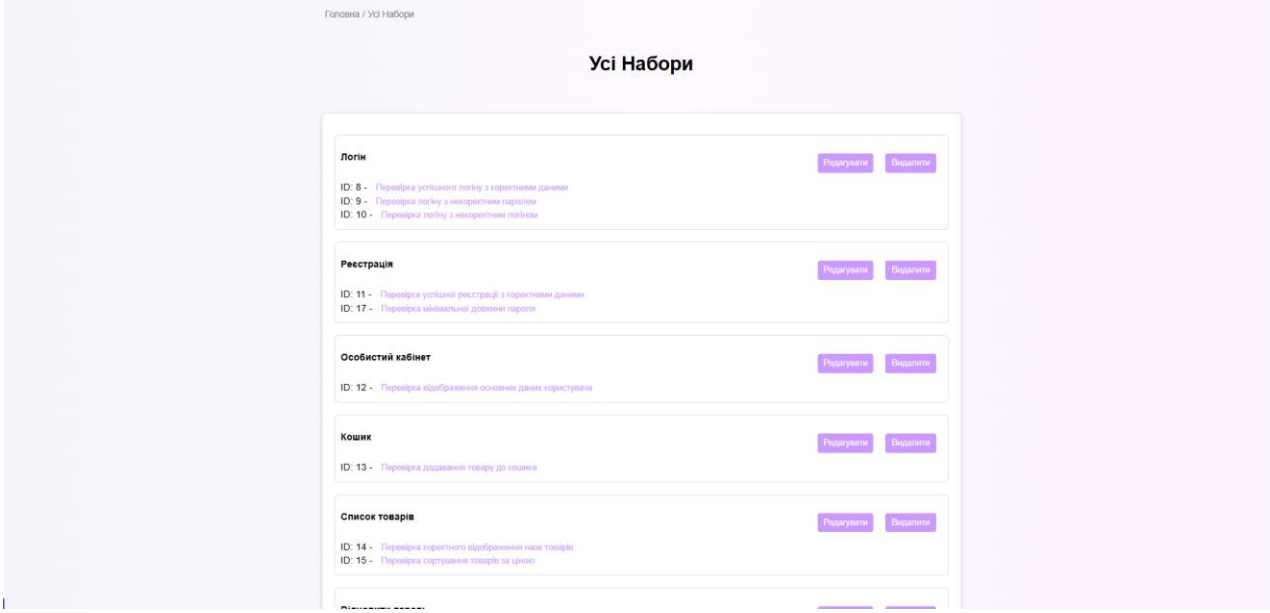


Рисунок 3.24 – Заповнені поля тест-кейсу

Під час виконання тестування програмного модуля для організації управління тест-кейсами, створено 12 шаблонів, 6 наборів тест-кейсів, 10 тест-кейсів, кілька разів перевірено функціональність редагування, видалення та

перегляду деталей наборів, тест-кейсів та шаблонів, а також переадресацію усіх посилань та кнопок. Весь функціонал працює відповідно до очікуваних результатів.

Порівняння розробленого програмного модуля з існуючими програмами-аналогами TestRail, qTest, Zephyr та TestLink наведена у таблиці 3.2. Характеристики оцінені за шкалою, де «+» – це найкращий показник, «-» – найгірший, «+/-» – середній.

Таблиця 3.2 – Порівняльна характеристика розробленого ПЗ

	TestRail	qTest	Zephyr	TestLink	Розроблена інформаційна технологія
Безкоштовний доступ до функціоналу	-	-	-	+	+
Наявність експертної системи, що забезпечує експертнупріоритеза-цію тест-кейсів	-	-	-	-	+
Можливість автоматичної зміни статусу тест-кейсів за допомогою системи Jira	-	-	-	-	+
Наявність шаблонів для тест-кейсів	-	-	-	-	+
Можливість роботи офлайн	-	-	+/-	-	+
Можливість створювати і зберігати тест-кейси та їх набори	+	+	+	+	+

Отже, з порівняльної таблиці видно, що розроблене програмне забезпечення має більш розширені функціональні можливості по відношенню до існуючих сучасних рішень та відповідає завданню на магістерську кваліфікаційну роботу.

3.6 Висновок до розділу 3

У третьому розділі обґрунтовано вибір вибір технологічних рішень для розробки інформаційної технології управління тест-кейсами, а саме мови, середовища програмування та фреймворку. Проаналізовано мови програмування, що потенційно підходять для розробки інформаційної технології. У результаті було обрано мову Python, яка забезпечує широкий спектр бібліотек і гнучкість у розробці веб-додатків. Як фреймворк для веб-інтерфейсу обрано Flask завдяки його легкості, простоті використання та можливості швидкого створення REST API для інтеграції з іншими сервісами. Для розробки проєкту використано середовище програмування PyCharm, яке надає зручні інструменти для роботи з Python, полегшуючи процес налагодження та тестування.

Розглянуто процес інтеграції системи управління тест-кейсами з інструментом Jira та розроблено алгоритм роботи інформаційної технології. Інтеграція передбачає можливість прив'язки тест-кейсів до завдань у Jira та автоматичного оновлення статусів на основі змін у цих завданнях.

Проведено тестування розробленої інформаційної технології для організації управління тест-кейсами. Результати тестування оброблено та порівняно із програмами-аналогами, що дозволило оцінити переваги і точність розробленої системи.

4 ЕКОНОМІЧНА ЧАСТИНА

Успішне впровадження науково-технічної розробки можливе за умови, що вона відповідає актуальним вимогам науково-технічного прогресу та враховує економічні аспекти. Важливим етапом є оцінка економічної ефективності результатів науково-дослідної роботи.

Магістерська кваліфікаційна робота, присвячена розробці та дослідженню на тему «Інформаційна технологія для організації управління тест-кейсами», належить до категорії науково-технічних проектів, орієнтованих на комерціалізацію. Рішення щодо виходу розробки на ринок може бути ухвалене ще в процесі її виконання, що відкриває можливість реалізації отриманих результатів у реальних умовах.

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту дослідження на тему «Інформаційна технологія для огранізації управління тест-кейсами» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Для проведення технологічного аудиту використано таблицю 4.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу [25].

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Таблиця 4.1 – Продовження

Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Таблиця 4.1 – Продовження

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використо-вуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використо-вуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 4.3. Експертами в опитуванні були: Роман Бралатан – .NET developer в компанії

Delphi, Володимир Кушнір – .NET developer в компанії Delphi, Павліченко Юрій – Automation QA в компанії Delphi

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Експерти		
	Роман Бралатан	Володимир Кушнір	Павліченко Юрій
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	2	2	1
3. Ринкові переваги (ціна продукту)	2	2	2
4. Ринкові переваги (технічні властивості)	3	3	3
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	3	2	3
7. Ринкові перспективи (конкуренція)	3	3	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	4	3	4
10. Практична здійсненність (необхідність нових матеріалів)	5	5	5
11. Практична здійсненність (термін реалізації)	4	3	4
12. Практична здійсненність (розробка документів)	5	5	5
Сума балів	$СБ_1$=42	$СБ_2$=39	$СБ_3$=40
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{42+39+40}{3} = 40$		

За результатами розрахунків, наведених в таблиці 4.3, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в таблиці 4.2.

Згідно проведених досліджень рівень комерційного потенціалу розробки, розрахований на основі висновків експертів склав 40 балів, що, відповідно до таблиці 4.2, свідчить про рівень комерційного потенціалу розробки вище середнього.

На початку розробка може бути реалізована шляхом розміщення програмного файлу у репозиторії GitHub та поширенням посилання та опису розробки через професійні соціальні мережі, на приклад, LinkedIn. Після отримання перших відгуків від реальних користувачів, розробка буде доопрацьована та розміщена на спеціально створеному сервері. Детальний опис програмного забезпечення та посилання на сервер буде оформлено у вигляді статті та розміщено на спеціалізованих веб-сайтах, на приклад, DOU.

Основною цільовою аудиторією є тестувальники програмного забезпечення, розробники та власники айті компаній.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія для організації управління тест-кейсами», під час планування, обліку і обчислення собівартості науково-дослідної роботи групуються за певними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, науковим, інженерно-технічним працівникам, технологам та іншим працівникам, безпосередньо зайнятим виконанням конкретної задачі.

Основна заробітна плата кожного з дослідників $З_о$ розраховуємо у відповідності до посадових окладів працівників, за формулою 4.1:

$$З_о = \sum_{i=1}^{k} \frac{M_{mi} \cdot t_i}{Т_р}, \quad (4.1)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

M_{mi} – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

$Т_р$ – середня кількість робочих днів в місяці, $Т_р$ = 21 дні.

$$З_о = 25000 \cdot \frac{30}{21} = 34091 \; грн.$$

Проведені розрахунки зведено до таблиці 4.4.

Таблиця 4.4 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник проекту	25000	1136,4	30	34091
Інженер-програміст	15000	681,8	40	34091
Всього				68182

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт НДР розраховуємо за формулою 4.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 4.3:

$$C_i = \frac{M_m \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (4.3)$$

де M_m – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо M_m = 8000 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати;

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$$C_i = \frac{8000 \cdot 1 \cdot 1{,}65}{21 \cdot 8} = 78{,}6 \; грн.$$

$$З_{p1} = 78{,}6 \cdot 2 = 157{,}2 \; грн.$$

Таблиця 4.5 – Величина витрат на основну заробітню плату робітників

Найменування робіт	Тривалість, год.	Розряд роботи	Погодинна тарифна ставка, грн.	Величина оплати на робітника, грн.
Збір вимог	2	1	65,8	157,2
Проектування	30	3	88,8	2665,0
Розробка	25	5	111,9	2796,7
Тестування	10	2	72,4	723,8
Реалізація	3	4	59,8	179,6
Всього				6522,2

Додаткова заробітна плата дослідників та робітників. Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 4.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (68182 + 6522{,}2) \cdot \frac{11}{100\%} = 8217{,}46 \text{ грн.}$$

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 4.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймемо 22%.

$$З_н = (68182 + 6522{,}2 + 8217{,}46) \cdot \frac{22}{100\%} = 376916{,}6 \text{ грн.}$$

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Інформаційної технології організації заходів".

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 4.6:

$$\text{М} = \sum_{j=1}^{n} H_j \cdot \text{Ц}_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot \text{Ц}_{\text{в}j}, \quad (4.6)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

Ц_j – вартість матеріалу j-го найменування, грн/кг;

K_j– коефіцієнт транспортних витрат, ($K_j = 1{,}1 \ldots 1{,}15$);

B_j – маса відходів j-го найменування, кг;

$\text{Ц}_{\text{в}j}$ – вартість відходів j-го найменування, грн/кг.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 шт, грн.	Норма витрат, шт.	Вартість витраченого матеріалу, грн.
Папір А4	180	1	180
Ручка	20	2	40
Диск оптичний OPTIMA CD	15,5	2	31
Flesh-пам'ять GOODRAM 64 C10A	300	1	300
Всього			551
З врахуванням коефіцієнта транспортування			606,1

4.2.4 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 4.5:

$$\text{А}_{\text{обл}} = \frac{\text{Ц}_{\text{б}}}{T_{\text{в}}} \cdot \frac{t_{\text{вик}}}{12}, \quad (4.7)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

K_c – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = \frac{45000}{1} \cdot \frac{2}{12} = 1875 \text{ грн.}$$

Таблиця 4.7 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Ноутбук	45000	2	1	1875,00
Монітор	11000	2	1	458,33
Приміщення	250000	20	1	791,67
Всього				3125,00

4.2.5 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію $В_е$ розраховуємо за формулою 4.8:

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot K_{впі}}{n_i}, \quad (4.8)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,5 грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

n_i – коефіцієнт корисної дії обладнання, $n_i < 1$.

$$В_е = \frac{0,25 \cdot 250,0 \cdot 7,5 \cdot 0,5}{0,8} = 295,97 \text{ грн.}$$

4.2.6 Службові відрядження

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 4.9:

$$З_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.9)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження». Приймемо 20%.

$$З_{св} = (68182 + 6522,2) \cdot \frac{20}{100\%} = 14935,69 \text{ грн.}$$

4.2.7 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 4.10:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.10)$$

де $І_в$ – норма нарахування за статтею «Інші витрати». Приймемо 50%.

$$І_в = (68182 + 6522{,}2) \cdot \frac{50}{100\%} = 37339{,}21 \text{ грн.}$$

4.2.8 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науковотехнічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 4.11:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.11)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати». Приймемо 100%.

$$Н_{нзв} = (68182 + 6522{,}2) \cdot \frac{100}{100\%} = 74678{,}43 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія для організації управління тест-кейсами», розраховуємо як суму всіх попередніх статей витрат за формулою 4.12:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.12)$$

$$В_{заг} = 68182 + 6522{,}6 + 8217{,}46 + 188236{,}47 + 606{,}1 + 1875 + 3125 + 292{,}97 + 14935{,}69 + 37339{,}21 + 74678{,}43 = 232109{,}76 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 4.13:

$$ЗВ = \frac{В_{заг}}{n}, \quad (4.11)$$

де n – коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи. Приймемо 0,7.

$$ЗВ = \frac{232106{,}92}{0{,}7} = 1331581{,}3 \textit{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із трьох років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою 4.12:

$$\Delta П_{\text{і}} = \sum_1^n (\Delta Ц_{\text{о}} * N * Ц_{\text{о}} * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\upsilon}{100}\right), \quad (4.12)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_{\text{о}}$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. ρ = 0,25;

υ – ставка податку на прибуток. У 2024 році – 18%.

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (1 \cdot 500 + 5000 \cdot 500) \cdot 0{,}83 \cdot 0{,}4 \cdot \left(1 - \frac{0{,}18}{100}\%\right)$$

$= 469858{,}29$ грн.

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = П2\ (1 \cdot 500 + 5000 \cdot (500 + 1000)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1 - 0{,}18/100\%)$$

$= 1409818{,}6$ грн.

$$\Delta П_3 = (1 \cdot 500 + 5000 \cdot (500 + 1000 + 2000)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1 - 0{,}18 /100\%) = 3288910{,}1$$ грн.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{4.13}$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо 2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 3282746,34 грн.

$$PV = 2 \cdot 3282746{,}34 = 663162{,}62 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \tag{4.14}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 3282746,34 грн;

$$\text{ПП} = \sum_1^T \frac{\Delta \Pi_\text{i}}{(1+\tau)^t}\ , \tag{4.13}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$Е_{абс} = 3282746{,}34 - 663162{,}62 = 2619583{,}72 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього користуються формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1\,, \qquad (4.14)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{2619583{,}72}{663162{,}62}} - 1 = 1{,}07.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f\,, \qquad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,14…0,2);

f – показник, що характеризує ризикованість вкладень, приймемо 0,25.

$$\tau_{min} = 0{,}1 + 0{,}25 = 0{,}35.$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$\mathrm{T_{ок}} = \frac{1}{\mathrm{E_в}}. \quad (4.16)$$

$$\mathrm{T_{ок}} = \frac{1}{0{,}7} = 0{,}9 \text{ (роки)}.$$

$\mathrm{T_{ок}} \leq 3$ років, що свідчить про комерційну привабливість науковотехнічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновок до розділу 4

Згідно проведених досліджень рівень комерційного потенціалу розробки на тему «Інформаційна технологія для організації управління тест-кейсами» становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього. При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,26 рази. Також термін окупності становить 9 місяців, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Отже можна зробити висновок про доцільність проведення науководослідної роботи на тему «Інформаційна технологія для організації управління тест-кейсами».

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології та програмного забезпечення розв'язання задачі організації управління тест-кейсами.

У першому розділі здійснено аналіз предметної області управління тест-кейсами, розглянуто сучасні підходи до їх організації, а також визначено основну задачу дослідження. Проведено огляд популярних інструментів для роботи з тест-кейсами, із зазначенням їх переваг та недоліків. Виконано порівняльний аналіз цих систем за критеріями зручності використання, функціональності, вартості та продуктивності. Окрему увагу приділено використанню експертних систем, які продемонстрували високий потенціал у сфері оптимізації управління тест-кейсами.

У другому розділі виконано аналіз процесів організації управління тест-кейсами. Здійснено класифікацію підходів до тестування програмного забезпечення, розглянуто структуру тест-кейсів та їх життєвий цикл. Проведено аналіз існуючих підходів до організації та управління наборами тест-кейсів. Визначено ключові фактори, що впливають на ефективність управління. Також досліджено сучасні технології експертних систем для автоматизації управління тест-кейсами, зокрема для автоматизованої пріоритезації тестів на основі об’єктивних критеріїв, що включають ризики, історичні дані та критичність функціональних елементів програмного забезпечення. Розроблено модель експертної системи для пріоритезації тест-кейсів та алгоритм її функціонування, що дозволяють інтегрувати ці підходи в систему для організації управління тест-кейсами. Розроблено структуру функціонування інформаційної технології та створено UML діаграму класів.

У третьому розділі обґрунтовано вибір вибір технологічних рішень для розробки інформаційної технології управління тест-кейсами. Розглянуто процес інтеграції системи управління тест-кейсами з інструментом Jira та розроблено алгоритм роботи інформаційної технології. Проведено тестування розробленої

інформаційної технології для організації управління тест-кейсами. Результати тестування оброблено та порівняно із програмами-аналогами, що дозволило оцінити переваги і точність розробленої системи.

У четвертому розділі виконано розрахунок витрат на розробку. Рівень комерційного потенціалу розробки становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень. При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,26 рази. Також термін окупності становить 9 місяців, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аналіз перспектив застосування експертних систем у сфері тестування програмного забезпечення / А.О. Єпіфанова, А.А. Яровий : Збірник матеріалів Міжнародної науково-практичної конференції "Молодь в науці: дослідження, проблеми, перспективи (МН-2025)". – В.: ВНТУ, 2024. – С. 1-3. [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/22821/18810

2. The Art of Service. Test Cases A Complete Guide - 2021 Edition, 2021. 307 с.

3. Test Case Prioritization Techniques and Metrics – [Електронний ресурс]. – Тип доступу: https://www.testrail.com/blog/test-case-prioritization/

4. TestRail. – [Електронний ресурс]. – Тип доступу: https://www.testrail.com/

5. QTest. – [Електронний ресурс]. – Тип доступу: https://www.tricentis.com/products/unified-test-management-qtest/test-case-manager

6. Zephyr – [Електронний ресурс]. – Тип доступу: https://smartbear.com/test-management/zephyr/

7. TestLink. – [Електронний ресурс]. – Тип доступу: https://testlink.org/

8. Роль та ефективність засобів штучного інтелекту в тестуванні. – [Електронний ресурс].– Тип доступу: https://journals.maup.com.ua/index.php/it/article/view/3950/4309

9. Dorothy Graham. Foundations of Software Testing ISTQB Certification, 2024. 288 с.

10. 7 Reasons Why Software Testing is Important. – [Електронний ресурс]. – Тип доступу: https://www.indiumsoftware.com/blog/why-software-testing/

11. Класифікаційний аналіз методів тестування програмного забезпечення / А.О. Єпіфанова, А.А. Яровий – Тези LII науково-технічної

конференції підрозділів ВНТУ (НТКП ВНТУ-2023). – [Електронний ресурс]. – Тип доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/18156

12. Istvan Forgacs. Modern Software Testing Techniques: A Practical Guide for Developers and Testers 1st ed. Edition, 2024. 284 с.

13. Що таке цілісність тест-кейса або як правильно описати тест-кейс, щоб перевірити функціонал? – [Електронний ресурс]. – Тип доступу: https://training.qatestlab.com/blog/course-materials/test-case-description/

14. Mauricio Aniche. Effective Software Testing, 2022. 328 с.

15. What is a Test Suite & Test Case? – [Електронний ресурс]. – Тип доступу: https://www.browserstack.com/guide/what-is-test-suite-and-test-case

16. I. Gupta. Artificial Intelligence and Expert Systems, 2020. 412 с.

17. Месюра В.І., Яровий А.А., Арсенюк І.Р. Експертні системи. Частина 1. Навчальний посібник. – Вінниця: ВНТУ, 2006. 114 с.

18. What is UML. – [Електронний ресурс]. – Тип доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

19. Python. – [Електронний ресурс]. – Тип доступу: https://www.python.org/

20. What is JavaScript. – [Електронний ресурс]. – Тип доступу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

21. What are HTML and CSS used for. – [Електронний ресурс]. – Тип доступу: https://www.futurelearn.com/info/blog/what-are-html-css-basics-of-coding

22. Pycharm. – [Електронний ресурс]. – Тип доступу: https://www.jetbrains.com/pycharm/

23. Flask. – [Електронний ресурс]. – Тип доступу: https://pythonbasics.org/what-is-flask-python/

24. Jira. – [Електронний ресурс]. – Тип доступу: https://www.atlassian.com/jira

25. Козловський В.О., Лесько О.Й., Кавецький В.В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт – Вінниця : ВНТУ, 2021. – 42 с.

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Інформаційна технологія для організації управління тест-кейсами

Тип роботи: магістерська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА

(кафедра, факультет)

Показники звіту подібності Turnitin

Оригінальність 97,00% Схожість 3,00%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи Єпіфанова А.О.

Керівник роботи Яровий А.А.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов’язковий) Лістинг програми

```
import requests
from flask import Flask, render_template, redirect, url_for, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from base64 import b64encode
from flask_migrate import Migrate

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test_management.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
migrate = Migrate(app, db)

# Моделі
class TestSet(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)

class JiraIssue(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    key = db.Column(db.String(50), nullable=False)
    link = db.Column(db.String(200), nullable=False)
    summary = db.Column(db.String(200), nullable=True)
    status = db.Column(db.String(50), nullable=True)
    type = db.Column(db.String(50), nullable=True)
    reporter = db.Column(db.String(100), nullable=True)
    priority = db.Column(db.String(50), nullable=True)
    created = db.Column(db.String(50), nullable=True)
    test_case_id = db.Column(db.Integer, db.ForeignKey('test_case.id'),
nullable=False)

class TestCase(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # Автоінкрементний ID
    name = db.Column(db.String(100), nullable=False)
    prerequisites = db.Column(db.Text, nullable=True)
    steps = db.Column(db.Text, nullable=True)
    data = db.Column(db.Text, nullable=True)
    expected_result = db.Column(db.Text, nullable=False)
    priority = db.Column(db.String(20), nullable=False, default="Середній")
```

```
    status = db.Column(db.String(20), nullable=True, default="Not Executed")
    set_id = db.Column(db.Integer, db.ForeignKey('test_set.id'), nullable=True)
    test_set = db.relationship('TestSet', backref=db.backref('test_cases', lazy=True))
    jira_issues = db.relationship('JiraIssue', backref='test_case', lazy=True,
cascade="all, delete-orphan")  # Зв'язок з JiraIssue

class Template(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    prerequisites = db.Column(db.Text, nullable=True)
    steps = db.Column(db.Text, nullable=True)
    data = db.Column(db.Text, nullable=True)
    expected_result = db.Column(db.Text, nullable=False)
    priority = db.Column(db.String(20), default="Середній")

# Створення бази даних
with app.app_context():
    db.create_all()

def fetch_jira_issue(issue_key):
    """Функція для отримання даних одного завдання з Jira"""
    url = f"{JIRA_API_URL}/issue/{issue_key}"
    headers = {
        "Authorization": f"Basic
{b64encode(f'{JIRA_EMAIL}:{JIRA_API_TOKEN}'.encode()).decode()}",
        "Accept": "application/json"
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        issue_data = response.json()

        # Перетворення дати створення у потрібний формат
        created_date = issue_data['fields']['created']
        formatted_date = datetime.strptime(created_date, "%Y-%m-
%dT%H:%M:%S.%f%z").strftime('%d/%m/%Y')

        return {
            "key": issue_key,
            "link": f"{JIRA_URL}/browse/{issue_key}",
            "summary": issue_data['fields']['summary'],
            "status": issue_data['fields']['status']['name'],
```

```
            "type": issue_data['fields']['issuetype']['name'],
            "reporter": issue_data['fields']['reporter']['displayName'],
            "priority": issue_data['fields']['priority']['name'],
            "created": formatted_date  # Дата у потрібному форматі
        }
    else:
        return None

def fetch_and_update_jira_issue(issue):
    """Отримує статус завдання з Jira і оновлює його в базі даних."""
    jira_data = fetch_jira_issue(issue.key)
    if jira_data:
        issue.status = jira_data['status']
        db.session.commit()

def update_test_case_status(test_case):
    # Оновлюємо статус кожного Jira-issue, прикріпленого до тест-кейсу
    for issue in test_case.jira_issues:
        fetch_and_update_jira_issue(issue)

    # Перевіряємо, чи всі оновлені статуси Jira-issue мають статус "Done"
    if test_case.jira_issues:
        all_resolved = all(issue.status == "Done" for issue in test_case.jira_issues)
    else:
        all_resolved = False

    if all_resolved and test_case.status != "Passed":
        test_case.status = "Passed"
        db.session.commit()  # Оновлюємо статус тест-кейсу

@app.route('/jira_info', methods=['POST'])
def get_jira_info():
    """Маршрут для отримання інформації про Jira завдання на основі списку
ідентифікаторів"""
    data = request.json
    issue_keys = data.get("issue_keys")
    if not issue_keys:
        return jsonify({"error": "Issue keys are missing"}), 400

    issues = []
    for key in issue_keys:
```

```
            issue_data = fetch_jira_issue(key)
            if issue_data:
                issues.append(issue_data)

    return jsonify(issues)


# Маршрут для додавання нового набору
@app.route('/add_set', methods=['GET', 'POST'])
def add_set():
    if request.method == 'POST':
        set_name = request.form.get('name')
        if set_name:
            new_set = TestSet(name=set_name)
            db.session.add(new_set)
            db.session.commit()
            return redirect(url_for('view_sets'))
    return render_template('add_set.html')

# Маршрут для редагування набору
@app.route('/edit_set/<int:set_id>', methods=['GET', 'POST'])
def edit_set(set_id):
    test_set = TestSet.query.get_or_404(set_id)
    if request.method == 'POST':
        test_set.name = request.form.get('name')
        db.session.commit()
        return redirect(url_for('view_sets'))
    return render_template('edit_set.html', test_set=test_set)

# Маршрут для видалення набору
@app.route('/delete_set/<int:set_id>', methods=['POST'])
def delete_set(set_id):
    test_set = TestSet.query.get_or_404(set_id)
    db.session.delete(test_set)
    db.session.commit()
    return redirect(url_for('view_sets'))

# Маршрут для додавання нового тест-кейса
@app.route('/add_test_case', methods=['GET', 'POST'])
def add_test_case():
    sets = TestSet.query.all()  # Для вибору набору зі списку
```

```
templates = Template.query.all()  # Для вибору шаблону

# Конвертація шаблонів у список словників
templates_data = [
    {
        "id": template.id,
        "name": template.name,
        "prerequisites": template.prerequisites,
        "steps": template.steps,
        "data": template.data,
        "expected_result": template.expected_result,
        "priority": template.priority
    }
    for template in templates
]

if request.method == 'POST':
    name = request.form.get('name')
    prerequisites = request.form.get('prerequisites')
    steps = request.form.get('steps')
    data = request.form.get('data')
    expected_result = request.form.get('expected_result')
    priority = request.form.get('priority')
    set_id = request.form.get('set_id')
    jira_keys = request.form.get('jiraLink').split(',')

    if name and expected_result:
        new_test_case = TestCase(
            name=name,
            prerequisites=prerequisites,
            steps=steps,
            data=data,
            expected_result=expected_result,
            priority=priority,
            status="Not Executed",
            set_id=set_id if set_id else None
        )
        db.session.add(new_test_case)
        db.session.commit()

        for key in jira_keys:
```

```
            key = key.strip()
            jira_data = fetch_jira_issue(key)
            if jira_data:
                jira_issue = JiraIssue(
                    key=jira_data['key'],
                    link=jira_data['link'],
                    summary=jira_data['summary'],
                    status=jira_data['status'],
                    type=jira_data['type'],
                    reporter=jira_data['reporter'],
                    priority=jira_data['priority'],
                    created=jira_data['created'],
                    test_case_id=new_test_case.id  # Зв'язок з тест-кейсом
                )
                db.session.add(jira_issue)
        db.session.commit()  # Збереження змін після додавання всіх завдань
        return redirect(url_for('view_test_cases'))
    return render_template('add_test_case.html', sets=sets, templates=templates_data)

# Маршрут для редагування тест-кейса
@app.route('/edit_test_case/<int:test_case_id>', methods=['GET', 'POST'])
def edit_test_case(test_case_id):
    test_case = TestCase.query.get_or_404(test_case_id)
    sets = TestSet.query.all()  # Список наборів для вибору
    if request.method == 'POST':
        test_case.name = request.form.get('name')
        test_case.prerequisites = request.form.get('prerequisites')
        test_case.steps = test_case.steps.replace("1.", "\n1.")
        test_case.data = request.form.get('data')
        test_case.expected_result = request.form.get('expected_result')
        test_case.priority = request.form.get('priority')
        test_case.set_id = request.form.get('set_id')

        # Оновлення Jira-ідентифікаторів
        jira_keys = request.form.get('jiraLink').split(',')
        jira_keys = [key.strip() for key in jira_keys if key.strip()]  # Очищення пробілів
і пустих значень

        # Видалення попередніх завдань Jira для тест-кейсу
        JiraIssue.query.filter_by(test_case_id=test_case.id).delete()
```

```
        # Додавання нових завдань Jira
        for key in jira_keys:
            jira_data = fetch_jira_issue(key)
            if jira_data:
                jira_issue = JiraIssue(
                    key=jira_data['key'],
                    link=jira_data['link'],
                    summary=jira_data['summary'],
                    status=jira_data['status'],
                    type=jira_data['type'],
                    reporter=jira_data['reporter'],
                    priority=jira_data['priority'],
                    created=jira_data['created'],
                    test_case_id=test_case.id  # Зв'язок з тест-кейсом
                )
                db.session.add(jira_issue)

        db.session.commit()
        return redirect(url_for('view_test_cases'))
    return render_template('edit_test_case.html', test_case=test_case, sets=sets)

@app.route('/update_status/<int:test_case_id>', methods=['POST'])
def update_status(test_case_id):
    data = request.json
    new_status = data.get('status')

    test_case = TestCase.query.get_or_404(test_case_id)
    test_case.status = new_status
    db.session.commit()

    return jsonify({"success": True})

# Маршрут для видалення тест-кейса
@app.route('/delete_test_case/<int:test_case_id>', methods=['POST'])
def delete_test_case(test_case_id):
    test_case = TestCase.query.get_or_404(test_case_id)
    db.session.delete(test_case)
    db.session.commit()
    return redirect(url_for('view_test_cases'))

# Маршрут для додавання нового шаблона
```

```
@app.route('/add_template', methods=['GET', 'POST'])
def add_template():
    if request.method == 'POST':
        name = request.form.get('name')
        prerequisites = request.form.get('prerequisites')
        steps = request.form.get('steps')
        data = request.form.get('data')
        expected_result = request.form.get('expected_result')
        priority = request.form.get('priority')

        if name and expected_result:
            new_template = Template(
                name=name,
                prerequisites=prerequisites,
                steps=steps,
                data=data,
                expected_result=expected_result,
                priority=priority
            )
            db.session.add(new_template)
            db.session.commit()
            return redirect(url_for('view_templates'))
    return render_template('add_template.html')

# Маршрут для редагування шаблона
@app.route('/edit_template/<int:template_id>', methods=['GET', 'POST'])
def edit_template(template_id):
    template = Template.query.get_or_404(template_id)
    if request.method == 'POST':
        template.name = request.form.get('name')
        template.prerequisites = request.form.get('prerequisites')
        template.steps = request.form.get('steps')
        template.data = request.form.get('data')
        template.expected_result = request.form.get('expected_result')
        template.priority = request.form.get('priority')
        db.session.commit()
        return redirect(url_for('view_templates'))
    return render_template('edit_template.html', template=template)

# Маршрут для видалення шаблона
@app.route('/delete_template/<int:template_id>', methods=['POST'])
```

```
def delete_template(template_id):
    template = Template.query.get_or_404(template_id)
    db.session.delete(template)
    db.session.commit()
    return redirect(url_for('view_templates'))

# Маршрути для головної та перегляду всіх елементів
@app.route('/')
def main():
    return render_template('main.html')

@app.route('/view_sets')
def view_sets():
    sets = TestSet.query.all()
    # Отримання всіх наборів із тест-кейсами
    sets_with_cases = [
        {
            "set": test_set,
            "cases": TestCase.query.filter_by(set_id=test_set.id).all()
        }
        for test_set in sets
    ]
    return render_template('view_sets.html', sets_with_cases=sets_with_cases)


@app.route('/view_test_cases')
def view_test_cases():
    # Отримання всіх тест-кейсів і оновлення їх статусу
    test_cases = TestCase.query.all()
    for test_case in test_cases:
        update_test_case_status(test_case)

    # Отримання тільки тих наборів, які мають тест-кейси
    sets_with_cases = [
        {"set": test_set, "test_cases": TestCase.query.filter_by(set_id=test_set.id).all()}
        for test_set in TestSet.query.all()
        if TestCase.query.filter_by(set_id=test_set.id).count() > 0
    ]

    return render_template('view_test_cases.html', sets_with_cases=sets_with_cases)
```

```
@app.route('/view_templates')
def view_templates():
    templates = Template.query.all()
    return render_template('view_templates.html', templates=templates)

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

add_set.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Додати Набір</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/addSet.css') }}">
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
        <li class="breadcrumb-item"><a href="{{ url_for('view_sets') }}">Усі
Набори</a></li>
        <li class="breadcrumb-item active" aria-current="page">Додати Набір</li>
    </ol>
</nav>
    <header>
        <h1>Додати Набір</h1>
    </header>
    <section>
        <form method="POST" action="{{ url_for('add_set') }}">
            <label for="name">Назва набору:</label>
            <input type="text" id="name" name="name" required>
            <button type="submit">Додати набір</button>
            <button type="button" onclick="location.href='{{ url_for('view_sets')
}}'">Скасувати</button>
        </form>
    </section>
</body>
</html>
```

add_template.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <title>Додати Шаблон</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='css/addTemplate.css')
}}">
   <script src="{{ url_for('static', filename='js/global.js') }}"></script>
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
   <ol class="breadcrumb">
      <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
      <li class="breadcrumb-item"><a href="{{ url_for('view_templates') }}">Усі
Шаблони</a></li>
      <li class="breadcrumb-item active" aria-current="page">Додати Шаблон</li>
   </ol>
</nav>
   <header>
      <h1>Додати Шаблон</h1>
   </header>
   <section>
      <form method="POST" action="{{ url_for('add_template') }}">
         <label for="name">Назва шаблону:</label>
         <input type="text" id="name" name="name" required>

         <label for="prerequisites">Передумови:</label>
         <textarea id="prerequisites" name="prerequisites"></textarea>

         <label for="steps">Кроки відтворення:</label>
         <textarea id="steps" name="steps"></textarea>

         <label for="data">Тестові дані:</label>
         <textarea id="data" name="data"></textarea>

         <label for="expected_result">Очікуваний результат:</label>
         <textarea id="expected_result" name="expected_result" required></textarea>

         <label for="priority">Пріоритет:</label>
```

```
        <div class="priority-container">
          <select id="priority" name="priority">
            <option value="" disabled selected hidden>Оберіть пріоритет</option>
            <option value="Середній">Середній</option>
            <option value="Високий">Високий</option>
            <option value="Низький">Низький</option>
          </select>
        </div>

        <button type="submit">Додати шаблон</button>
        <button type="button" onclick="location.href='{{ url_for('view_templates')
}}'">Скасувати</button>
      </form>
  </section>
</body>
</html>
```

add_test_case.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Додати Тест-кейс</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/addTestCase.css')
}}">
  <script>
    const templatesData = {{ templates | tojson | safe }};
  </script>
  <script src="{{ url_for('static', filename='js/global.js') }}"></script>
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
    <li class="breadcrumb-item"><a href="{{ url_for('view_test_cases') }}">Усі
Тест-кейси</a></li>
    <li class="breadcrumb-item active" aria-current="page">Додати Тест-
кейс</li>
  </ol>
</nav>
```

```
    <header>
        <h1>Додати Тест-кейс</h1>
    </header>
    <section>
        <form method="POST" action="{{ url_for('add_test_case') }}">
            <label for="name">Назва тест-кейсу:</label>
            <input type="text" id="name" name="name" required>

            <label for="prerequisites">Передумови:</label>
            <textarea id="prerequisites" name="prerequisites"></textarea>

            <label for="steps">Кроки відтворення:</label>
            <textarea id="steps" name="steps"></textarea>

            <label for="data">Тестові дані:</label>
            <textarea id="data" name="data"></textarea>

            <label for="expected_result">Очікуваний результат:</label>
            <textarea id="expected_result" name="expected_result" required></textarea>

            <label for="priority">Пріоритет:</label>
            <div class="priority-container">
                <select id="priority" name="priority" required>
                    <option value="" disabled selected hidden>Оберіть пріоритет</option>
                    <option value="Середній">Середній</option>
                    <option value="Високий">Високий</option>
                    <option value="Низький">Низький</option>
                </select>
                <button type="button" onclick="openModal()">Визначити
пріоритет</button>
            </div>

            <div id="priorityModal" class="modal">
                <div class="modal-content-priority">
                    <span class="close" onclick="closeModal()">&times;</span>
                    <h2>Визначення пріоритету</h2>
                    <p id="question-text"></p>
                    <div id="answers"></div>
                </div>
            </div>
```

```
        <div>
            <label for="jiraLink">Ідентифікатори завдань Jira (через кому):</label>
            <input type="text" id="jiraLink" name="jiraLink"
                value=""
                placeholder="PROJ-123, PROJ-456">
            <button type="button" onclick="fetchJiraIssues()">Отримати статус
багів</button>

            <div id="jiraResults" style="display: none;">
                <button type="button" class="close-button"
onclick="closeJiraResults()">×</button>
                <div id="jiraContent"></div>
            </div>
        </div>

        <label for="set_id">Набір:</label>
        <select id="set_id" name="set_id">
        <option value="" disabled selected hidden>Оберіть набір</option>
            {% for set in sets %}
                <option value="{{ set.id }}">{{ set.name }}</option>
            {% endfor %}
        </select>

        <button type="submit">Додати тест-кейс</button>
        <button type="button" onclick="location.href='{{ url_for('view_test_cases')
}}'">Скасувати</button>
        <button type="button" onclick="openTemplateModal()">Використати
шаблон</button>
    </form>

    <div id="templateModal" class="modal">
        <div class="modal-content-template">
            <span class="close" onclick="closeTemplateModal()">&times;</span>
            <h2>Виберіть шаблон</h2>

            <input type="text" id="templateSearch" placeholder="Пошук за
назвою..." oninput="filterTemplates()">

            <ul id="templateList">
                {% for template in templates %}
                    <li class="template-item" data-name="{{ template.name }}">
```

```
                    {{ template.name }}
                    <button type="button-modal" onclick="useTemplateById({{
template.id }})">Вибрати</button>
                  </li>
                {% endfor %}
              </ul>
            </div>
        </div>
    </section>
</body>
</html>
```

edit_set.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Редагувати Набір</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/editSet.css') }}">
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
        <li class="breadcrumb-item"><a href="{{ url_for('view_sets') }}">Усі
Набори</a></li>
        <li class="breadcrumb-item active" aria-current="page">Редагувати Набір</li>
    </ol>
</nav>
    <header>
        <h1>Редагувати Набір</h1>
    </header>
    <section>
        <form method="POST" action="{{ url_for('edit_set', set_id=test_set.id) }}">
            <label for="name">Назва набору:</label>
            <input type="text" id="name" name="name" value="{{ test_set.name }}"
required>
            <button type="submit">Зберегти набір</button>
            <button type="button" onclick="location.href='{{ url_for('view_sets')
}}'">Скасувати</button>
```

```
        </form>
    </section>
</body>
</html>
```

edit_template.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Редагувати Шаблон</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/editTemplate.css')
}}">
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
        <li class="breadcrumb-item"><a href="{{ url_for('view_templates') }}">Усі
Шаблони</a></li>
        <li class="breadcrumb-item active" aria-current="page">Редагувати
Шаблон</li>
    </ol>
</nav>
    <header>
        <h1>Редагувати Шаблон</h1>
    </header>
    <section>
        <form method="POST" action="{{ url_for('edit_template',
template_id=template.id) }}">
            <label for="name">Назва шаблону:</label>
            <input type="text" id="name" name="name" value="{{ template.name }}"
required>

            <label for="prerequisites">Передумови:</label>
            <textarea id="prerequisites" name="prerequisites">{{ template.prerequisites
}}</textarea>

            <label for="steps">Кроки відтворення:</label>
            <textarea id="steps" name="steps">{{ template.steps }}</textarea>
```

```
        <label for="data">Тестові дані:</label>
        <textarea id="data" name="data">{{ template.data }}</textarea>

        <label for="expected_result">Очікуваний результат:</label>
        <textarea id="expected_result" name="expected_result" required>{{
template.expected_result }}</textarea>

        <label for="priority">Пріоритет:</label>
        <select id="priority" name="priority">
          <option value="Середній" {% if template.priority == "Середній"
%}selected{% endif %}>Середній</option>
          <option value="Високий" {% if template.priority == "Високий"
%}selected{% endif %}>Високий</option>
          <option value="Низький" {% if template.priority == "Низький"
%}selected{% endif %}>Низький</option>
        </select>

        <button type="submit">Зберегти шаблон</button>
        <button type="button" onclick="location.href='{{ url_for('view_templates')
}}'">Скасувати</button>
      </form>
  </section>
</body>
</html>
```

edit_test_case.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Редагувати Тест-кейс</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/editTestCase.css')
}}">
  <script src="{{ url_for('static', filename='js/global.js') }}"></script>
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
```

```
      <li class="breadcrumb-item"><a href="{{ url_for('view_test_cases') }}">Усі
Тест-кейси</a></li>
      <li class="breadcrumb-item active" aria-current="page">Редагувати Тест-
кейс</li>
   </ol>
</nav>
   <header>
      <h1>Редагувати Тест-кейс</h1>
   </header>
   <section>
      <form method="POST" action="{{ url_for('edit_test_case',
test_case_id=test_case.id) }}">

         <label for="id">ID тест-кейса:</label>
         <input type="text" id="id" name="id" value="{{ test_case.id }}" readonly>

         <label for="name">Назва тест-кейсу:</label>
         <input type="text" id="name" name="name" value="{{ test_case.name }}"
required>

         <label for="prerequisites">Передумови:</label>
         <textarea id="prerequisites" name="prerequisites">{{ test_case.prerequisites
}}</textarea>

         <label for="steps">Кроки відтворення:</label>
         <textarea id="steps" name="steps">{{ test_case.steps }}</textarea>

         <label for="data">Тестові дані:</label>
         <textarea id="data" name="data">{{ test_case.data }}</textarea>

         <label for="expected_result">Очікуваний результат:</label>
         <textarea id="expected_result" name="expected_result" required>{{
test_case.expected_result }}</textarea>

         <label for="priority">Пріоритет:</label>
         <div class="priority-container">
            <select id="priority" name="priority">
               <option value="Середній" {% if test_case.priority == "Середній"
%}selected{% endif %}>Середній</option>
               <option value="Високий" {% if test_case.priority == "Високий"
%}selected{% endif %}>Високий</option>
```

```
            <option value="Низький" {% if test_case.priority == "Низький"
%}selected{% endif %}>Низький</option>
        </select>
        <button type="button" onclick="openModal()">Визначити
пріоритет</button>
    </div>

    <div id="priorityModal" class="modal">
        <div class="modal-content-priority">
            <span class="close" onclick="closeModal()">&times;</span>
            <h2>Визначення пріоритету</h2>
            <p id="question-text"></p>
            <div id="answers"></div>
        </div>
    </div>

    <label for="jiraLink">Ідентифікатори завдань Jira (через кому):</label>
    <input type="text" id="jiraLink" name="jiraLink"
        value="{% for issue in test_case.jira_issues %}{{ issue.key }}{% if not
loop.last %}, {% endif %}{% endfor %}"
        placeholder="PROJ-123, PROJ-456">
    <button type="button" onclick="fetchJiraIssues()">Отримати статус
багів</button>

    <div id="jiraResults" style="display: none;">
        <button type="button" class="close-button"
onclick="closeJiraResults()">×</button>
        <div id="jiraContent"></div> <!-- Весь контент з Jira-інформацією -->
    </div>

    <label for="set_id">Набір:</label>
    <select id="set_id" name="set_id">
        {% for set in sets %}
            <option value="{{ set.id }}" {% if test_case.set_id == set.id
%}selected{% endif %}>{{ set.name }}</option>
        {% endfor %}
    </select>

    <button type="submit">Зберегти тест-кейс</button>
    <button type="button" onclick="location.href='{{ url_for('view_test_cases')
}}'">Скасувати</button>
```

```
        </form>
    </section>
</body>
</html>
```

main.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Інформаційна технологія для організації управління тест-кейсами</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
</head>
<body>
<!-- Breadcrumbs для головної сторінки -->
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item active" aria-current="page">Головна</li>
    </ol>
</nav>
    <header>
        <h1>Інформаційна технологія для організації управління тест-кейсами</h1>
    </header>
    <div class="container">
        <section class="block block-sets">
            <h2>Набори</h2>
            <button class="main-page-button" onclick="location.href='/add_set'">Додати набір</button>
            <button class="main-page-button" onclick="location.href='/view_sets'">Переглянути всі набори</button>
        </section>
        <section class="block block-test-cases">
            <h2>Тест-кейси</h2>
            <button class="main-page-button" onclick="location.href='/add_test_case'">Додати тест-кейс</button>
            <button class="main-page-button" onclick="location.href='/view_test_cases'">Переглянути всі тест-кейси</button>
```

```
        </section>
        <section class="block block-templates">
            <h2>Шаблони</h2>
            <button class="main-page-button"
onclick="location.href='/add_template'">Додати шаблон</button>
            <button class="main-page-button"
onclick="location.href='/view_templates'">Переглянути всі шаблони</button>
        </section>
    </div>
</body>
</html>
```

view_sets.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Усі Набори</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/viewSets.css') }}">
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
        <li class="breadcrumb-item active" aria-current="page">Усі Набори</li>
    </ol>
</nav>
    <header>
        <h1>Усі Набори</h1>
    </header>
    <section>
        <ul>
            {% for item in sets_with_cases %}
                <li class="set-item">
                    <div class="set-header">
                        <span>{{ item.set.name }}</span>
                        <div class="action-buttons">
                            <button type="button" onclick="location.href='/edit_set/{{
item.set.id }}'">Редагувати</button>
                            <form action="{{ url_for('delete_set', set_id=item.set.id) }}"
```

```
method="post" style="display:inline;">
                        <button type="submit" onclick="return confirm('Ви впевнені, що хочете видалити цей набір?')">Видалити</button>
                    </form>
                </div>
            </div>

            <ul class="case-list">
                {% for case in item.cases %}
                    <li class="case-item">
                        ID: {{ case.id }} -
                        <a href="{{ url_for('edit_test_case', test_case_id=case.id) }}">{{ case.name }}</a>
                    </li>
                {% endfor %}
            </ul>
        </li>
    {% endfor %}
</ul>
<button onclick="location.href='{{ url_for('add_set') }}'" class="main-page-button">Додати набір</button>
<button onclick="location.href='{{ url_for('main') }}'" class="main-page-button">На головну</button>
</section>
</body>
</html>
```

view_templates.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Усі Шаблони</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/viewTemplates.css') }}">
</head>
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
    <ol class="breadcrumb">
        <li class="breadcrumb-item"><a href="{{ url_for('main') }}">Головна</a></li>
```

```
      <li class="breadcrumb-item active" aria-current="page">Усі Шаблони</li>
   </ol>
</nav>
   <header>
      <h1>Усі Шаблони</h1>
   </header>
   <section>
      <ul>
         {% for template in templates %}
            <li>
               {{ template.name }}
               <div class="action-buttons">
                  <button type="button" onclick="location.href='/edit_template/{{
template.id }}'">Редагувати</button>
                  <form action="{{ url_for('delete_template', template_id=template.id)
}}" method="post" style="display:inline;">
                     <button type="submit" onclick="return confirm('Ви впевнені, що
хочете видалити цей шаблон?')">Видалити</button>
                  </form>
               </div>
            </li>
         {% endfor %}
      </ul>
      <button onclick="location.href='{{ url_for('add_template') }}'" class="main-
page-button">Додати шаблон</button>
      <button onclick="location.href='{{ url_for('main') }}'" class="main-page-
button">На головну</button>
   </section>
</body>
</html>
```

view_test_cases.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <title>Усі Тест-кейси</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='css/viewTestCases.css')
}}">
   <script src="{{ url_for('static', filename='js/global.js') }}"></script>
</head>
```

```
<body>
<nav aria-label="breadcrumb" class="breadcrumb-container">
   <ol class="breadcrumb">
      <li class="breadcrumb-item"><a href="{{ url_for('main')
}}">Головна</a></li>
      <li class="breadcrumb-item active" aria-current="page">Усі Тест-кейси</li>
   </ol>
</nav>
<header>
   <h1>Усі Тест-кейси</h1>
</header>
<section>
   {% for test_set in sets_with_cases %}
      <div class="test-set-section">
         <div class="set-header">
            <button class="toggle-set" onclick="toggleSet('{{ test_set.set.id
}}')">▸ </button>
            <h2>{{ test_set.set.name }}</h2>
         </div>
         <ul id="set-{{ test_set.set.id }}" class="test-case-list" style="display: none;">
            {% for test_case in test_set.test_cases %}
               <div class="ramka">
                  <li class="test-case-item">
                     <div class="test-case-header">
                        <!-- Кнопка розгортання для тест-кейсу -->
                        <button id="toggle-button-{{ test_case.id }}" class="toggle-
details" onclick="toggleDetails({{ test_case.id }})">▸ </button>


                        <span class="test-case-name">{{ test_case.name }}</span>
                     </div>

                     <div class="action-buttons">
                        <button type="button" onclick="location.href='/edit_test_case/{{
test_case.id }}'">Редагувати</button>
                        <form action="{{ url_for('delete_test_case',
test_case_id=test_case.id) }}" method="post" style="display:inline;">
                           <button type="submit" onclick="return confirm('Ви впевнені,
що хочете видалити цей тест-кейс?')">Видалити</button>
                        </form>
                     </div>
```

```
                </li>

                <div class="test-case-status-container">
                    <label for="status-{{ test_case.id }}">Статус:</label>
                    <select id="status-{{ test_case.id }}" class="test-case-status"
onchange="updateStatus({{ test_case.id }}, this.value)">
                        <option value="Not Executed" {% if test_case.status == "Not
Executed" %}selected{% endif %}>Not Executed</option>
                        <option value="Passed" {% if test_case.status == "Passed"
%}selected{% endif %}>Passed</option>
                        <option value="Failed" {% if test_case.status == "Failed"
%}selected{% endif %}>Failed</option>
                        <option value="Blocked" {% if test_case.status == "Blocked"
%}selected{% endif %}>Blocked</option>
                        <option value="Skipped" {% if test_case.status == "Skipped"
%}selected{% endif %}>Skipped</option>
                    </select>
                </div>

                {% if test_case.jira_issues %}
                <div class="jira-issues">
                    <span class="jira-label">Завдання Jira:</span>
                    <ul class="jira-list">
                        {% for issue in test_case.jira_issues %}
                            <li class="tooltip">
                                <a href="{{ issue.link }}" target="_blank">{{ issue.key
}}</a> - {{ issue.summary }}
                                <span class="tooltiptext">
                                    <span><strong>Статус:</strong> {{ issue.status
}}</span><br>
                                    <span><strong>Тип:</strong> {{ issue.type
}}</span><br>
                                    <span><strong>Репортер:</strong> {{ issue.reporter
}}</span><br>
                                    <span><strong>Пріоритет:</strong> {{ issue.priority
}}</span><br>
                                    <span><strong>Дата створення:</strong> {{
issue.created }}</span>
                                </span>
                            </li>
                        {% endfor %}
```

```
                    </ul>
                </div>
                {% endif %}

                <div id="details-{{ test_case.id }}" class="test-case-details"
style="display: none;">
                    <p><strong>ID:</strong> {{ test_case.id }}</p>
                    <p><strong>Передумови:</strong> <span class="formatted-
text">{{ test_case.prerequisites }}</span></p>
                    <p><strong>Кроки відтворення:</strong>
                        <pre class="steps-text">{{ test_case.steps }}</pre>
                    </p>
                    <p><strong>Тестові дані:</strong> <span class="formatted-
text">{{ test_case.data }}</span></p>
                    <p><strong>Очікуваний результат:</strong> <span
class="formatted-text">{{ test_case.expected_result }}</span></p>
                </div>

            </div>
        {% endfor %}
    </ul>
</div>
{% endfor %}

<button onclick="location.href='{{ url_for('add_test_case') }}'" class="main-page-
button">Додати тест-кейс</button>
<button onclick="location.href='{{ url_for('main') }}'" class="main-page-
button">На головну</button>
</section>

<script>
function toggleSet(setId) {
    const setList = document.getElementById(`set-${setId}`);
    const button = document.querySelector(`.toggle-
set[onclick="toggleSet('${setId}')"]`);
    if (setList.style.display === "none") {
        setList.style.display = "block";
        button.textContent = " ▾ ";
    } else {
        setList.style.display = "none";
        button.textContent = " ▸ ";
```

```
    }
}

function toggleDetails(testCaseId) {
    const details = document.getElementById(`details-${testCaseId}`);
    details.style.display = details.style.display === "none" ? "block" : "none";
}

function toggleDetails(testCaseId) {
    const details = document.getElementById(`details-${testCaseId}`);
    const toggleButton = document.getElementById(`toggle-button-${testCaseId}`);

    // Перемикання видимості блоку деталей
    if (details.style.display === "none") {
        details.style.display = "block";
        toggleButton.textContent = " ▿ ";  // Змінюємо на стрілочку вниз
    } else {
        details.style.display = "none";
        toggleButton.textContent = " ▹ ";  // Змінюємо на стрілочку вправо
    }
}


</script>

</body>
</html>
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДЛЯ ОРГАНІЗАЦІЇ УПРАВЛІННЯ ТЕСТ-КЕЙСАМИ

Виконав: студент 2-го курсу,
групи 1КН-23м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Єпіфанова А.О.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
_____ Яровий А.А.
(прізвище та ініціали)
«05» 12 2024 р.

Вінниця ВНТУ - 2024 рік

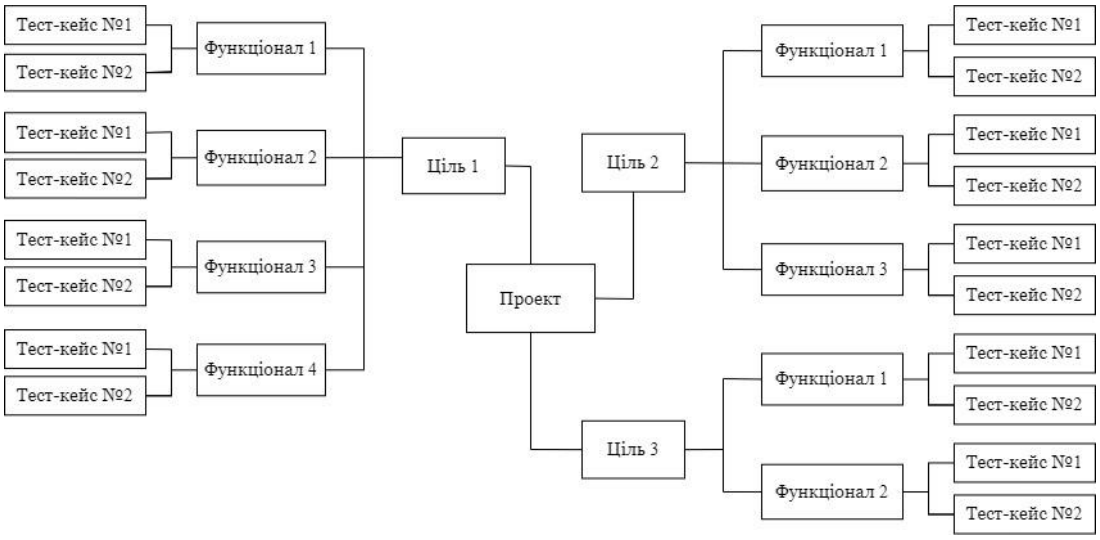


Рисунок В.1 – Приклад організації тест-кейсів

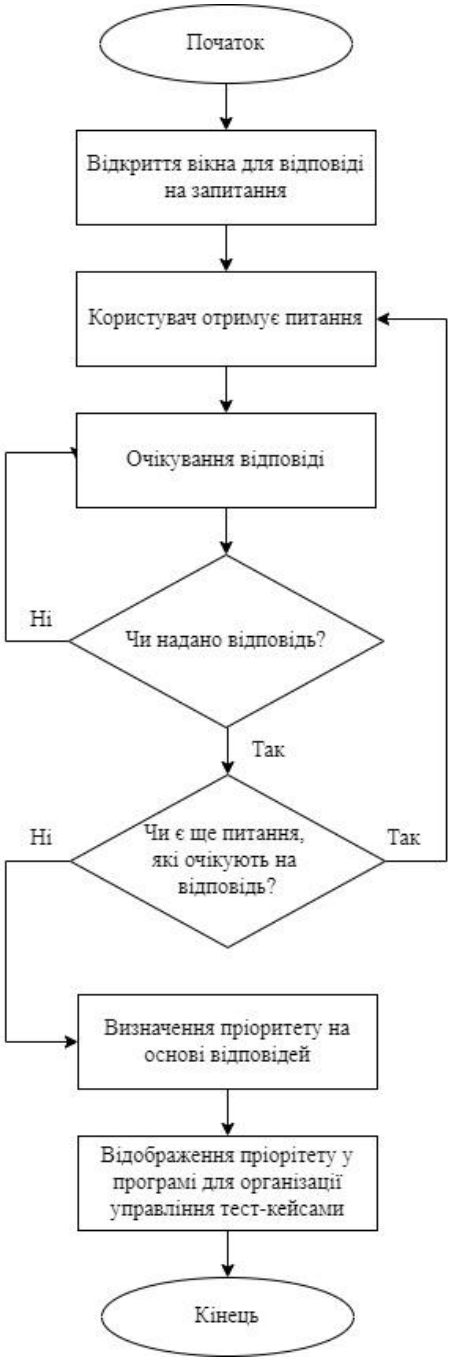


Рисунок В.2 – Узагальнений алгоритм роботи експертної системи для пріоритезації, інтегрована у систему для організації управління тест-кейсами

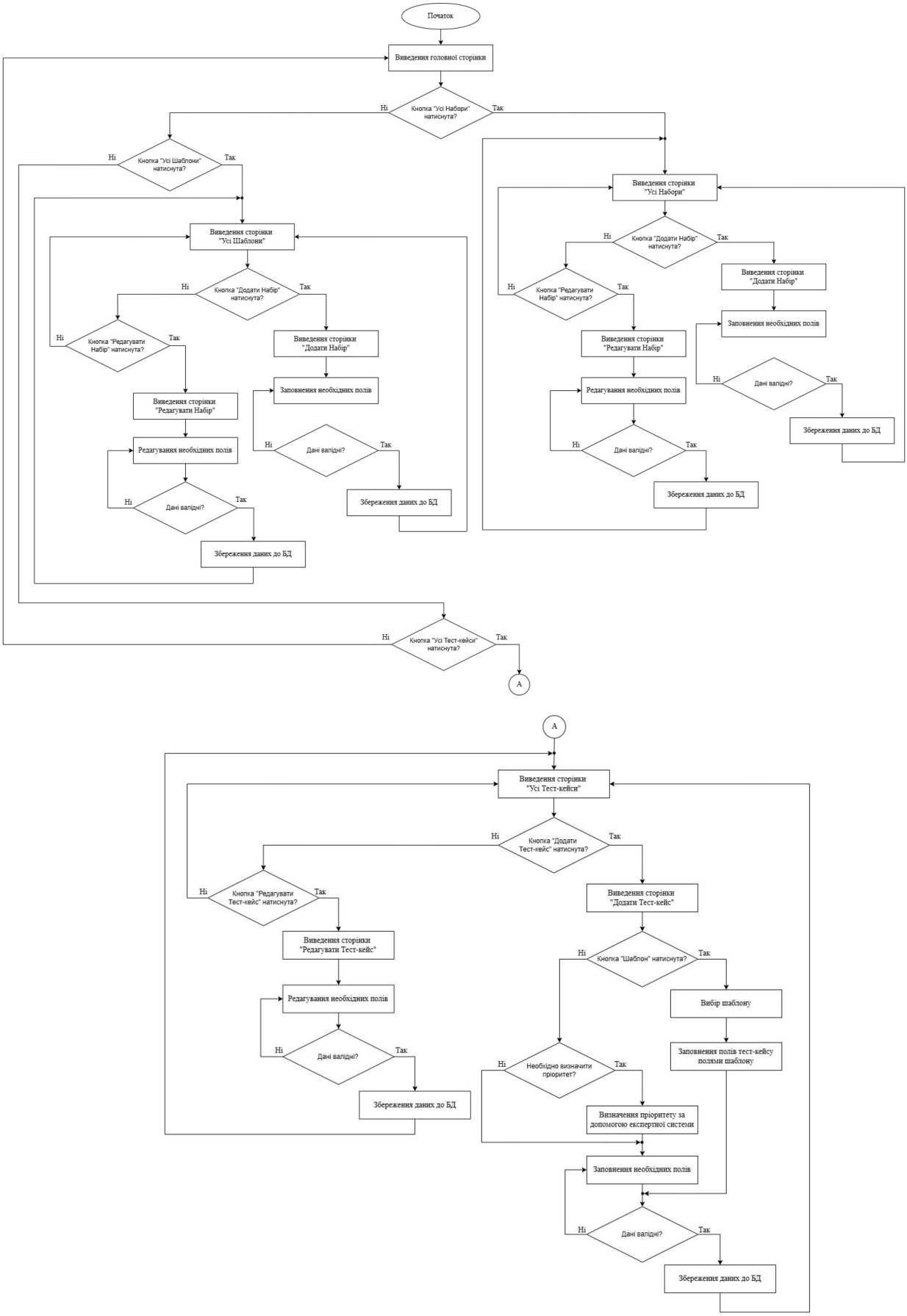
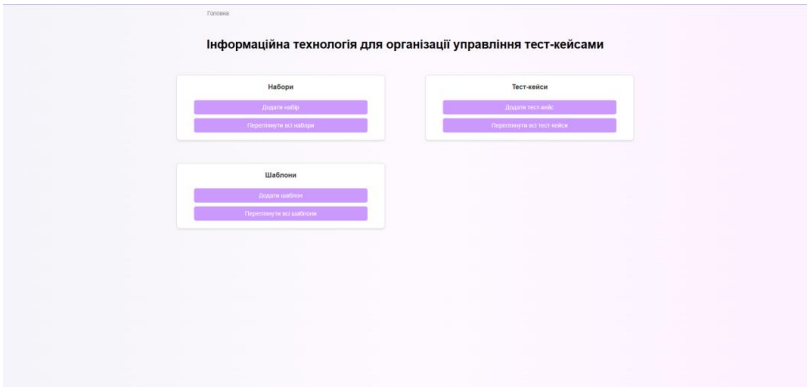
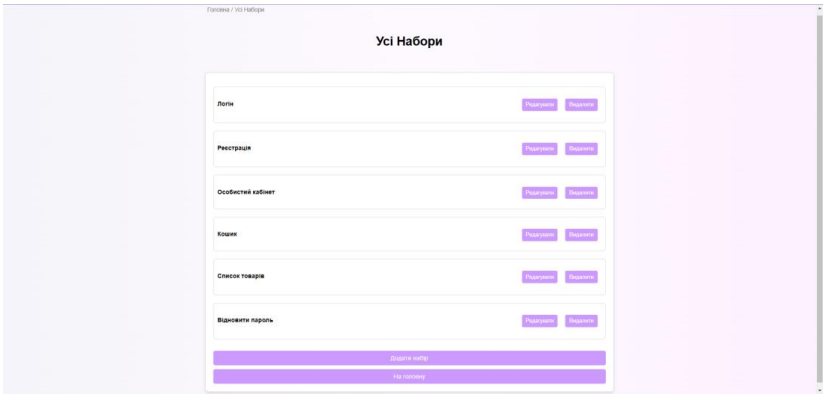
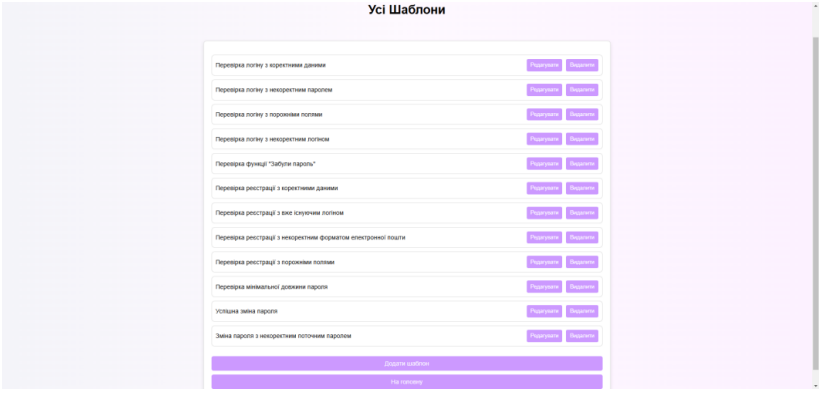


Рисунок В.3 – Узагальнений алгоритм роботи програмного продукту







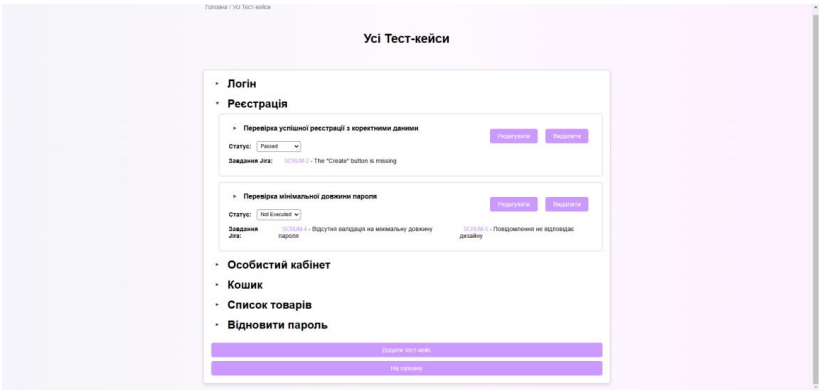


Рисунок В.4 – Робочі вікна програми

Додаток Г (довідниковий) Інструкція користувача

Для того, щоб запустити програму, потрібно відкрити консоль, перейти у папку розпашування програми, використовуючи програму «cd» та написати команду «flask run». Після цього потрібно відкрити у браузері посилання http://127.0.0.1:5000, щоб перейти на головну сторінку сайту.

Для того, щоб додати набір тест-кейсів, потрібно натиснути на кнопку «Додати набір» на головній сторінці. Після цього необхідно заповнити поле «Назва набору» та натиснути кнопку «Додати набір». Щоб повернутися до головної сторінки, не додаючи набір, потрібно натиснути на кнопку «На головну».

Щоб переглянути усі існуючі набори тест-кейсів, потрібно натиснути на кнопку «Переглянути всі набори». Щоб видалити набір, необхідно натиснути на кнопку «Видалити» навпроти відповідного набору. Щоб переглянути деталі тест-кейсу, що входить до набору, потрібно натиснути на відповідний тест-кейс. Для повернення на головну сторінку, потрібно натиснути на кнопку «На головну».

Для того, щоб додати новий шаблон, потрібно натиснути на кнопку «Додати шаблон» на головній сторінці. Необхідно заповнити усі поля та натиснути на кнопку «Додати шаблон». Щоб повернутися до головної сторінки, не додаючи набір, потрібно натиснути на кнопку «На головну».

Щоб переглянути усі існуючі в системі шаблони, потрібно натиснути на кнопку «Переглянути всі шаблони». Для того, щоб видалити шаблон, необхідно натиснути на кнопку «Видалити» навпроти потрібного шаблону. Для того, щоб переглянути деталі шаблону, необхідно натиснути на кнопку «Деталі» навпроти потрібного шаблону. Для того, щоб редагувати шаблон, необхідно натиснути на кнопку «Редагувати» навпроти потрібного шаблону. Щоб повернутися до головної сторінки, потрібно натиснути на кнопку «На головну». При натисненні на кнопку «Деталі», для редагування шаблону, потрібно натиснути на кнопку «Редагувати шаблон», щоб повернутися до сторінки зі всіма шаблонами, потрібно натиснути на кнопку «До всіх шаблонів». При натисненні на кнопку

«Редагувати», можна змінити текст будь-якого поля шаблону. Для збереження змін потрібно натиснути на кнопку «Оновити шаблон». Для повернення до сторінки зі всіма шаблонами без застосування змін, потрібно натиснути на кнопку «Назад до всіх шаблонів».

Для того, щоб додати новий тест-кейс, необхідно натиснути на кнопку «Додати тест-кейс» на головній сторінці. Потрібно заповнити всі поля, обрати набір з випадного списку та натиснути на кнопку «Додати тест-кейс». Для використання шаблону, необхідно натиснути на кнопку «Шаблони». У вікні, що відкрилося, необхідно натиснути на кнопку «Використати шаблон». Для того, щоб не використовувати шаблон, потрібно натиснути на кнопку «Закрити» або натиснути на будь-яку частину екрану за областю вікна. Після використання шаблону поля можна редагувати та додати тест-кейс за допомогою кнопки «Додати тест-кейс». Для того, щоб визначити пріоритет за допомогою експертної системи, необхідно натиснути на кнопку «Визначити пріоритет» та дати відповідь на питання, що з’являються у модальному вікні. Після успішного визначення пріоритету він з’явиться у полі «Пріоритет». Для того, щоб переглянути інформацію про завдання у Jira та додати їх до тест-кейсу, необхідно ввести ідентифікатори у поле «Ідентифікатори завдань Jira», та натиснути на кнопку «Отримати статус багів». Для того, щоб повернутися на головну сторінку не додаючи тест-кейс, потрібно натиснути на кнопку «На головну».

Щоб переглянути усі тест-кейси, потрібно натиснути на кнопку «Переглянути всі тест-кейси» на головній сторінці. Після цього потрібно натиснути на будь-який набір, щоб побачити тест-кейси, які до нього входять. Для того, щоб повернутися на головну сторінку, потрібно натиснути на кнопку «На головну». Для того, щоб згорнути набір тест-кейсів, потрібно натиснути на нього ще раз. Для того, щоб вибрати статус тест-кейса, потрібно натиснути на випадний список «Статус» та обрати потрібний статус зі списку. Для того, щоб видалити тест-кейс, необхідно натиснути на кнопку «Видалити». Щоб редагувати тест-кейс, потрібно натиснути на кнопку «Редагувати». Для перегляду деталей тест-кейсу, потрібно натиснути на кнопку «Деталі». При

перегляді деталей тест-кейсу, для їх редагування, потрібно натиснути на кнопку «Редагувати». Для того, щоб повернутися до сторінки зі списком усіх тест-кейсів, необхідно натиснути на кнопку «До всіх тест-кейсів». При редагуванні тест-кейсу, для збереження змін після редагування необхідних полів, потрібно натиснути на кнопку «Оновити тест-кейс». Для повернення на сторінку зі всіма тест-кейсами без збереження змін, необхідно натиснути на кнопку «Назад до всіх тест-кейсів».