Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп’ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Комп'ютерна система підтримки прийняття рішень при контрольних випробуваннях технічних систем»

Виконав: студента групи К23-1М
Спеціальність 122 «Комп’ютерні науки»
Коцуба С.М
(прізвище та ініціали)

Керівник: д.т.н., професор кафедри Яковенко В.О.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент: Університет митної справи та фінансів
(місто роботи)
доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доц. Клим В. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Коцуба С.М. Комп’ютерна система підтримки прийняття рішень при контрольних випробуваннях технічних систем.

Дипломна робота (проект) на здобуття освтнього ступеня магістр за спеціальністю 122 «Комп’ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Об’єктом дослідження є процес прийняття рішень при проведенні контрольних випробувань технічних систем.

Предмет дослідження це методи, алгоритми та засоби розробки комп’ютерної системи підтриммки прийняття рішень.

Основною метою даної роботи є створення сучасної та ефективної системи підтримки прийняття рішень, призначеної для забезпечення якісного аналізу та оцінки результатів контрольних випробувань технічних систем. Для досягнення поставленої мети було проведено комплексне дослідження предметної області, що охоплює аналіз існуючих рішень, які представлені на ринку, їх сильні та слабкі сторони, а також актуальні тенденції в галузі.

На основі отриманих даних здійснено ретельний вибір архітектури системи, що відповідає сучасним вимогам продуктивності, надійності та масштабованості. Проведено проектування бази даних, яка забезпечує ефективне зберігання та доступ до інформації, а також визначено внутрішню структуру системи, яка включає ключові модулі та їх взаємодію.

У результаті роботи створено функціональну систему, яка забезпечує ефективне зберігання та обробку даних, зручний інтерфейс для користувачів, автоматизовані алгоритми аналізу та високий рівень безпеки. Тестування підтвердило коректність роботи системи та її відповідність вимогам.

Ключові слова: Python, Flask, SQLAlchemy, система підтримки прийняття рішень.

ABSTRACT

Kotsuba S. M. Computer decision support system for control tests of technical systems.

Diploma thesis (project) for the degree of Master's Degree in specialty 122 «Computer Science». – University of Customs and Finance, Dnipro, 2024.

The object of research is decision-making process during the control testing of technical systems.

The subject of the research is methods. Algoritms, and tools for developing a decision support computer system.

The main goal of this work is to create a modern and efficient decision support system designed to ensure high-quality analysis and evaluation of the results of control testing of technical systems. To achieve this goal, a comprehensive study of the subject area was conducted, which includes an analysis of existing solutions available on the market, their strengths and weaknesses, as well as current trends in the field.

Based on the obtained data, a careful selection of the system architecture was made, ensuring compliance with modern performance, reliability, and scalability requirements. The design of the database was carried out, ensuring effective storage and access to information, and the internal structure of the system was defined, including the key modules and their interactions.As result of the work, a functional system was created that provides efficient data storage and processing, a user-friendly interface, automated analysis algorithms, and a high level of security. Testing confirmed the correctness of the system's operation and its compliance with the requiments.

Keywords: Python, Flask, SQLAlchemy, decision support system.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8

1.1 Системи підтримки прийняття рішень 8

1.2 Огляд існуючих рішень 11

1.3 Постановка задачі 14

1.4 Вибір технічної системи та критеріїв для тестування 15

1.5 Висновки до першого розділу 16

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ 17

2.1 Вибір архітектури 17

2.2 Вибір методу контрольних випробувань 19

2.2.1 Аналіз обраного методу 19

2.2.2 Аналіз інших методів 20

2.3 Проектування бази даних 21

2.4 Проектування внутрішньої будови 25

2.6 Вибір засобів розробки 40

2.7 Висновки до другого розділу 43

РОЗДІЛ 3. РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ 44

3.1 Розробка бази даних 44

3.2 Розробка основного функціоналу 53

3.3 Оцінка параметрів для обранної технічної системи 64

3.4 Тестування комп'ютерної системи 65

3.4 Висновки до третього розділу 69

ВИСНОВКИ 70

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ 75

ВСТУП

У сучасних умовах розвитку технологій та зростання складності технічних систем питання ефективного прийняття рішень набуває особливої актуальності. Збільшення обсягів даних, що генеруються під час контрольних випробувань технічних систем, потребує використання інноваційних підходів для їх аналізу та інтерпретації. Система підтримки прийняття рішень (СППР) стає невід'ємною частиною процесу управління технічними системами, забезпечуючи об'єктивність, швидкість та точність прийняття рішень на основі аналізу великих обсягів інформації.

Актуальність теми розробки комп’ютерної системи підтримки прийняття рішень при контрольних випробуваннях технічних систем полягає в контексті швидкого розвитку технічних систем і зростаючої складності їх компонентів, ефективність та якість контрольних випробувань набувають критичного значення. Прийняття рішень на основі результатів таких випробувань впливає на подальше впровадження, експлуатацію та модернізацію технічних систем. Традиційні підходи до обробки даних і аналізу результатів часто є недостатньо оперативними, схильними до людських помилок та не враховують комплексність сучасних технічних завдань.

Розробка комп’ютеної системи підтримки прийняття рішень дозволяє автоматизувати процеси аналізу результатів випробувань, знизити влпви суб’єктивного фактору, підвищити точність і швидкість прийняття рішень. Це особливо важливо для галузей, де критичним є швидке реагування та висока точність, таких як авіація, енергетика, автомобілебудування та інші сектори.

Новизна роботи полягає в інтеграції сучасних технологій розробки програмного забезпечення з методами аналізу даних для створення СППР, яка відповідає специфічним вимогам контрольних випробувань технічних систем.

Метою дослідження є розробка та впровадження ефективної системи підтримки прийняття рішень для контрольних випробувань технічних систем, яка забезпечить автоматизацію процесу оцінки, підвищення точності аналізу та оптимізацію прийняття рішень.

Для досягнення поставленої мети були використані такі методи дослідження, як аналіз та синтез інформації, моделювання систем, розробка програмного забезпечення, а також тестування та валідація розробленої системи.

Для досяггення поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання:

1. Провести аналіз існуючих систем підтримки прийняття рішень для визначення їх переваг та недоліків.
2. Вибір архітектури для подальшої роботи.
3. Проектування варіантів використання, бази данних та внутрішньої будови.
4. Вибір засобів розробки.
5. Розробка бази данних та основного функціоналу.

Методи дослідження: в роботі використовувалися веб-фрейморки Flask та SQLAlchemy, мова програмування Python.

Об’єкт дослідження: Процес прийняття рішень при проведенні контрольних випробувань технічних систем.

Предмет дослідження: методи, алгоритми та засоби розробки комп’ютерної системи підтриммки прийняття рішень.

Практичне значення дослідження полягає у можливості застосування розробленої системи в різних галузях промисловості, де необхідно проводити комплексні оцінки технічних систем для прийняття обґрунтованих рішень.

Очікуваними результатами є створення функціональної та ефективної СППР, яка забезпечить автоматизований процес збору, обробки та аналізу даних, підвищить точність оцінок технічних систем та дозволить приймати швидкі та

обґрунтовані рішення. Крім того, система має потенціал для подальшого розвитку та масштабування, що дозволить адаптувати її до змінних потреб підприємств та вдосконалювати функціональні можливості відповідно до нових вимог ринку.

Структура роботи:

Розділ 1 Аналіз предметної області. Перший розділ охоплює аналіз систем підтримки прийняття рішень та огляд існуючих рішень на ринку.

Розділ 2 Проектування системи. Другий розділ присвячений проектуванню системи, включаючи вибір архітектури, створення варіантів використання та розробку бази даних.

Розділ 3 Розробка системи. Третій розділ описує процес розробки основного функціоналу, вибір засобів розробки, реалізацію маршрутів та функцій, а також проведення тестування системи.

Робота складається зі вступу, 3-х розділів, висновків, списку використанної літератури з 44 джерел, 1 додатку. Обсяг роботи 66 сторінок, 3 рисунка та 1 формула.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Системи підтримки прийняття рішень

Система підтримки прийняття рішень (СППР) — це інформаційна система, яка підтримує діяльність з прийняття бізнес- або організаційних рішень. СППР обслуговують рівень управління, операцій і планування організації (зазвичай середнього та вищого керівництва) і допомагають людям приймати рішення щодо проблем, які можуть швидко змінюватися та нелегко визначити заздалегідь, тобто неструктурованих і напівструктурованих проблем прийняття рішень. Системи підтримки прийняття рішень можуть бути або повністю комп'ютеризованими, або керованими людиною, або поєднувати обидва.

Хоча науковці сприймали DSS як інструмент для підтримки процесів прийняття рішень, користувачі DSS бачать DSS як інструмент для полегшення організаційних процесів [1]. Деякі автори розширили визначення СППР, включивши будь-яку систему, яка може підтримувати прийняття рішень, а деякі СППР включають програмний компонент для прийняття рішень; Sprague (1980) [2] визначає правильний термін DSS наступним чином:

1. DSS, як правило, націлений на менш добре структуровану, недостатньо визначену проблему, з якою зазвичай стикаються керівники вищого рівня;
2. DSS намагається поєднати використання моделей або аналітичних методів із традиційними функціями доступу та пошуку даних;
3. DSS особливо зосереджується на функціях, які полегшують їх використання людьми, які не володіють комп'ютером, в інтерактивному режимі;
4. DSS наголошує на гнучкості та здатності адаптуватися до змін у середовищі та підході користувача до прийняття рішень.

СППР включають системи, засновані на знаннях. Правильно розроблена СППР — це інтерактивна система на основі програмного забезпечення, призначена допомогти особам, які приймають рішення, зібрати корисну інформацію з комбінації необроблених даних, документів, особистих знань та/або бізнес-моделей для виявлення та вирішення проблем і прийняття рішень.

Типова інформація, яку може збирати та представляти програма підтримки прийняття рішень, включає:

– інвентаризація інформаційних активів (включаючи успадковані та реляційні джерела даних, куби, сховища даних і вітрини даних);

– порівняльні показники продажів між одним періодом і наступним;

– прогнозовані показники доходу, засновані на припущеннях щодо продажів продукції.

СППР теоретично можна створити в будь-якій області знань. Одним із прикладів є система підтримки клінічних рішень для медичної діагностики. Є чотири етапи в еволюції системи підтримки клінічних рішень (CDSS): примітивна версія є автономною та не підтримує інтеграцію; друге покоління підтримує інтеграцію з іншими медичними системами; третій базується на стандартах, а четвертий базується на моделі обслуговування [3].

DSS широко використовується в бізнесі та менеджменті. Виконавча інформаційна панель та інше програмне забезпечення для підвищення ефективності бізнесу дозволяють швидше приймати рішення, виявляти негативні тенденції та краще розподіляти бізнес-ресурси. Завдяки СППР вся інформація від будь-якої організації представлена у вигляді схем, графіків, тобто в зведеному вигляді, що допомагає керівництву приймати стратегічні рішення. Наприклад, одним із застосувань СППР є управління та розвиток складних антитерористичних систем [4]. Інші приклади включають кредитний спеціаліст банку, який перевіряє кредит заявника на позику, або інжинірингову фірму, яка

подала заявки на кілька проектів і хоче знати, чи можуть вони бути конкурентоспроможними за своїми витратами.

Сфера застосування, концепцій, принципів і методів СППР зростає в сільськогосподарському виробництві, маркетингу для сталого розвитку. Сільськогосподарські DSS почали розробляти та просувати в 1990-х. Наприклад, пакет DSSAT4 [5]. Система підтримки прийняття рішень щодо трансферу агротехнологій [6], розроблений за фінансової підтримки USAID протягом 1980-х та 1990-х років, дозволив швидко оцінити декілька систем сільськогосподарського виробництва в усьому світі для полегшення прийняття рішень на рівні ферми та політики. DSS також поширена в управлінні лісами, де довгий горизонт планування та просторовий вимір проблем планування вимагають конкретних вимог. Усі аспекти управління лісами, від транспортування колод, планування врожаю до сталого розвитку та захисту екосистем, розглядаються сучасними СППР. У цьому контексті, розгляд однієї або кількох цілей управління, пов’язаних із наданням товарів і послуг, які продаються або не продаються, і часто підпадають під обмеження ресурсів і проблеми з прийняттям рішень. Спільнота практикуючих систем підтримки прийняття рішень у лісовому господарстві надає велике сховище знань про створення та використання систем підтримки прийняття рішень у лісовому господарстві [8].

Конкретний приклад стосується канадської національної залізничної системи, яка регулярно тестує своє обладнання за допомогою системи підтримки прийняття рішень. Проблема, з якою стикається будь-яка залізниця, — це зношені або несправні рейки, що може призвести до сотень сходів з рейок на рік. Відповідно до DSS системі канадської національної залізниці вдалося зменшити кількість сходжень з рейок у той час, як інші компанії зазнали зростання.

DSS використовувався для оцінки ризику для інтерпретації даних моніторингу великих інженерних споруд, таких як дамби, вежі, собори або

кам’яні будівлі. Наприклад, Mistral — експертна система для моніторингу безпеки дамб, розроблена в 1990-х роках компанією Ismes (Італія). Він отримує дані з автоматичної системи моніторингу та проводить діагностику стану дамби. Його перша копія, встановлена в 1992 році на дамбі Рідраколі (Італія), досі працює 24/7/365 [9] Він був встановлений на кількох дамбах в Італії та за кордоном (наприклад, дамба Ітайпу в Бразилії) [10] і на пам'ятниках під назвою Калейдос [11]. Mistral є зареєстрованою торговою маркою CESI. ГІС успішно використовується з 90-х років у поєднанні з DSS для відображення на карті оцінки ризику в реальному часі на основі даних моніторингу, зібраних у районі катастрофи Валь Пола (Італія) [12].

1.2 Огляд існуючих рішень

На ринку інформаційних систем підтримки прийняття рішень існує широкий спектр програмних продуктів, призначених для різних галузей промисловості та бізнесу. Ці системи варіюються за своїми функціональними можливостями, складністю інтеграції та рівнем адаптивності до специфічних потреб підприємств. Проте, незважаючи на їхню різноманітність, багато з них мають певні обмеження, що створює потребу в розробці нових рішень, здатних ефективніше відповідати сучасним вимогам ринку [13].

Однією з провідних систем на ринку є SAP BusinessObjects. Ця система забезпечує широкий спектр інструментів для бізнес-аналітики, включаючи звітність, аналіз даних та візуалізацію інформації. SAP BusinessObjects дозволяє користувачам створювати детальні звіти та аналітичні панелі, що сприяють прийняттю обґрунтованих рішень [14]. Проте, система відома своєю складністю у налаштуванні та високими витратами на впровадження, що робить її менш доступною для середніх та малих підприємств. Крім того, інтеграція з іншими

системами може бути трудомісткою та вимагати значних ресурсів, що обмежує гнучкість використання та адаптацію до змінних потреб бізнесу [15].

Іншим популярним рішенням є IBM Cognos Analytics. Ця система також надає потужні інструменти для бізнес-аналітики, включаючи створення звітів, панелей моніторингу та прогнозування. IBM Cognos відзначається високим рівнем інтеграції з різними джерелами даних та можливістю обробки великих обсягів інформації [16]. Проте, користувачі часто відзначають складність у використанні інтерфейсу та необхідність спеціалізованого навчання для ефективної роботи з системою. Також, висока вартість ліцензій може бути бар'єром для невеликих підприємств, що прагнуть впровадити сучасні рішення для підтримки прийняття рішень [17].

Microsoft Power BI є ще одним популярним інструментом у сфері бізнес-аналітики. Ця система відома своєю інтуїтивно зрозумілою інтерфейсною частиною, що дозволяє користувачам без спеціальних технічних знань створювати візуалізації даних та аналітичні звіти [18]. Power BI інтегрується з різноманітними джерелами даних, включаючи хмарні сервіси та локальні бази даних, що забезпечує гнучкість у використанні [19]. Однак, незважаючи на свої переваги, Power BI може мати обмеження у масштабованості та обробці дуже великих обсягів даних, що може бути критичним для великих підприємств з високими вимогами до обробки інформації. Крім того, деякі користувачі відзначають необхідність постійних оновлень та підтримки для забезпечення стабільної роботи системи.

Tableau є ще одним важливим гравцем на ринку інструментів для бізнес-аналітики. Ця система відома своїми потужними можливостями візуалізації даних, що дозволяють створювати інтерактивні та динамічні панелі моніторингу. Tableau дозволяє користувачам легко досліджувати дані та виявляти приховані закономірності, що сприяє глибшому аналізу та прийняттю обґрунтованих рішень [20]. Проте, високі витрати на ліцензії та складність у налаштуванні

можуть обмежувати доступність Tableau для малих підприємств. Крім того, інтеграція з деякими специфічними джерелами даних може вимагати додаткових зусиль та ресурсів, що підвищує загальні витрати на впровадження та експлуатацію системи.

QlikView є ще одним відомим інструментом для бізнес-аналітики, який надає користувачам можливість здійснювати гнучкий аналіз даних та створювати візуалізації. Система відзначається швидкістю обробки даних та можливістю інтерактивного дослідження інформації, що дозволяє користувачам отримувати швидкі відповіді на свої запитання [21]. Проте, QlikView може мати обмеження у масштабованості та управлінні великими обсягами даних, що може бути критичним для великих підприємств з високими вимогами до обробки інформації.

Незважаючи на високу ефективність та потужні можливості, існуючі рішення мають певні обмеження, які можуть створювати бар'єри для їхнього широкого використання в різних умовах. Серед основних недоліків можна виділити високу вартість впровадження та експлуатації, складність у налаштуванні та використанні, обмежену гнучкість та масштабованість, а також проблеми з інтеграцією з іншими системами та забезпеченням безпеки даних. Ці фактори можуть суттєво знижувати ефективність використання інформаційних систем підтримки прийняття рішень та обмежувати їхню застосовність у різних умовах.

У зв'язку з цим, розробка нової системи підтримки прийняття рішень стає актуальною задачею, яка спрямована на усунення існуючих недоліків та забезпечення більшої гнучкості, адаптивності та доступності. Нова система повинна бути розроблена з урахуванням специфічних потреб підприємств, забезпечувати легку інтеграцію з існуючими інформаційними системами, бути масштабованою та гнучкою, а також відповідати високим стандартам безпеки та конфіденційності даних. Крім того, система повинна мати інтуїтивно зрозумілий

інтерфейс, що дозволить користувачам з різним рівнем технічної підготовки ефективно використовувати її можливості.

1.3 Постановка задачі

Розробка комп'ютерної системи підтримки прийняття рішень при контрольних випробуваннях технічних систем передбачає створення інтегрованого рішення, яке забезпечить автоматизацію процесів збору, обробки та аналізу даних, а також надання точних та своєчасних рекомендацій для прийняття обґрунтованих рішень. Основною задачею системи є забезпечення ефективної взаємодії між користувачами та інформаційними ресурсами, що дозволить оптимізувати процеси випробувань та підвищити якість прийнятих рішень.

Першим ключовим функціональним елементом системи є інтерфейс користувача, який повинен бути інтуїтивно зрозумілим та зручним для експертів, відповідальних за проведення випробувань. Інтерфейс повинен дозволяти легко вводити дані, такі як параметри випробувань, оцінки результатів та іншу релевантну інформацію. Це забезпечить мінімізацію людських помилок та підвищить ефективність роботи користувачів.

Другим важливим компонентом є модуль збору та зберігання даних. Система повинна забезпечувати автоматизований збір даних з різних джерел, включаючи сенсори, датчики та інші пристрої, що використовуються під час випробувань. Зібрані дані мають зберігатися в централізованій базі даних, що забезпечить їхню доступність для подальшої обробки та аналізу. Важливою вимогою є забезпечення високого рівня безпеки та конфіденційності збережених даних, що передбачає використання сучасних методів шифрування та контролю доступу.

Ключовим елементом системи є також модуль генерації звітів та візуалізацій. Цей модуль повинен надавати користувачам можливість отримувати детальні звіти, графіки та інші візуальні представлення результатів аналізу даних. Візуалізації повинні бути зрозумілими та доступними, що дозволить швидко оцінювати результати випробувань та приймати необхідні заходи для їхнього покращення.

Інтеграція системи з існуючими інформаційними ресурсами та інструментами підприємства є ще одним важливим завданням. Це дозволить забезпечити безперебійний обмін даними між різними відділами та підвищити загальну ефективність бізнес-процесів. Інтеграція повинна бути здійснена таким чином, щоб не порушувати існуючі робочі процеси та забезпечувати високу продуктивність системи.

Нарешті, система повинна мати можливість масштабування та адаптації до зростаючих обсягів даних та розширення функціональних можливостей. Це дозволить забезпечити довгострокову ефективність системи та її здатність відповідати на нові виклики та вимоги ринку.

1.4 Вибір технічної системи та критеріїв для тестування

Обраною технічною системою стала відеокарта. Це електронний пристрій, призначений для генерації та обробки зображень з подальшим виведенням на екран периферійного пристрою. Відеокарти мають різні характеристики, що дає змогу створити критерії для її оцінювання як технічної системи та порівняти декілька технічних систем для вибору найкращої.

Обраними критеріями технічної системи є:

- Продуктивність - здатність відеокарти ефективно виконувати графічні обчислення;

- Енергоспоживання – кількість енергії, яку споживає відеокарта, відносно її продуктивності;
- Вартість – співвідношення ціни та якості відеокарти;
- Зручність використання – фактичні розміри.

Обраними параметрами тестування технічної системи є:

- Для оцінки продуктивності – продуктивність з плаваючою точкою, кількість ядер, тактова частота ядра, пропускна здатність пам’яті та об’єм відеопам’яті;
- Для оцінки енергоспоживання – енергоспоживання у Вт;
- Для оцінки вартості – ціна відеокарти;
- Для оцінки зручності – довжина (чим більше – тим гірше).

Після збору даних буде проведений аналіз декілької технічних систем, проведено порівняння та будуть зроблені висновки по результатам.

1.5 Висновки до першого розділу

Постановка задачі розробки комп'ютерної системи підтримки прийняття рішень включає створення інтуїтивно зрозумілого інтерфейсу та зберігання даних, розробку потужних модулів аналізу, генерацію зрозумілих звітів, інтеграцію з існуючими системами та забезпечення масштабованості.

Обранною технічною системою стала відеокарта. Це достатньо складна система для демонстрації можливостей системи підтримки прийняття рішень, але одночасно достатньо зрозуміла для аналізу з доступмними даними для аналізу по обраним критеріям та параметрам.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Вибір архітектури

У процесі розробки комп'ютерної системи підтримки прийняття рішень було ретельно розглянуто різні архітектурні підходи з метою забезпечення максимальної ефективності, гнучкості та масштабованості системи. Одним із ключових факторів вибору архітектури було забезпечення чіткого розділення логіки додатка, інтерфейсу користувача та управління даними, що дозволяє спростити процес розробки, підтримки та подальшого вдосконалення системи [22].

Вибір архітектури базується на використанні веб-фреймворку Flask, який реалізує паттерн Model-View-Template (MVT). Цей підхід дозволяє структурувати додаток таким чином, що кожен компонент виконує чітко визначену функцію [23]. Модель відповідає за управління даними та бізнес-логікою, представлення (View) забезпечує інтерфейс для взаємодії користувача з системою, а шаблони (Template) визначають, як саме дані будуть відображатися у користувацькому інтерфейсі [24].

Однією з основних переваг вибору Flask та архітектури MVT є висока гнучкість та легкість у налаштуванні. Flask є мікрофреймворком, що дозволяє розробникам створювати додатки з мінімальними початковими налаштуваннями, одночасно надаючи можливість додавання необхідних розширень та компонентів за потребою [25]. Це особливо важливо для розробки системи підтримки прийняття рішень, оскільки дозволяє адаптуватися до специфічних вимог та змінних умов проекту без значних перебудов архітектури [26].

Архітектура MVT сприяє ефективному управлінню складними бізнес-процесами та великими обсягами даних, що є суттєвим для системи підтримки прийняття рішень при контрольних випробуваннях технічних систем. Завдяки

чіткому розділенню компонентів, розробники можуть легко модифікувати та вдосконалювати окремі частини системи без впливу на інші компоненти, що значно спрощує процес впровадження нових функцій та оновлень.

Іншою важливою причиною вибору Flask є його легкість у інтеграції з різними базами даних та іншими зовнішніми системами. Це дозволяє забезпечити безперебійний обмін даними між системою підтримки прийняття рішень та іншими інформаційними ресурсами підприємства [27], що є критично важливим для забезпечення актуальності та точності аналізу даних [28]. Крім того, Flask підтримує різні бібліотеки та інструменти для роботи з даними, що дозволяє розробникам використовувати найсучасніші методи та алгоритми для аналізу інформації [29].

Вибір архітектури також враховував потребу в забезпеченні високого рівня безпеки та конфіденційності даних. Flask надає можливості для впровадження різноманітних механізмів захисту, таких як шифрування даних, аутентифікація користувачів та управління доступом до ресурсів. Це дозволяє створити надійну систему, яка захищає конфіденційну інформацію від несанкціонованого доступу та забезпечує відповідність вимогам законодавства щодо захисту персональних даних [30].

Архітектура MVT також сприяє покращенню продуктивності та швидкості роботи системи. Завдяки розділенню логіки обробки даних та відображення результатів, система може ефективно обробляти великі обсяги інформації без значних затримок, що є критично важливим для оперативного прийняття рішень [31]. Крім того, використання шаблонів дозволяє оптимізувати процес генерації HTML-сторінок, забезпечуючи швидке та ефективне відображення даних користувачам [32].

Важливою перевагою Flask та архітектури MVT є також їхня масштабованість. Система може бути легко розширена за рахунок додавання нових модулів та компонентів без необхідності суттєвих змін у базовій структурі

[33]. Це дозволяє забезпечити довгострокову ефективність системи та її здатність відповідати на нові виклики та вимоги ринку [34]. Масштабованість системи забезпечується завдяки використанню сучасних технологій та гнучких архітектурних рішень, що дозволяють легко адаптуватися до зростаючих обсягів даних та розширення функціональних можливостей [35].

Таким чином, вибір архітектури на базі Flask та паттерну Model-View-Template є оптимальним рішенням для розробки системи підтримки прийняття рішень при контрольних випробуваннях технічних систем. Цей підхід забезпечує високу гнучкість, масштабованість, ефективність та безпеку системи, що дозволяє створити надійний та потужний інструмент для автоматизації процесів аналізу даних та прийняття обґрунтованих рішень [36]. Впровадження такої архітектури сприятиме підвищенню загальної продуктивності та конкурентоспроможності підприємства, забезпечуючи його стабільний розвиток у швидкозмінному ринковому середовищі [37].

2.2 Вибір методу контрольних випробувань

2.2.1 Аналіз обраного методу

Вибір методу контрольних випробувань для комп’ютерної системи є важливою частиною її проектування. Від нього залежать точність результатів, алгоритм та складність реалізації. Обраним методом став метод зваженої суми балів (Weighted Scoring Method або WSM).

Його алгоритм заключається у тому, що для початку потрібно визначити критерії оцінки та їх ваги. Потім проводиться оцінка системи за кожним критерієм (наприклад, від 0 до 100), після чого кожна оцінка перемножується на вагу відповідного критерію, та зважені бали підсумовуються для отримання загального результату. Рішення приймається на основі підсумку балів, тобто, якщо оцінка системи проходить певний пороговий бал, то системиа приймається.

Перевагами даного методу можна назвати простоту реалізації та прозорий процес прийняття рішень. Також він є гнучким із-за вибору ваг у критеріїв, але це є і недоліком, бо на результат впливає суб'єктивність вибору цих ваг. Але недокліками є відсутність врахування взаємозв'язків між критеріями та такий метод не підходить для систем з великим ступенем невизначеності.

Формула данного методу виглядає наступним чином:

$$A_{\mathrm{i}} = \sum_{j=1}^{n} w_{\mathrm{j}} a_{\mathrm{ij}}, \text{for } i = 1,2,3 \dots, m. \tag{2.1}$$

2.2.2 Аналіз інших методів

В альтернативу обраному методу можна привести метод TOPSIS (Technique for Order Preference by Similatiry to Ideal Solution). Його алгоритм заключається у тому, що для початку потрібно сформувати матрицю рішень, в якій строки – це альтернативні стратегії, а стовпці – критерії оцінки. Після чого дана матриця нормалізується, щоб усі значення були приведені до одного масштабу, та множиться на вектор ваг критеріїв. Потім обчислюється відстань кожної альтернативи до ідеального та антиідеального рішення та обирається альтернатива з найбільшою близкістю. Перевагами даного методу є зручне порівняння декількох альтернатив, але присутня залежність результату від вибору ваг.

Також схожим на попередній метод можна назвати PROMETHEE (Preference Ranking Organization Method for Enrichment Evaluation). Він також використовує альтернативи для розрахунку результату, але алгоритм інший. Для початку визначаються критерії оцінки та їх ваги. Потім розраховується функція переваги для кожної альтернативи, тобто, йде порівняння пар об'єктів по

кожному критерію. Функції переваги мають поріг байдужості (якщо їх різниця в парі об’єктів несуттєва, то вони еквівалентні по заданому критерію). Після чого розраховують відносну перевагу для кожної альтернативи щодо інших та підссумовують результати для отримання остаточного ранжування. Перевагами даного методу є гнучкість у налаштуванні функцій переваг та можливість враховування суб’єктивних переваг, але сам метод, відносно інших, складний для розуміння та реалізації.

2.3 Проектування бази даних

Проектування бази даних є ключовим етапом розробки комп'ютерної системи підтримки прийняття рішень, оскільки від цього залежить ефективність зберігання, обробки та доступу до даних, необхідних для прийняття обґрунтованих рішень під час контрольних випробувань технічних систем. У даному проекті була обрана реляційна модель бази даних, що дозволяє забезпечити структуроване та організоване зберігання інформації, а також забезпечує цілісність та узгодженість даних через використання первинних та зовнішніх ключів.

Основними сутностями, що визначені в базі даних, є "Система", "Критерій" та "Оцінка". Кожна з цих сутностей має свої атрибути та взаємозв'язки, що дозволяє ефективно організувати інформацію та забезпечити її доступність для аналізу та прийняття рішень.

Сутність "Система" представляє технічні системи, які підлягають контрольним випробуванням. Вона містить такі атрибути, як унікальний ідентифікатор системи (id), назву системи (name), а також зв'язок з оцінками, які надаються для цієї системи. Поле name є унікальним та обов'язковим для заповнення, що забезпечує унікальність кожної системи в базі даних. Взаємозв'язок між сутностями "Система" та "Оцінка" реалізовано через

зовнішній ключ system_id, що забезпечує зв'язок кожної оцінки з конкретною системою.

Сутність "Критерій" відображає різні аспекти, за якими оцінюються технічні системи під час випробувань. Кожен критерій має унікальний ідентифікатор (id), назву критерія (name) та вагу критерію (weight), яка визначає важливість цього критерію в загальній оцінці системи. Назва критерію також є унікальною та обов'язковою для заповнення, що запобігає дублюванню критеріїв у базі даних. Вага критерію є числовим значенням, що дозволяє об'єктивно оцінювати внесок кожного критерію в загальну оцінку системи.

Сутність "Оцінка" є проміжною таблицею, яка реалізує зв'язок між сутностями "Система" та "Критерій". Вона містить унікальний ідентифікатор оцінки (id), зовнішній ключ system_id, що посилається на конкретну систему, зовнішній ключ criterion_id, що посилається на конкретний критерій, а також оцінку (score), яка представляє числове значення, надане за відповідним критерієм для конкретної системи. Поля system_id та criterion_id є обов'язковими для заповнення, що забезпечує цілісність даних та дозволяє уникнути некоректних зв'язків між системами та критеріями. Взаємозв'язок між сутностями реалізовано через відношення "багато до одного" (many-to-one), де одна система може мати багато оцінок за різними критеріями, а один критерій може бути використаний для оцінки багатьох систем.

Для забезпечення ефективного управління даними та підтримки бізнес-процесів система підтримки прийняття рішень повинна мати можливість додавання нових систем, критеріїв та оцінок, а також редагування та видалення існуючих записів. Це передбачає наявність відповідних форм для введення даних, а також механізмів для валідації та обробки введеної інформації. У нашому проекті були реалізовані форми "SystemForm" та "EvaluationForm", які дозволяють користувачам додавати нові системи та оцінки відповідно. Форми включають необхідні валідатори для забезпечення коректності введених даних,

такі як перевірка наявності значень у обов'язкових полях та обмеження на допустимі значення оцінок.

Ініціалізація критеріїв відбувається через спеціальну функцію, яка додає стандартні критерії до бази даних, якщо вони ще не існують. Це дозволяє забезпечити наявність базового набору критеріїв для оцінки систем при першому запуску системи, що спрощує процес налаштування та використання системи новими користувачами.

Для обчислення загального балу системи використовуються ваги критеріїв, що дозволяє здійснювати об'єктивну оцінку систем на основі важливості кожного критерію. Функція calculate_total_score системи бере всі оцінки, надані для конкретної системи, множить їх на відповідні ваги критеріїв та обчислює агрегований бал. На основі цього балу система приймає рішення про прийняття або відхилення системи, що дозволяє автоматизувати процес прийняття рішень та зменшити вплив людського фактору.

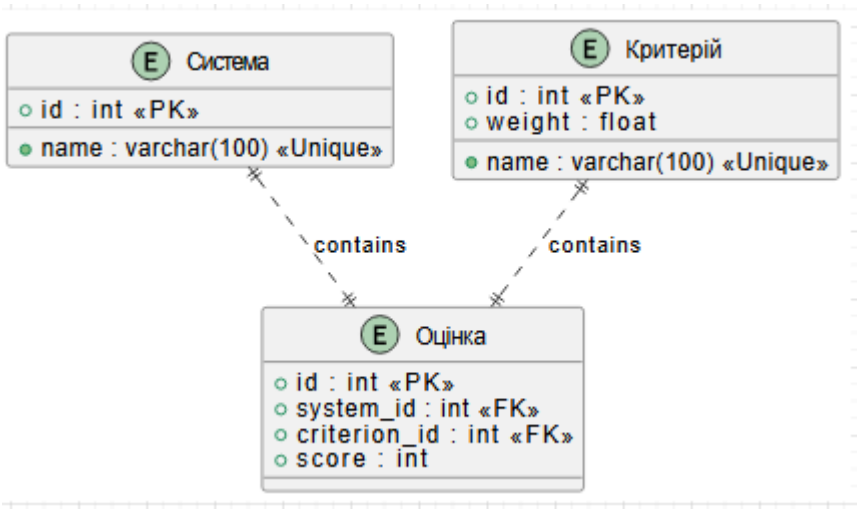

Рисунок 2.2 — Схема бази даних

Схема бази даних складається з трьох основних сутностей: "Система", "Критерій" та "Оцінка". Кожна сутність представлена таблицею з відповідними

атрибутами та ключами, що визначають її унікальність та зв'язки з іншими таблицями.

Таблиця "Система" містить інформацію про технічні системи, які підлягають контрольним випробуванням. Кожен запис у цій таблиці має унікальний ідентифікатор (id), який служить первинним ключем, та назву системи (name), яка також є унікальною. Це забезпечує однозначну ідентифікацію кожної системи в базі даних та запобігає дублюванню записів.

Таблиця "Критерій" зберігає інформацію про критерії, за якими оцінюються технічні системи. Кожен критерій має унікальний ідентифікатор (id), назву (name) та вагу (weight), яка визначає важливість цього критерію в загальній оцінці системи. Унікальність назви критерію забезпечується за допомогою обмеження <<Unique>>, що запобігає дублюванню критеріїв у таблиці.

Таблиця "Оцінка" є проміжною таблицею, що реалізує зв'язок між сутностями "Система" та "Критерій". Вона містить унікальний ідентифікатор оцінки (id), зовнішній ключ system_id, що посилається на таблицю "Система", зовнішній ключ criterion_id, що посилається на таблицю "Критерій", та оцінку (score), яка представляє числове значення, надане за відповідним критерієм для конкретної системи. Взаємозв'язок між таблицями реалізовано через зовнішні ключі, що забезпечує цілісність даних та дозволяє встановлювати зв'язки між системами та їхніми оцінками за різними критеріями.

Взаємозв'язки між таблицями відображені на схемі бази даних, де одна система може мати багато оцінок за різними критеріями, а один критерій може бути використаний для оцінки багатьох систем. Це реалізовано через відношення "один до багатьох" між таблицями "Система" та "Оцінка", а також між таблицями "Критерій" та "Оцінка".

Ця структура бази даних забезпечує ефективне зберігання та обробку даних, необхідних для підтримки процесу прийняття рішень. Вона дозволяє легко додавати нові системи, критерії та оцінки, забезпечуючи гнучкість та

адаптивність системи до змінних вимог. Крім того, використання реляційної моделі забезпечує можливість виконання складних запитів для аналізу даних, що є важливим аспектом для системи підтримки прийняття рішень.

Узгодженість та цілісність даних забезпечується через використання первинних та зовнішніх ключів, а також через обмеження на рівні бази даних, що запобігає некоректному введенню даних та забезпечує їхню правильну зв'язок. Це дозволяє уникнути дублювання даних, забезпечує їхню актуальність та точність, а також полегшує процес їхнього аналізу та інтерпретації.

Таким чином, проектування бази даних системи підтримки прийняття рішень при контрольних випробуваннях технічних систем включає створення трьох основних таблиць з відповідними атрибутами та ключами, а також визначення зв'язків між ними. Ця структура забезпечує ефективне та гнучке зберігання даних, що дозволяє системі підтримки прийняття рішень виконувати свої функції з високою точністю та надійністю, сприяючи підвищенню ефективності процесів випробувань та прийняття рішень.

2.4 Проектування внутрішньої будови

Проектування внутрішньої будови комп'ютерної системи підтримки прийняття рішень при контрольних випробуваннях технічних систем є важливим етапом розробки, який визначає структуру та взаємодію компонентів системи. Цей розділ детально описує основні класи, їхні атрибути та методи, а також взаємозв'язки між ними, що забезпечують ефективну та гнучку роботу системи. Для ілюстрації внутрішньої архітектури було створено діаграму класів, яка відображає основні компоненти системи та їхні взаємодії.

Основною частиною системи є моделі даних, які реалізовані за допомогою бібліотеки SQLAlchemy. Ці моделі відповідають таблицям бази даних і забезпечують ORM (Object-Relational Mapping) для спрощення роботи з даними.

У системі визначено три основні моделі: System, Criterion та Evaluation. Кожна з цих моделей має свої унікальні атрибути та зв'язки, що дозволяє ефективно організувати зберігання та обробку інформації.

Клас System відповідає за зберігання інформації про технічні системи, які підлягають випробуванням. Він має атрибути id, name, а також зв'язок з моделлю Evaluation через атрибут evaluations. Цей зв'язок реалізовано за допомогою параметра backref='system', що дозволяє легко доступатися до оцінок, пов'язаних з конкретною системою. Метод __repr__ забезпечує зручне представлення об'єктів класу у вигляді рядків, що полегшує відлагодження та роботу з об'єктами у консолі.

Клас Criterion визначає різні критерії, за якими оцінюються технічні системи. Він містить атрибути id, name та weight, де weight визначає вагу критерію у загальній оцінці системи. Атрибут weight є числовим значенням, яке дозволяє об'єктивно оцінювати внесок кожного критерію у загальну оцінку системи. Зв'язок з моделлю Evaluation реалізовано через атрибут evaluations, що дозволяє відстежувати, які системи оцінюються за цим критерієм.

Клас Evaluation є проміжною моделлю, яка реалізує зв'язок між системами та критеріями. Він містить атрибути id, system_id, criterion_id та score. Атрибути system_id та criterion_id є зовнішніми ключами, що посилаються на відповідні таблиці System та Criterion. Атрибут score представляє оцінку, надану за відповідним критерієм для конкретної системи. Завдяки цьому зв'язку одна система може мати багато оцінок за різними критеріями, а один критерій може бути використаний для оцінки багатьох систем.

Класи SystemForm та EvaluationForm реалізують форми для введення даних користувачами через веб-інтерфейс. Вони успадковуються від класу FlaskForm і містять відповідні поля, такі як name для системи, system, criterion та score для оцінок. Використання форм забезпечує валідацію введених даних та спрощує процес їхнього збирання та обробки.

Для управління взаємодією користувачів з системою використовуються маршрути (routes), які визначаються у Flask-додатку. Маршрути відповідають за обробку запитів до певних URL-адрес, виконання відповідних дій та відображення відповідних шаблонів (templates). Наприклад, маршрут /add_system відповідає за додавання нових систем, а маршрут /add_evaluation – за додавання оцінок до систем.

Важливим компонентом внутрішньої будови системи є функція calculate_total_score, яка відповідає за обчислення загального балу системи на основі оцінок за різними критеріями. Ця функція враховує ваги критеріїв, що дозволяє об'єктивно оцінювати внесок кожного критерію у загальну оцінку системи. Результатом роботи функції є агрегований бал та рішення щодо прийняття або відхилення системи.

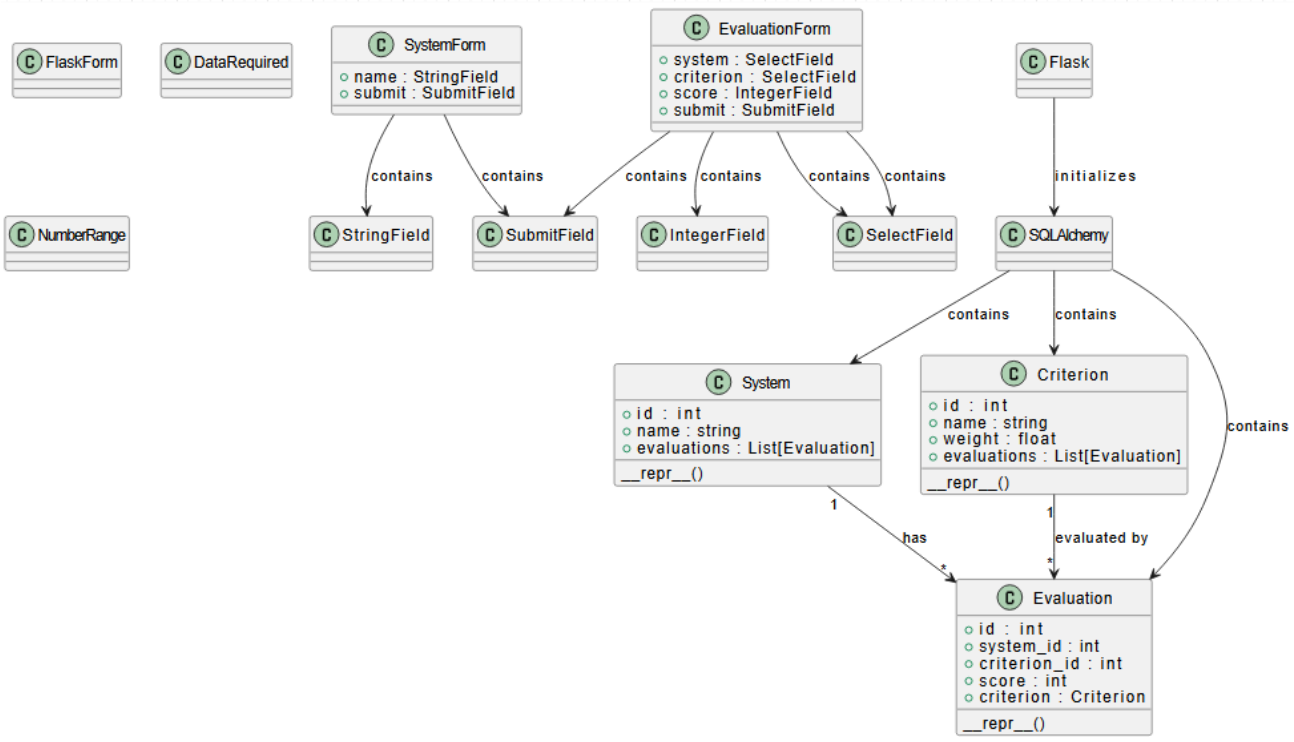

Рисунок 2.3 — Внутрішня будова системи

Діаграма класів представляє основні компоненти внутрішньої будови системи підтримки прийняття рішень. Вона включає класи, які відповідають за моделі даних, форми для введення інформації та основні компоненти фреймворку Flask, такі як Flask та SQLAlchemy.

Клас FlaskForm є базовим класом для всіх форм у системі. Він містить загальні методи та атрибути, які успадковуються іншими формами, такими як SystemForm та EvaluationForm. Ці форми використовуються для введення даних користувачами через веб-інтерфейс.

Класи StringField, SubmitField, IntegerField, SelectField, DataRequired та NumberRange представляють різні типи полів та валідаторів, які використовуються у формах. Вони забезпечують валідацію введених даних та визначають типи інформації, що вводиться користувачем.

Клас Flask відповідає за ініціалізацію Flask-додатку, налаштування конфігурації та управління маршрутизацією запитів. Клас SQLAlchemy є ORM-бібліотекою, яка використовується для взаємодії з базою даних, забезпечуючи зручний спосіб роботи з моделями даних у вигляді класів Python.

Класи System, Criterion та Evaluation представляють основні моделі даних системи. Клас System містить інформацію про технічні системи, які підлягають випробуванням, та має зв'язок з оцінками через атрибут evaluations. Клас Criterion визначає критерії, за якими оцінюються системи, та має зв'язок з оцінками через атрибут evaluations. Клас Evaluation реалізує зв'язок між системами та критеріями, зберігаючи оцінки, надані за кожним критерієм для конкретної системи.

Класи SystemForm та EvaluationForm представляють форми для введення даних користувачами. SystemForm використовується для додавання нових систем, а EvaluationForm – для додавання оцінок до систем за різними критеріями. Обидві форми містять поля, які відповідають атрибутам відповідних моделей, та валідатори для забезпечення коректності введених даних.

Взаємозв'язки між класами демонструють, як різні компоненти системи взаємодіють один з одним. Flask ініціалізує SQLAlchemy, яка потім використовується для створення та управління моделями даних. Класи System та Criterion мають зв'язок "один до багатьох" з класом Evaluation, що дозволяє одній системі мати багато оцінок за різними критеріями та одному критерію оцінювати багато систем.

Функція calculate_total_score відповідає за обчислення загального балу системи на основі оцінок за різними критеріями та їхніх ваг. Ця функція використовує зв'язок між системами та оцінками для отримання необхідних даних та обчислення агрегованого балу, який визначає рішення про прийняття або відхилення системи.

Крім того, система включає маршрути, які визначають поведінку додатку при обробці запитів користувачів. Маршрути відповідають за відображення головної сторінки, додавання нових систем та оцінок, а також перегляд деталей конкретних систем. Взаємодія з базою даних здійснюється через SQLAlchemy, що дозволяє ефективно виконувати CRUD-операції (створення, читання, оновлення, видалення) над даними.

Детальний опис класів та їхніх взаємозв'язків

Клас System має наступні атрибути:

- id: Унікальний ідентифікатор системи, який автоматично генерується базою даних.
- name: Назва системи, яка є унікальною та обов'язковою для заповнення.
- evaluations: Зв'язок з моделлю Evaluation, що дозволяє отримувати всі оцінки, пов'язані з цією системою.

Метод __repr__ повертає строкове представлення об'єкта класу, що полегшує його відображення у консолі та журналах.

Клас Criterion включає такі атрибути:

- id: Унікальний ідентифікатор критерію.
- name: Назва критерію, яка є унікальною.
- weight: Вага критерію у загальній оцінці системи, яка визначає його важливість.

Метод __repr__ також реалізований для зручності відображення об'єктів класу.

Клас Evaluation має наступні атрибути:

- id: унікальний ідентифікатор оцінки.
- system_id: зовнішній ключ, що посилається на таблицю System.
- criterion_id: зовнішній ключ, що посилається на таблицю Criterion.
- score: оцінка, надана за відповідним критерієм для конкретної системи.

Взаємозв'язок між моделями реалізовано через атрибути system та criterion, які дозволяють легко доступатися до пов'язаних об'єктів System та Criterion відповідно.

Класи SystemForm та EvaluationForm включають поля, які відповідають атрибутам моделей. SystemForm містить поле name для введення назви системи та кнопку submit для підтвердження додавання системи. EvaluationForm включає поля system та criterion, які є селектерами для вибору системи та критерію відповідно, поле score для введення оцінки та кнопку submit для підтвердження додавання оцінки.

Взаємодія між користувачем та системою здійснюється через веб-інтерфейс, де користувачі можуть вводити дані, переглядати результати аналізу та отримувати рекомендації. Маршрути відповідають за відображення відповідних шаблонів та обробку введених даних. Наприклад, маршрут /add_system відповідає за відображення форми для додавання нової системи та обробку введених даних, а маршрут /add_evaluation – за додавання нових оцінок до систем.

Функція calculate_total_score є критично важливою для системи, оскільки вона автоматизує процес обчислення загального балу системи на основі введених оцінок та їх ваг. Вона забезпечує об'єктивність та точність прийнятих рішень, враховуючи важливість кожного критерію. Результат роботи цієї функції використовується для прийняття рішення про прийняття або відхилення системи, що дозволяє автоматизувати процес прийняття рішень та зменшити вплив людського фактору.

Обрана внутрішня будова системи забезпечує кілька ключових переваг, які сприяють її ефективній роботі та гнучкості:

1. Модульність та розширюваність: завдяки чіткому розділенню моделей даних, форм та маршрутів, система є модульною та легко розширюється. Це дозволяє додавати нові функціональні можливості без суттєвих змін у існуючій архітектурі.

2. Інтуїтивно зрозумілий інтерфейс: веб-інтерфейс розроблений таким чином, щоб користувачі могли легко вводити та переглядати дані, що забезпечує високу зручність використання системи.

3. Гнучкість налаштувань: система дозволяє налаштовувати критерії оцінки та ваги відповідно до потреб підприємства, що забезпечує гнучкість та адаптивність до змінних умов ринку.

4. Інтеграція з іншими системами: завдяки використанню Flask та SQLAlchemy, система легко інтегрується з іншими інформаційними ресурсами підприємства, що сприяє більш ефективному управлінню даними та бізнес-процесами.

2.5 Проектування алгоритмів системи

Проектування алгоритмів системи підтримки прийняття рішень при контрольних випробуваннях технічних систем є ключовим етапом розробки,

оскільки від цього залежить ефективність та надійність роботи всієї системи. Алгоритми визначають логіку обробки даних, взаємодію між компонентами системи та забезпечують реалізацію основних бізнес-процесів. У даному підрозділі детально розглядаються основні алгоритми, реалізовані у системі, їхні функції, логіка роботи та взаємозв'язки між ними на основі розроблених блок-схем.

Рисунок 2.4 — Блок-схема алгоритму init_criteria()

Першим важливим алгоритмом є функція init_criteria(), яка відповідає за ініціалізацію бази даних стандартними критеріями оцінки технічних систем. Цей алгоритм починається зі створення списку критеріїв, кожен з яких має назву та вагу, що визначає його важливість у загальній оцінці системи. Для кожного

критерію перевіряється його наявність у базі даних за допомогою запиту Criterion.query.filter_by(name=crit['name']).first(). Якщо критерій з такою назвою ще не існує, створюється новий об'єкт Criterion з відповідними атрибутами та додається до сесії бази даних за допомогою db.session.add(new_criterion). Після обробки всіх критеріїв здійснюється коміт транзакції за допомогою db.session.commit(), що зберігає нові критерії у базі даних. Цей алгоритм гарантує, що при першому запуску системи база даних буде заповнена необхідними критеріями, що спрощує подальшу роботу користувачів із системою.

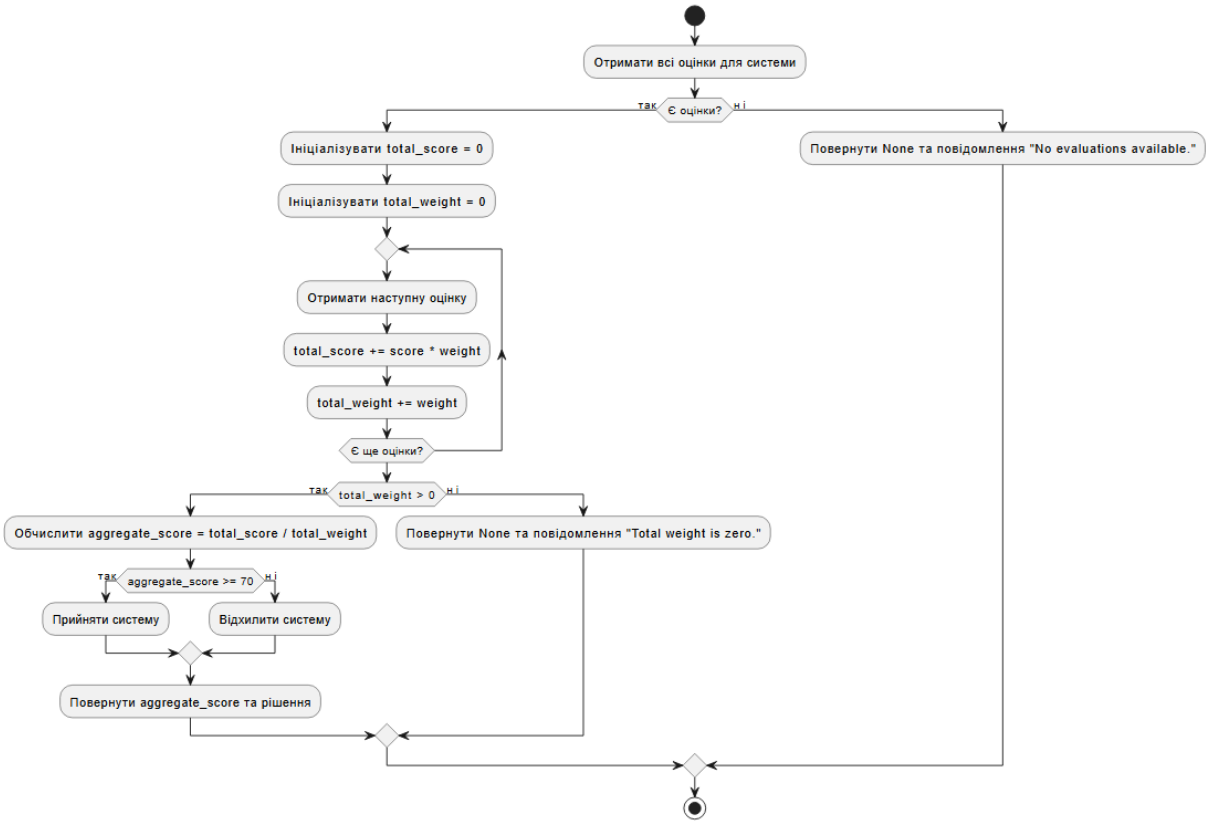

Рисунок 2.5 — Блок-схема алгоритму calculate_total_score(system)

Наступним ключовим алгоритмом є функція calculate_total_score(system), яка відповідає за обчислення агрегованого балу системи на основі її оцінок за різними критеріями.

Процес починається з отримання всіх оцінок, пов'язаних з конкретною системою, за допомогою запиту Evaluation.query.filter_by(system_id=system.id).all(). Якщо оцінок немає, функція повертає None та повідомлення "No evaluations available.", що сигналізує про відсутність даних для обчислення. Якщо оцінки присутні, алгоритм ініціалізує змінні total_score та total_weight для накопичення сумарного балу та сумарної ваги критеріїв відповідно. Далі здійснюється цикл, де кожна оцінка множиться на вагу відповідного критерію і додається до total_score, а вага критерію додається до total_weight. Після обробки всіх оцінок перевіряється, чи не дорівнює сумарна вага нулю. Якщо це так, функція повертає None та повідомлення "Total weight is zero.", що вказує на некоректну конфігурацію ваг критеріїв. В іншому випадку, агрегований бал обчислюється як співвідношення total_score до total_weight. На основі цього балу приймається рішення: якщо агрегований бал більший або дорівнює 70, система приймається, інакше — відхиляється. Функція повертає агрегований бал та відповідне рішення, що дозволяє користувачам швидко оцінювати стан системи та приймати обґрунтовані рішення щодо її подальшого використання або вдосконалення.

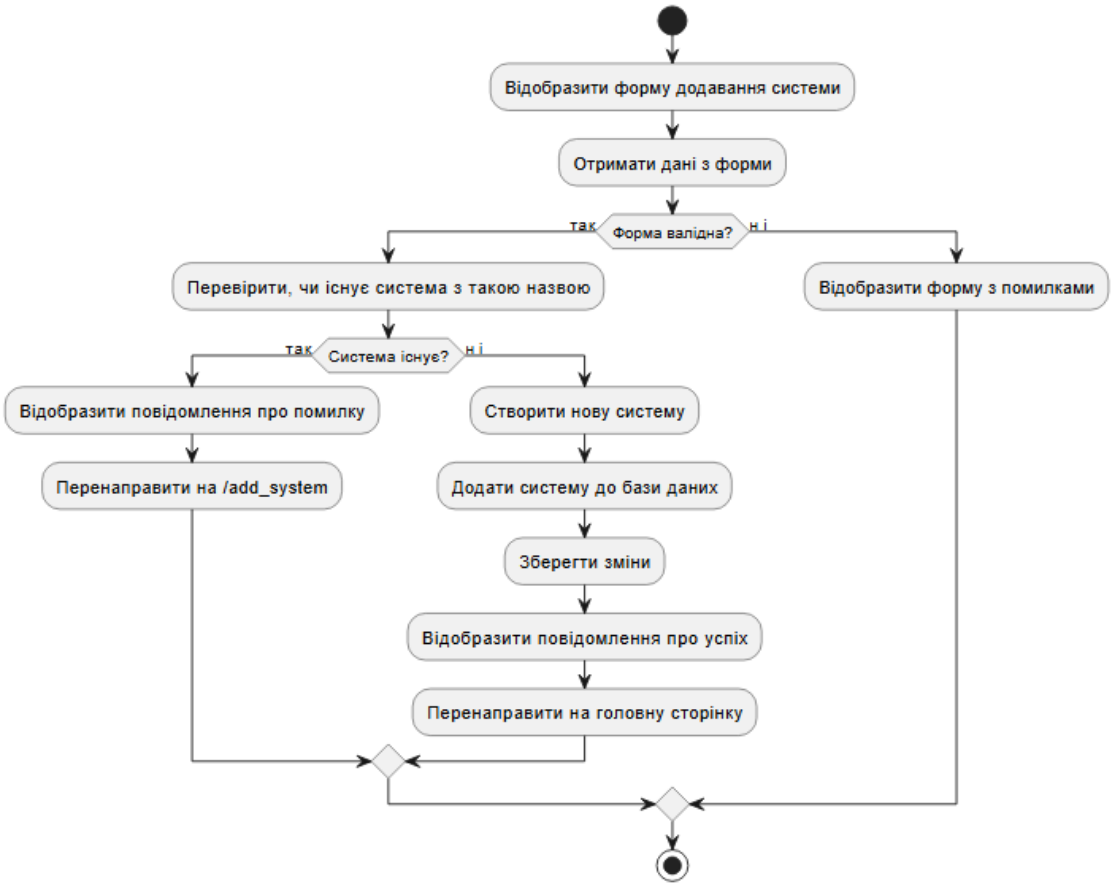

Рисунок 2.6 — Блок-схема алгоритму додавання системи

Маршрутизація у системі реалізована за допомогою Flask-додатку, де кожен маршрут відповідає за певну функціональність системи. Наприклад, маршрут /add_system відповідає за додавання нових систем до бази даних. Алгоритм роботи цього маршруту починається з відображення форми SystemForm, де користувач вводить назву нової системи. Після подання форми здійснюється перевірка валідності введених даних за допомогою методу validate_on_submit(). Якщо дані валідні, виконується перевірка наявності системи з такою ж назвою у базі даних за допомогою запиту System.query.filter_by(name=form.name.data).first(). Якщо система вже існує, користувачу відображається повідомлення про помилку через функцію flash, і відбувається перенаправлення на сторінку додавання системи. Якщо система з такою назвою відсутня, створюється новий об'єкт System з введеною назвою,

який додається до сесії бази даних та зберігається за допомогою db.session.commit().

Після успішного додавання користувач отримує повідомлення про успіх та перенаправляється на головну сторінку системи. Цей алгоритм забезпечує уникнення дублювання записів та підтримку цілісності даних у базі.

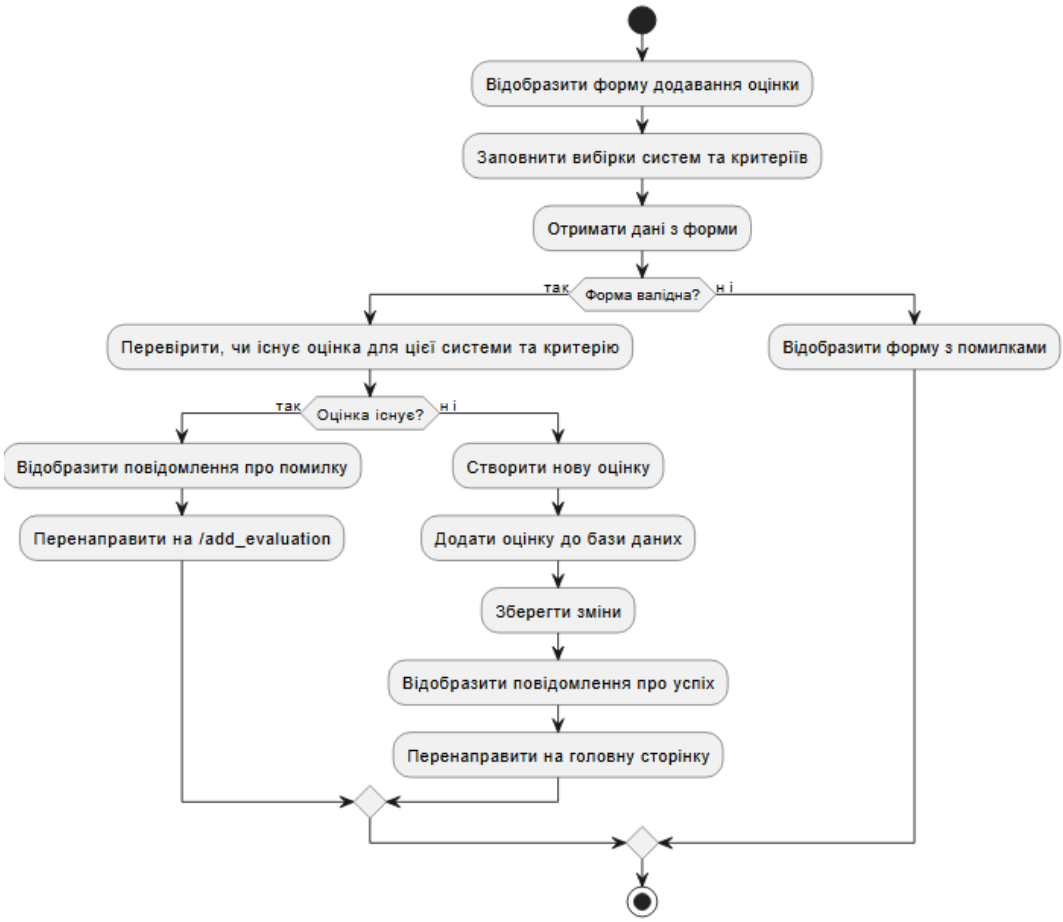

Рисунок 2.7 — Блок-схема алгоритму додавання оцінок

Аналогічним чином працює маршрут /add_evaluation, який відповідає за додавання оцінок до систем за певними критеріями. Спочатку заповнюються вибіркові поля форми EvaluationForm на основі наявних систем та критеріїв у базі даних. Це здійснюється за допомогою генерації списків варіантів для полів system та criterion через запити System.query.order_by('name') та

Criterion.query.order_by('name'). Після заповнення та подання форми здійснюється перевірка валідності даних. Далі перевіряється, чи вже існує оцінка для вибраної системи та критерію за допомогою запиту Evaluation.query.filter_by(system_id=form.system.data, criterion_id=form.criterion.data).first(). Якщо така оцінка вже існує, користувачу відображається повідомлення про помилку через функцію flash, і здійснюється перенаправлення на сторінку додавання оцінки. Якщо оцінка відсутня, створюється новий об'єкт Evaluation з відповідними полями system_id, criterion_id та score, який додається до сесії бази даних та зберігається за допомогою db.session.commit(). Після успішного додавання користувач отримує повідомлення про успіх та перенаправляється на головну сторінку. Цей алгоритм забезпечує унікальність оцінок для кожної системи та критерію, а також підтримку цілісності даних.

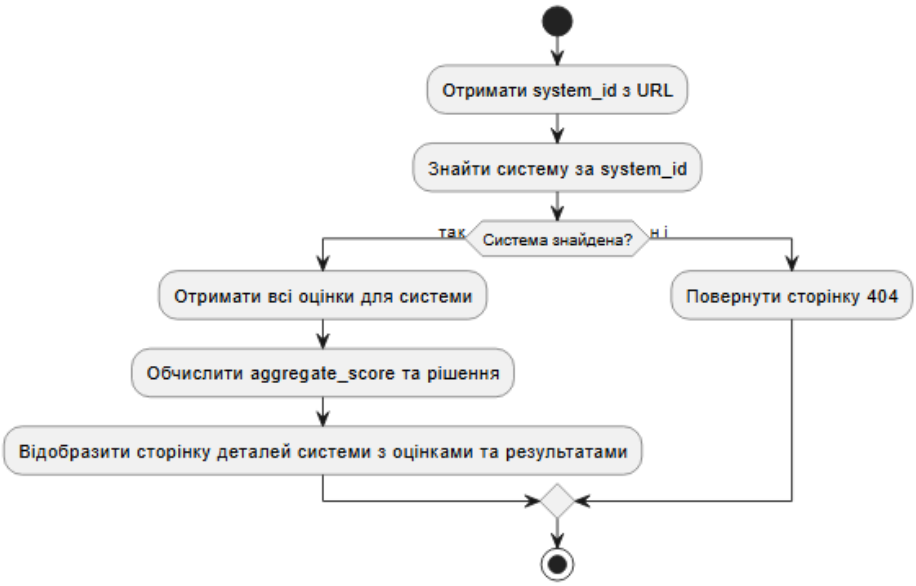

Рисунок 2.8 — Блок-схема алгоритму перегляду системи

Маршрут /system/<int:system_id> відповідає за відображення детальної інформації про конкретну систему, включаючи всі її оцінки та агрегований бал.

Алгоритм роботи цього маршруту починається з отримання system_id з URL-адреси та пошуку відповідної системи у базі даних за допомогою методу System.query.get_or_404(system_id). Якщо система не знайдена, користувачу повертається сторінка з помилкою 404. Якщо система знайдена, отримуються всі оцінки, пов'язані з цією системою, за допомогою запиту Evaluation.query.filter_by(system_id=system.id).all(). Далі викликається функція calculate_total_score(system), яка обчислює агрегований бал та прийняте рішення на основі отриманих оцінок. Отримані дані передаються до шаблону system_detail.html для відображення детальної інформації про систему, її оцінки, агрегований бал та рішення. Цей алгоритм дозволяє користувачам детально аналізувати оцінки конкретної системи та приймати обґрунтовані рішення щодо її подальшого використання або вдосконалення.

Кожен з маршрутових алгоритмів реалізований з урахуванням принципів модульності та розділення обов'язків, що забезпечує чистоту та організованість коду. Це полегшує його підтримку, тестування та подальший розвиток, дозволяючи легко вносити зміни та вдосконалення без впливу на інші частини системи. Використання Flask як мікрофреймворку забезпечує простоту та гнучкість у створенні маршрутів та обробці запитів, тоді як SQLAlchemy як ORM-бібліотека спрощує роботу з базою даних та забезпечує ефективне управління зв'язками між таблицями.

Функція init_criteria() та маршрути /add_system та /add_evaluation забезпечують основні операції додавання даних до системи, перевіряючи їх на валідність та унікальність, що запобігає виникненню помилок та дублюванню записів. Функція calculate_total_score(system) забезпечує об'єктивний підхід до оцінки систем, враховуючи ваги критеріїв та забезпечуючи точність результатів. Маршрут /system/<int:system_id> дозволяє детально переглядати оцінки конкретної системи, що сприяє глибшому аналізу та прийняттю обґрунтованих рішень.

Взаємодія між компонентами системи організована таким чином, щоб забезпечити ефективну обробку даних та швидку реакцію на запити користувачів. Маршрути відповідають за обробку запитів до певних URL-адрес, виконання відповідних дій та відображення результатів через шаблони. Цей підхід дозволяє розділити логіку додатку від представлення даних, забезпечуючи чистоту та організованість коду, що полегшує його підтримку та подальший розвиток.

Алгоритми введення даних експертами забезпечують точність та коректність введених даних через використання валідаторів у формах. Це гарантує, що всі введені дані відповідають вимогам системи та не призводять до помилок при обробці. Використання форм SystemForm та EvaluationForm з відповідними валідаторами забезпечує надійність та стабільність роботи системи, запобігаючи введенню некоректних або неповних даних.

Функція flash використовується для відображення повідомлень про успіх чи помилку, що дозволяє надавати зворотний зв'язок користувачам про результати їхніх дій. Це покращує взаємодію з системою та підвищує її зручність у використанні. Перенаправлення користувачів на відповідні сторінки після виконання певних дій забезпечує послідовність користувацького досвіду та допомагає уникнути повторного виконання дій при оновленні сторінок.

Всі алгоритми розроблені з урахуванням принципів надійності та безпеки, що забезпечує стабільну та передбачувану роботу системи. Використання механізмів обробки помилок та валідації даних дозволяє системі коректно реагувати на нестандартні ситуації та запобігати можливим збоям у роботі. Це підвищує загальну надійність системи та забезпечує її стабільну роботу навіть у випадку виникнення непередбачуваних ситуацій.

2.6 Вибір засобів розробки

Вибір засобів розробки є критично важливим етапом у процесі створення комп'ютерної системи підтримки прийняття рішень, оскільки визначає ефективність, гнучкість та масштабованість кінцевого продукту. У даному проекті було обрано мову програмування Python, веб-фреймворк Flask та ORM-бібліотеку SQLAlchemy. Ці технології були обрані з огляду на їхні численні переваги, сумісність з вимогами проекту та підтримку спільноти розробників.

Python був обраний як основна мова програмування завдяки своїй простоті, читабельності та широкому спектру застосувань. Python є високорівневою мовою, що дозволяє розробникам швидко та ефективно створювати програмні рішення з мінімальними витратами часу на написання коду. Завдяки чіткій синтаксичній структурі, Python забезпечує високу читабельність коду, що полегшує процес його підтримки та подальшого розвитку. Крім того, Python має велику стандартну бібліотеку, яка містить широкий набір модулів для різноманітних задач, що дозволяє значно скоротити час розробки.

Іншим важливим аспектом вибору Python є його універсальність. Python підтримує різні парадигми програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування, що дозволяє розробникам використовувати найбільш ефективний підхід для вирішення конкретних завдань. Крім того, Python має потужну екосистему бібліотек та фреймворків, що робить його ідеальним вибором для розробки веб-додатків, аналітики даних, машинного навчання та інших сфер.

Вибір веб-фреймворку Flask був обумовлений його легкістю, гнучкістю та мінімалістичним підходом до розробки веб-додатків. Flask є мікрофреймворком для Python, який надає базові інструменти для створення веб-додатків, залишаючи розробнику велику свободу у виборі додаткових компонентів та бібліотек. Це дозволяє створювати кастомізовані рішення, що максимально

відповідають специфічним вимогам проекту. Flask має невеликий розмір та просту архітектуру, що робить його легким для навчання та швидким для розробки.

Однією з ключових переваг Flask є його розширюваність. Завдяки широкому вибору розширень, таких як Flask-WTF для роботи з формами, Flask-Login для управління автентифікацією користувачів та Flask-Migrate для міграції бази даних, Flask дозволяє легко додавати нові функціональні можливості до додатку без необхідності значних змін у його основній структурі. Це забезпечує високу гнучкість та можливість адаптації додатку до змінних вимог бізнесу.

Flask також підтримує використання шаблонів для розділення логіки додатка від представлення, що сприяє чистоті коду та полегшує його підтримку. Використання шаблонів дозволяє створювати динамічні веб-сторінки, які можуть бути легко оновлені та змінені без необхідності змінювати основний код додатка. Це особливо корисно у випадках, коли дизайн веб-інтерфейсу потребує регулярних змін або оновлень.

Вибір ORM-бібліотеки SQLAlchemy був зроблений з огляду на її потужні можливості та сумісність з Flask. SQLAlchemy є однією з найбільш популярних ORM-бібліотек для Python, що забезпечує високий рівень абстракції для роботи з базами даних. Вона дозволяє розробникам працювати з базою даних у вигляді об'єктів Python, що значно спрощує процес написання та підтримки SQL-запитів.

Однією з головних переваг SQLAlchemy є її підтримка як реляційних, так і нереляційних баз даних, що забезпечує високу гнучкість у виборі сховища даних. SQLAlchemy підтримує широкий спектр баз даних, включаючи PostgreSQL, MySQL, SQLite, Oracle та інші, що дозволяє легко змінювати тип бази даних без необхідності значних змін у коді додатка. Це є важливим аспектом для проектів, які можуть вимагати масштабування або зміни бази даних у майбутньому.

SQLAlchemy також підтримує потужні функції для управління транзакціями, забезпечуючи цілісність та консистентність даних навіть у

випадках виникнення помилок або збоїв. Це є критично важливим для системи підтримки прийняття рішень, де точність та надійність даних мають вирішальне значення для прийняття обґрунтованих рішень. Крім того, SQLAlchemy надає можливість використовувати складні запити та зв'язки між таблицями, що дозволяє ефективно обробляти великі обсяги даних та виконувати складні аналітичні завдання.

Важливою характеристикою SQLAlchemy є її підтримка міграцій бази даних через розширення Flask-Migrate. Це дозволяє автоматизувати процес внесення змін до структури бази даних, що значно спрощує управління версіями бази даних та зменшує ризик виникнення помилок при ручному внесенні змін. Flask-Migrate інтегрується з Alembic, інструментом для міграцій бази даних, що забезпечує плавний перехід між різними версіями схеми бази даних без втрати даних.

Крім технічних переваг, Python, Flask та SQLAlchemy мають активну та підтримуючу спільноту розробників, що забезпечує доступ до великої кількості ресурсів, документації та прикладів коду. Це сприяє швидкому вирішенню виникаючих проблем та постійному вдосконаленню проекту завдяки внескам з боку спільноти. Наявність численних плагінів та розширень також дозволяє легко інтегрувати додаткові функціональні можливості до системи без необхідності розробки їх з нуля.

Вибір цих технологій також обумовлений їхньою популярністю та стабільністю на ринку розробки програмного забезпечення. Python є однією з найпопулярніших мов програмування у світі, що забезпечує високу доступність кваліфікованих розробників та постійне оновлення мовних конструкцій та бібліотек. Flask, будучи мікрофреймворком, надає можливість створювати легкі та швидкі веб-додатки, що є особливо важливим для проектів, які потребують швидкого запуску та ефективного використання ресурсів.

SQLAlchemy, як ORM-бібліотека, забезпечує зручний та ефективний спосіб роботи з базами даних, що дозволяє розробникам зосередитися на логіці додатка замість витрат часу на написання та оптимізацію SQL-запитів. Це сприяє підвищенню продуктивності розробки та зменшенню кількості помилок у коді, що може призвести до підвищення надійності та стабільності системи.

Вибір Python, Flask та SQLAlchemy також узгоджується з принципами модульності та розширюваності, які є важливими для забезпечення довгострокової підтримки та розвитку системи. Ці технології дозволяють легко додавати нові функціональні можливості, інтегруватися з іншими системами та адаптуватися до змінних вимог бізнесу без необхідності значних змін у основній архітектурі додатка.

Загалом, обрані засоби розробки – Python, Flask та SQLAlchemy – забезпечують оптимальне поєднання простоти, гнучкості, потужності та підтримки спільноти, що робить їх ідеальним вибором для створення ефективної та надійної системи підтримки прийняття рішень при контрольних випробуваннях технічних систем. Використання цих технологій дозволяє розробникам швидко та ефективно реалізовувати функціональні можливості системи, забезпечуючи високу якість та надійність кінцевого продукту.

2.7 Висновки до другого розділу

Обраною архітектурою став веб-фреймворк Flask. Його перевагою є висока гнучкість та легкість у налаштувані. Також він дозволяє розробникам створювати додатки з мінімальними початковим налаштуванням, одночасно надаючи можливість додавання необхідних розширень та компонентів. Основна частина системи даних була створенна за допомогою бібліотеки SQLAlchemy.

РОЗДІЛ 3. РОЗРОБКА КОМП’ЮТЕРНОЇ СИСТЕМИ

3.1 Розробка бази даних

Розробка бази даних є критично важливим етапом у створенні комп'ютерної системи підтримки прийняття рішень при контрольних випробуваннях технічних систем. Вона визначає структуру зберігання даних, їхню організацію та взаємозв'язки, що забезпечує ефективну обробку та доступ до інформації. У даному проекті для розробки бази даних було обрано бібліотеку SQLAlchemy, яка є потужним інструментом для роботи з реляційними базами даних у середовищі Python. SQLAlchemy дозволяє визначати структуру бази даних у вигляді класів, що відповідають таблицям, та забезпечує зручний інтерфейс для виконання операцій над даними.

Нижче представлено код моделей, розроблених за допомогою SQLAlchemy, разом з детальними описами кожної з них. Кожна модель відповідає окремій таблиці в базі даних та включає атрибути, що відображають поля таблиці, а також взаємозв'язки між таблицями через зовнішні ключі.

```
# Моделі даних
class System(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)
    evaluations = db.relationship('Evaluation', backref='system', lazy=True)

    def __repr__(self):
        return f"<System {self.name}>"
```

Клас System представляє таблицю, що зберігає інформацію про технічні системи, які підлягають випробуванням. Він має три основні атрибути:

1. id: це поле є первинним ключем таблиці, що забезпечує унікальну ідентифікацію кожної системи. Воно має тип Integer і автоматично інкрементується при додаванні нових записів.

2. name: поле зберігає назву системи. Воно має тип String з максимальною довжиною 100 символів. Поле є обов'язковим для заповнення (nullable=False) та унікальним (unique=True), що запобігає дублюванню назв систем у базі даних.

3. evaluations: це відношення (relationship) до таблиці Evaluation, яке встановлює зв'язок "один до багатьох" між системою та її оцінками. Параметр backref='system' дозволяє зворотний доступ до системи з об'єкта оцінки, а параметр lazy=True вказує на те, що зв'язок буде завантажуватися "на вимогу", тобто лише коли до нього звертаються.

Метод __repr__ повертає строкове представлення об'єкта класу, що полегшує відлагодження та вивід інформації про об'єкти системи у консоль або журнали.

```
class Criterion(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)
    weight = db.Column(db.Float, nullable=False)  # Вес критерия

    def __repr__(self):
        return f"<Criterion {self.name}>"
```

Клас Criterion відповідає за таблицю, що зберігає інформацію про критерії оцінки технічних систем. Він включає три атрибути:

1. id: первинний ключ таблиці з типом Integer, що забезпечує унікальну ідентифікацію кожного критерію.

2. name: назва критерію з типом String та максимальною довжиною 100 символів. Це поле також є обов'язковим (nullable=False) та унікальним (unique=True), що запобігає дублюванню назв критеріїв.

3. weight: поле зберігає вагу критерію з типом Float. Воно є обов'язковим для заповнення (nullable=False) і визначає важливість цього критерію у загальній оцінці системи. Вага критерію використовується для обчислення агрегованого балу системи на основі її оцінок за різними критеріями.

Метод __repr__ забезпечує зрозуміле представлення об'єктів класу Criterion.

```
class Evaluation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    system_id = db.Column(db.Integer, db.ForeignKey('system.id'), nullable=False)
    criterion_id = db.Column(db.Integer, db.ForeignKey('criterion.id'), nullable=False)
    score = db.Column(db.Integer, nullable=False)

    criterion = db.relationship('Criterion')

    def __repr__(self):
        return f"<Evaluation System:{self.system_id} Criterion:{self.criterion_id} Score:{self.score}>"
```

Клас Evaluation представляє таблицю, що зберігає оцінки технічних систем за різними критеріями. Він включає наступні атрибути:

1. id: первинний ключ таблиці з типом Integer, що забезпечує унікальну ідентифікацію кожної оцінки.

2. system_id: зовнішній ключ (ForeignKey), що посилається на поле id таблиці System. Це поле є обов'язковим для заповнення (nullable=False) і встановлює зв'язок між оцінкою та відповідною системою.

3. criterion_id: зовнішній ключ (ForeignKey), що посилається на поле id таблиці Criterion. Це поле також є обов'язковим (nullable=False) і встановлює зв'язок між оцінкою та відповідним критерієм.

4. score: поле зберігає оцінку системи за певним критерієм з типом Integer. Воно є обов'язковим для заповнення (nullable=False) і представляє числове значення, яке надається за відповідним критерієм.

Відношення між оцінкою та критерієм реалізовано через атрибут criterion, що дозволяє зворотний доступ до критерію з об'єкта оцінки. Це забезпечує зручний спосіб отримання інформації про критерій, за яким проводиться оцінка.

Метод __repr__ повертає зрозуміле строкове представлення об'єкта оцінки, включаючи ідентифікатори системи та критерію, а також саму оцінку.

```
class SystemForm(FlaskForm):
    name = StringField('System Name', validators=[DataRequired()])
    submit = SubmitField('Add System')

class EvaluationForm(FlaskForm):
    system = SelectField('System', coerce=int, validators=[DataRequired()])
    criterion = SelectField('Criterion', coerce=int, validators=[DataRequired()])
    score = IntegerField('Score', validators=[DataRequired(), NumberRange(min=0, max=100)])
    submit = SubmitField('Add Evaluation')
```

Класи SystemForm та EvaluationForm реалізують форми для введення даних користувачами через веб-інтерфейс. Вони успадковуються від класу FlaskForm та включають відповідні поля з необхідними валідаторами для забезпечення коректності введених даних.

Клас SystemForm містить поле name, яке є обов'язковим для заповнення (validators=[DataRequired()]) і відповідає за введення назви нової системи. Також він включає кнопку submit для підтвердження додавання системи.

Клас EvaluationForm містить поля system та criterion, які є селекторами (SelectField) для вибору системи та критерію відповідно. Ці поля використовують параметр coerce=int, що перетворює вибране значення на ціле число, відповідно до типу даних у базі. Поле score є числовим полем (IntegerField) з валідаторами DataRequired() та NumberRange(min=0, max=100), що забезпечує введення числових значень у діапазоні від 0 до 100. Клас також включає кнопку submit для підтвердження додавання оцінки.

```
def init_criteria():
    criteria = [
        {'name': 'Performance', 'weight': 0.4},
        {'name': 'Reliability', 'weight': 0.3},
        {'name': 'Usability', 'weight': 0.2},
        {'name': 'Maintainability', 'weight': 0.1},
    ]
    for crit in criteria:
        existing = Criterion.query.filter_by(name=crit['name']).first()
        if not existing:
            new_criterion = Criterion(name=crit['name'], weight=crit['weight'])
            db.session.add(new_criterion)
    db.session.commit()
```

Функція init_criteria відповідає за ініціалізацію стандартних критеріїв оцінки у базі даних. Вона створює список словників, кожен з яких містить назву критерію та його вагу. Для кожного критерію перевіряється наявність у базі даних за допомогою запиту Criterion.query.filter_by(name=crit['name']).first(). Якщо критерій з такою назвою ще не існує, створюється новий об'єкт Criterion з відповідними атрибутами та додається до сесії бази даних. Після завершення циклу всі нові критерії зберігаються у базі даних за допомогою виклику db.session.commit().

Ця функція забезпечує наявність базового набору критеріїв для оцінки систем при першому запуску системи, що спрощує процес налаштування та використання системи новими користувачами.

```
with app.app_context():
    db.create_all()
    init_criteria()
```

Блок коду with app.app_context() забезпечує контекст додатку Flask, необхідний для виконання операцій з базою даних. Метод db.create_all() створює всі таблиці, визначені моделями, у базі даних, якщо вони ще не існують. Після цього викликається функція init_criteria() для ініціалізації стандартних критеріїв.

```
def calculate_total_score(system):
    evaluations = Evaluation.query.filter_by(system_id=system.id).all()
    if not evaluations:
        return None, "No evaluations available."

    total_score = 0
    total_weight = 0
```

```
    for eval in evaluations:
        total_score += eval.score * eval.criterion.weight
        total_weight += eval.criterion.weight

    if total_weight == 0:
        return None, "Total weight is zero."

    aggregate_score = total_score / total_weight
    decision = "Accept" if aggregate_score >= 70 else "Reject"
    return aggregate_score, decision
```

Функція calculate_total_score відповідає за обчислення загального балу системи на основі її оцінок за різними критеріями. Вона приймає об'єкт системи як аргумент та виконує наступні кроки:

1. Отримує всі оцінки, пов'язані з цією системою, за допомогою запиту Evaluation.query.filter_by(system_id=system.id).all().
2. Якщо оцінок немає, функція повертає None та повідомлення "No evaluations available."
3. Ініціалізує змінні total_score та total_weight для накопичення сумарного балу та ваги.
4. Проходить по всіх оцінках та додає до total_score добуток оцінки на вагу відповідного критерію. Вагу додає до total_weight.
5. Якщо загальна вага дорівнює нулю, функція повертає None та повідомлення "Total weight is zero."
6. Обчислює агрегований бал як співвідношення total_score до total_weight.

7. Приймає рішення про прийняття або відхилення системи на основі агрегованого балу: якщо бал більше або дорівнює 70, рішення "Accept", інакше "Reject".

8. Повертає агрегований бал та прийняте рішення.

Реалізовано три основні моделі даних: System, Criterion та Evaluation. Ці моделі відповідають за зберігання інформації про технічні системи, критерії оцінки та оцінки відповідно. Кожна модель має свої унікальні атрибути та взаємозв'язки, що дозволяє ефективно організувати та управляти даними у системі.

Модель System відповідає за зберігання інформації про технічні системи, які підлягають випробуванням. Основними атрибутами цієї моделі є id та name. Атрибут id є первинним ключем, що забезпечує унікальну ідентифікацію кожної системи, а атрибут name зберігає назву системи. Поле name є обов'язковим для заповнення та має унікальне обмеження, що запобігає дублюванню назв систем у базі даних. Відношення між системою та оцінками реалізовано через атрибут evaluations, який встановлює зв'язок "один до багатьох" між системою та її оцінками. Це дозволяє одній системі мати багато оцінок за різними критеріями.

Модель Criterion відповідає за зберігання інформації про критерії оцінки, за якими оцінюються технічні системи. Вона містить атрибути id, name та weight. Атрибут id є первинним ключем, а атрибут name зберігає назву критерію. Поле weight зберігає вагу критерію, яка визначає його важливість у загальній оцінці системи. Вага критерію є числовим значенням, що дозволяє об'єктивно оцінювати внесок кожного критерію у загальну оцінку системи. Відношення між критерієм та оцінками реалізовано через атрибут evaluations, що дозволяє одному критерію оцінювати багато систем.

Модель Evaluation є проміжною таблицею, яка реалізує зв'язок між системами та критеріями. Вона містить атрибути id, system_id, criterion_id та

score. Атрибут system_id є зовнішнім ключем, що посилається на поле id таблиці System, а атрибут criterion_id є зовнішнім ключем, що посилається на поле id таблиці Criterion. Атрибут score зберігає оцінку системи за певним критерієм. Таким чином, модель Evaluation дозволяє встановлювати зв'язок між системами та критеріями, зберігаючи оцінки, надані за кожним критерієм для конкретної системи.

Взаємозв'язки між моделями

Взаємозв'язки між моделями System, Criterion та Evaluation реалізовані через використання зовнішніх ключів та відношень relationship у sqlalchemy. Це забезпечує цілісність та узгодженість даних у базі даних, дозволяючи ефективно управляти зв'язками між різними таблицями.

Модель System має відношення "один до багатьох" з моделлю Evaluation, що дозволяє одній системі мати багато оцінок за різними критеріями. Це реалізовано через атрибут evaluations у класі System, який встановлює зв'язок з моделлю Evaluation через параметр backref='system'. Це означає, що з об'єкта оцінки можна отримати доступ до відповідної системи через атрибут system.

Модель Criterion також має відношення "один до багатьох" з моделлю Evaluation, що дозволяє одному критерію оцінювати багато систем. Це реалізовано через атрибут evaluations у класі Criterion, який встановлює зв'язок з моделлю Evaluation через параметр backref='criterion'. Таким чином, з об'єкта оцінки можна отримати доступ до відповідного критерію через атрибут criterion.

Модель Evaluation є проміжною таблицею, яка реалізує зв'язок між моделями System та Criterion. Вона містить зовнішні ключі system_id та criterion_id, що посилаються на відповідні таблиці. Це дозволяє зберігати оцінки систем за різними критеріями, забезпечуючи можливість виконання комплексних запитів та аналізу даних.

3.2 Розробка основного функціоналу

Розробка основного функціоналу системи підтримки прийняття рішень передбачає створення маршрутів та функцій, які забезпечують взаємодію користувачів із системою, обробку введених даних, взаємодію з базою даних та відображення результатів через веб-інтерфейс. Використовуючи Flask як веб-фреймворк та SQLAlchemy як ORM-бібліотеку, було реалізовано кілька ключових маршрутів, кожен з яких відповідає за певну функціональність системи. Нижче наведено детальний опис цих маршрутів та функцій разом із відповідним кодом.

Перш за все, головний маршрут додатку відповідає за відображення головної сторінки, де користувачі можуть переглядати список усіх систем з їхніми агрегованими балами та рішеннями щодо прийняття або відхилення. Цей маршрут використовує функцію calculate_total_score для обчислення загального балу кожної системи на основі її оцінок за різними критеріями. Результати передаються до шаблону index.html для відображення.

```
@app.route('/')
def index():
    systems = System.query.all()
    system_data = []
    for system in systems:
        aggregate_score, decision = calculate_total_score(system)
        system_data.append({
            'system': system,
            'aggregate_score': aggregate_score,
            'decision': decision
        })
```

```
    return render_template('index.html', system_data=system_data)
```

Функція index виконує наступні дії: вона отримує всі системи з бази даних за допомогою методу System.query.all(), а потім для кожної системи обчислює агрегований бал та прийняте рішення за допомогою функції calculate_total_score. Отримані дані зберігаються у списку system_data, який передається до шаблону для відображення на головній сторінці. Це дозволяє користувачам бачити загальну картину оцінок систем та прийнятих рішень.

Наступним маршрутом є /add_system, який відповідає за додавання нових систем до бази даних. Цей маршрут підтримує методи GET та POST, що дозволяє одночасно відображати форму для введення даних та обробляти подані дані.

```
@app.route('/add_system', methods=['GET', 'POST'])
def add_system():
    form = SystemForm()
    if form.validate_on_submit():
        existing = System.query.filter_by(name=form.name.data).first()
        if existing:
            flash('Система з такою назвою вже існує.', 'danger')
            return redirect(url_for('add_system'))
        new_system = System(name=form.name.data)
        db.session.add(new_system)
        db.session.commit()
        flash('Система успішно додана!', 'success')
        return redirect(url_for('index'))
    return render_template('add_system.html', form=form)
```

Функція add_system створює екземпляр форми SystemForm та перевіряє її валідність за допомогою методу validate_on_submit(). Якщо форма валідна, виконується перевірка наявності системи з такою ж назвою у базі даних. Якщо така система вже існує, користувач отримує повідомлення про помилку та перенаправляється на сторінку додавання системи. Якщо системи з такою назвою немає, створюється новий об'єкт System, який додається до сесії бази даних та зберігається за допомогою методу db.session.commit(). Після успішного додавання користувач отримує повідомлення про успіх та перенаправляється на головну сторінку.

Маршрут /add_evaluation відповідає за додавання оцінок до систем за певними критеріями. Цей маршрут також підтримує методи GET та POST і включає логіку для заповнення вибіркових полів формою на основі наявних систем та критеріїв у базі даних.

```
@app.route('/add_evaluation', methods=['GET', 'POST'])
def add_evaluation():
    form = EvaluationForm()
    form.system.choices = [(s.id, s.name) for s in System.query.order_by('name')]
    form.criterion.choices = [(c.id, c.name) for c in Criterion.query.order_by('name')]
    if form.validate_on_submit():
        existing_eval = Evaluation.query.filter_by(system_id=form.system.data, criterion_id=form.criterion.data).first()
        if existing_eval:
            flash('Оцінка для цієї системи та критерію вже існує.', 'danger')
            return redirect(url_for('add_evaluation'))
        new_evaluation = Evaluation(
            system_id=form.system.data,
```

```
            criterion_id=form.criterion.data,
            score=form.score.data
        )
        db.session.add(new_evaluation)
        db.session.commit()
        flash('Оцінка успішно додана!', 'success')
        return redirect(url_for('index'))
    return render_template('add_evaluation.html', form=form)
```

Функція add_evaluation спочатку заповнює вибіркові поля system та criterion на основі даних з таблиць System та Criterion. Це забезпечує актуальність вибору систем та критеріїв при додаванні оцінок. Після перевірки валідності форми здійснюється перевірка наявності оцінки для вибраної системи та критерію. Якщо така оцінка вже існує, користувач отримує повідомлення про помилку та перенаправляється на сторінку додавання оцінки. Якщо оцінка відсутня, створюється новий об'єкт Evaluation, який додається до бази даних та зберігається. Після успішного додавання користувач отримує повідомлення про успіх та перенаправляється на головну сторінку.

Маршрут /system/<int:system_id> відповідає за відображення детальної інформації про конкретну систему, включаючи всі її оцінки за різними критеріями та агрегований бал.

```
@app.route('/system/<int:system_id>')
def system_detail(system_id):
    system = System.query.get_or_404(system_id)
    evaluations = Evaluation.query.filter_by(system_id=system.id).all()
    aggregate_score, decision = calculate_total_score(system)
```

```
    return render_template('system_detail.html', system=system, evaluations=evaluations, aggregate_score=aggregate_score, decision=decision)
```

Функція system_detail використовує метод get_or_404 для отримання системи за заданим ідентифікатором system_id. Якщо система не знайдена, повертається сторінка з помилкою 404. Після цього функція отримує всі оцінки, пов'язані з цією системою, та обчислює агрегований бал та рішення за допомогою функції calculate_total_score. Отримані дані передаються до шаблону system_detail.html для відображення детальної інформації про систему.

Маршрут /about відповідає за відображення сторінки "Про систему", яка містить загальну інформацію про проект, його цілі та функціональні можливості.

```
@app.route('/about')
def about():
    return render_template('about.html')
```

Функція about просто повертає шаблон about.html, що містить статичну інформацію про систему. Цей маршрут не потребує взаємодії з базою даних або обробки даних, оскільки призначений для надання загальної інформації користувачам.

Функція calculate_total_score є основною для обчислення агрегованого балу системи на основі її оцінок за різними критеріями. Вона враховує ваги критеріїв, що забезпечує об'єктивну оцінку системи.

```
def calculate_total_score(system):
    evaluations = Evaluation.query.filter_by(system_id=system.id).all()
    if not evaluations:
        return None, "Немає оцінок."
```

```
total_score = 0
total_weight = 0
for eval in evaluations:
    total_score += eval.score * eval.criterion.weight
    total_weight += eval.criterion.weight

if total_weight == 0:
    return None, "Загальна вага дорівнює нулю."

aggregate_score = total_score / total_weight
decision = "Прийнято" if aggregate_score >= 70 else "Відхилено"
return aggregate_score, decision
```

Функція calculate_total_score виконує наступні кроки: вона отримує всі оцінки, пов'язані з конкретною системою, та перевіряє їх на наявність. Якщо оцінок немає, повертається повідомлення про відсутність оцінок. Далі функція обчислює сумарний бал як суму добутків оцінок на ваги відповідних критеріїв, а також сумарну вагу критеріїв. Якщо загальна вага дорівнює нулю, повертається повідомлення про некоректну конфігурацію ваг. В іншому випадку, агрегований бал обчислюється як співвідношення сумарного балу до сумарної ваги. На основі агрегованого балу приймається рішення: якщо бал більший або дорівнює 70, система приймається, інакше відхиляється. Функція повертає агрегований бал та прийняте рішення для подальшого використання у веб-інтерфейсі.

Всі маршрути та функції взаємодіють з базою даних через SQLAlchemy, що забезпечує ефективну роботу з даними та їхню цілісність. Важливою частиною розробки є забезпечення валідності введених даних, що досягається за допомогою використання валідаторів у формах. Це гарантує, що всі введені дані відповідають вимогам системи та не призводять до помилок при обробці.

Для забезпечення зручності користувачів та покращення взаємодії з системою були реалізовані механізми відображення повідомлень про успіх чи помилки за допомогою функції flash. Це дозволяє надавати зворотний зв'язок користувачам про результати їхніх дій, покращуючи загальний користувацький досвід.

Крім того, розробка маршрутових функцій включає використання перенаправлень через функцію redirect, що дозволяє направляти користувачів на відповідні сторінки після виконання певних дій, таких як додавання нової системи або оцінки. Це забезпечує послідовність користувацького досвіду та допомагає уникнути повторного виконання дій при оновленні сторінок.

Забезпечення безпеки даних та контролю доступу до функціональностей системи є критично важливим аспектом розробки. Це досягається за рахунок впровадження механізмів автентифікації та авторизації користувачів, що дозволяє обмежувати доступ до певних функцій системи відповідно до ролі користувача. Наприклад, адміністратори мають повний доступ до управління користувачами та налаштуваннями системи, тоді як звичайні користувачі можуть додавати та переглядати системи та оцінки, але не мають прав на зміну конфігурації системи.

У контексті розробки основного функціоналу також важливо забезпечити оптимізацію продуктивності системи. Використання SQLAlchemy як ORM-бібліотеки дозволяє ефективно управляти запитами до бази даних та мінімізувати час відповіді системи. Оптимізовані запити та ефективна структура бази даних дозволяють системі справлятися з великими обсягами даних без втрати продуктивності.

Взаємодія між різними компонентами системи організована таким чином, щоб забезпечити гнучкість та масштабованість. Наприклад, додавання нових маршрутів та функцій може бути здійснено без необхідності зміни існуючих

компонентів системи, що дозволяє легко розширювати функціональність системи відповідно до зростаючих потреб підприємства.

Забезпечення підтримки багатомовності також може бути інтегровано у систему за допомогою відповідних бібліотек та налаштувань Flask, що дозволяє надавати користувачам інтерфейс на різних мовах, покращуючи доступність системи для широкого кола користувачів.

Крім того, розробка основного функціоналу передбачає реалізацію механізмів тестування та валідації функцій, щоб гарантувати їхню коректну роботу та відповідність вимогам системи. Це включає модульне тестування окремих функцій, інтеграційне тестування взаємодії між компонентами системи та системне тестування для перевірки повної функціональності системи.

Важливим аспектом є також забезпечення масштабованості системи, що досягається за рахунок використання ефективних алгоритмів обробки даних та оптимізації структури бази даних. Це дозволяє системі підтримувати високу продуктивність навіть при збільшенні обсягу даних та кількості користувачів.

Нижче наведено додаткові маршрути та функції, які можуть бути реалізовані для покращення функціональності системи:

```
@app.route('/delete_system/<int:system_id>', methods=['POST'])
def delete_system(system_id):
    system = System.query.get_or_404(system_id)
    db.session.delete(system)
    db.session.commit()
    flash('Система успішно видалена!', 'success')
    return redirect(url_for('index'))
```

Функція delete_system відповідає за видалення системи з бази даних. Вона приймає параметр system_id, отримує відповідну систему за допомогою методу

get_or_404, видаляє її з бази даних та зберігає зміни. Після успішного видалення користувач отримує повідомлення про успіх та перенаправляється на головну сторінку.

```
@app.route('/edit_system/<int:system_id>', methods=['GET', 'POST'])
def edit_system(system_id):
    system = System.query.get_or_404(system_id)
    form = SystemForm(obj=system)
    if form.validate_on_submit():
        existing = System.query.filter_by(name=form.name.data).first()
        if existing and existing.id != system.id:
            flash('Система з такою назвою вже існує.', 'danger')
            return redirect(url_for('edit_system', system_id=system.id))
        system.name = form.name.data
        db.session.commit()
        flash('Система успішно оновлена!', 'success')
        return redirect(url_for('system_detail', system_id=system.id))
    return render_template('edit_system.html', form=form, system=system)
```

Функція edit_system відповідає за редагування існуючої системи. Вона приймає параметр system_id, отримує відповідну систему з бази даних, створює екземпляр форми SystemForm з даними системи, та перевіряє валідність введених даних. Якщо форма валідна, здійснюється перевірка наявності системи з такою ж назвою, крім поточної системи. Якщо така система існує, користувач отримує повідомлення про помилку. В іншому випадку, оновлюються дані системи та зберігаються зміни у базі даних. Після успішного оновлення користувач перенаправляється на сторінку деталей системи з повідомленням про успіх.

```
@app.route('/delete_evaluation/<int:evaluation_id>', methods=['POST'])
def delete_evaluation(evaluation_id):
    evaluation = Evaluation.query.get_or_404(evaluation_id)
    db.session.delete(evaluation)
    db.session.commit()
    flash('Оцінка успішно видалена!', 'success')
    return redirect(url_for('system_detail', system_id=evaluation.system_id))
```

Функція delete_evaluation відповідає за видалення оцінки з бази даних. Вона приймає параметр evaluation_id, отримує відповідну оцінку за допомогою методу get_or_404, видаляє її з бази даних та зберігає зміни. Після успішного видалення користувач перенаправляється на сторінку деталей системи, до якої належить ця оцінка, з повідомленням про успіх.

```
@app.route('/edit_evaluation/<int:evaluation_id>', methods=['GET', 'POST'])
def edit_evaluation(evaluation_id):
    evaluation = Evaluation.query.get_or_404(evaluation_id)
    form = EvaluationForm(obj=evaluation)
    form.system.choices = [(s.id, s.name) for s in System.query.order_by('name')]
    form.criterion.choices = [(c.id, c.name) for c in Criterion.query.order_by('name')]
    if form.validate_on_submit():
        existing_eval = Evaluation.query.filter_by(system_id=form.system.data, criterion_id=form.criterion.data).first()
        if existing_eval and existing_eval.id != evaluation.id:
            flash('Оцінка для цієї системи та критерію вже існує.', 'danger')
            return redirect(url_for('edit_evaluation', evaluation_id=evaluation.id))
        evaluation.system_id = form.system.data
```

```
            evaluation.criterion_id = form.criterion.data
            evaluation.score = form.score.data
            db.session.commit()
            flash('Оцінка успішно оновлена!', 'success')
            return redirect(url_for('system_detail', system_id=evaluation.system_id))
        return render_template('edit_evaluation.html', form=form, evaluation=evaluation)
```

Функція edit_evaluation відповідає за редагування існуючої оцінки. Вона приймає параметр evaluation_id, отримує відповідну оцінку з бази даних, створює екземпляр форми EvaluationForm з даними оцінки, а також заповнює вибіркові поля систем та критеріїв. Після перевірки валідності форми здійснюється перевірка наявності оцінки для вибраної системи та критерію, крім поточної оцінки. Якщо така оцінка існує, користувач отримує повідомлення про помилку. В іншому випадку, оновлюються дані оцінки та зберігаються зміни у базі даних. Після успішного оновлення користувач перенаправляється на сторінку деталей системи з повідомленням про успіх.

Всі маршрути та функції розроблені таким чином, щоб забезпечити інтуїтивно зрозумілий та ефективний користувацький досвід. Використання шаблонів для відображення даних дозволяє створювати чистий та організований інтерфейс, що полегшує навігацію та взаємодію користувачів із системою. Крім того, забезпечення валідності даних та обробки помилок сприяє підвищенню надійності системи та уникненню потенційних помилок при введенні даних.

Розробка основного функціоналу системи підтримки прийняття рішень включає створення та налаштування маршрутів та функцій, які забезпечують взаємодію користувачів із системою, обробку введених даних, взаємодію з базою даних та відображення результатів через веб-інтерфейс. Використання Flask як веб-фреймворку та SQLAlchemy як ORM-бібліотеки дозволяє створювати

ефективні, гнучкі та масштабовані веб-додатки, що відповідають сучасним вимогам до систем підтримки прийняття рішень. Це сприяє підвищенню продуктивності та надійності системи, забезпечуючи її здатність адаптуватися до змінних потреб підприємства та забезпечувати високу якість прийнятих рішень на основі актуальних та точних даних.

Функціональність системи підтримує необхідні бізнес-процеси, пов'язані з оцінкою технічних систем, забезпечуючи точність та об'єктивність прийнятих рішень. Це досягається завдяки автоматизованим алгоритмам обчислення агрегованих балів, гнучкості у налаштуванні критеріїв оцінки та їх ваг, а також ефективній взаємодії між компонентами системи. Всі ці аспекти сприяють створенню надійної та ефективної системи підтримки прийняття рішень, яка відповідає сучасним вимогам до управління даними та аналітики.

Окрім цього, особлива увага приділяється забезпеченню надійності та безпеки роботи системи, що є критично важливим для застосування в різних галузях, таких як енергетика, транспорт, виробництво та інші сфери, де технічні системи відіграють ключову роль.

3.3 Оцінка параметрів для обранної технічної системи

Обраними технічними системами для перевірки системи стали дві відеокарти – GeForce RTX 4090 та GeForce RTX 3080.

Таблиця 3.1

Параметри обох технічних систем [38]

	RTX 4090	RTX 3080
Продуктивність з плаваючою точкою	82.58 TFLOPS	29.77 TFLOPS
Тактова частота	2235 МГц	1440 Мгц

Пропускна здатність	1010 GB/s	760 Gb/s
Об’єм відеопам’яті	24 Гб	10 Гб
Енергоспоживання	450 Вт	320 Вт
Ціна	1599$	699$
Довжина	304 мм	285 мм

Ваги та оцінка критеріїв була виключно суб’єктивною, операючись на дані обоих систем та загальні дані по іншим відеокартам.

Таблиця 3.2

Критерії, ваги та оцінка технічних систем

Критерії	Ваги	RTX 4090	RTX 3080
Продуктивність	0.4	100	80
Енергоспоживання	0.3	30	40
Вартість	0.6	40	80
Зручність використання	0.2	60	70

3.4 Тестування комп’ютерної системи

Тестування системи підтримки прийняття рішень є невід'ємною частиною процесу розробки, оскільки забезпечує перевірку коректності роботи функціональних компонентів, виявлення та усунення помилок, а також підтвердження відповідності системи встановленим вимогам. Основною метою тестування було забезпечення стабільної та надійної роботи системи, а також гарантія того, що всі функціональні можливості працюють згідно з очікуваннями користувачів та технічними специфікаціями.

При запускі програма генерує HTML сторінку (Рисунок 3.1) з даними про системи та кнопками для переходу на інші сторінки. Також на головній сторінці можна порівняти дві системи за допомогою кнопки «Compare Systems». Порівнюватися вони будуть лише по спільним критеріям та, якщо спільних критеріїв у систем нема, про це повідомить система.

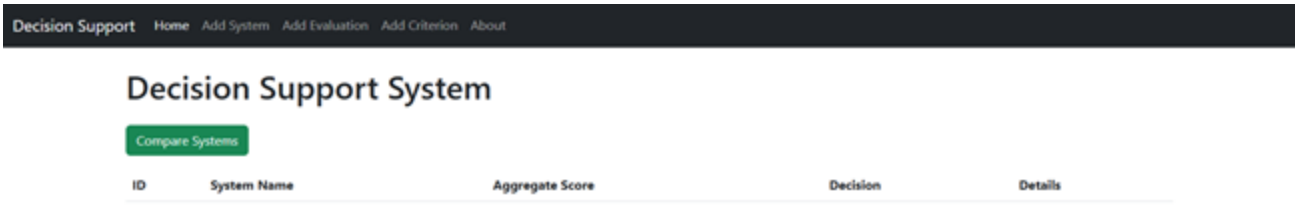

Рисунок 3.1 – головна сторінка

Для початку тестування систем спочатку потрібно їх додати. Це можна зробити за допомогою кнопки «Add System» у шапці сторінки. При потрапляні на сторінку (Рисунок 3.2), користувачу потрібно ввести назву системи для її збереження у базу даних. Після додавання системи користувач автоматично перейде до головної сторінки, на якій буде додана система з обранною назвою.

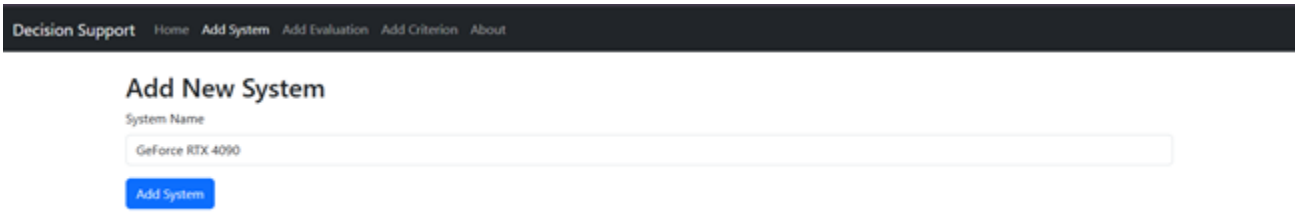

Рисунок 3.2 – додавання системи

Також користувач повинен додати критерії для своєї системи. Система, по замовчуванню, пропонує такі критерії як Maintainability, Perfomance, Reability та Usability з вже виставленими вагами, але, в данному випадку, потрібно створити інші критерії. Для цього користувачу потрібно обрати «Add Criterion» у шапці сторінки. На сторінці (Рисунок 3.3) користувачу потрібно ввести назву та вагу

критерію у відсотках. Для прикладу буде створенний критерій «Price» для наступного тестування технічних систем.

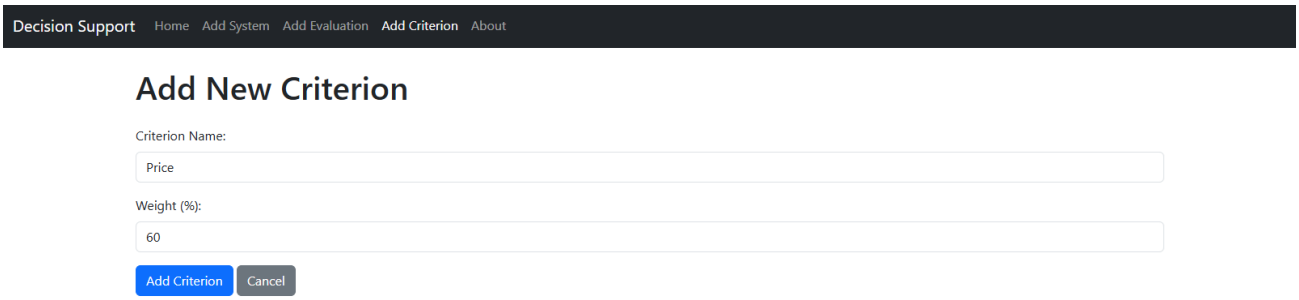

Рисунок 3.3 – додавання нового критерію

Після додавання критеріїв, користувачу потрібно додати їх до систем. Також, з критерієм, додається оцінка цього критерію.

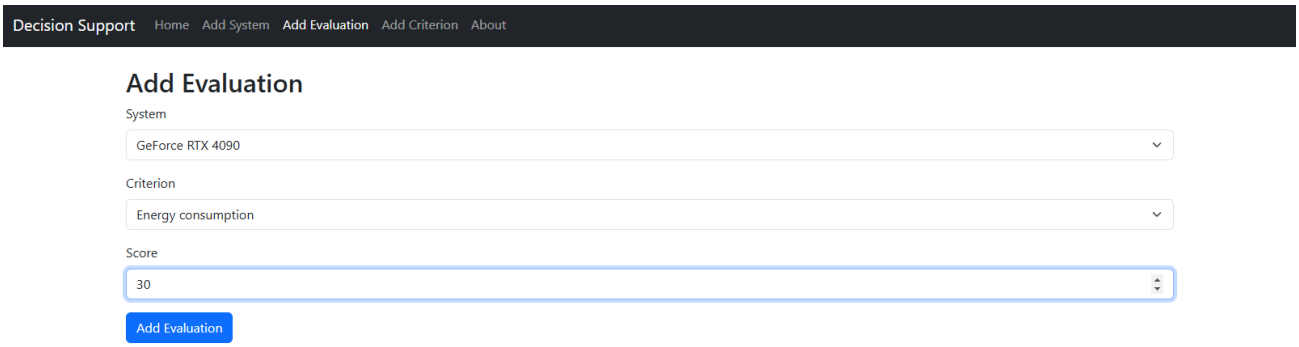

Рисунок 3.4 – додавання оцінки критерію

Після додавання усіх критеріїв, на головній сторінці, система показує наступні результати:

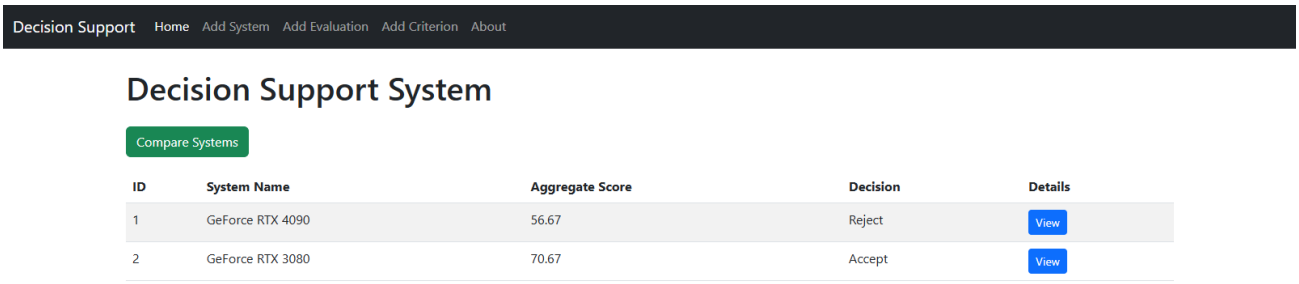

Рисунок 3.5 – результати

За результатами розробленної комп’ютерної системи, по заданим даним, технічну систему GeForce RTX 4090 було відхилено. Також користувач може порівняти дві системи, використавши кнопку «Compare Systems» та обравши інтересуючи системи для порівняння (Рисунок 3.6):

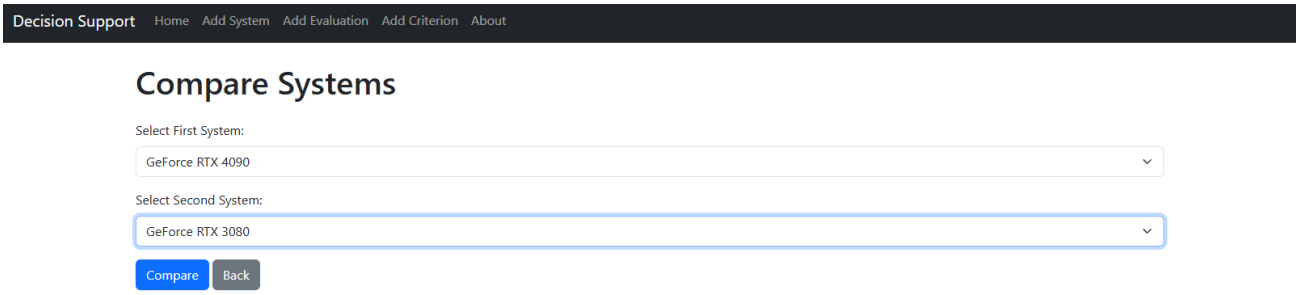

Рисунок 3.6 – вікно порівняння систем

При натисненні на «Compare» система створює сторінку з таблицею для порівняння двох технічних систем за критеріями (Рисунок 3.7).

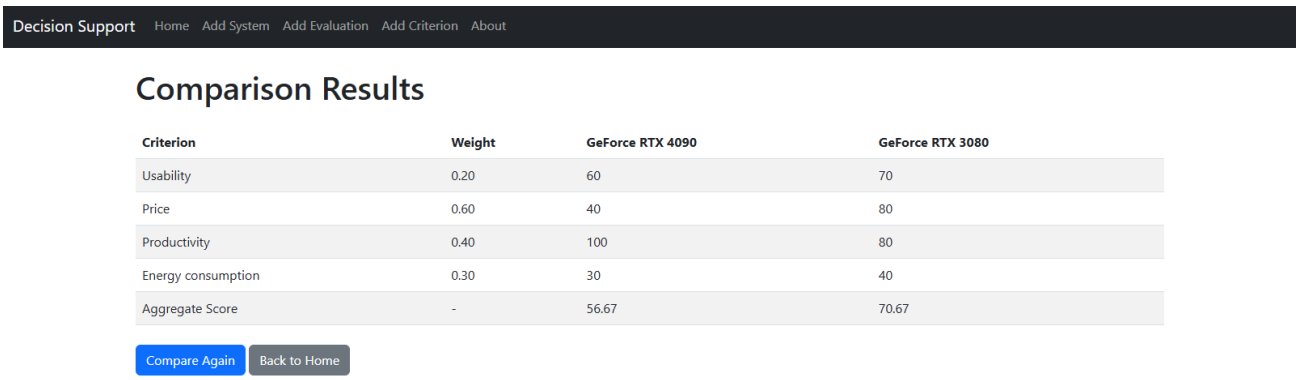

Рисунок 3.7 – порівняння двох систем

Також, з головної сторінки за допомогою кнопки «View» (Рисунок 3.5) користувач може отримати таблицю з оцінкою критеріїв, вагами та взаженими оцінками по вибранній системі (Рисунок 3.8).

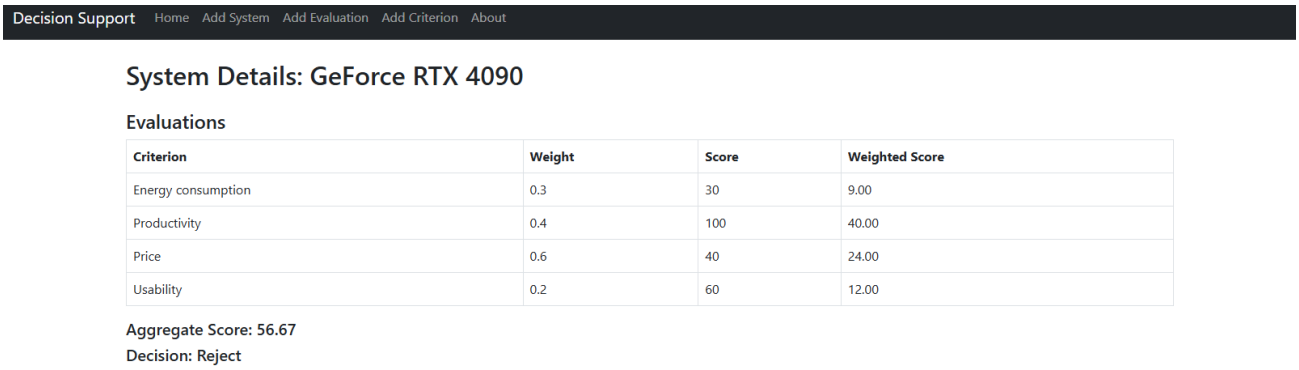

Рисунок 3.8 – деталі по системі

3.4 Висновки до третього розділу

Проведене тестування підтверджує відповідність системи встановленим вимогам та забезпечує впевненість у її здатності ефективно підтримувати процеси прийняття рішень при контрольних випробуваннях технічних систем. Завдяки ретельному тестуванню було виявлено та усунуто всі можливі помилки.

Розроблена система підтримки прийняття рішень відповідає сучасним вимогам ринку та бізнесу, сприяє підвищенню конкурентоспроможності підприємства, надаючи інструменти для оперативного та точного прийняття стратегічних рішень. Це забезпечує не лише розвиток організації, але й створює довготривалі конкурентні переваги у динамічному середовищі сучасного ринку.

ВИСНОВКИ

У результаті проведеного дослідження була розроблена та впроваджена система підтримки прийняття рішень для контрольних випробувань технічних систем, яка відповідає сучасним вимогам щодо автоматизації та оптимізації процесів оцінки. Використання мови програмування Python, веб-фреймворку Flask та ORM-бібліотеки SQLAlchemy дозволило створити гнучку, масштабовану та надійну систему, яка забезпечує ефективну взаємодію з базою даних, обробку великих обсягів інформації та зручний інтерфейс для користувачів.

Розроблені моделі даних, включаючи класи System, Criterion та Evaluation, забезпечують логічну та структуровану організацію інформації про технічні системи, критерії оцінки та їхні оцінки відповідно. Впровадження механізмів валідації даних через форми дозволяє гарантувати коректність введених користувачами даних, що є важливим аспектом для забезпечення точності та надійності системи. Алгоритми обчислення агрегованих балів, реалізовані у функції calculate_total_score, забезпечують об'єктивність та точність прийняття рішень на основі зібраних даних.

Практичне застосування розробленої системи дозволяє підприємствам ефективно управляти процесами контрольних випробувань технічних систем, знижуючи ризики пов'язані з людським фактором та підвищуючи загальну продуктивність. Система підтримує швидке та обґрунтоване прийняття рішень, забезпечуючи високу точність оцінок та оптимізацію ресурсів підприємства.

Отже, проведене дослідження успішно досягло поставлених цілей, створивши ефективну та надійну систему підтримки прийняття рішень для контрольних випробувань технічних систем. Розроблена система має високу практичну цінність та може бути адаптована до різних галузей промисловості, сприяючи підвищенню якості та надійності технічних систем, що підлягають випробуванням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Keen, P. Decision support systems : a research perspective / P. Keen. – Cambridge, Massachusetts : Center for Information Systems Research, Alfred P. Sloan School of Management, 1980. – hdl:1721.1/47172.
2. Sprague, R. A Framework for the Development of Decision Support Systems / R. Sprague // MIS Quarterly. – 1980. – Vol. 4, No. 4. – P. 1–25.
3. Keen, P. G. W. Decision support systems: an organizational perspective / P. G. W. Keen. – Reading, Mass. : Addison-Wesley Pub. Co., 1978. – 234 p. – ISBN 0-201-03667-3.
4. Turban, E. Decision Support Systems and Intelligent Systems / E. Turban, J. E. Aronson, T.-P. Liang. – 8th ed. – 2008. – 574 p.
5. Wright, A. A framework and model for evaluating clinical decision support architectures q / A. Wright, D. Sittig // Journal of Biomedical Informatics. – 2008. – Vol. 41, No. 6. – P. 982–990. – doi:10.1016/j.jbi.2008.03.009.
6. Zhang, S. X. An evolutionary real options framework for the design and management of projects and systems with complex real options and exercising conditions / S. X. Zhang, V. Babovic // Decision Support Systems. – 2011. – Vol. 51, No. 1. – P. 119–129. – doi:10.1016/j.dss.2010.12.001.
7. Official Home of the DSSAT Cropping Systems Model [Електронний ресурс] // DSSAT.net. – Режим доступу: https://dssat.net.
8. Stephens, W. Why has the uptake of Decision Support Systems been so poor? / W. Stephens, T. Middleton // Crop-soil simulation models in developing countries / Eds. R. B. Matthews, W. Stephens. – Wallingford : CABI, 2002. – P. 129–148.
9. Community of Practice Forest Management Decision Support Systems [Електронний ресурс]. – Режим доступу: http://www.forestdss.org/ – Дата звернення: 28.11.2024.

10. Salvaneschi, P. Applying AI to structural safety monitoring and evaluation / P. Salvaneschi, M. Cadei, M. Lazzari // IEEE Expert. – 1996. – Vol. 11, No. 4. – P. 24–34. – doi:10.1109/64.511774.
11. Masera, A. Integrated approach to dam safety / A. Masera et al. // Comitê Brasileiro de Barragens [Електронний ресурс]. – Режим доступу: https://www.cbdb.org.br/. – Дата звернення: 01.12.2024.
12. Lancini, S. Diagnosing Ancient Monuments with Expert Software / S. Lancini, M. Lazzari, A. Masera, P. Salvaneschi // Structural Engineering International. – 1997. – Vol. 7, No. 4. – P. 288–291. – doi:10.2749/101686697780494392.
13. Lazzari, M. Embedding a Geographic Information System in a Decision Support System for Landslide Hazard Monitoring / M. Lazzari, P. Salvaneschi // Natural Hazards. – 1999. – Vol. 20, No. 2–3. – P. 185–195. – doi:10.1023/A:1008187024768.
14. Peerspot. SAP BusinessObjects Business Intelligence Platform Overview [Електронний ресурс]. – Режим доступу: https://www.peerspot.com. – Дата звернення: 02.12.2024.
15. Peerspot. IBM Cognos Analytics vs SAP BusinessObjects [Електронний ресурс]. – Режим доступу: https://www.peerspot.com. – Дата звернення: 04.12.2024.
16. Saasworthy. Power BI vs Tableau Comparison [Електронний ресурс]. – Режим доступу: https://www.saasworthy.com. – Дата звернення: 06.12.2024.
17. Sourceforge. IBM Cognos Analytics vs Tableau [Електронний ресурс]. – Режим доступу: https://sourceforge.net. – Дата звернення: 08.12.2024.
18. Revopsteam. Best Business Intelligence Platforms [Електронний ресурс]. – Режим доступу: https://revopsteam.com. – Дата звернення: 10.12.2024.
19. Google AI Platform Overview [Електронний ресурс]. – Режим доступу: https://cloud.google.com/ai-platform/. – Дата звернення: 14.12.2024.

20. Amazon SageMaker Documentation [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/sagemaker/. – Дата звернення: 16.12.2024.
21. Tableau Software. Overview and Features [Електронний ресурс]. – Режим доступу: https://www.tableau.com. – Дата звернення: 18.12.2024.
22. G2. Tableau vs QlikView Comparison [Електронний ресурс]. – Режим доступу: https://www.g2.com. – Дата звернення: 20.12.2024.
23. Saasworthy. QlikView Features and Limitations [Електронний ресурс]. – Режим доступу: https://www.saasworthy.com. – Дата звернення: 28.12.2024.
24. Amazon SageMaker Documentation [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/sagemaker/. – Дата звернення: 29.12.2024.
25. Flask Documentation. Flask Overview [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com. – Дата звернення: 4.01.2025.
26. Real Python. Flask and MVT Pattern [Електронний ресурс]. – Режим доступу: https://realpython.com. – Дата звернення: 01.01.2025.
27. GeeksforGeeks. Flask Framework Features [Електронний ресурс]. – Режим доступу: https://geeksforgeeks.org. – Дата звернення: 02.01.2025.
28. Medium. Why Choose Flask for Web Development [Електронний ресурс]. – Режим доступу: https://medium.com. – Дата звернення: 04.01.2025.
29. Microsoft Learn. Flask Integration with Databases [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com. – Дата звернення: 06.01.2025.
30. Towards Data Science. Flask for Scalable Web Applications [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com. – Дата звернення: 07.01.2025.
31. TechTarget. Data Analysis in Flask Applications [Електронний ресурс]. – Режим доступу: https://techtarget.com. – Дата звернення: 12.01.2025.
32. DigitalOcean. Flask Security Features [Електронний ресурс]. – Режим доступу: https://digitalocean.com. – Дата звернення: 07.01.2025.

33. Flask Security Features Overview [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com. – Дата звернення: 07.01.2025.
34. Towards Data Science. Optimizing Performance in Flask Applications [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com. – Дата звернення: 07.01.2025.
35. Real Python. Using Templates in Flask [Електронний ресурс]. – Режим доступу: https://realpython.com. – Дата звернення: 08.01.2025.
36. GeeksforGeeks. Scalability with Flask Framework [Електронний ресурс]. – Режим доступу: https://geeksforgeeks.org. – Дата звернення: 09.01.2025.
37. Nvidia Graphics Cards [Електронний ресурс], - Режим доступу: https://www.nvidia.com/en-eu/geforce/graphics-cards/. – Дата звернення: 10.01.2025

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
from flask import Flask, render_template, request, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, IntegerField, SelectField
from wtforms.validators import DataRequired, NumberRange

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key_here'  # Замените на свой секретный ключ
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)

# Модели данных
class System(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)
    evaluations = db.relationship('Evaluation', backref='system', lazy=True)

    def __repr__(self):
        return f"<System {self.name}>"

class Criterion(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)
```

```
    weight = db.Column(db.Float, nullable=False)  # Вес критерия

    def __repr__(self):
        return f"<Criterion {self.name}>"

class Evaluation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    system_id = db.Column(db.Integer, db.ForeignKey('system.id'), nullable=False)
    criterion_id = db.Column(db.Integer, db.ForeignKey('criterion.id'), nullable=False)
    score = db.Column(db.Integer, nullable=False)

    criterion = db.relationship('Criterion')

    def __repr__(self):
        return f"<Evaluation System:{self.system_id} Criterion:{self.criterion_id} Score:{self.score}>"

# Формы
class SystemForm(FlaskForm):
    name = StringField('System Name', validators=[DataRequired()])
    submit = SubmitField('Add System')

class EvaluationForm(FlaskForm):
    system = SelectField('System', coerce=int, validators=[DataRequired()])
    criterion = SelectField('Criterion', coerce=int, validators=[DataRequired()])
```

```
    score = IntegerField('Score', validators=[DataRequired(), NumberRange(min=0, max=100)])
    submit = SubmitField('Add Evaluation')

# Инициализация критериев (можно настроить по необходимости)
def init_criteria():
    criteria = [
        {'name': 'Performance', 'weight': 0.4},
        {'name': 'Reliability', 'weight': 0.3},
        {'name': 'Usability', 'weight': 0.2},
        {'name': 'Maintainability', 'weight': 0.1},
    ]
    for crit in criteria:
        existing = Criterion.query.filter_by(name=crit['name']).first()
        if not existing:
            new_criterion = Criterion(name=crit['name'], weight=crit['weight'])
            db.session.add(new_criterion)
    db.session.commit()

with app.app_context():
    db.create_all()
    init_criteria()

# Вспомогательная функция для расчета итогового балла
def calculate_total_score(system):
    evaluations = Evaluation.query.filter_by(system_id=system.id).all()
    if not evaluations:
        return None, "No evaluations available."
```

```
    total_score = 0
    total_weight = 0
    for eval in evaluations:
        total_score += eval.score * eval.criterion.weight
        total_weight += eval.criterion.weight

    if total_weight == 0:
        return None, "Total weight is zero."

    aggregate_score = total_score / total_weight
    decision = "Accept" if aggregate_score >= 70 else "Reject"
    return aggregate_score, decision

# Маршруты
@app.route('/')
def index():
    systems = System.query.all()
    system_data = []
    for system in systems:
        aggregate_score, decision = calculate_total_score(system)
        system_data.append({
            'system': system,
            'aggregate_score': aggregate_score,
            'decision': decision
        })
    return render_template('index.html', system_data=system_data)
```

```
@app.route('/add_system', methods=['GET', 'POST'])
def add_system():
    form = SystemForm()
    if form.validate_on_submit():
        existing = System.query.filter_by(name=form.name.data).first()
        if existing:
            flash('System with this name already exists.', 'danger')
            return redirect(url_for('add_system'))
        new_system = System(name=form.name.data)
        db.session.add(new_system)
        db.session.commit()
        flash('System added successfully!', 'success')
        return redirect(url_for('index'))
    return render_template('add_system.html', form=form)

@app.route('/add_evaluation', methods=['GET', 'POST'])
def add_evaluation():
    form = EvaluationForm()
    form.system.choices = [(s.id, s.name) for s in System.query.order_by('name')]
    form.criterion.choices = [(c.id, c.name) for c in Criterion.query.order_by('name')]
    if form.validate_on_submit():
        # Проверка, что оценка по данному критерию для системы еще не существует
        existing_eval = Evaluation.query.filter_by(system_id=form.system.data, criterion_id=form.criterion.data).first()
        if existing_eval:
            flash('Evaluation for this criterion and system already exists.', 'danger')
```

```
            return redirect(url_for('add_evaluation'))
        new_evaluation = Evaluation(
            system_id=form.system.data,
            criterion_id=form.criterion.data,
            score=form.score.data
        )
        db.session.add(new_evaluation)
        db.session.commit()
        flash('Evaluation added successfully!', 'success')
        return redirect(url_for('index'))
    return render_template('add_evaluation.html', form=form)

@app.route('/system/<int:system_id>')
def system_detail(system_id):
    system = System.query.get_or_404(system_id)
    evaluations = Evaluation.query.filter_by(system_id=system.id).all()
    aggregate_score, decision = calculate_total_score(system)
    return render_template('system_detail.html', system=system,
evaluations=evaluations, aggregate_score=aggregate_score, decision=decision)

@app.route('/about')
def about():
    return render_template('about.html')

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