Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка сервісно-орієнтованого програмного забезпечення для оцінювання та планування програмних проєктів малого і середнього масштабу»

Виконав: студент IV курсу, групи СПс-41
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Кікцьо Т.Ю.
(підпис) (прізвище та ініціали)

Керівник _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Стадник М.А.
(підпис) (прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«_____» _____ 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Кікцьо Тарасу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка сервісно-орієнтованого програмного забезпечення для оцінювання та планування програмних проєктів малого і середнього масштабу»

Керівник роботи Цуприк Галина Богданівн, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «6» квітня 2026 року № 4/9-172
2. Термін подання студентом завершеної роботи «22» червня 2026 р.
3. Вихідні дані до роботи: Завдання на розробку програмного забезпечення
4. Зміст роботи (перелік питань, які потрібно розробити)
Обрати напрям дослідження та об'єкт, сформулювати, погодити та затвердити тему кваліфікаційної роботи; розробити та затвердити завдання;
проаналізувати завдання, зробити огляд та проаналізувати бібліографічні матеріали щодо стану справ та тенденій в обраному напрямку дослідження;
сформулювати завдання, обґрунтувати цілі дослідження;
розробити та спроєктувати;
описати технології, етапи розробки та сам продукт,
розробити план тестування програмного продукту; апробувати роботу,
оформити допоміжну документацію;
проаналізувати роботу щодо питань з дотримання положень про безпеку життєдіяльності та основи охорони праці; оформити пояснювальну записку; зробити відповідні висновки за результатами виконаної роботи, представити роботу на розгляд ЕК.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративна частина кваліфікаційної роботи оформляється у вигляді слайдів, висвітлює основні розділи та етапи роботи та містить:
тему роботи, мету, предмет та об'єкт дослідження,
сформульовані завдання, етапи виконання кваліфікаційної роботи,
короткий аналіз актуальності теми, проєктування та розробка,

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці			
Нормоконтроль	Стоянов Ю.М.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та погодження теми з керівником.	23.03-05.04.26	виконано
2.	Затвердження теми КР наказом ректора Створення та затвердження завдання	06.04.2026	виконано
3.	Аналіз технічного завдання, підбір та аналіз	20.04.2026	виконано
	бібліографічних матеріалів необхідних для виконання		
	кваліфікаційної роботи		
4.	Проєктування та розробка програмного продукту	11.05.2026	виконано
5.	Тестування та дослідна експлуатація програмного	18.05.2026	виконано
	забезпечення		
6.	Створення допоміжної документації. Написання та	25.05.2026	виконано
	перевірка на відповідність вимогам пояснювальної		
	записки		
7.	Формування записки кваліфікаційної роботи	28.05.2026	виконано
8.	Перевірка програмою антиплагіат	.06.2026	виконано
9.	Безпека життєдіяльності, основи охорони праці	.06.2026	виконано
10.	Нормоконтроль	.06.2026	виконано
11.	Попередній захист	09.06.2026	виконано
12.	Захист кваліфікаційної роботи	.06.2026	

Студент _____ Кікцьо Т.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка сервісно-орієнтованого програмного забезпечення для оцінювання та планування програмних проєктів малого і середнього масштабу // Кваліфікаційна робота освітнього рівня «Бакалавр» // Тарас Юрійович Кікцьо // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп`ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-41 // Тернопіль, 2026 // С. ___, рис. – ___, табл. – ___, додат. – ___, бібліогр. – ___.

Ключові слова: програмний продукт; підтримка прийняття рішень; оцінювання параметрів проєкту; аналіз ризиків; сценарії реалізації.

Кваліфікаційна робота присвячена розробці сервісно-орієнтованого програмного рішення для оцінювання, аналізу і планування програмних проєктів.

В першому розділі проведено аналіз вимог до програмного продукту.

В другому розділі описано основні етапи проєктування та розробки програмного рішення.

В третьому розглянуто основні аспекти тестування, розгортання, системні вимоги та верифікацію програмного рішення для підтримки оцінювання проєктів.

В четвертому розділі висвітлено важливість безпеки життєдіяльності та основ з охорони праці.

Об’єкт дослідження: комплексна задача оцінювання, аналізу та планування програмних проєктів малого та середнього масштабу на етапі підготовки.

Предмет дослідження: сервісно-орієнтоване програмне рішення для порівняльного аналізу готових програмних продуктів або окремих показників, на етапі їх підготовки з урахуванням ризиків, ресурсів та альтернативних сценаріїв реалізації.

ABSTRACT

Development of service-oriented software for evaluating and planning small and medium-scale software projects // Qualification work of the educational level «Bachelor» // Taras Kiktso // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Software Engineering Department, group SPs-41 // Ternopil, 2026 // P. ___, fig. – ___, tabl. – ___, annexes. – ___, references – ___.

Keywords: software product; decision support; project parameter assessment; risk analysis; implementation scenarios.

The qualification work to the study of the development of a service-oriented software solution for the evaluation, analysis and planning of software projects.

The first chapter analyzes the requirements for the software product.

The second chapter describes the main stages of designing and developing a software solution.

The third chapter considers the main aspects of testing, deployment, system requirements and verification of a software solution to support project evaluation.

The fourth chapter of importance of life safety and the basics of occupational health and safety.

Object of research: a complex task of evaluating, analyzing and planning small and medium-scale software projects at the preparation stage.

Subject of the study: a service-oriented software solution for comparative analysis of finished software products or individual indicators at the preparation stage, taking into account risks, resources and alternative implementation scenarios.

ЗМІСТ

ВСТУП

Актуальність теми зумовлена тим, що перш ніж почати, будь-яку роботу потрібно розуміти її перспективність, мати об’єктивну та неупереджену інформацію на яку можна опиратись при прийнятті оптимального рішення, через обґрунтований вибір. Такий підхід є правильним та бажаним для будь-якого напрямку діяльності, в тому числі і з інженерії програмного забезпечення, при роботі з програмними проєктами. Адже це дає змогу як визначити та «побачити» можливі ризики, проблеми та виклики так і оцінити потенційний результат і чи він відповідає очікуванням, а також чи виправдана затратна частина. Враховуючи результати техніко-економічного обґрунтування, приймаючий рішення має можливість чітко визначити відповідність проєкту цілям та вимогам замовника, наперед виявити та оцінити потенційні перешкоди та їх критичність та скласти план дій на випадок можливих непередбачуваних ситуацій. Таким чином такий системний підхід може з високою ймовірністю гарантувати доцільність початку роботи в принципі, а після її початку ефективно розприділяти та контролювати ресурси, а проєкт, що реалізується за таким підходом має високі шанси на успіх та реалізовність.

Метою кваліфікаційної роботи є проєктування та розробка сервісно-орієнтованого програмного рішення для підтримки оцінювання, аналізу і планування програмних проєктів малого та середнього масштабу, яке забезпечуватиме автоматизацію процесів оцінювання та планування на етапі планування (підготовки проєкту до реалізації), а також дозволить аналізувати та порівнювати альтернативні сценарії його реалізації з метою підтримки прийняття обґрунтованих оптимальних рішень. А це, в свою чергу, дасть можливість зацікавленим сторонам зрозуміти як потенційну рентабельність інвестцій так і прибутковість проєкту та реальні терміни повернення коштів при його потенційній успішній реалізації.

Відповідно до мети я окреслив наступні задачі, а саме:

1 - проаналізувати предметну область в напрямку оцінювання та планування сучасних програмних проєктів.

2 - дослідити існуючі підходи, методи та моделі оцінювання параметрів програмних проєктів.

3 - визначити функціональні та нефункціональні вимоги до розроблюваної програмного рішення.

4 - спроєктувати архітектуру сервісно-орієнтованої програмного продукту.

5 - розробити логічну та фізичну моделі бази даних.

6 - побудувати UML-діаграми (варіантів використання, класів, діяльності, станів).

7 - реалізувати основні сервіси та бізнес-логіку програми.

8 - провести тестування та верифікацію розробки.

9 - виконати порівняльний аналіз запропонованих моделей з існуючими методами оцінювання.

Об’єктом дослідження є програмні проєкти, процес їх оцінювання та планування з метою визначення на відповідність цілям та вимогам замовника та попереднього виявлення та оцінки потенційних ризиків та їх критичність, можливих непередбачуваних ситуацій, для отримання можливості відповідальній особі приймати оптимальні та ефективні рішення щодо вибору альтернативних варіантів, сценаріїв чи, навіть, доцільності реалізації проєкту.

Предметом дослідження є методи, моделі та сервісно-орієнтовані програмні засоби підтримки прийняття оптимальних рішень при оцінюванні, аналізі та плануванні програмних проєктів, які забезпечують автоматизований та якісний розрахунок основних параметрів проєкта, а також можливість аналізу та порівняння альтернативних варіантів та сценаріїв з метою вибору оптимального.

На відміну від існуючих рішень, які орієнтовані здебільшого на порівняння готових програмних продуктів, у моїй роботі основну увагу я приділив аналізу та оцінюванню параметрів сучасного програмного проєкту ще на етапі його планування. Особливістю розроблюваного програмного рішення є підтримка порівняння альтернативних сценаріїв реалізації проєкту та використання сервісно-орієнтованої архітектури, що забезпечує гнучкість і можливість

інтеграції з іншими інструментами життєвого циклу програмного забезпечення. Також в моєму програмному рішенні я враховую невизначеність та ризики,і це, в свою чергу, дозволить формувати альтернативні сценарії реалізації проєкту.

Відповідно до вимог зазначених в Методичних вказівках призначених для ознайомлення здобувачів вищої освіти з вимогами, правилами оформлення та оцінювання випускних кваліфікаційних робіт бакалавра для здобувачів, які навчаються за освітньо-професійною програмою «Інженерія програмного забезпечення» однойменної спеціальності, авторства відповідального за випуск Петрика М.Р., д.ф.-м.н., професор; Михалика Д.М., к.т.н., доцент; Цуприк Г.Б., к.т.н., доцент, Бревуса В.М., PhD передбачено апробацію результатів кваліфікаційної роботи на конференції де будуть доповідатись основні результати проведеного дослідження з наступним включенням тез до збірника матеріалів роботи конференції та їх публікацією.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО СЕРВІСНО-ОРІЄНТОВАНОГО ПРОДУКТУ ПЛАНУВАННЯ ПРОЄКТНОЇ ДІЯЛЬНОСТІ

Розділ присвячений аналізу вимог до програмного продукту, що є основою для його подальшого проєктування та розробки. Спочатку я проаналізую предметну область, окреслю ключові проблеми та очікування майбутніх користувачів [1,2]. Окрему увагу буде приділено формуванню цілей і чіткому визначенню завдання, яке повинна виконувати система. Крім того, визначусь з основними учасниками взаємодії та сценаріями використання, які будуть відображати взаємодію системи з користувачами. Результати цього розділу я використаю як основу для наступних проєктних рішень.

1.1 Аналіз предметної області

В процесі розробки програмного забезпечення вважаю, що потрібен комплексний підхід не лише до самої розробки, а й до оцінювання та планування проєктів, і це є одними з ключових процесів у сучасній інженерії програмного забезпечення. Особливо це важливо для малих і середніх команд, де часто є обмеження по ресурсам та часу необхідних для реалізації проєкту, крім цього ключовим є оцінити ризики та сформувати реалістичний бюджет, що дасть можливість оперативно обирати оптимальні варіанти етапів виконання проєкту [3-7].

На практиці процес оцінювання досить часто базується на досвіді керівників або експертів. Звичацйно, що такий підхід може бути швидким, але він, на мою думку, є досить суб’єктивний і не завжди дає точні результати, в силу своєї специфіки. Планування без чіткої структури задач і залежностей між ними ускладнює контроль за виконанням та підвищує ризик перевитрат, затримок, чи зривів проєктів.

Сьогодні, на ринку сьогодні існує декілька категорій інструментів для оцінювання та планування проєктів:

1. Комерційні системи керування проєктами (такі як: FlexProject, Microsoft Project, Jira, Wrike). Їх перевагами є широкий функціонал для планування задач, контроль за ресурсами та термінами, інтеграція з інструментами керування командою. До недоліків я б відніс складність в налаштуванні для саме для невеликих команд, обмежену автоматизація оцінювання, високу вартість ліцензії [8-11].

2. Відкриті або безкоштовні інструменти (для прикладу: OpenProject, Taiga). Їхніми перевагами є доступність, базові функції планування та можливість відстежування задач та інтеграція з системами контролю версій.

До недоліків, на мою думку варта віднести відсутність комплексної автоматизації оцінювання, обмежені можливості для порівняння альтернативних сценаріїв, складнощі з розширенням функціоналу під специфічні потреби [12,13].

3. Спеціалізовані системи оцінювання і планування (наприклад, калькулятори типу COCOMO II, PlanView, Primavera). Вони цікаві тим, що в результаті користувач може отримати точний розрахунок трудомісткості, підтримують різні моделі оцінювання, передбачено можливість аналіза ризиків. Але характеризуються складністю до інтеграції, недостатньою гнучкістю для малих та середніх команд, обмеженою підтримкою альтернативних сценаріїв, не завжди передбачають веб-інтерфейс [14-16].

4. Сервісно-орієнтовані та інтегровані підходи (маються на увазі наукові прототипи та корпоративні рішення). До їх переваг цих інструментів я б відніс автоматизацію процесів, можливість масштабування, інтеграцію з іншими сервісами та базами даних. Проте вони часто є експериментальними та потребують спеціальних ,достатньо специфічних навиків для впровадження [17-19].

Таким чином відповідно до проведеного аналізу предметної області я зробив висновок про те, що існуючі системи є або занадто складними для малих і середніх команд, або не підтримують комплексної автоматизації оцінювання та планування, а саме більшість з інструментів не підтримують порівняння

альтернативних сценаріїв і не інтегровані у сервісно-орієнтовану архітектуру, що обмежує гнучкість і масштабованість.

Отже, актуальним є створення сервісно-орієнтованої програмної системи, яка б поєднувала:

1 - автоматизоване оцінювання ключових параметрів проєкту (трудомісткість, ресурси, терміни, бюджет);

2 – планування і порівняння альтернативних сценаріїв;

3 – формування рекомендацій для прийняття обґрунтованих управлінських рішень;

4 – інтеграцію з інструментами життєвого циклу програмного забезпечення;

5 - модульну архітектуру для підключення нових сервісів та компонентів.

На відміну від існуючих систем управління проєктами, запропоноване рішення орієнтоване не лише на організацію виконання завдань, а передусім на етап попереднього оцінювання та планування. Особливістю системи є підтримка порівняння альтернативних сценаріїв реалізації проєкту та використання сервісно-орієнтованої архітектури, що забезпечує гнучкість і можливість інтеграції з іншими інструментами життєвого циклу програмного забезпечення.

Розробка такої системи відповідатиме сучасним вимогам інженерії програмного забезпечення та моїй спеціальності (121 Інженерія програмного забезпечення) та є актуальною саме для малих та середніх команд, забезпечуючи поєднання архітектурних рішень, сервісно-орієнтованого підходу та інноваційних інструментів підтримки життєвого циклу програмного забезпечення.

1.2 Постановка завдання та цілей програмної розробки

Метою моєї кваліфікаційної роботи є проєктування та розробка сервісно-орієнтованої програмної системи для оцінювання та планування програмних проєктів малого та середнього масштабу, яка дозволяє автоматизувати процеси визначення ключових параметрів проєкту, аналізувати альтернативні сценарії

виконання задач та формувати обґрунтовані рекомендації для прийняття управлінських рішень [2-7].

Для досягнення поставленої мети я запланував вирішити наступні завдання:

1 - Аналіз предметної області (досліджу існуючі підходи до оцінювання та планування програмних проєктів, виявлю переваги та недоліки сучасних рішеннях);

2 - Дослідження методів і моделей оцінювання (планую вивчити математичні та експертні методи оцінювання, наприклад, такі як COCOMO II, Use Case Points, а також маю намір оцінити їх на можливість застосування з метою автоматизації процесу) [14];

3 - Визначення вимог до системи (сформулюю функціональні та нефункціональні вимоги до програмної системи, визначу користувачів, їх ролі та рівні доступу);

4 - Проєктування архітектури системи (створю модульну та сервісно-орієнтовану структури, що забезпечують масштабованість і інтеграцію нових компонентів);

5 - Розробка бази даних та моделей данних (побудую логічну та фізичну моделі для збереження інформації про проєкти, завдання та ресурси);

6 - Побудова UML-діаграм (створю діаграми варіантів використання, класів, станів і діяльності для опису поведінки системи);

7 - Реалізація сервісів та бізнес-логіки (розроблю програмні модулі для оцінювання, планування та аналізу альтернативних сценаріїв);

8 - Тестування та верифікація системи (перевірю коректність роботи всіх функцій, точність розрахунків та відповідність вимогам);

9 - Порівняльний аналіз запропонованих рішень (порівняю реалізовані моделі з існуючими методами оцінювання та планування проєктів).

В результаті вирішення запланованих завдань я планую отримати функціонуючу сервісно-орієнтовану систему для оцінювання та планування проєктів в якій буде передбачено автоматизований розрахунок ключових параметрів проєкту, таких як трудомісткість, ресурси, терміни, бюджет та з

можливістю порівняння альтернативних сценаріїв планування та рекомендації щодо оптимального вибору. Систему можна буде інтегрувати з іншими інструментами життєвого циклу програмного забезпечення. Також обґрунтую оптимальність прийнятих в процесі розробки рішень з UML-діаграмами та приклади тестування, що демонструватимуть роботу системи.

В разі успішної реалізації розроблюваної системи я зможу отримати підвищення ефективності управління проєктами, зменшити вплив суб'єктивних факторі (наприклад, так званого людського фактору) та забезпечити більш обґрунтоване прийняття рішень у малих і середніх командах розробників.

1.3 Пошук акторів та варіантів використання

Метою підрозділу є визначення ключових акторів системи та основних функцій, які вони виконують. Це дозволяє побудувати основу для подальшого опису сценаріїв використання та побудови UML-діаграм. Визначення акторів також допомагає чітко окреслити межі системи та взаємодію між користувачами та модулями [20-22].

Для визначення функціональних можливостей програмної системи я провів аналіз користувачів, які взаємодіятимуть із нею, а також їх цілей і типових сценаріїв роботи. При цьому для мене важливим було враховувати, щоб розроблювана система орієнтована не на безпосереднє управління виконанням завдань, а на етап попереднього оцінювання, планування та обґрунтування рішень щодо реалізації програмного проєкту.

Такий підхід дозволить розглядати розроблювану систему як інструмент підтримки прийняття рішень, що відрізняє її від традиційних систем управління проєктами. Відповідно, коло користувачів формується не лише навколо виконання задач, а передусім навколо аналізу даних і вибору оптимальних варіантів реалізації проєкту.

У результаті проведеного аналізу було визначено основних акторів системи, а саме:

Таблиця 1.1 – Основні актори проєктованого програмного рішення

Актор	Загальна роль
«КЕРІВНИК ПРОЄКТУ»	– є основним користувачем системи, заємодіє з більшістю сценаріїв (оціночні та планувальні дії);
«АНАЛІТИК»	– забезпечує підготовку вхідних даних для оцінювання, оцінка ризиків, аналіз варіантів;
«ЗАМОВНИК ПРОЄКТУ»	– зацікавлений у результатах оцінювання та планування, перегляд результатів та звітів;
«АДМІНІСТРАТОР»	– відповідає за технічне функціонування системи, управління користувачами і правами доступу;
«МОДУЛЬ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ»	– реалізує функції обробки даних, автоматичне виконання розрахунків та формування рекомендацій для підтримки прийняття рішень щодо реалізації програмного проєкту, порівняння варіантів.

«Керівник проєкту» – ключова роль, оскільки саме він ініціює створення проєкту, обирає підходи до оцінювання та приймає остаточні рішення щодо доцільності його реалізації. У процесі роботи «Керівник проєкту» формує альтернативні сценарії виконання, аналізує результати розрахунків і на їх основі обирає найбільш ефективний варіант.

Важливу допоміжну роль виконує «Аналітик». Він відповідає за точність вхідних даних та їх повноту, здійснює оцінку ризиків і бере участь у формуванні можливих варіантів реалізації проєкту. Якість роботи аналітика безпосередньо впливає на достовірність результатів, отриманих у системі.

Окремим учасником взаємодії є «Замовник проєкту». Він не приймає жодної участі у розрахунках, проте аналізує отримані результати, порівнює альтернативні варіанти та може впливати на прийняття остаточного рішення щодо запуску проєкту.

Підтримку роботи системи забезпечує «Адміністратор». До його обов’язків належать управління обліковими записами користувачів, налаштування прав доступу, а також забезпечення надійності збереження даних і стабільності роботи системи в цілому.

Окрім користувачів, важливу роль у функціонуванні системи відіграє «Модуль підтримки прийняття рішень», який є внутрішнім компонентом системи.

Саме він виконує обробку вхідних даних, здійснює розрахунок основних параметрів проєкту, оцінює ризики та формує альтернативні сценарії реалізації. Крім того, цей модуль забезпечує порівняння отриманих варіантів і формує рекомендації, які використовуються керівником проєкту для прийняття рішень.

Таким чином, взаємодія між визначеними акторами та аналітичним модулем формує основу функціонування системи та визначає її ключові можливості.

На основі визначених акторів сформовано перелік ключових варіантів використання, які відображають основні сценарії взаємодії із системою. До них належать:

- створення та налаштування проєкту;
- введення та редагування параметрів проєкту;
- вибір методу оцінювання;
- автоматизований розрахунок параметрів проєкту;
- оцінювання ризиків та їх впливу на проєкт;
- формування альтернативних сценаріїв реалізації проєкту;
- моделювання варіантів розподілу ресурсів;
- аналіз впливу змін параметрів на результати оцінювання;
- порівняння варіантів виконання проєкту;
- визначення критичних факторів, що впливають на успішність проєкту;
- формування рекомендацій щодо вибору оптимального сценарію;
- прийняття рішення щодо реалізації проєкту;
- формування та перегляд звітів;
- збереження та використання історичних даних про проєкти;
- управління користувачами та доступом до системи.

Далі я визначився, дещо скорегував та розписав сценарії взаємодії із системою.

Таблиця 1.2 – Актори та відповідний кожному перелік ключових варіантів використання, які відображають основні сценарії взаємодії із системою.

Актор	Сценарії взаємодії із системою
«КЕРІВНИК ПРОЄКТУ»	1-Формує новий проєкт. 2-Вибирає методи оцінювання (COCOMO, експертна оцінка тощо). 3-Проводить оцінку трудомісткості та ресурсів. 4-Планує графік виконання завдань (PERT / CPM). 5-Аналізує альтернативні сценарії виконання проєкту. 6-Формує звіти та отримує рекомендації від системи.
«АНАЛІТИК» / Аналітичний модуль	1-Підготовка та перевірка вхідних даних. 2-Оцінка ризиків і ключових показників проєкту. 3-Формування рекомендацій для керівника проєкту. 4-Виконання розширеного аналізу сценаріїв (“Що якщо” аналіз).
«МОДУЛЬ ПРИЙНЯТТЯ РІШЕНЬ»	Внутрішній аналітичний компонент (модуль підтримки прийняття рішень) не є зовнішнім актором, а є внутрішнім компонентом системи. Модуль підтримки прийняття рішень на етапі аналізу розглядається як логічна сутність, а на етапі проєктування розглядатиметься як внутрішня компонента системи. 1-Автоматизує аналіз даних проєкту. 2-Генерує рекомендації для оптимізації плану. 3-Порівнює альтернативні варіанти виконання та пропонує оптимальні.
«АДМІНІСТРАТОР СИСТЕМИ»	1-Додавання та видалення користувачів, призначення ролей. 2-Підтримка сервісів і бази даних. 3-Резервне копіювання та відновлення.
«ЗАМОВНИК ПРОЄКТУ»	4-Перегляд результатів оцінки та планування. 5-Ознайомлення зі звітами та рекомендаціями. 6-Можливість коментувати результати (за потреби).

На основі отриманого я побудував UML-діаграму варіантів використання системи управління проєктами.

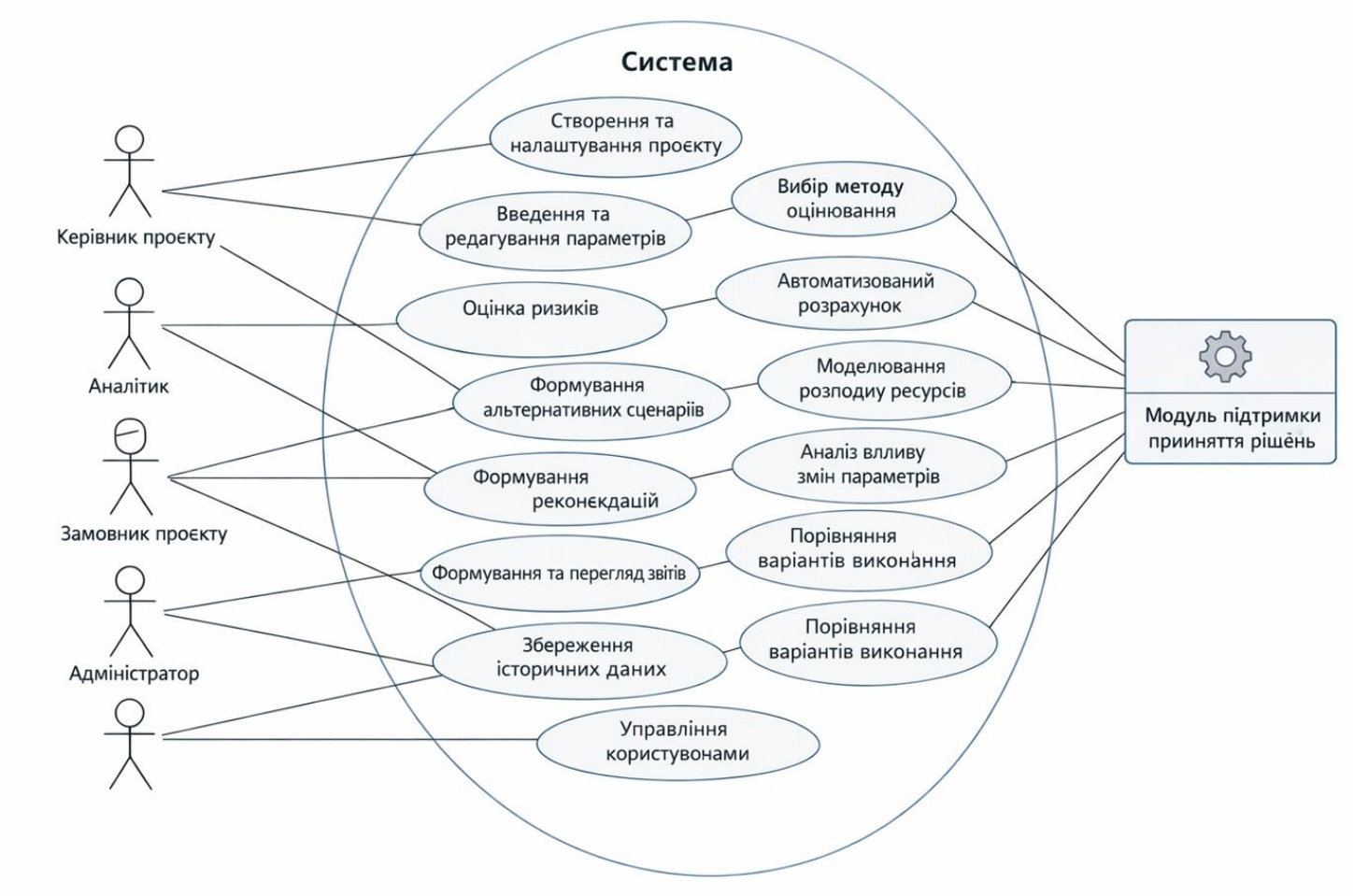

Рис. 1.1 – UML-діаграма варіантів використання системи управління проєктами [23]

Таким чином, я розробляю систему не призначену для управління процесом виконання завдань, а орієнтовану на аналітичну підтримку прийняття рішень на етапі планування проєкту, що вирізняє її від існуючих систем управління проєктами та інструментів оцінювання.

1.4 Опис ключових варіантів використання

Після визначення основних акторів та варіантів використання вважаю за доцільне більш детально розглянути ключові сценарії взаємодії користувачів із

системою. На мою думку, це дозволить краще зрозуміти логіку її функціонування та послідовність виконання основних операцій.

З огляду на специфіку розроблюваної системи, особливу увагу в приділю сценаріям, які пов’язані із оцінюванням параметрів проєкту, аналізом ризиків та підтримкою прийняття рішень.

Таблиця 1.3 – Створення та налаштування проєкту

Актор:	«Керівник проєкта»
Опис:	Сценарій передбачає створення нового проєкту та визначення його основних характеристик для подальшого оцінювання.
Передумови:	Користувач авторизований у системі; Користувач має відповідні права доступу.
Результат:	Користувач авторизований у системі; Користувач має відповідні права доступу.
Основний сценарій:	1. Користувач обирає функцію створення проєкту. 2. Вводить назву та опис проєкту. 3. Задає основні параметри (обсяг робіт, ресурси, обмеження). 4. Обирає метод оцінювання. 5. Підтверджує створення проєкту. 6. Система зберігає дані.
Альтернативні сценарії:	2а. Введено некоректні або неповні дані: система повідомляє про помилку. 4а. Метод оцінювання не обрано: система пропонує значення за замовчуванням.

Таблиця 1.4 – Оцінювання параметрів проєкту

Актор:	«Керівник проєкту», «Аналітик», «Модуль підтримки прийняття рішень»
Опис:	На цьому етапі система виконує розрахунок ключових характеристик проєкту на основі введених даних та обраної моделі оцінювання.
Передумови:	Проєкт створено; Всі необхідні параметри задані.
Результат:	Сформовано кількісну оцінку основних параметрів проєкту (трудомісткість, терміни виконання, ресурси), яка може бути використана для подальшого аналізу та планування.
Основний сценарій:	1. Користувач ініціює процес оцінювання. 2. Система перевіряє коректність вхідних даних. 3. Модуль підтримки прийняття рішень виконує розрахунки. 4. Формуються результати оцінювання. 5. Результати відображаються користувачу.

Продовження таблиці 1.4

Альтернативні сценарії:	2а. Дані неповні: система повідомляє про необхідність їх доповнення. 3а. Виникає помилка обчислення: система повідомляє користувача або повторює розрахунок.

Таблиця 1.5 – Оцінка ризиків та аналіз сценаріїв

Актор:	«Аналітик», «Керівник проєкта», «Модуль підтримки прийняття рішень»
Опис:	Сценарій спрямований на виявлення потенційних ризиків та формування альтернативних варіантів реалізації проєкту.
Передумови:	Виконано базове оцінювання проєкту; Доступні дані для аналізу ризиків.
Результат:	Визначено ключові ризики проєкту та сформовано альтернативні сценарії його реалізації з урахуванням можливих впливів.
Основний сценарій:	1. Аналітик вводить або уточнює інформацію про ризики. 2. Система оцінює їх вплив на параметри проєкту. 3. Формуються альтернативні сценарії реалізації. 4. Виконується моделювання розподілу ресурсів. 5. Користувач аналізує отримані варіанти.
Альтернативні сценарії:	1а. Дані про ризики відсутні: система використовує типові шаблони ризиків. 3а. Недостатньо даних для сценаріїв: система повідомляє користувача.

Таблиця 1.6 – Формування рекомендацій та прийняття рішення

Актор:	«Керівник проєкту», «Замовник», «Модуль підтримки прийняття рішень»
Опис:	Тут система аналізує всі сформовані сценарії та пропонує оптимальний варіант реалізації проєкту.
Передумови:	Сформовано альтернативні сценарії; Виконано оцінювання параметрів і ризиків.
Результат:	Сформовано обґрунтовані рекомендації щодо вибору оптимального варіанту реалізації проєкту або доцільності його виконання.
Основний сценарій:	1. Система порівнює альтернативні сценарії. 2. Визначає найбільш ефективний варіант. 3. Формує рекомендації. 4. Керівник проєкту аналізує результати. 5. Приймається рішення щодо реалізації проєкту. 36. амовник ознайомлюється з результатами.
Альтернативні сценарії:	2а. Відсутній явний оптимальний варіант: система пропонує декілька альтернатив. 5а. Рішення не прийнято: можливе повторне оцінювання або уточнення даних.

Таблиця 1.7 – Формування та перегляд звітів

Актор:	«Керівник проєкту», «Замовник»
Опис:	Сценарій передбачає отримання результатів у зручному для аналізу вигляді звітів і для подальшого їх використання.
Передумови:	Виконано оцінювання та аналіз проєкту.
Результат:	Сформовано структуровані звіти з результатами оцінювання та аналізу проєкту, доступні для перегляду та подальшого використання.
Основний сценарій:	1. Користувач обирає функцію формування звіту. 2. Система генерує звіт на основі наявних даних. 3. Відображає результати у зручному вигляді. 4. Користувач переглядає або експортує звіт.
Альтернативні сценарії:	2а. Дані відсутні: система повідомляє про неможливість формування звіту. 4а. Помилка експорту: система пропонує інший формат.

Описаними варіантами використання я продемонстрував логіку функціонування системи та взаємодію між її основними компонентами. Особливістю розроблюваної системи є наявність модуля підтримки прийняття рішень, який забезпечує не лише обробку даних, а й формування рекомендацій, що дозволяє підвищити обґрунтованість управлінських рішень ще на етапі планування проєкту.

Далі побудую діаграму прийняття рішень (Activity Diagram)

Діаграма діяльності відображатиме послідовність виконання основних етапів оцінювання програмного проєкту та прийняття рішення щодо доцільності його реалізації. Процес розпочинається з введення параметрів проєкту та перевірки їх повноти. У разі коректності даних виконується розрахунок основних характеристик проєкту, оцінка ризиків та формування альтернативних сценаріїв.

На слідуючому етапі реалізується порівняння сформованих варіантів та визначення найбільше ефективного з них. На основі отриманих результатів системою формуються рекомендації для прийняття рішення. У випадку недостатності даних або ж відсутності оптимального варіанта я передбачив можливість для уточнення параметрів та повторного аналізу.

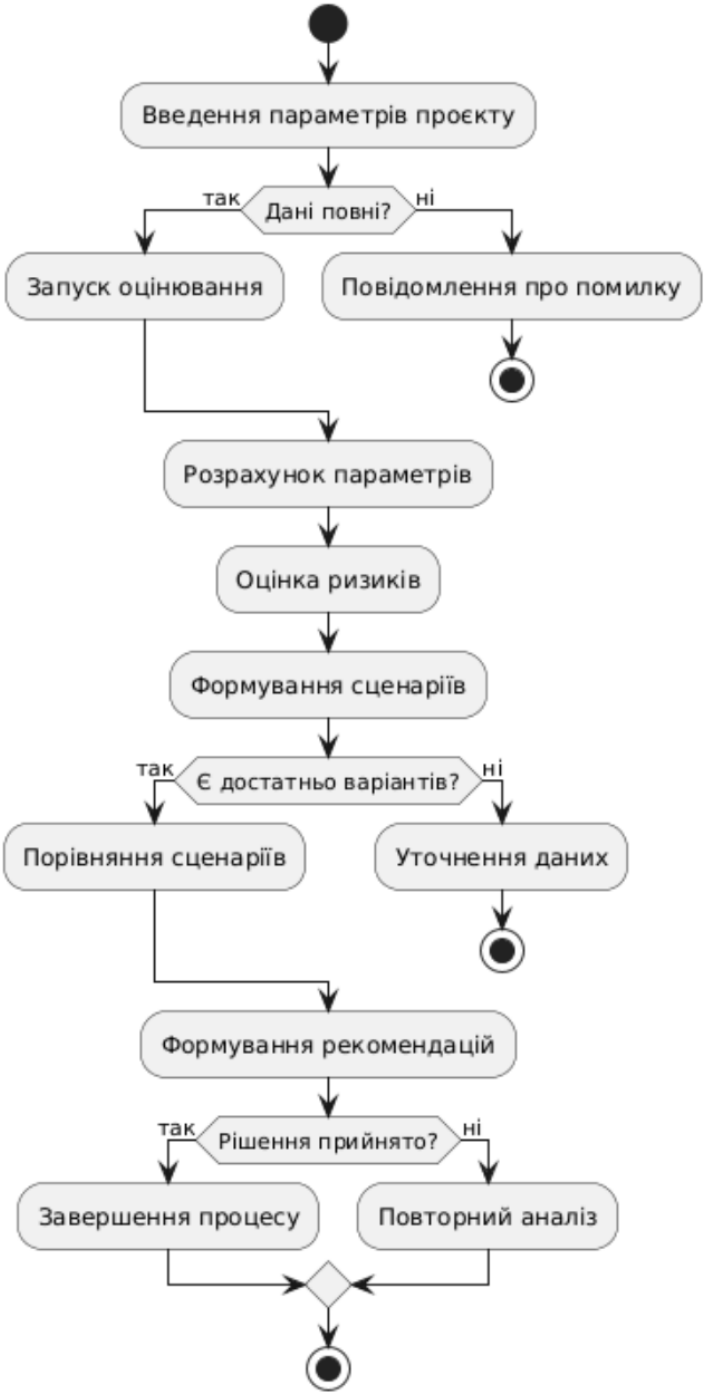

Рис. 1.2 – Activity Diagram - діаграма діяльності процесу оцінювання та прийняття рішення щодо реалізації програмного проєкту

Таким чином, представлена діаграма демонструє логіку роботи системи як інструменту для підтримки прийняття оптимальних і найбільш ефектьивних рішень, що дасть змогу забезпечити обґрунтованість вибору варіанту реалізації програмного проєкту.

1.5 Функціональні та нефункціональні вимоги до програмного рішення

На основі проведеного аналізу предметної області, визначених акторів та варіантів використання я сформував вимоги до розроблюваної мною програми, якими визначаються її функціональні можливості, а також обмеження та характеристики, яким система повинна відповідати в процесі розробки та експлуатації.

Функціональні вимоги визначають перелік основних можливостей системи, що забезпечують виконання задач оцінювання та планування програмних проєктів.

До основних функціональних вимог належать:

1- Система повинна забезпечувати створення, редагування та видалення програмних проєктів.

2 - Система повинна підтримувати введення, збереження та обробку параметрів проєкту, необхідних для оцінювання.

3 - Система повинна надавати можливість вибору методів оцінювання параметрів проєкту.

4 - Система повинна виконувати автоматизований розрахунок основних характеристик проєкту (трудомісткість, терміни виконання, ресурси).

5 - Система повинна забезпечувати оцінювання ризиків та визначення їх впливу на параметри проєкту.

6 - Система повинна формувати альтернативні сценарії реалізації проєкту з урахуванням заданих параметрів та обмежень.

7 - Система повинна забезпечувати моделювання варіантів розподілу ресурсів.

8 - Система повинна надавати можливість порівняння альтернативних варіантів реалізації проєкту.

9 - Система повинна формувати рекомендації щодо вибору оптимального варіанту реалізації проєкту.

10 - Система повинна забезпечувати підтримку прийняття рішень щодо доцільності реалізації проєкту.

11 - Система повинна формувати та відображати звіти за результатами оцінювання та аналізу.

12 - Система повинна зберігати історичні дані про проєкти та забезпечувати доступ до них.

13 - Система повинна підтримувати управління користувачами та розмежування прав доступу.

14 - Система повинна забезпечувати можливість інтеграції з іншими програмними системами (за потребою).

Нефункціональні вимоги визначають якісні характеристики системи, що забезпечують її ефективність, надійність та зручність використання.

1 - Вимоги до архітектури: система повинна бути реалізована на основі сервісно-орієнтованої архітектури (SOA); система повинна підтримувати модульну структуру для забезпечення масштабованості та розширюваності.

2 - Вимоги до продуктивності: система повинна забезпечувати обробку даних у прийнятний час без значних затримок; система повинна підтримувати одночасну роботу декількох користувачів;

3 - Вимоги до зручності використання: інтерфейс користувача повинен бути інтуїтивно зрозумілим; система повинна забезпечувати зручну навігацію та доступ до основних функцій.

4 - Вимоги до безпеки: система повинна забезпечувати автентифікацію та авторизацію користувачів; повинно бути забезпечено захист даних від несанкціонованого доступу.

5 - Вимоги до надійності: система повинна забезпечувати збереження даних у разі збоїв; повинна бути реалізована можливість резервного копіювання.

6 - Вимоги щодо сумісності: система повинна працювати у веб-середовищі (через браузер); повинна підтримуватись інтеграція з базами данних та зовнішніми сервісами.

Таким чином я сформував функціональні та не функціональні вимоги, якими визначаються основні характеристики розроблюваної мною програмної системи та, які будуть основою для її подальшого проєктування. Описані вимоги вже відображають специфіку системи як інструменту для підтримки прийняття рішень у процесі оцінювання та планування програмних проєктів [24,25].

На відміну від існуючих рішень та систем, розроблювана система орієнтована не на управління виконанням проєкту, а на аналітичну підтримку прийняття рішень на етапі його підготовки.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ ДЛЯ ОЦІНЮВАННЯ І ПЛАНУВАННЯ ПРОГРАМНИХ ПРОЄКТІВ

На наступному етапі, вже після проведення аналізу предметної області та визначення основних вимог до програмної системи я буду займатись проєктуванням та практичною реалізацією. Я сформую загальну структуру мого програмного продукту, визначу його основні компоненти, логіку їх взаємодії та засоби, які є необхідними для реалізації поставлених завдань.

В цілому проєктування системи має важливе значення, бо від обраних мною рішень залежитиме не лише функціональність програмного продукту, а й зручність його використання, подальшого супроводу та можливого розширення. Продумана побудова системи дасть змогу забезпечити цілісність її роботи, уникнути зайвої складності під час реалізації та підвищити ефективність виконання основних функцій.

У цьому розділі розглянуто особливості проєктування та розробки програмної системи для оцінювання і планування програмних проєктів малого та середнього масштабу, відображається практичний етап створення та демонструється, яким чином сформульовані вимоги було реалізовано у вигляді цілісного програмного рішення. Зокрема, буде подано опис архітектури системи, структури її основних модулів, принципів організації взаємодії між ними, а також технологій та інструментальних засобів, що були використані під час розробки.

2.1 Вибір процесу розробки

У процесі проєктування програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу одним із ключових етапів є вибір підходу до його розробки, оскільки саме він визначає послідовність виконання робіт, взаємодію між учасниками та гнучкість реагування на зміни вимог [26,27,28].

Врахувуючи специфіку сервісно-орієнтованого продукту для планування проєктної діяльності, який орієнтований на підтримку прийняття рішень у процесі оцінювання та планування програмних проєктів на етапі їх підготовки з урахуванням ризиків, ресурсів та альтернативних сценаріїв реалізації, було обрано ітеративний підхід до розробки з елементами гнучких методологій. Такий вибір для мене був оптимальним за рахунок того, що вимоги до подібних систем можуть змінюватись та уточнюватися вже під час, в процесі розробки, зокрема щодо методів оцінювання, моделей аналізу та способів представлення результатів.

На відміну від жорстко структурованих моделей життєвого циклу, ітеративний підхід дозволить мені розбити процес розробки на окремі ітераційні етапи, кожен з яких буде завершуватись отриманням якогось проміжного результату. Це дасть мені можливість поступово допрацьовувати моє програмне рішення, також перевіряти коректність реалізованих функцій та вчасно вносити необхідні зміни.

У межах кожного ітераційного кроку я планую виконання наступних основних етапів:

1 - аналіз та уточнення вимог;

2 - проєктування окремих компонентів системи;

3 - реалізацію функціональності;

4 - тестування та перевірку отриманих результатів.

Особливістю обраного ітераційного підходу є те, що основну увагу я зможу приділити реалізації саме ключового функціонала системи – модулю підтримки прийняття рішень ще на етапі підготовки проєктів з урахуванням ризиків, ресурсів та альтернативних сценаріїв реалізації. Власне саме цей компонент і потребуватиме поетапного уточнення алгоритмів, перевірки результатів обчислень та адаптації до різних сценаріїв використання.

Крім цього, ітеративний підхід добре узгоджується із сервісно-орієнтованою архітектурою системи, оскільки дозволяє реалізовувати окремі

сервіси незалежно один від одного, поступово розширюючи функціональні можливості програмного продукта.

Таким чином, обраний підхід також дозволяє поступово перевіряти ефективність запропонованих моделей оцінювання на практичних прикладах. Обраний ітеративний процес розробки дасть змогу забезпечити оптимальну гнучкість, можливість поступового вдосконалення системи та зменшити ймовірні ризики, пов’язані із реалізацією достатньо складної логіки оцінювання та планування програмних проєктів малого та середнього масштабу.

2.2 Проєктування архітектури програмного рішення

У процесі проєктування програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу я особливу увагу приділяв вибору архітектурного підходу, бо на мою думку саме архітектура визначає структуру системи, принципи взаємодії її компонент та можливість подальшого розширення її функціональності [26,33,35].

Враховуючи вимоги щодо гнучкості, масштабованості та можливості інтеграції з іншими системами, для розроблюваного продукту я прийняв рішення обрати найоптимальнішу, сервісно-орієнтовану архітектуру. Такий підхід дозволить мені розділити програму на окремі незалежні сервіси, кожен з яких відповідатме за виконання визначеної конкретної функції.

Особливістю моєї розробки є наявність окремого компонента, яким реалізується логіка підтримки прийняття рішень. Саме ця компонента об’єднуватиме результати оцінювання, аналіз ризиків та моделювання сценаріїв. Таким чином, в результаті користувач отримає сформовані рекомендації.

В межах проєкту я виділив такі основні компоненти:

1. Клієнтський рівень/користувацький інтерфейс. Забезпечується взаємодія користувачів із системою. На цьому рівні реалізується введення даних, відображення результатів оцінювання, сценаріїв та рекомендацій. Акцент рівня я робив на простоту використання та наочність представлення інформації.

2. Сервер прикладної логіки (Backend). Тут реалізується обробка запитів користувачів, координується робота системи та забезпечується взаємодія між компонентами роброблюваного продукту. Також на цьому рівні буде реалізовуватись основна бізнес-логіка системи.

3. Модуль підтримки прийняття рішень – є ключовою внутрішньо аналітичною компонентою системи, якою реалізуються: розрахунок параметрів проєкта; оцінювання ризиків; формування альтернативних сценаріїв;порівняння варіантів реалізації; генерація рекомендацій. Цим модулем я реалізую інтелектуальну складову системи, чим і визначу її основну цінність.

4. Сервіс управління проєктами відповідає за створення, збереження та редагування даних про проєкти, а також за керування їхніми параметрами.

5. Сервіс аналітики та оцінювання забезпечуватиме виконання обчислень, які пов’язані із визначенням характеристик проєкту, наприклад таких як трудомісткість, ресурси, терміни, тощо.

6. Сервіс роботи з даними, тобто база даних забезпечить зберігання всієї інформації, включаючи параметри проєктів, результати оцінки, історичні дані, облікові записи користувачів.

7. Сервісом управління користувачами я реалізовуватиму автентифікацію, авторизацію та розмежування прав доступу.

Стосовно взаємодії, то користувач взаємодіятиме із системою через інтерфейс, через надсилання запитів до серверної частини. Сервер прикладної логіки оброблятиме ці запити та передаватиме їх відповідним сервісам. В разі виконання оцінювання або аналізу дані передаватимуться до модуля підтримки прийняття рішень, який здійснюватиме необхідні розрахунки та поверне результати. А вже отримані результати зберігатимуться в базі даних та відображатимуться користувачу у вигляді зрозумілої інформації, якою він і зможе скористатися для підтримки (обґрунтування) прийняття рішень.

До переваг архітектурного підходу який я обрав, з врахуванням специфіки розроблюваного прогамного продукту я відніс модульність та незалежність компонент, можливість масштабування системи, спрощення супроводу та

модифікації, можливість інтеграції з іншими сервісами, гнучкість у реалізації алгоритмів оцінювання та аналізу.

Особливо важливим є те, що така сервісно-орієнтована архітектура дозволить незалежно розвивати модуль підтримки прийняття рішень, що є ключовим елементом системи.

Отже, обрана архітектура є оптимальною, та забезпечить ефективну організацію розробки, дозволить реалізувати складну логіку оцінювання та планування проєктів, а також створить основу для подальшого розвитку та вдосконалення системи.

У межах роботи сервісно-орієнтована архітектура реалізована на логічному рівні шляхом розподілу системи на окремі функціональні сервіси, що взаємодіють між собою через сервер прикладної логіки .

Узагальнену структуру програмної системи та взаємодію між її основними компонентами доцільно представити у вигляді компонентної діаграми, яка відображає розподіл функціональності між сервісами та їх взаємозв’язки.

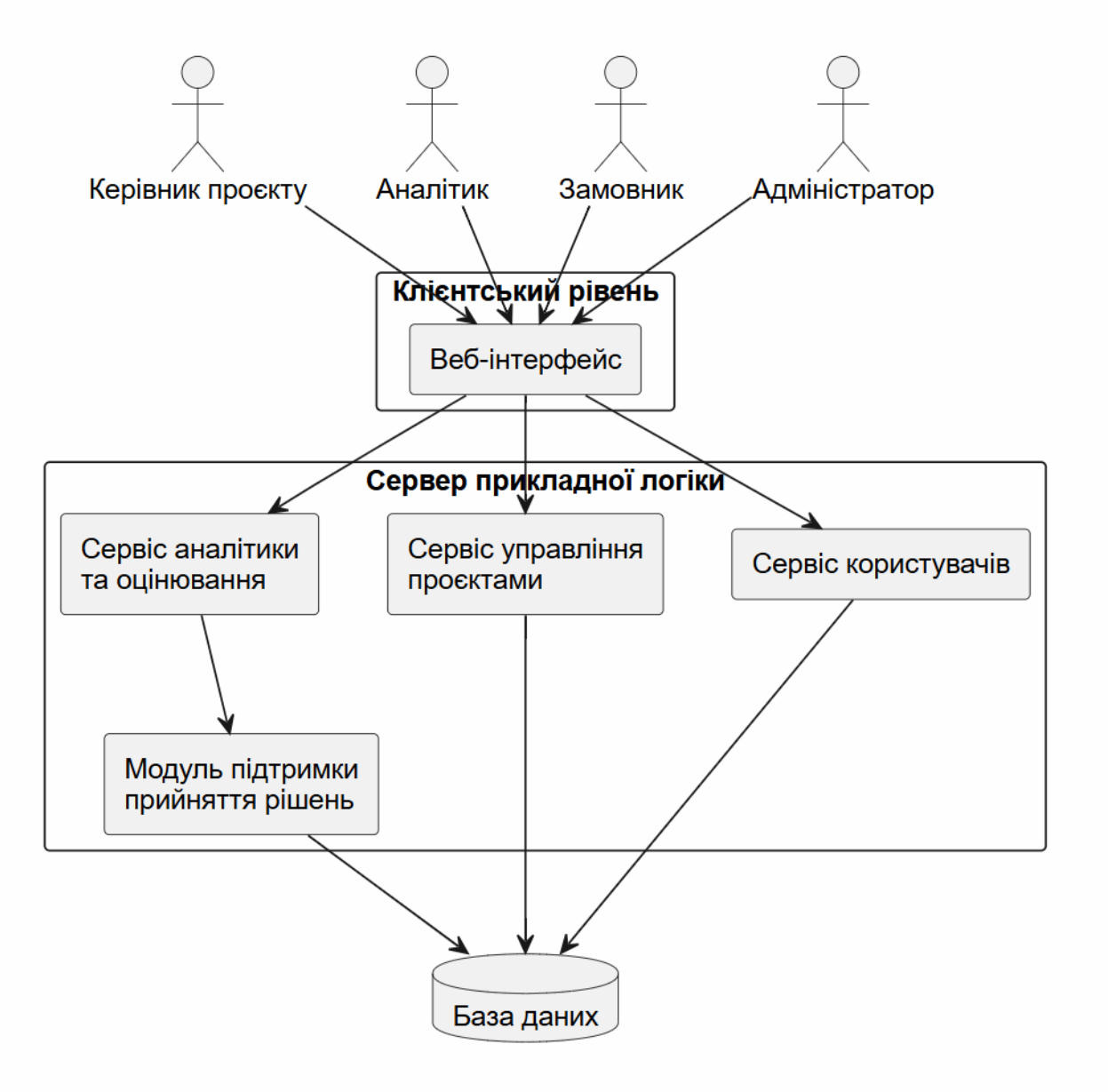

Рисунок 2.1 – Компонентна діаграма архітектури програмної системи

У попередньому підрозділі «Модуль підтримки прийняття рішень» було розглянуто в контексті взаємодії із системою як окрему сутність, що бере участь у виконанні сценаріїв. Однак слід зазначити, що з точки зору архітектурного проєктування даний модуль не є зовнішнім актором, а належить до внутрішніх компонентів системи.

Таке представлення я використав для підкреслення його функціональної ролі у процесі обробки даних та формування результатів. У подальшому, при проєктуванні архітектури, модуль підтримки прийняття рішень розглядається як

складова серверної частини системи, що реалізує бізнес-логіку та забезпечує виконання аналітичних операцій.

Таким чином, відмінність у представленні я поясню різними рівнями абстракції: на етапі аналізу вимог модуль може розглядатися як окрема логічна сутність, тоді як на етапі проєктування він виступатиме як внутрішній компонент системи.

Компонентна діаграма відображає загальну структуру програмної системи та взаємодію між її основними складовими. Користувачі системи, зокрема: «Керівник проєкту», «Аналітик», «Замовник проєкту» та «Адміністратор», взаємодіють із системою через клієнтський рівень, представлений веб-інтерфейсом.

Сервер прикладної логіки включає набір функціональних сервісів, кожен з яких відповідає за виконання окремих задач. Сервіс управління проєктами забезпечує роботу з даними проєктів, сервіс аналітики та оцінювання виконує обчислення параметрів проєкту, а модуль підтримки прийняття рішень реалізує формування рекомендацій на основі результатів аналізу. Окремо виділено сервіс управління користувачами, що відповідає за контроль доступу до системи. Зберігання та обробка даних здійснюється при допомозі бази даних, з якою взаємодіють відповідні сервіси.

Таким чином такою структурою, що відповідає принципам сервісно-орієнтованої архітектури, я зможу забезпечити розподіл функціональності між компонентами системи.

2.3 Побудова схеми бази даних

Для забезпечення зберігання, опрацювання та подальшого використання данних у розроблюваному програмному рішенні я спроєктував структуру бази даних, яка повинна забезпечувати ефективне збереження інформації про проєкти, про їх параметри, про результати їх оцінювання, про сценарії реалізації та про користувачів системи [26,33,34].

Проєктування бази даних я виконував з врахуванням функціональних вимог системи (п.п. 1. Користувач (User)1.5 Функціональні та нефункціональні вимоги до програмної системи), а також необхідності підтримки аналітичних операцій і формування рекомендацій. Особливу увагу я приділив на структуризацію даних. Тут мені важливо було забезпечити можливість зберігання альтернативних варіантів реалізації проєктів та результатів їх порівняння.

Далі в таблиці 2.1. я перелічив та описав основні сутності та атрибути структури бази даних.

Таблиця 2.1 – Основні сутності бази даних

Сутність	Опис	Основні атрибути
1. Користувач (User)	Містить інформацію про користувачів системи, їх ролі та облікові дані.	-ідентифікатор користувача; -ім’я; -електронна пошта; -пароль; -роль.
2. Проєкт (Project)	Описує загальну інформацію про програмний проєкт.	-ідентифікатор проєкту; -назва; -опис; -дата створення; -автор (користувач).
3. Параметри проєкту (ProjectParameters)	Зберігає вхідні дані для оцінювання.	-ідентифікатор; -трудомісткість; -кількість ресурсів; -бюджет; -терміни.
4. Оцінка (Evaluation)	Містить результати розрахунків параметрів проєкту.	-ідентифікатор; -розрахована трудомісткість; -тривалість; -використані ресурси; -дата оцінювання.

Продовження таблиці 2.1

Сутність	Опис	Основні атрибути
5. Ризик (Risk)	Описує потенційні ризики проєкту.	-ідентифікатор; -назва; -ймовірність; -вплив; -опис.
6. Сценарій (Scenario)	Містить альтернативні варіанти реалізації проєкту.	ідентифікатор; назва; опис; оцінка ефективності.
7. Рекомендація (Recommendation)	Зберігає результати аналізу та вибору оптимального варіанту.	ідентифікатор; опис рекомендації; обраний сценарій; дата формування.

Після визначення основних сутностей бази даних доцільно розглянути зв’язки між ними. Саме зв’язки відображають логіку організації даних у системі та саме вони ж і забезпечують цілісність інформаційної моделі. Встановлення взаємозв’язків між сутностями дає мені можливість коректно структурувати дані та тим самим уникнути їх дублювання, а також забезпечити логічну, «правильну» взаємодію між окремими компонентами програмного рішення.

Отже, зв’язки між сутностями бази даних виглядають наступним чином:

- один користувач може створювати декілька проєктів (1:N);
- один проєкт має один набір параметрів (1:1);
- один проєкт може мати декілька оцінок (1:N);
- один проєкт може містити декілька ризиків (1:N);
- один проєкт може мати декілька сценаріїв (1:N);
- один сценарій може бути обраний у рекомендації (1:1);
- один проєкт може мати декілька рекомендацій (1:N).

Таким чином, я запропонував структуру бази даних, яка дозволить мені:

- зберігати історію оцінювання проєктів;
- аналізувати різні варіанти реалізації;
- підтримувати процес прийняття рішень;

- забезпечувати гнучкість у додаванні нових параметрів та моделей оцінювання.

В результаті, спроєктована мною база даних програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу забезпечить збереження всіх необхідних даних для повноцінного функціонування системи, буде підтримувати аналітичні процеси та є основою для реалізації модулю підтримки прийняття рішень.

З метою наочного представлення структури бази даних та зв’язків між її основними сутностями я побудував ER-діаграму, яка відображатиме логічну модель даних розроблюваної сервісно-орієнтованої програми для планування проєктної діяльності.

ER-діаграма відображає логічну структуру бази даних та взаємозв’язки між основними сутностями програмного рішення, центральною сутністю якого є проєкт, з яким пов’язані інші дані, що використовуються у процесі оцінювання та планування.

В свою чергу користувачі системи можуть створювати проєкти, що визначає зв’язок між сутностями «User» та «Project». Для кожного такого проєкту зберігатимуться параметри, необхідні для проведення оцінювання, а також результати розрахунків, представлені у вигляді сутності «Evaluation».

Важливою складовою є «Risk». Ця сутність надає можливість враховувати потенційні ризики, що можуть вплинути на успішність реалізації проєкту.

Сутність «Scenario» містить альтернативні варіанти виконання проєкту. Вони формуються на основі аналізу параметрів та можливих ризиків.

На основі отриманих результатів формується сутність «Recommendation». Ця сутність відображає обраний варіант реалізації проєкту або рекомендації щодо його виконання.

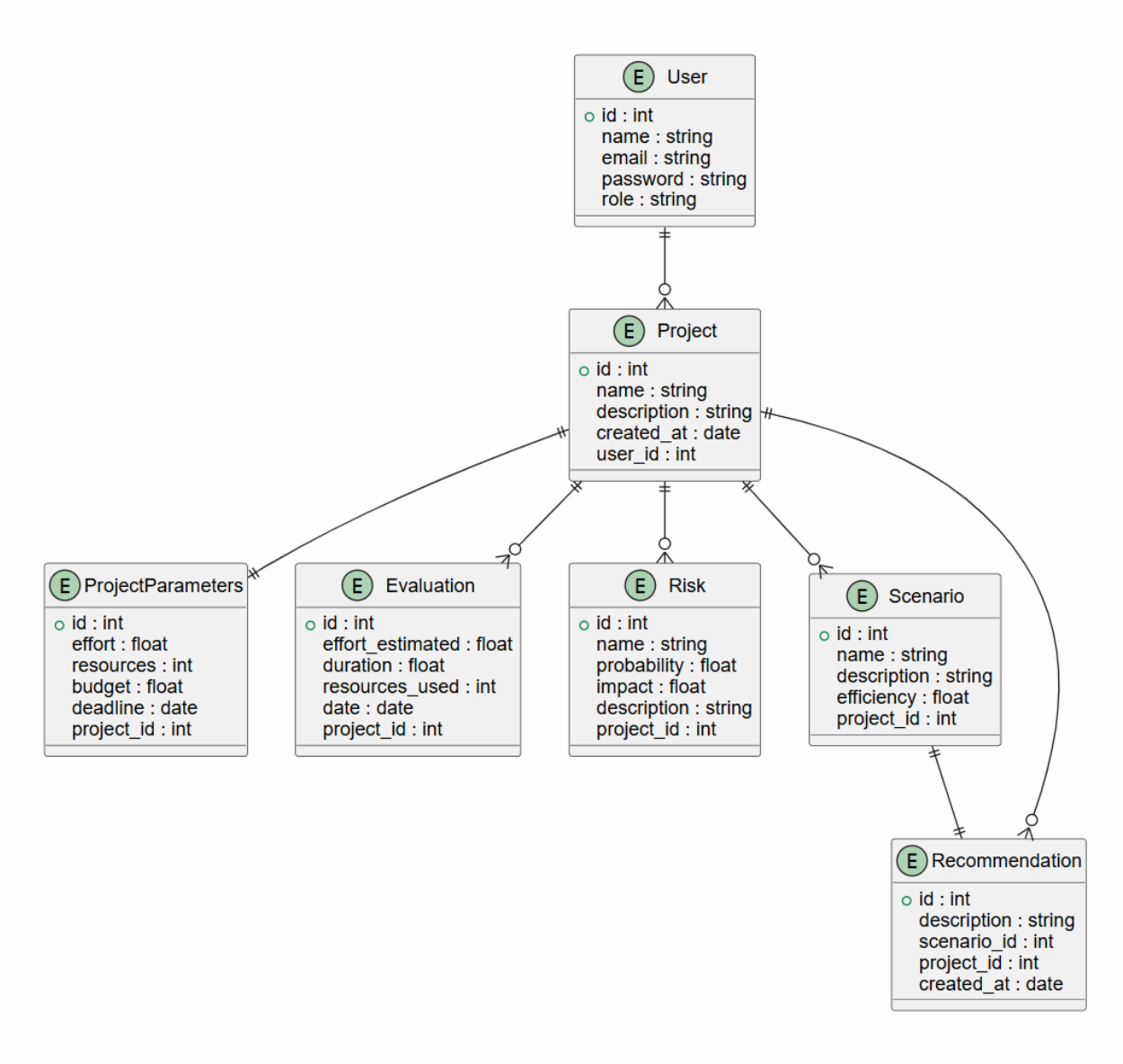

Рисунок 2.2 – ER-діаграма бази даних програмного рішення

Таким чином, запропонована структура бази даних забезпечує збереження необхідної інформації для підтримки процесів оцінювання, аналізу та прийняття рішень у межах програмної системи.

2.4 Побудова UML-діаграми класів

Одним із важливих етапів процесу проєктування розроблюваного програмного продукту є формування його структури на рівні програмної реалізації. Для цього я використав UML-діаграму класів, за допомогою якої

представлю основні класи моєї сервісно-орієнтованої системи планування проєктної діяльності, їх атрибути, методи та взаємозв’язки між ними.

Побудову діаграми класів я виконав базуючись на спроєктовану структуру бази даних описану в п.п. 2.3 Побудова схеми бази даних. Тако само я врахував сформульовані в п.п. 1.5 Функціональні та нефункціональні вимоги до програмної системи функціональні вимог до програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу. Це дозволило мені логічно узгодити структуру збереження даних із логікою їх обробки.

Для наочного представлення структури сервісно-орієнтованої системи планування проєктної діяльності та взаємодії між її складовими далі, на рисунку 2.3, я наведу побудовану UML-діаграму класів, яка відображатиме внутрішню структуру та дозволить деталізувати її реалізацію на рівні програмного коду. Крім цього поєднає у собі як структуру даних, так і логіку їх опрацювання, що дасть змогу мати вже цілісне уявлення про функціонування системи.

У побудованій діаграмі виділено групи класів, які відповідатимуть за різні напрямки роботи розроблюваного програмного продукту, зокрема до першої групи я відніс класи, якими описуються основні сутності предметної області, («User», «Project», «ProjectParameters», «Evaluation», «Risk», «Scenario» та «Recommendation»), оскільки вони «відповідають» за збереження та представлення даних, а також є основою для виконання подальших обчислень.

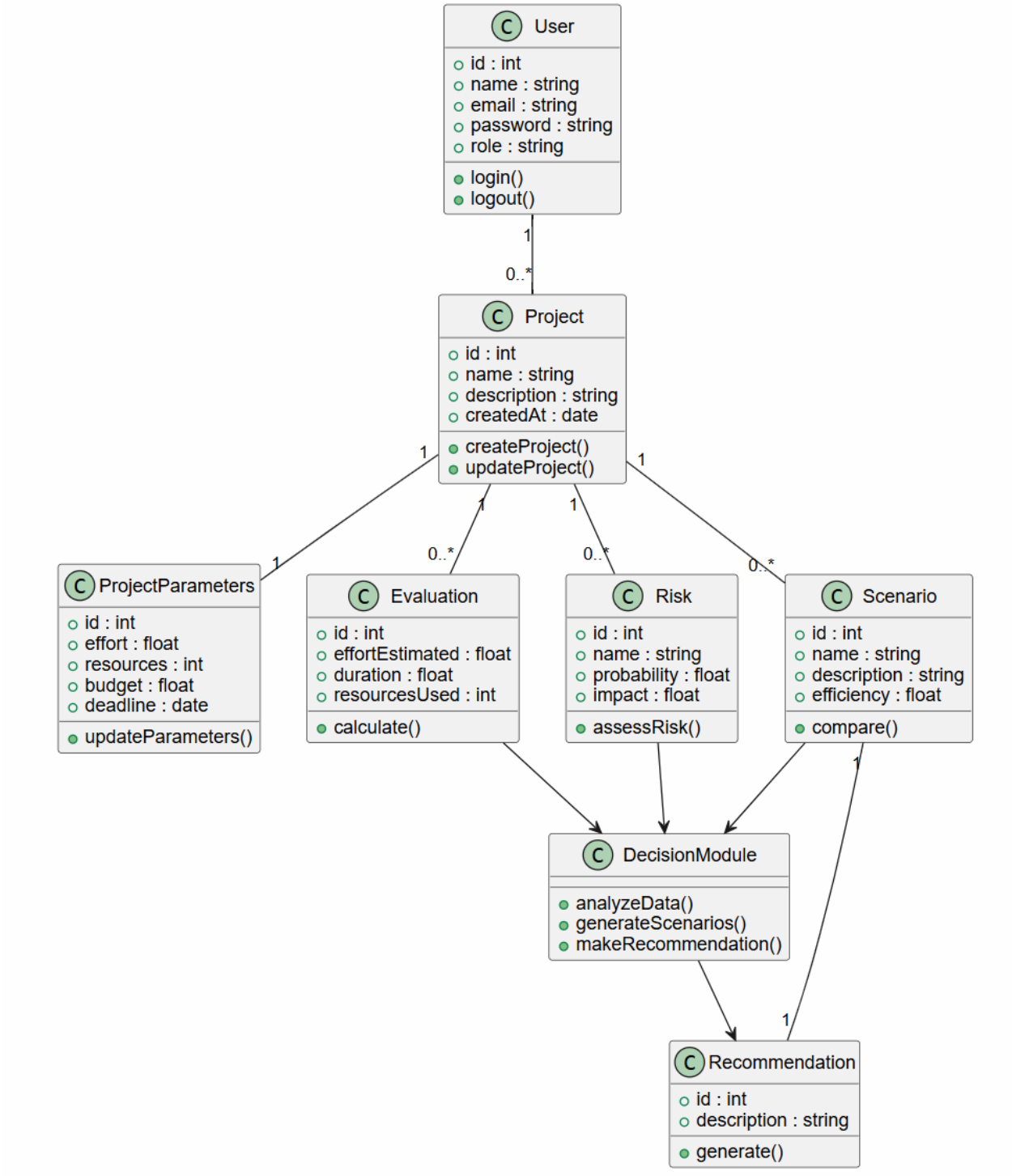

Рисунок 2.3 – UML-діаграма класів програмної системи

Далі я наведу більш детальний опис основних класів.

Клас «User» реалізує функціональність, пов’язану із роботою користувачів системи, зокрема автентифікацію та управління доступом, а його зв’язок із класом «Project» показує можливість одного користувача створювати та керувати декількома проєктами.

Клас «Project» вважаю є центральним елементом системи, оскільки він об’єднує всі інші сутності. Саме через клас «Project» реалізовано доступ до параметрів проєкту, результатів оцінювання, ризиків та сценаріїв.

Клас «ProjectParameters» містить вхідні дані, необхідні для проведення оцінювання. Має зв’язок один до одного з класом «Project».

Клас «Evaluation» відповідає за збереження результатів розрахунків, що дозволяє виконувати повторні оцінювання та аналізувати зміни параметрів.

Клас «Risk» використовується для опису потенційних ризиків. Також я його враховую під час формування сценаріїв реалізації проєкту.

Клас «Scenario» представляє альтернативні варіанти реалізації проєкту, які можуть бути порівняні між собою за визначеними критеріями.

Клас «Recommendation» відображає результат роботи системи у вигляді обґрунтованого вибору оптимального варіанта.

Клас «DecisionModule»Окрему має окрему важливу роль. Ним реалізується логіка підтримки прийняття рішень. Він не зберігає дані, а лише їх аналізу, взаємодіючи з класами «Evaluation», «Risk» та «Scenario». Також він формує рекомендації.

Отже, за домогою діаграми класів я наочно демонструю взаємозв’язок між структурою даних та логікою роботи системи. Це є основою для подальшої реалізації програмного забезпечення.

2.5 Вибір мови та середовища розробки

У процесі розробки програмної системи важливим етапом є вибір технологічного стеку, який має відповідати специфіці поставленої задачі, забезпечувати зручність реалізації функціоналу та можливість подальшого

розвитку системи. А так як розроблювана система орієнтована на виконання аналітичних обчислень, обробку параметрів проєкту, оцінювання ризиків та формування рекомендацій, ключовим критерієм вибору мови програмування стала її придатність до роботи з даними та реалізації алгоритмів [29-33].

Отже, для реалізації серверної частини обрано мову програмування Python. Такий вибір зумовлений тим, що Python має простий та зрозумілий синтаксис, що дозволить мені зосередитися безпосередньо на реалізації логіки системи, а не на технічних деталях. Крім цього, мова Python досить поширено використовується саме для обробки даних та побудови аналітичних моделей, а це є важливим якраз для мого випадку і особливості – для реалізації модуля підтримки прийняття рішень, якиє є внутрішнім аналітичним компонентом. Використання Python дозволить мені ефективно реалізувати алгоритми оцінювання, опрацювання можливих ризиків та формування альтернативних сценаріїв.

У якості фреймворку для серверної частини я зупнив свій вибір на FastAPI. Основною причиною мого вибору є його орієнтація на створення високопродуктивних веб-сервісів. FastAPI підтримує асинхронну обробку запитів. Це дозволяє ефективно працювати з декількома запитами одночасно і є важливим у випадку виконання обчислень або обробки великих обсягів даних [36-40]. Крім того, фреймворк FastAPI забезпечує зручне створення API, що відповідає принципам сервісно-орієнтованої архітектури, обраної для розроблюваного мною програмного рішення.

Для збереження даних я буду використовувати систему керування базами даних PostgreSQL. Цей вибір обумовлений, тим, що вона є надійною, підтримує складні запити, та є високопродуктивною в роботі зі структурованими даними [36-40]. А враховуючи особливості моєї розробки, в якій планується зберігати не лише базову інформацію про проєкти, а й результати їх оцінювання, сценарії та рекомендації, то виникла необхідність в гнучкій та стабільній базі даних. Саме тому мені добре підходить PostgreSQL оскільки вона розрахована для реалізації таких задач як мої і забезпечить мені можливість масштабування мого програмного рішення у майбутньому.

Клієнтську частину мого програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу я реалізую реалізується з використанням стандартних веб-технологій, зокрема HTML, CSS та JavaScript. Такий вибір пояснюється їх універсальністю та можливістю створення зручного інтерфейсу користувача без прив’язки до конкретної платформи. Використання JavaScript дозволяє динамічно відображати результати аналізу, змінювати параметри та взаємодіяти із серверною частиною через API.

В якості середовища розробки я обрав Visual Studio Code. Свій вибір я можу обґрунтувати його гнучкістю, підтримкою необхідних мов програмування та великою кількістю розширень. Це дозволить мені працювати як із серверною, так і з клієнтською частинами розроблюваного програмного продукту в одному середовищі.

Обраний стек технологій є взаємодоповнюючим, оскільки мова Python забезпечить реалізацію аналітичної логіки, FastAPI – ефективну взаємодію між компонентами системи, а система керівання базами даних PostgreSQL – надійне збереження даних. Веб-технології в свою чергу дадуть змогу отримати зручний доступ всіх користувачів до функціоналу системи.

Отже, враховуючи вище наведене обґрунтування, з впевненістю можу сказати, що вибір мови програмування та середовища розробки є обґрунтованим з точки зору поставлених задач та забезпечить ефективну реалізацію сервісно-орієнтованої програмної системи підтримки прийняття рішень. Обраний технологічний стек також дозволить передбачити в майбутньому інтегрувати систему з іншими інформаційними ресурсами та розширювати її функціональні можливості.

2.6 Реалізація основних класів та методів

Опираючись на розроблену UML-діаграму класів я виконав реалізацію основних компонент розроблюваного програмного рішення. Представлена

реалізація охоплює класи (що відповідають за збереження даних) та модуль (який забезпечує обробку інформації та підтримку прийняття рішень) [29-32].

При реалізації я обрав та використав підхід об’єктно-орієнтованого програмування, який дозволить мені чітко розмежувати «відповідальність» між окремими компонентами програмного рішення.

У системі виділено наступну групу класів, які відповідають за збереження та представлення даних.

Лістинг 2.1 – Клас Project

```
class Project:
    def __init__(self, name, description, user_id):
        self.name = name
        self.description = description
        self.user_id = user_id
```

Клас «Project» використовується для збереження основної інформації про проєкт.

Лістинг 2.2 – Клас Evaluation

```
class Evaluation:
    def __init__(self, effort, resources):
        self.effort = effort
        self.resources = resources

    def calculate_duration(self):
        return self.effort / self.resources if self.resources else 0
```

Клас «Evaluation» забезпечує виконання базових розрахунків, зокрема визначення тривалості виконання проєкту.

Лістинг 2.3 – Клас Risk

```
class Risk:
    def __init__(self, probability, impact):
        self.probability = probability
        self.impact = impact

    def calculate_risk_level(self):
        return self.probability * self.impact
```

Клас «Risk» дозволяє кількісно оцінити рівень ризику на основі його ймовірності та впливу.

Ключовим елементом системи є модуль підтримки прийняття рішень, який реалізовано у вигляді окремого класу.

Лістинг 2.4 – Клас DecisionModule

```
class DecisionModule:
    def evaluate_project(self, evaluation, risks):
        total_risk = sum(r.calculate_risk_level() for r in risks)
        return evaluation.calculate_duration() + total_risk

    def make_recommendation(self, scenarios):
        return min(scenarios, key=lambda s: s.score)
```

Метод `evaluate_project()` виконує узагальнення результатів оцінювання та ризиків, формуючи інтегральний показник. Метод `make_recommendation()` дозволяє обрати найбільш ефективний варіант реалізації проєкту серед доступних сценаріїв.

Далі, для демонстрації взаємодії між класами та процесу оцінювання проєкту з урахуванням ризиків, я наведу приклад.

Лістинг 2.5 – Приклад взаємодії між класами та процес оцінювання проєкту з урахуванням ризиків

```
project = Project("Проєкт", "Опис", 1)
evaluation = Evaluation(100, 5)

risk1 = Risk(0.3, 5)
risk2 = Risk(0.2, 4)

decision = DecisionModule()
result = decision.evaluate_project(evaluation, [risk1, risk2])
```

Таким чином, реалізація основних класів та методів забезпечує виконання ключових функцій системи, а саме обробку параметрів проєкту, оцінювання ризиків та підтримку прийняття рішень. Запропонована структура є гнучкою та може бути розширена відповідно до подальших вимог.

Розширена реалізація класів та приклади програмного коду наведені у додатку.

2.7 Розробка інтерфейса користувача

Розробка користувацького інтерфейса одним із важливих етапів створення програмної системи, оскільки саме через нього я забезпечу взаємодію користувачів із функціональними можливостями розроблюваного програмного рішення. Моєю основною метою при проєктуванні інтерфейса було створення зручного, зрозумілого та інтуїтивного середовища для виконання задач, пов’язаних із оцінюванням параметрів проєкту та підтримкою прийняття оптимальних рішень [30,31].

Під час розробки інтерфейсу я враховував ролі користувачів, визначені на етапі аналізу вимог, зокрема керівника проєкту, аналітика, замовника проєкту та адміністратора. Кожен із зазначених користувачів взаємодіє із системою відповідно до своїх функцій: керівник проєкту здійснює загальне управління та прийняття рішень, аналітик відповідає за введення та аналіз даних, замовник

проєкту переглядає результати оцінювання та сформовані рекомендації, а адміністратор забезпечує технічну підтримку та управління доступом до системи. Таким чином, це дозволило мені сформувати структуру інтерфейсу відповідно до їх потреб та сценаріїв використання.

Інтерфейс програмного рішення я реалізую у вигляді вебзастосунку, що забезпечить доступ до функціоналу через браузер. Необхідності встановлювати додаткове програмне забезпечення не буде. Такий підхід дасть мені змогу підвищити зручність використання сервісно-орієнтованої системи планування проєктної діяльності та забезпечить доступність з різних пристроїв.

Далі перелічу основні елементи інтерфейса, який включає наступні компоненти:

- сторінка авторизації користувача;
- панель керування з переліком проєктів;
- форма створення та редагування проєкту;
- модуль відображення результатів оцінювання;
- блок формування рекомендацій.

Саме ця структура забезпечить логічну послідовність роботи користувача від введення даних до отримання кінцевих результатів.

Для реалізації інтерфейсу я використав стандартні веб-технології, зокрема HTML, CSS та JavaScript.

Далі наведу приклад форми введення параметрів проєкта. Наведений фрагмент кода продемонструє базову форму введення параметрів проєкту, яка використовуватиметься для передачі даних до серверної частини програмне рішення для підтримки оцінювання проєктів малого та середнього масштабу.

Лістинг 2.6 – Форма для введення параметрів проєкта

```
<form>
  <h3>Створення проєкту</h3>

  <label>Назва проєкту:</label>
  <input type="text" name="name" required>

  <label>Опис:</label>
  <input type="text" name="description">

  <label>Трудомісткість:</label>
  <input type="number" name="effort" required>

  <label>Кількість ресурсів:</label>
  <input type="number" name="resources" required>

  <button type="submit">Розрахувати</button>
</form>
```

Для обміну даними між інтерфейсом та серверною частиною я використав API. Введені користувачем дані передаватимуться на серверну сторону, де здійснюватиметься їх опрацювання, виконання розрахунків та формування результатів.

Отримані результати повертаються до інтерфейсу та відображаються у зручному для користувача вигляді, що дозволяє аналізувати параметри проєкту та обирати оптимальні варіанти його реалізації.

Важливу роль у функціонуванні системи відіграє модуль підтримки прийняття рішень, який є внутрішнім компонентом системи. Хоча він не є безпосереднім користувачем інтерфейсу, результати його роботи відображаються у вигляді рекомендацій та аналітичних показників. Таким чином, користувачі отримують доступ до результатів обробки даних без прямої взаємодії з даним модулем.

Робота користувача із системою відбуватиметься в декілька етапів:

1 - користувач проходить авторизацію та переходить до панелі керування;

2 - створюється новий проєкт або обирається існуючий.

Далі, вже після введення параметрів система виконуватиме оцінювання з врахуванням ризиків та сформує альтернативні сценарії. Результати аналіза відображатимуться в вигляді структурованої інформації, яка дозволяить користуваччеві прийняти обґрунтоване рішення.

Отже, розроблений інтерфейс користувача забезпечить зручну взаємодію із системою, підтримуватиме основні сценарії її використання. Також він дозволить ефективно виконувати задачі, які пов’язані із аналізом і плануванням проєктів.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО РІШЕННЯ ДЛЯ ПІДТРИМКИ ОЦІНЮВАННЯ ПРОЄКТІВ МАЛОГО ТА СЕРЕДНЬОГО МАСШТАБУ

Після завершення розробки програми мені треба переконатися, що вона коректно виконує покладені на неї функції, може бути використана в реальних умовах і не потребує значного доопрацювання перед початком експлуатації.

Перевірка роботи дасть змогу оцінити правильність реалізації окремих функціональних можливостей, виявити можливі помилки та визначити, наскільки стабільно програмне рішення працює в цілому. Водночас впровадження передбачає не тільки технічне розгортання системи, а й створення умов для її практичного використання. Подальша підтримка, у свою чергу, необхідна для забезпечення стабільної роботи, своєчасного виявлення і усунення недоліків, можливого вдосконалення функціонала.

У цьому розділі розглянуто основні аспекти тестування, впровадження та підтримки програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу. Окрему увагу приділено перевірці роботи системи, особливостям її використання в робочому середовищі, а також питанням подальшого супроводу та оновлення.

Таким чином, цей розділ відображає завершальний етап реалізації програмного продукту та дає змогу оцінити його готовність до практичного застосування [41-46].

3.1. Тестування програмного рішення

Тестування сервісно-орієнтованого програмного продукту для планування проєктної діяльності є невід’ємною складовою процесу його розробки, оскільки дозволить забезпечити коректність реалізації його оптимальних функціональних можливостей, стабільність роботи та відповідність розробки визначеним вимогам

замовника. Проведення тестування дасть мені змогу виявити помилки на різних етапах функціонування, так само як і оцінити його готовність до використання.

У межах розроблюваної системи тестування спрямоване на реалізацію перевірки основних функціональних компонент. Йдеться про процеси введення та обробку параметрів проєкта, виконання розрахунків, оцінювання ризиків, формування альтернативних сценаріїв, а також ґенерацію рекомендацій щодо реалізації самого програмного проєкту.

Особливо я вважаю за необхідне приділити увагу перевірці модулю для підтримки прийняття рішень «DecisionModule», яким реалізується ключова аналітична логіка моєї програмної розробки, оскільки власне цей модуль відповідає за узагальнення (зведення) результатів оцінювання, врахування ризиків та формування обґрунтованих рекомендацій. На мою думку, цей модуль вигідно вирізняє мій продукт від подібних в предметній області, і саме тому його коректна робота є критично важливою для всієї системи.

Тестування також охоплює перевірку взаємодії між окремими компонентами системи. Йде мова про взаємодію поміж клієнтською та серверною частиною, а також коректність передачі та опрацювання данних через API. Це дозволить мені забезпечити узгоджену роботу увсіх складових розроблюваного програмного рішення.

Окремим аспектом для тестування є перевірка на поведінку системи у випадках при некоректному введенні даних. Зокрема, йдеться про аналіз реакції системи оцінювання і планування програмних проєктів на відсутність обов’язкових параметрів, введення некоректних значень або якісь нетипові, граничні випадки. Такий підхід дозволить підвищити надійність, а також уникнути помилок під час експлуатації.

Окрім того, тестування скероване на оцінку зручності використання кінцевого програмного продукту. Здійснюється перевірка логічності структури інтерфейса, зрозумілості елементів управління та коректності представлення результатів. Це дозволить мені забезпечити ефективну взаємодію користувача з програмою.

Таким чином, тестування програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу вважаю, що є комплексним процесом. Він включає перевірку функціональних можливостей, взаємодії компонент, обробки даних та зручності використання. Результати тестування дозволяють мені підтвердити працездатність кінцевого програмного рішення для оцінювання і планування програмних проєктів та його готовність до подальшого впровадження для наступного використання.

3.1.1 Види та план тестування

Для забезпечення надійності та коректності роботи розроблюваної програмної системи я прийняв рішення застосувати комплексний підхід до тестування. Це передбачає використання декількох видів тестів. Такий підхід дозволить мені перевірити як окремі компоненти, так і їх взаємодію, а також оцінити відповідність програмного рішення функціональним вимогам [41,42].

Враховуючи специфіку розроблюваного програмного продукту, яка заключається в обробці даних, виконання розрахунків та підтримку прийняття рішень, особливу увагу я вирішив зосередити на перевірці правильності алгоритмів та логіки роботи основних модулів.

Для досягнення поставлених цілей тестування я вважаю за доцільне застосувати декілька видів тестування, кожен із яких буде спрямовано на перевірку окремих аспектів функціонування системи. Саме такий підхід дозволить мені якнайкраще забезпечити ґрунтовну перевірку програмної розробки, а саме від її окремих компонент до вже їх взаємодії та відповідності до вимог користувачів.

Мій вибір конкретних видів тестування обумовлено особливостями розроблюваної системи та її функціональним призначенням. Отже, я пропоную застосувати:

1 - модульне тестування (перевірка окремих компонент системи незалежно один від одного. В представленій мною роботі тестуванню підлягають основні класи, зокрема «Project», «Evaluation», «Risk» та «DecisionModule»).

Тут я пропоную до перевірки коректність роботи методів, правильність виконання розрахунків та обробки даних. Наприклад, для класу «Evaluation» тестую метод визначення тривалості проєкту. Для класу «Risk» – правильність обчислення рівня ризику. Особливу увагу я приділяю перевірці методів «DecisionModule» – модуля підтримки прийняття рішень, оскільки саме з їх допомогою реалізую ключову функціональність розроблюваного програмного продукту.

2 - інтеграційне тестування (йде мова про перевірку взаємодії між окремими компонентами пррограми. Тут я пропоную до первірки коректність передачі даних між класами, а також узгодженості роботи як серверної так і клієнтської частини).

Зокрема, тестуванню піддаються процес передачі параметрів проєкту від інтерфейса до серверної частини, обробка цих получених даних та повернення вже результатів у вигляді оцінок і рекомендацій. Це дозволить переконатися у правильності реалізації взаємодії між компонентами системи.

3 - функціональне тестування (перевірка відповідності системи визначеним функціональним вимогам).

При цьому виді тестування перевіряються основні сценарії використання розроблюваного програмного продукту, зокрема:

- створення проєкту;
- налаштування проєкта;
- введення параметрів;
- виконання розрахунків;
- оцінка ризиків;
- формування альтернативних сценаріїв;
- надання рекомендацій.

Функціональне тестування дозволить мені оцінняти, наскільки програмне рішення для підтримки оцінювання проєктів малого та середнього масштабу відповідає поставленим задачам. Також я можу зрозуміти чи забезпечує розроблюване програмне рішення необхідний функціонал.

4 - тестування користувацького інтерфейса (перевірка на зручність використання програми та коректності відображення даних).

При цьому виді тестування я оцінюю логіку розташування елементів, зрозумілість форм введення і відображення результатів. Особливу увагу я надаю перевірці на коректність представлення роботи «DecisionModule» – модуля підтримки прийняття рішень, зокрема рекомендацій та сценаріїв.

5 - тестування обробки помилок. Даний вид тестування передбачає перевірку поведінки програмного продукта у випадках введення некоректних або неповних даних. Тут перевіряється реакція системи на:

- відсутність обов’язкових параметрів,
- введення нульових або від’ємних значень;
- некоректні типи даних.

Тут метою є забезпечення стабільності роботи системи та запобігання виникнення критичних помилок під час її використання та роботи.

6 – тестування план тестування (визначає послідовність виконання тестових робіт).

План тестування складається з наступних основних етапів:

- визначення об’єктів під тестування (класи, модулі, функції);
- підготовка тестових данних;
- виконання тестових сценаріїв;
- фіксація результатів тестування;
- аналіз виявлених помилок;
- повторне тестування після усунення виявлених на попередньому етапі помилок.

Представлений мною підхід дозволить мені якнайкраще системно перевірити розроблене програмне рішення та забезпечити його якісне функціонування.

3.1.2 Розробка тестових сценаріїв

Для забезпечення комплексної перевірки працездатності розробленого програмного рішення я розробив набір тестових сценаріїв, які охоплять його основні функціональні можливості. Тестові сценарії, які я пропоную застосувати, дозволять перевірити правильність роботи системи у типових та граничних умовах, а також оцінити її поведінку при введенні некоректних даних [41,42].

Розробку тестових сценаріїв я здійснював беручи до уваги визначені варіанти використання, що були сформовані на етапі аналізу вимог (див. п.п. 1.5 Функціональні та нефункціональні вимоги до програмної системи). Такий підхід забезпечить узгодженість між функціональними вимогами та процесом тестування, а також дозволить перевірити реалізацію буквально кожного ключового сценарія взаємодії між користувачем та системою.

Особливу увага я звертав на тестування процесів, які пов’язані з обробкою параметрів проєкта, виконанням розрахунків, оцінюванням ризиків та формуванням рекомендацій. Крім того, я врахував перевірку на роботу «DecisionModule» – модуля підтримки прийняття рішень, який вважаю центральним елементом системи.

В цілому, до кожного тестового сценарію я включив вхідні дані, опис дій користувача, очікуваний результат та фактичний результат виконання. Це дозволить мені не тільки перевіряти коректність роботи сервісно-орієнтованої програми для планування проєктної діяльності, а й зафіксувати результати проведеного тестування для подальшого їх аналіза.

Отже, я розробив тестові сценарії, які, умовно звичайно, варта поділити на такі групи:

1) позитивні сценарії: які перевіряють коректну роботу програмного рішення при введенні коректних даних;
2) негативні сценарії. Дозволять оцінити поведінку система оцінювання і планування програмних проєктів у випадках введення некоректних або неповних даних;
3) граничні сценарії. За допомогою них я можу перевірити роботу сервісно-орієнтована система планування проєктної діяльності при граничних (крайніх) значеннях параметрів.

Обраний мною підхід дозволить забезпечити всестороннє тестування. Також маю можливість підвищити надійність програмного рішення для підтримки оцінювання проєктів малого та середнього масштаба.

Для узагальнення та структурованого представлення розроблених мною тестових сценаріїв, для наочності та кращого сприйняття, я сформував таблицю тестових випадків. У таблиці 3.1 наведено ключові сценарії тестування, їх вхідні параметри, очікувані результати та результати виконання. Це дозволить мені комплексно оцінити роботоспроможність програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу

Таблиця 3.1 – Тестові сценарії системи

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат
1	Створення проєкта	назва, опис	проєкт створено	-відповідає
2	Введення параметрів	effort=100, resources=5	дані збережено	-відповідає
3	Розрахунок тривалості	effort=100, resources=5	duration=20	-відповідає
4	Оцінка ризику	probability=0.3, impact=5	risk=1.5	-відповідає
5	Формування сценаріїв	набір параметрів	створено сценарії	-відповідає

Продовження таблиці 3.1

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат
6	Формування рекомендації	список сценаріїв	обрано оптимальний варіант	-відповідає
7	Некоректні / неповні дані	resources=0	обробка без помилки	-відповідає

Як видно з представленої вище таблиці 3.1 всі основні функції системи працюють коректно, а також відповідають очікувальним результатам.

Далі, з метою перевірки коректності роботи окремих функцій системи, пропоную застосувати модульне тестування. Для прикладу наведу тест перевірки розрахунку тривалості виконання проєкту.

Лістинг 3.1 – Перевірка розрахунку тривалості виконання проєкту

```
def test_calculate_duration():
    evaluation = Evaluation(100, 5)
    result = evaluation.calculate_duration()
    assert result == 20
```

Даний тест демонструє варіант перевірка на відповідність фактичного до результату очікуваному значенню. Якщо б була невідповідність тест вважається неуспішним, що сигналізує про наявність помилки у реалізації.

Лістинг 3.2 – Очікуваний результат не відповідає фактичному значенню

```
def test_calculate_duration_error():
    evaluation = Evaluation(100, 0)  # некоректні дані (0 ресурсів)
    result = evaluation.calculate_duration()

    # очікуємо, що система поверне 0
    assert result == 20
```

У наведеному прикладі очікуваний результат не відповідає фактичному значенню. При введенні нульового значення кількості ресурсів система повертає 0, однак у тесті задано очікуване значення 20. У результаті тест не проходить, що свідчить саме про невідповідність очікуваного та фактичного результатів. Така ситуація дозволить виявити помилки при формуванні тестових сценаріїв або некоректні очікування щодо роботи програмного продукту.

Отже, враховуючи результати виконання тестових сценаріїв я можу стверджувати, що програмний продукт коректно реалізує основні функціональні можливості. Усі позитивні сценарії були виконані успішно. Це свідчить про правильність реалізації алгоритмів обробки даних та розрахунків.

В свою чергу негативні сценарії показали, що система коректно обробляє помилки введення та не допускає «аварійного» завершення роботи. Граничні сценарії підтвердили стабільність роботи сервісно-орієнтованої системи планування проєктної діяльності навіть у випадку з граничними (крайніми) значеннями параметрів.

Також, особливо хочу акцентувати увагу на те, що «DecisionModule» – модуль підтримки прийняття рішень формує логічно обґрунтовані результати, які відповідають очікуванням користувача.

Таким чином, підсумовуючи, можу сказати, що розроблені тестові сценарії дозволили всебічно перевірити функціональність програмної системи та підтвердити її працездатність. Отримані результати свідчать про готовність системи до впровадження та подальшого використання. Фрагменти програмної реалізації тестування наведені у додатках Б,В,Г. Наведені тести демонструють здійснення перевірки основних функціональних можливостей програмного рішення, зокрема обчислення параметрів проєкту, оцінювання ризиків та роботу модуля підтримки прийняття рішень. Окремо наведено приклад тесту з невідповідністю, що ілюструє можливість виявлення помилок під час тестування.

3.2 Розгортання програмного продукту та системні вимоги

Розгортання програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу буде завершальним етапом його розробки. На цьому етапі програмний продукт переходить із стадії створення до практичного використання. Зараз важливо забезпечити запуск програми. Також не менш важливим є і створення умов для її стабільної та безперебійної роботи [44].

Розроблювана система реалізована у вигляді вебзастосунка. Це суттєво спрощує процес її впровадження. Користувачам не потрібно встановлювати додаткове програмне забезпечення, бо достатньо мати доступ до браузера та мережі Інтернет. Це особливо важливо в сучасних умовах, при діючому військовому стані, коли система може використовуватися різними категоріями користувачів, які можуть мати різні технічні можливості.

При розгортанні системи я врахував її архітектурні особливості, зокрема поділ на клієнтську та серверну частини (п.п. 2.2 Проєктування архітектури програмного рішення). Основна логіка обробки даних (п.п. 2.3 Побудова схем бази даних), включаючи виконання розрахунків та формування рекомендацій, зосереджена на сервері, тоді як інтерфейс користувача забезпечує зручну взаємодію із програмою.

Процес розгортання я умовно поділив на декілька послідовних етапів, на кожному з яких виконується своя функція, а саме:

На I етапі здійснюється підготовка серверного середовища. Це включає встановлення операційної системи та базових компонентів, необхідних для подальшої роботи програмного рішення. При цьому вибір якоїсь конкретної операційної системи не є критичним, оскільки система може функціонувати як у середовищі Windows, так і Linux.

На II етапі виконується встановлення середовища виконання та необхідних бібліотек. Програму я реалізував із використанням сучасних засобів програмування, тому важливо забезпечити сумісність версій програмного забезпечення та коректну роботу залежностей.

На наступному ІІІ етапі є розгортання серверної частини системи. Тут програмний код розміщується на сервері, налаштовуються необхідні параметри запуску та перевіряється коректність роботи основних модулів.

Окрему увагу приділено налаштуванню бази даних. Створюється структура таблиць та визначаються зв’язки між ними. Також забезпечується підключення системи до бази даних. Це дозволить організувати збереження з наступним використанням інформації про проєкти.

Далі на етапі IV виконується налаштування вебсервера, що забезпечує доступ до системи через мережу. Саме на цьому етапі перевіряється коректність відображення інтерфейсу та взаємодія між клієнтською та серверною частинами.

І на останньому, V етапі, здійснюється перевірка працездатності програми вже після розгортання. Виконується тестування основних функцій у реальному середовищі. Це дозволить переконатись в готовності програмного рішення вже до безпосереднього використання замовниками.

Для забезпечення стабільної роботи програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу я визначив мінімальні та рекомендовані системні вимоги. В них я враховую особливості обробки даних та необхідність виконання обчислень у процесі роботи сервісно-орієнтованої системи планування проєктної діяльності.

Отже, вимоги до серверної частини:

1 - операційна система: Windows або Linux;

2 - процесор: двохядерний або вище;

3 - оперативна пам’ять: не менше 4 ГБ;

4 - вільне місце на диску: від 1 ГБ;

5 - встановлене середовище виконання (Python або інше відповідне);

6 - система управління базами даних (SQLite, PostgreSQL чи аналогічна);

7 - наявність веб-сервера.

До вимог до клієнтської частини я відніс:

1 - сучасний веб-браузер (Google Chrome, Mozilla Firefox, Microsoft Edge);

2 - доступ до мережі Інтернет;

З - пристрій із базовими характеристиками (персональний комп’ютер, ноутбук або планшет).

Обрана модель розгортання має як переваги так і недоліки. Щодо переваг, то по-перше, централізоване розміщення серверної частини дозволить спростити адміністрування розробки та забезпечити контроль за її роботою. По друге, користувачі отримують доступ до актуальної версії програмного продукту без необхідності в оновленні програмного забезпечення.

Крім вище наведеного, такий підхід дозволить масштабувати програмне рішення при збільшенні кількості користувачів або обсягів даних. Це є важливим фактором для подальшого розвитку програмного продукту.

Таким чином, процес розгортання програмної системи є, на мою думку, оптимальним та логічним завершенням її розробки. Ним забезпечується перехід до практичного використання. Визначені системні вимоги та організація процесу розгортання дозволять забезпечити стабільну, надійну та зручну роботу системи для користувачів різних категорій та можливостей в тому числі технічних.

3.3 Верифікація програмного рішення

Верифікація розробленого програмного продукту є важливим етапом при оцінці його якості. Вона дозволить мені встановити чи відповідає реалізація початковим вимогам та поставленим задачам. На відміну від тестування, суть якого полягає в перевірці працездатності розробки, основна ціль при верифікації – це підтвердити правильність її проєктування та реалізації [45,46].

В представленій мною роботі верифікація здійснюється через аналіз відповідності функціональних можливостей програмної системи для оцінки і планування програмних проєктів вимогам, визначеним на етапі аналізу предметної області. Особливу увагу я приділив перевірці наскільки програмний продукт забезпечує підтримку прийняття рішень, оскільки це і є його основною відмінністю від існуючих рішень та його основним призначенням.

Таким чином, зазвичай, верифікація проводиться на основі порівняння очікуваних результатів із фактичними результатами роботи розробленого програмного продукту. Для цього я використаю тестові сценарії, які розробив у підрозділі 3.1.2, а також застосую аналіз логіки функціонування основних компонент. Крім цього, буду оцінювати відповідність реалізованої архітектури та структури, мого варіанту програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу, початковим проєктним рішенням. Це дозволить пересвідчитися в тому, що всі компоненти системи працюють узгоджено та забезпечують виконання поставлених задач.

Таким чином, в процесі верифікації я встановив, що розроблена сервісно-орієнтована програмна система планування проєктної діяльності реалізує всі основні функції, визначені на етапі аналізу вимог, зокрема:

- забезпечує створення проєктів,
- введення параметрів,
- обробку параметрів,
- виконання розрахунків,
- оцінювання ризиків,
- формування альтернативних сценаріїв,
- генерацію рекомендацій.

Для більш наочного підтвердження відповідності реалізованого програмного рішення визначеним вимогам я сформував узагальнену таблицю (див. таблицю 3.2), в якій представив взаємозв’язок між функціональними вимогами та результатами їх реалізації.

Таблиця 3.2 – Верифікація функціональних вимог

№	Функціональна вимога	Опис реалізації	Результат
1	Створення проєкта	реалізовано форму введення даних	Виконано
2	Введення параметрів	передбачено інтерфейс введення	Виконано
3	Розрахунок параметрів	реалізовано у класі «Evaluation»	Виконано

Продовження таблиці 3.2

№	Функціональна вимога	Опис реалізації	Результат
4	Оцінка ризиків	використовується клас «Risk»	Виконано
5	Формування сценаріїв	ґенерація варіантів реалізації	Виконано
6	Прийняття рішень	реалізовано у «DecisionModule»	Виконано
7	Відображення результатів	інтерфейс відображає результати	Виконано
8	Обробка помилок	перевірка некоректних даних	Виконано

Відповідно до наведених в таблиці 3.2 даних, програмним рішенням для підтримки оцінювання проєктів малого та середнього масштабу було реалізовано всі визначені функціональні вимоги. Результатами верифікації я підтверджую, що система коректно виконує поставлені задачі та забезпечує необхідний функціонал.

Також я хочу відмітити, що ключовим елементом системи є «DecisionModule» модуль підтримки прийняття рішень, тому його верифікації я приділив окрему увагу. В результаті встановив, що «DecisionModule» коректно опрацьовує вхідні дані, з врахуванням можливих ризиків. Важливим є такою виконання коректного формування обґрунтованих рекомендацій.

Крім вище зазначеного, я верифікував узгодженість між окремими компонентами програмного рішення. Йдеться про взаємодію між інтерфейсом користувача та серверною частиною, та коректність передачі даних.

Отже, результатом проведеної верифікації встановлено, що в цілому робота програми відповідає визначеним вимогам та поставленим завданням, а реалізовані функціональні можливості забезпечують ефективну підтримку процесу прийняття рішень. В загальному ж структура програмної системи оцінювання і планування програмних проєктів узгоджена та стабільна. Таким чином,отриманими результатами я підтверджую готовність розробленого програмного продукту до його практичного використання та застосування в реальних умовах розробки програмного забезпечення.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Безпека життєдіяльності та основи охорони праці є невід'ємною складовою сучасної інженерної діяльності, зокрема в галузі розробки програмного забезпечення. Незважаючи на те, що основна робота програміста або команди розробників відбувається у цифровому середовищі, відсутність належних заходів охорони праці може призвести до професійних захворювань, травм або психологічного вигорання.

У сучасних IT-компаніях особлива увага приділяється організації робочого місця, ергономіці, використанню безпечного обладнання та дотриманню правил електробезпеки. Крім того, розробка та програмного забезпечення передбачає взаємодію з електронними пристроями, серверним обладнанням та локальними мережами, що накладає

Мета цього розділу – узагальнити основні принципи безпеки життєдіяльності та основ охорони праці, з адаптацією до специфіки розробки програмного забезпечення та сучасних інформаційних технологій.

4.1 Безпека життєдіяльності

Безпека життєдіяльності є невід'ємною складовою будь-якої організованої діяльності. Вона охоплює комплекс заходів, спрямованих на захист та збереження життя, здоров'я, психологічного та психічного стану учасників. Для успішної роботи мого програмного продукту необхідно враховувати всі можливі ризики: фізичні, психологічні та інформаційні.

У цьому підрозділі розглянуто основні положення щодо організації безпеки в надзвичайних ситуаціях, методи запобігання надзвичайним ситуаціям, а також інтеграцію цих заходів у роботу цифрової платформи [48-50, 52].

Безпека життєдіяльності і цілому – це комплекс організаційних, технічних та психологічних заходів, спрямованих на захист життя і здоров'я учасників волонтерської діяльності. Для моєї розробки безпека включає: фізичну,

психологічну, інформаційну безпеку та готовність до надзвичайних ситуацій [48,49,51].

Далі в таблиці 4.2 я наведу основні ризики та заходи для їх мінімізації чи уникнення.

Таблиця 4.2. – Основні ризики, приклади ситуацій та заходи щодо їх мінімізації чи уникнення

Категорія ризику	**Приклади ситуацій**	**Заходи на платформі**
Фізичні	Травми під час подій, акцій, падіння	Онлайн-інструкції, алгоритми дій у НС, обов'язковий інструктаж перед подією [48,49]
Психологічні	Стрес, вигорання	Тренінги з управління стресом, рекомендації психологів [51]
Інформаційні	Витік даних користувачів	Двофакторна аутентифікація, шифрування, контроль доступу [52]
Надзвичайні ситуації	Пожежі, медичні інциденти	План евакуації, автоматичні сповіщення, форма для повідомлення про НС [50]

Також варта зазначити про можливості інтеграції безпеки в тому числі і при настанні надзвичайних ситуацій у саме програмне забезпечення. Сюди я відніс:

- модуль довідкових матеріалів (тексти, відео, інфографіка) щодо дій у надзвичайних ситуаціях [51];
- онлайн-тести та сертифікація [48,51];
- система сповіщень: нагадування про правила безпеки [48];
- форма для повідомлення про небезпеку [49].

В свою чергу навчання та інструктажі охоплюють правила безпеки, алгоритми дій у надзвичайних ситуаціях, використання засобів захисту, правила обробки персональних даних [52]. Результати інструктажів зберігаються на платформі для контролю організаторами [51].

4.2. Основи охорони праці

Для забезпечення належного рівня охорони праці в ІТ-сфері важливе дотримання чинного законодавства та державних стандартів [47]. Правова база

визначає обов'язки роботодавців і працівників, встановлює вимоги до організації безпечного робочого середовища та передбачає заходи запобігання надзвичайним ситуаціям. Нижче наведено основні нормативні документи, на які орієнтуються при створенні систем безпеки на робочому місці.

Закон України «Про охорону праці» № 2694-XII [48] Закон встановлює правові, економічні та організаційні основи охорони праці в Україні. Він визначає обов'язки роботодавців та працівників щодо створення безпечних і нешкідливих умов праці, правила розслідування нещасних випадків, вимоги до навчання та інструктажу працівників. Цей закон є основним нормативним документом, на який спираються підприємства та організації при впровадженні систем охорони праці, включно з IT-компаніями та командами розробників програмного забезпечення.

ДСТУ 2293:99 «Охорона праці. Терміни та визначення» [49] Державний стандарт України визначає основні терміни та поняття, що використовуються у сфері охорони праці. Він забезпечує єдине тлумачення ключових понять, таких як ризик, небезпека, засоби індивідуального захисту, професійні захворювання тощо. ДСТУ 2293:99 є важливим для правильного застосування норм законодавства та розробки внутрішніх положень безпеки на підприємстві, у тому числі в IT-сфері.

ДСТУ 9327:2025 «Пожежна безпека будівель і приміщень» [50] Стандарт встановлює вимоги до протипожежного захисту будівель і приміщень, що застосовуються для офісних, виробничих та навчальних приміщень. Він описує заходи запобігання займанням, організацію евакуаційних шляхів, облаштування систем пожежної сигналізації та вогнегасників. Для команд розробників веб-платформ дотримання цього стандарту гарантує безпеку робочого простору та мінімізує ризик надзвичайних ситуацій у приміщеннях із комп'ютерним та серверним обладнанням.

Охорона праці в умовах використання веб-платформ та інформаційних технологій базується на загальних принципах забезпечення безпечних і здорових умов праці, визначених нормативно-правовими актами та державними

стандартами України [48]. Особливістю такої діяльності є переважна робота з персональними комп’ютерами, що зумовлює необхідність урахування специфічних ризиків та факторів впливу на здоров’я користувачів.

Одним із ключових принципів є раціональна організація робочих місць, яка передбачає дотримання вимог ергономіки, достатнього рівня освітлення та належної вентиляції приміщень [49]. Правильна організація робочого середовища сприяє зменшенню фізичного навантаження, запобігає перевтомі та негативному впливу на зір і опорно-руховий апарат користувачів.

Важливе місце займає принцип технічної безпеки, що полягає у дотриманні правил експлуатації персональних комп’ютерів, електричного та периферійного обладнання [51]. У межах веб-платформи реалізація цього принципу можлива шляхом розміщення інформаційних матеріалів і рекомендацій щодо безпечної роботи з технічними засобами.

З метою збереження здоров’я користувачів особлива увага приділяється профілактиці професійних захворювань, пов’язаних із тривалою роботою за комп’ютером. До таких заходів належать регламентовані перерви в роботі, виконання вправ для очей та дотримання режиму праці й відпочинку, що відповідає вимогам чинних нормативних документів [49].

Не менш важливим є принцип навчання та інструктажу з охорони праці, який включає проведення вступних, первинних і повторних інструктажів, а також навчання з надання першої домедичної допомоги [51].

Ефективна реалізація принципів охорони праці забезпечується через контроль та відповідальність, що передбачає ведення журналів інструктажів, фіксацію фактів навчання та контроль за дотриманням вимог [48]. Адміністратори можуть здійснювати такий контроль у цифровому форматі, що підвищує прозорість і зручність управління процесами охорони праці.

Все вище наведене регламентується міжнародними стандартами ISO 45001:2018 — управління охороною праці, аналіз ризиків, навчання персоналу [51] та ISO/IEC 27001:2022 — інформаційна безпека, захист даних користувачів [52].

ВИСНОВКИ

Метою кваліфікаційної роботи на тему «Розробка сервісно-орієнтованого програмного забезпечення для оцінювання та планування програмних проєктів малого і середнього масштабу» є проєктування та розробка програмного рішення основним призначенням якого є підтримка оцінювання і планування програмних проєктів малого та середнього масштабу, яка забезпечуватиме автоматизацію процесів оцінювання та планування на етапі підготовки проєкту до реалізації, а також дозволяє аналізувати та порівнювати альтернативні сценарії його виконання з метою підтримки прийняття обґрунтованих оптимальних управлінських рішень.

Відповідно до мети я окреслив та вирішив наступні задачі:

1 - проаналізував предметну область в напрямку оцінювання та планування сучасних програмних проєктів.

2 - дослідив існуючі підходи, методи та моделі оцінювання параметрів проєктів.

3 - визначив функціональні та нефункціональні вимоги до розробки.

4 - спроєктував архітектуру сервісно-орієнтованої програмного рішення.

5 - розробив логічну та фізичну моделі бази даних.

6 - побудував UML-діаграми (варіантів використання, класів, діяльності, станів).

7 - реалізував основні сервіси та бізнес-логіку програмного рішення.

8 - провів тестування та верифікацію продукту.

9 - виконав порівняльний аналіз.

У першому розділі «Аналіз вимог до програмного сервісно-орієнтованого продукту планування проєктної діяльності» було проведено аналіз вимог до програмного продукту, що стало основою для його подальшого проєктування та розробки. Я проаналізував предметну область, окреслив ключові проблеми та очікування майбутніх користувачів. Окрему увагу буде я приділив формуванню цілей і чіткому визначенню завдання, яке повинна виконувати програма. Крім того, визначився з основними акторами, їх взаємодією із користувачами та сценаріями використання.

У другому розділі «Проєктування та розробка програмного рішення для оцінювання і планування програмних проєктів», я сформую загальну структуру мого програмного продукту, визначу його основні компоненти, логіку їх взаємодії та засоби, які є необхідними для реалізації поставлених завдань.

У третьому розділі «Тестування, впровадження та супровід програмного продукту для підтримки оцінювання проєктів малого та середнього масштабу» розглянуто основні аспекти тестування, впровадження та підтримки програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу. Окрему увагу приділено перевірці роботи системи, особливостям її використання в робочому середовищі, а також питанням подальшого супроводу та оновлення.

На відміну від існуючих рішень, які орієнтовані здебільшого на порівняння готових програмних продуктів, у моїй роботі основну увагу я приділив аналізу та оцінюванню параметрів ІТ-проєкту на етапі його планування. Особливістю системи є підтримка порівняння альтернативних сценаріїв реалізації проєкту та використання сервісно-орієнтованої архітектури, що забезпечує гнучкість і можливість інтеграції з іншими інструментами життєвого циклу програмного забезпечення. Також в моєму програмному рішенні я враховую невизначеність та ризики, і це, в свою чергу, дозволить формувати альтернативні сценарії реалізації проєкту.

Відповідно до вимог зазначених в Методичних вказівках призначених для ознайомлення здобувачів вищої освіти з вимогами, правилами оформлення та оцінювання випускних кваліфікаційних робіт бакалавра для здобувачів, які навчаються за освітньо-професійною програмою «Інженерія програмного забезпечення» однойменної спеціальності, авторства відповідального за випуск Петрика М.Р., д.ф.-м.н., професор; Михалика Д.М., к.т.н., доцент; Цуприк Г.Б., к.т.н., доцент, Бревуса В.М., PhD роботу апробовано: основні результати оприлюднено в доповіді з наступним включенням тез до збірника матеріалів роботи IX Міжнародної студентської науково-технічної конференції «Природничі та гуманітарні науки. Актуальні питання» та їх публікацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 45 с. (https://elartu.tntu.edu.ua/handle/lib/50317)

2. Wiegers Karl, Beatty Joy. Software Requirements. – Microsoft Press, 2013. – 84 p.

3. Harvard Business Review. URL: https://hbr.org/topic/subject/teams

4. MindTools. URL: https://www.mindtools.com/teams.html

5. Virtual Team Builders. URL: https://www.virtualteambuilders.com. 4.Innovation Leader. URL: https://www.innovationleader.com

6. Fast Company – Innovation.

URL: https://www.fastcompany.com/section/innovation

7. MIT Sloan Management Review – Innovation.

URL: https://sloanreview.mit.edu/tag/innovation

8. FlexiProject : офіційний сайт [Електронний ресурс]. – Режим доступу: https:// https://flexi-project.com/

9. Microsoft Project: офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/microsoft-365/project

10. Jira Software: офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/software/jira

11. Wrike : офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.wrike.com

12. OpenProject : офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.openproject.org

13. Taiga : офіційний сайт [Електронний ресурс]. – Режим доступу: https://taiga.io

14. COCOMO II : офіційний ресурс моделі оцінювання вартості програмного забезпечення [Електронний ресурс]. – Режим доступу: https://csse.usc.edu/tools/cocomoii.php

15. Planview : офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.planview.com

16. Oracle Primavera P6 : офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.oracle.com/construction-engineering/primavera/p6

17. IBM. What is Service-Oriented Architecture (SOA)? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/soa

18. Amazon Web Services. What are Microservices? [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/microservices/

19. Amazon Web Services. Implementing Microservices on AWS [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/

20. Malik M. I., Sindhu M. A., Abbasi R. A. Extraction of Use Case Diagram Elements Using Natural Language Processing and Network Science // PLOS ONE. – 2023. – Vol. 18(6). – DOI: 10.1371/journal.pone.0287502.

21. Alyami A., Pileggi S. F., Sohaib O., Hawryszkiewycz I. Seamless Transformation from Use Case to Sequence Diagrams // PeerJ Computer Science. – 2023. – Vol. 9. – DOI: 10.7717/peerj-cs.1444.

22. Kautz O., Rumpe B., Wachtmeister L. Semantic Differencing of Use Case Diagrams // Journal of Object Technology. – 2022. – Vol. 21(3).

23. PlantUML. Use Case Diagram Syntax and Features [Електронний ресурс]. – Режим доступу: https://plantuml.com/use-case-diagram

24. Веб-розробка [Електронний ресурс]. Режим доступу: https://skillsbuild.org/uk/adult-learners/explore-learning/web-developer

25. Функціональні та нефункціональні вимоги (з прикладами) [Електронний ресурс]. Режим доступу: https://visuresolutions.com/uk/alm-guide/https://visuresolutions.com/uk/alm-guide/функціональні-та-нефункціональні-вимоги/

26. Dennis A., Wixom B. H., Tegarden D. Systems Analysis and Design. – 8th ed. – Hoboken : John Wiley & Sons, 2021. – 656 p.

27. Kerzner H. Project Management: A Systems Approach to Planning, Scheduling, and Controlling. – 13th ed. – Hoboken : John Wiley & Sons, 2022. – 848 p.

28. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK® Guide). – 7th ed. – Newtown Square : PMI, 2021. – 370 p.

29. Кантор І. Сучасний підручник з JavaScript [Електронний ресурс]. – Режим доступу: uk.javascript.info

30. JavaScript Guide [Електронний ресурс]. – Mozilla Developer Network. – Режим доступу: developer.mozilla.org

31. React Documentation [Електронний ресурс]. – Режим доступу: react.dev

32. Node.js Documentation [Електронний ресурс]. – Режим доступу: nodejs.org

33. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: postgresql.org

34. PlantUML Language Reference Guide [Електронний ресурс]. – Режим доступу: plantuml.com

35. UML Diagrams [Електронний ресурс]. – Режим доступу: uml-diagrams.org

36. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

37. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

38. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring

and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

39. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72-76

40. Yaroslav Kotov, Evhenia Yavorska, Halyna Tsupryk, Róża Dzierżak 1 , Oleksandr Reshetnik, Viktoriia Bokovets (2025) Evaluating interoperability and data quality in FHIR-based AI assessment pipelines. Proc. SPIE 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025, 140091F (30 December 2025) https://doi.org/10.1117/12.3100561

41. ISTQB. Certified Tester Foundation Level Syllabus Version 4.0.1 [Електронний ресурс]. – Режим доступу: https://www.istqb.org/

42. Thompson G., Morgan P., Samaroo A. Software Testing: An ISTQB-BCS Certified Tester Foundation Level Guide (CTFL v4.0). – 5th ed. – Swindon : BCS Learning & Development Ltd, 2024. – 288 p.

43. Stapp L., Roman A., Pilaeten M. ISTQB® Certified Tester Foundation Level: A Self-Study Guide Syllabus v4.0. – Cham : Springer Nature Switzerland, 2024. – 406 p.

44. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/

45. Прокоф’єв І. Метод статичного аналізу якості коду з допомогою машинного навчання // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2025. – № 1. – С. 115–121.

46. Гринько А. М. Дослідження впливу SATD на якість програмного коду [Електронний ресурс]. – Режим доступу: https://openarchive.nure.ua/entities/publication/d60623a8-d842-4d06-a3cd-e1bf381ced61

47. Охорона праці : Навчальний посібник [Текст] Геврик Є.О. - Ельга: Ніка-Центр. - 2003. – 279 с.

48. Закон України «Про охорону праці» № 2694-XII. – Відомості Верховної Ради України, 1992. – № 24. – Ст. 194.

49. ДСТУ 2293-99. Охорона праці. Терміни та визначення основних понять. – Київ: Держспоживстандарт України, 1999. – 25 с.

50. ДСТУ 9327:2025. Пожежна безпека. Проєктування огороджувальних конструкцій. – Київ: Держспоживстандарт України, 2025. – 32 с.

51. ISO 45001:2018. Occupational health and safety management systems — Requirements with guidance for use. – Geneva: ISO, 2018. – 40 p.

52. ISO/IEC 27001:2022. Information technology — Security techniques — Information security management systems — Requirements. – Geneva: ISO/IEC, 2022. – 45 p.

53. Навчально-методичний посібник до практичних заняттяздисципліни«Безпека життєдіяльності, основи охорони праці» для студентівосвітньогоступеня „бакалавр" усіх спеціальностей та форм навчання / Укладачі : О. Я. Гурик,І. Б. Окіпний, В. С. Сенчишин, С. Ю. Мариненко, О. І. Король. Тернопіль:ТНТУ 70імені Івана Пулюя, 2025. 123 с. URL: http://elartu.tntu.edu.ua/handle/lib/48496

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.41
Кікцьо Т.–ст. гр. СПс-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ ДЛЯ ПІДТРИМКИ ОЦІНЮВАННЯ ТА ПЛАНУВАННЯ ІТ-ПРОЄКТІВ

Науковий керівник: к.т.н., доцент Цуприк Г. Б.

Kiktso T.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A SOFTWARE SOLUTION FOR SUPPORTING ESTIMATION AND PLANNING OF IT PROJECTS

Supervisor: PhD, Associate Professor H. B.Tsupryk

Ключові слова: розробка, програмне забезпечення, прийняття рішення.
Keywords: Development, Software, Decision Making.

Метою кваліфікаційної роботи є проєктування та розробка сервісно-орієнтованого програмного рішення для підтримки оцінювання і планування програмних проєктів малого та середнього масштабу, яке забезпечуватиме автоматизацію процесів оцінювання та планування на етапі підготовки проєкту до реалізації, а також дозволить аналізувати та порівнювати альтернативні сценарії його виконання з метою підтримки прийняття обґрунтованих оптимальних управлінських рішень. А це, в стою чергу, дасть можливість зацікавленим сторонам зрозуміти потенційну рентабельність інвестицій, прибутковість проєкту, орієнтуватись в реальних термінах повернення коштів при його потенційній успішній реалізації.

Об'єктом дослідження є програмні проєкти, процес їх оцінювання та планування з метою визначення на відповідність цілям та вимогам замовника та попереднього виявлення та оцінки потенційних ризиків та їх критичність, попереднє виявлення можливих непередбачуваних ситуацій, для отримання можливості відповідальній особі приймати оптимальні та ефективні рішення щодо вибору альтернативних варіантів чи, навіть, доцільності реалізації проєкту.

Предметом дослідження є методи, моделі та сервісно-орієнтовані програмні засоби підтримки прийняття рішень при оцінюванні та плануванні програмних проєктів, які забезпечують автоматизацію розрахунків основних параметрів проєкту, а також можливість аналізу та порівняння альтернативних варіантів.

Враховуючи специфіку програмного продукту для планування проєктної діяльності, який орієнтований на підтримку прийняття рішень у процесі оцінювання та планування програмних проєктів на етапі їх підготовки з урахуванням ризиків, ресурсів та альтернативних сценаріїв реалізації, було обрано ітеративний підхід до розробки з елементами гнучких методологій. Такий вибір для мене був оптимальним за рахунок того, що вимоги до подібних систем можуть змінюватись вже під час, в процесі розробки, зокрема щодо методів оцінювання, моделей аналізу та способів представлення результатів.

Враховуючи вимоги щодо гнучкості, масштабованості та можливості інтеграції з іншими системами, для розроблюваного продукту я прийняв рішення обрати

найоптимальнішу, сервісно-орієнтовану архітектуру, яка дозволить незалежно розвивати модуль підтримки прийняття рішень, що є ключовим елементом системи.

Для реалізації серверної частини я обрав мову програмування Python, а в якості фреймворку я зупинив свій вибір на FastAPI. Для збереження даних я буду використовувати систему керування базами даних PostgreSQL. Клієнтську частину мого програмного рішення для підтримки оцінювання проєктів малого та середнього масштабу я реалізую з використанням «стандартних» веб-технологій, зокрема HTML, CSS та JavaScript. При реалізації я обрав та використав парадигми ООП. Інтерфейс програмного рішення я реалізую у вигляді вебзастосунку, що забезпечить доступ до функціоналу через браузер. Для обміну даними між інтерфейсом та серверною частиною я використав API. Для забезпечення надійності та коректності роботи розроблюваної програмної системи обрав комплексний підхід до тестування.

Обраний стек технологій є взаємодоповнюючим, оскільки мова Python забезпечить реалізацію аналітичної логіки, FastAPI – ефективну взаємодію між компонентами системи, а система керування базами даних PostgreSQL – надійне збереження даних. Веб-технології в свою чергу дадуть змогу отримати зручний доступ всіх користувачів до функціоналу системи.

На відміну від існуючих систем управління проєктами, запропоноване рішення орієнтоване не лише на організацію виконання завдань, а передусім на етап попереднього оцінювання та планування. Особливістю системи є підтримка порівняння альтернативних сценаріїв реалізації проєкту та використання сервісно-орієнтованої архітектури, що забезпечує гнучкість і можливість інтеграції з іншими інструментами життєвого циклу програмного забезпечення.

Таким чином, розробка такого програмного рішення відповідатє сучасним вимогам інженерії програмного забезпечення та моїй спеціальності (121 Інженерія програмного забезпечення) та є актуальною саме для малих та середніх команд, забезпечуючи поєднання архітектурних рішень, сервісно-орієнтованого підходу та інноваційних інструментів підтримки життєвого циклу програмного забезпечення.

Література:

1. Шапа Н. М., Вечеров В. Т. Огляд наукових методів і підходів проектного управління та оцінки вартості проекту // Економічний простір. – 2023. – Режим доступу: https://prostir.pdaba.dp.ua/index.php/journal/article/view/1364

2. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

3. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

4. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

Додаток Б

Лістинг Б.1 – UML-діаграма варіантів використання системи управління проєктами

Цей фрагмент демонструє побудову UML-діаграма варіантів використання системи управління проєктами

```
@startuml
left to right direction
skinparam shadowing false

actor "Керівник проєкту" as PM
actor "Аналітик" as Analyst
actor "Замовник проєкту" as Customer
actor "Адміністратор" as Admin
actor "Модуль підтримки\nприйняття рішень" as Decision

rectangle "Система оцінювання та планування проєктів" {

usecase "Створення проєкту" as UC1
usecase "Введення параметрів" as UC2
usecase "Вибір методу оцінювання" as UC3
usecase "Розрахунок параметрів" as UC4

usecase "Оцінка ризиків" as UC5
usecase "Формування сценаріїв" as UC6
usecase "Розподіл ресурсів" as UC7
usecase "Аналіз змін" as UC8

usecase "Порівняння варіантів" as UC9
usecase "Формування рекомендацій" as UC10
usecase "Прийняття рішення" as UC11

usecase "Формування звітів" as UC12
usecase "Історичні дані" as UC13
usecase "Управління користувачами" as UC14
}

' Керівник
PM --> UC1
PM --> UC2
PM --> UC3
PM --> UC4
PM --> UC5
PM --> UC6
PM --> UC7
PM --> UC8
PM --> UC9
```

```
PM --> UC10
PM --> UC11
PM --> UC12

' Аналітик
Analyst --> UC2
Analyst --> UC5
Analyst --> UC6
Analyst --> UC7
Analyst --> UC8
Analyst --> UC9
Analyst --> UC10

' Замовник
Customer --> UC11
Customer --> UC12

' Адміністратор
Admin --> UC13
Admin --> UC14

' Модуль прийняття рішень
Decision --> UC4
Decision --> UC5
Decision --> UC6
Decision --> UC7
Decision --> UC8
Decision --> UC9
Decision --> UC10

@enduml
```

Лістинг Б.2 – Activity Diagram -діаграма оцінювання та прийняття рішень

Цей фрагмент демонструє логіку роботи системи як інструменту для підтримки прийняття оптимальних і найбільш ефектьивних рішень, що дасть змогу забезпечити обґрунтованість вибору варіанту реалізації програмного проєкту.

```
@startuml
start

:Введення параметрів проєкту;

if (Дані повні?) then (так)
:Запуск оцінювання;
else (ні)
:Повідомлення про помилку;
stop
endif

:Розрахунок параметрів;
:Оцінка ризиків;
:Формування сценаріїв;

if (Є достатньо варіантів?) then (так)
:Порівняння сценаріїв;
else (ні)
:Уточнення даних;
stop
endif

:Формування рекомендацій;

if (Рішення прийнято?) then (так)
:Завершення процесу;
else (ні)
:Повторний аналіз;
endif

stop
@enduml
```

Лістинг Б.3 – Діаграма архітектури (Component Diagram)

Тут демонструється узагальнена структура розробки та взаємодія між її основними компонентами у вигляді компонентної діаграми, яка відображатиме розподіл функціональності між сервісами та їх взаємозв’язки.

```
@startuml
skinparam componentStyle rectangle

actor "Керівник проєкту" as PM
actor "Аналітик" as Analyst
actor "Замовник" as Client
actor "Адміністратор" as Admin

rectangle "Клієнтський рівень" {
component "Веб-інтерфейс" as UI
}

rectangle "Сервер прикладної логіки" {

component "Сервіс управління\nпроєктами" as ProjectService
component "Сервіс аналітики\nта оцінювання" as AnalyticsService
component "Модуль підтримки\nприйняття рішень" as DecisionModule
component "Сервіс користувачів" as UserService

}

database "База даних" as DB

' Взаємодія акторів
PM --> UI
Analyst --> UI
Client --> UI
Admin --> UI

' Взаємодія інтерфейсу з backend
UI --> ProjectService
UI --> AnalyticsService
UI --> UserService

' Взаємодія сервісів
ProjectService --> DB
AnalyticsService --> DecisionModule
DecisionModule --> DB
UserService --> DB

@enduml
```

Лістинг Б.4 – ER-діаграму логічної моделі даних розроблюваної сервісно-орієнтованої програми для планування проєктної діяльності

Далі представлення структури бази даних та зв’язків між її основними сутностями.

```
@startuml
entity User {
+id : int
name : string
email : string
password : string
role : string
}

entity Project {
+id : int
name : string
description : string
created_at : date
user_id : int
}

entity ProjectParameters {
+id : int
effort : float
resources : int
budget : float
deadline : date
project_id : int
}

entity Evaluation {
+id : int
effort_estimated : float
duration : float
resources_used : int
date : date
project_id : int
}

entity Risk {
+id : int
name : string
probability : float
impact : float
description : string
project_id : int
}

entity Scenario {
```

```
+id : int
name : string
description : string
efficiency : float
project_id : int
}

entity Recommendation {
+id : int
description : string
scenario_id : int
project_id : int
created_at : date
}

User ||--o{ Project
Project ||--|| ProjectParameters
Project ||--o{ Evaluation
Project ||--o{ Risk
Project ||--o{ Scenario
Scenario ||--|| Recommendation
Project ||--o{ Recommendation

@enduml
```

Лістинг Б.5 – UML-діаграма класів, яка дозволяє відобразити основні класи системи, їх атрибути, методи та взаємозв’язки між ними.

Далі представлення структури програмної системи та взаємодії між її складовими задопомогою UML-діаграми класів.

```
@startuml

class User {
+id : int
+name : string
+email : string
+password : string
+role : string
+login()
+logout()
}

class Project {
+id : int
+name : string
+description : string
+createdAt : date
+createProject()
+updateProject()
}

class ProjectParameters {
+id : int
+effort : float
+resources : int
+budget : float
+deadline : date
+updateParameters()
}

class Evaluation {
+id : int
+effortEstimated : float
+duration : float
+resourcesUsed : int
+calculate()
}

class Risk {
+id : int
+name : string
+probability : float
+impact : float
+assessRisk()
```

```
}

class Scenario {
+id : int
+name : string
+description : string
+efficiency : float
+compare()
}

class Recommendation {
+id : int
+description : string
+generate()
}

class DecisionModule {
+analyzeData()
+generateScenarios()
+makeRecommendation()
}

User "1" -- "0..*" Project
Project "1" -- "1" ProjectParameters
Project "1" -- "0..*" Evaluation
Project "1" -- "0..*" Risk
Project "1" -- "0..*" Scenario
Scenario "1" -- "1" Recommendation

Evaluation --> DecisionModule
Scenario --> DecisionModule
Risk --> DecisionModule
DecisionModule --> Recommendation

@enduml
```

Додаток В -Фрагменти програмної реалізації системи

Тут наведено розширені фрагменти програмної реалізації основних компонентів системи, що демонструють підхід до побудови класів та реалізації логіки оцінювання і підтримки прийняття рішень.

Лістинг В.1 – Клас Project

```
class Project:
    def __init__(self, name, description, user_id):
        self.name = name
        self.description = description
        self.user_id = user_id
        self.parameters = None
        self.evaluations = []
        self.risks = []

    def set_parameters(self, parameters):
        self.parameters = parameters

    def add_evaluation(self, evaluation):
        self.evaluations.append(evaluation)

    def add_risk(self, risk):
        self.risks.append(risk)
```

Лістинг В.2 – Клас Evaluation

```
class Evaluation:
    def __init__(self, effort, resources):
        self.effort = effort
        self.resources = resources
        self.duration = 0

    def calculate_duration(self):
        if self.resources == 0:
            self.duration = 0
        else:
            self.duration = self.effort / self.resources
        return self.duration
```

Лістинг В.3 – Клас Risk

```
class Risk:
    def __init__(self, name, probability, impact):
        self.name = name
        self.probability = probability
        self.impact = impact

    def calculate_risk_level(self):
        return self.probability * self.impact
```

Лістинг В.4 – Клас Scenario

```
class Scenario:
    def __init__(self, name):
        self.name = name
        self.score = 0

    def update_score(self, score):
        self.score = score
```

Лістинг В.5 – Модуль DecisionModule

```
class DecisionModule:

    def calculate_total_risk(self, risks):
        return sum(r.calculate_risk_level() for r in risks)

    def evaluate_project(self, evaluation, risks):
        duration = evaluation.calculate_duration()
        risk_value = self.calculate_total_risk(risks)
        return duration + risk_value

    def generate_scenarios(self, project, evaluation):
        base_score = self.evaluate_project(evaluation,
project.risks)

        scenario1 = Scenario("Базовий")
        scenario1.update_score(base_score)

        scenario2 = Scenario("Більше ресурсів")
        scenario2.update_score(base_score * 0.8)

        return [scenario1, scenario2]

    def make_recommendation(self, scenarios):
        return min(scenarios, key=lambda s: s.score)
```

Лістинг В.6 – Приклад використання

```
project = Project("Система", "Аналіз", 1)
evaluation = Evaluation(100, 5)

project.add_risk(Risk("Технічний", 0.3, 5))
project.add_risk(Risk("Організаційний", 0.2, 4))

decision = DecisionModule()
scenarios = decision.generate_scenarios(project, evaluation)
best = decision.make_recommendation(scenarios)
```

Додаток Г – Фрагменти тестування програмного рішення

У даному додатку наведено приклади тестування основних компонентів програмної системи, що дозволяють перевірити коректність реалізації функціональних можливостей та обробку різних типів вхідних даних.

Лістинг Г.1 – Тест розрахунку тривалості проєкту (коректні дані)

```
def test_calculate_duration():
    evaluation = Evaluation(100, 5)
    result = evaluation.calculate_duration()
    assert result == 20
```

Лістинг Г.2 – Тест розрахунку тривалості (граничний випадок)

```
def test_calculate_duration_zero_resources():
    evaluation = Evaluation(100, 0)
    result = evaluation.calculate_duration()
    assert result == 0
```

Лістинг Г.3 – Тест оцінювання ризику

```
def test_risk_calculation():
    risk = Risk("Технічний", 0.3, 5)
    result = risk.calculate_risk_level()
    assert result == 1.5
```

Лістинг Г.4 – Тест роботи модуля прийняття рішень

```
def test_decision_module():
    evaluation = Evaluation(100, 5)
    risks = [
        Risk("Технічний", 0.3, 5),
        Risk("Організаційний", 0.2, 4)
    ]

    decision = DecisionModule()
    result = decision.evaluate_project(evaluation, risks)

    assert result > 0
```

Лістинг Г.5 – Тест формування рекомендації

```
def test_make_recommendation():
    scenario1 = Scenario("A")
    scenario1.update_score(10)

    scenario2 = Scenario("B")
    scenario2.update_score(5)

    decision = DecisionModule()
    best = decision.make_recommendation([scenario1, scenario2])

    assert best.name == "B"
```

Лістинг Г.6 – Тест з невідповідністю (виявлення помилки)

```
def test_calculate_duration_error():
    evaluation = Evaluation(100, 0)
    result = evaluation.calculate_duration()

    # некоректне очікування
    assert result == 20”
```