МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення інформаційної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення»

здобувача Ляскович Ангеліни Олександрівни
(ПІБ, підпис)

Науковий керівник: д.т.н., професор Шевченко К.Л.
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: ____к.е.н., доцент Тішков Б.О.
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____Устенко С.В.
«____» _____202_ р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
«____» _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Ляскович Ангеліни Олександрівні
(прізвище, ім'я, по батькові)
очної (денної) форми навчання
очної (денної), заочної, дистанційної

на підготовку кваліфікаційної магістерської роботи
на тему: «Розроблення інформаційної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення»

Тему затверджено наказом ректора Університету від 07» березня 2025 р. № 464-ст

Кваліфікаційна магістерська робота виконується на матеріалах
_ наукових публікацій та відкритих джерел з мережі інтернет

План кваліфікаційної магістерської роботи

Розділ І _ Дослідження та аналіз підходів до створення інформаційних систем управління ролями та відповідальністю в процесі розробки програмного забезпечення
Розділ ІІ Характеристика інформаційних управляючих систем та методи і моделі
Розділ ІІІ Розроблення проектних рішень та їх реалізація
Об'єкт дослідження: процеси управління ролями та відповідальністю в межах реалізації ІТ-проєктів
Предмет дослідження: методи та інформаційні технології для підтримки процесів управління проєктними ролями
Мета кваліфікаційної магістерської роботи: спроєктувати інформаційну управляючу систему для підтримки процесу розподілу ролей і відповідальності в проєктній діяльності з використанням інтелектуальних методів аналізу.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі І Дослідити предметну області управління ролями та відповідальністю в процесі розробки програмного забезпечення. Проаналізувати сучасні підходи до розподілу ролей у командах, оцінити можливості існуючих інформаційних систем, виявити їхні обмеження, а також обґрунтувати доцільність застосування інтелектуальних моделей для прогнозування ризиків, виявлення відхилень і підтримки прийняття рішень.

У розділі ІІ Розробити проєктні рішення щодо створення інтелектуальної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення, зокрема: визначити вимоги до системи, змоделювати її поведінку та структуру з використанням сучасних методологій проєктування інформаційних систем; дослідити інструменти інтелектуального аналізу даних, методи та моделі, які забезпечують призначення ролей, прогнозування ризиків перевантаження та виявлення порушень політик відповідальності.

У розділі ІІІ Розробити проєктні рішення щодо реалізації інтелектуальної системи управління ролями та відповідальністю в процесі розробки програмного забезпечення, зокрема в частині проєктування бази даних, засобів інтелектуального аналізу дій користувачів, програмних модулів і компонентів системи, а також вибору відповідного технічного забезпечення. Реалізувати контрольний приклад функціонування запропонованих рішень із моделюванням ключових сценаріїв прогнозування ризиків соціальної взаємодії.

Завдання підготував
науковий керівник _____ _Шевченко Констянтин Леонідович
(підпис) *(ініціали, прізвище)*

«____» _____ 202_ р.

Завдання одержав
здобувач _____ Ляскович Ангеліна Олександрівна
(підпис) *(ініціали, прізвище)*

«____» _____ 202_ р.

Відгук

про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в економіці»
освітньо-професійної програми «Інформаційні управляючі системи і технології»
Ляскович Ангеліни Олександрівни
(прізвище, ім'я, по батькові)
на тему «Розроблення інформаційної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення»

1. Актуальність теми: тема роботи досить актуальна, особливо при розробці великих проєктів, що потребують залучення значної кількості виконавців

2. Позитивні риси кваліфікаційної магістерської роботи: в роботі пропонується використання системного підходу до аналізу ланцюжків та шляхів реалізації окремих функціональних завдань з врахуванням ступеню завантаженості виконавців

3. Наявність самостійних розробок автора: автор зробив спробу формування архітектури інформаційної системи, вибору методів аналізу поведінкових та структурних даних

4. Цінність теоретичних висновків та практичних рекомендацій: цінність роботи, у випадку її доведення до практичного впровадження, полягає у можливості підвищення ефективності управління командою розробників шляхом контролю за виконанням поставлених задач та своєчасного реагування на ризики у реальних організаційних умовах

5. Наявність недоліків: Робота має певні стилістичні неточності . Представлене проєктне рішення потребує подальшого вдосконалення й розвитку

6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: в цілому кваліфікаційна магістерська робота Ляскович А.О. відповідає вимогам до магістерських робіт за спеціальністю 122 – комп'ютерні науки та може бути рекомендована до захисту

Науковий керівник:
професор кафедри ІСЕ, д.т.н., доцент _____ Шевченко К.Л.

Рецензія

на кваліфікаційну магістерську роботу
здобувача вищої освіти
Ляскович Ангеліна Олександрівна

Тема Розроблення інформаційної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення

В умовах стрімкого розвитку проєктної діяльності у сфері розробки програмного забезпечення дедалі більшого значення набуває ефективне управління командною взаємодією, зокрема через чітке визначення ролей, зон відповідальності та контроль виконання завдань. Водночас традиційні підходи до управління не враховують динамічних змін у навантаженні, поведінкових особливостей учасників команди та ризиків порушення відповідальності, що негативно впливає на результативність проєктів і ускладнює прийняття оперативних управлінських рішень. Це зумовлює актуальність створення інтелектуальної інформаційної системи, здатної не лише фіксувати ролі та дії користувачів, а й здійснювати аналіз прихованих відхилень, прогнозування потенційних критичних ситуацій і адаптацію до змін у структурі проєктної команди.

Якість проведеного дослідження свідчить про високий рівень опрацювання теми, обґрунтованість вибору методів та системний підхід до вирішення поставлених завдань. Автором проведено глибокий аналіз актуальних проблем управління ролями та відповідальністю в умовах командної розробки програмного забезпечення, що супроводжується впровадженням інтелектуальних інструментів аналізу даних. Представлені рішення мають практичне значення, а логічна послідовність викладу, обґрунтованість висновків і використання сучасних методів моделювання засвідчують належну якість наукової роботи.

До позитивних рис кваліфікаційної магістерської роботи можна віднести актуальність обраної теми, що відповідає сучасним тенденціям у сфері розробки інформаційних систем із застосуванням інтелектуальних технологій. Робота вирізняється цілісною структурою, логічністю викладу матеріалу та наявністю чітко сформульованих цілей і завдань дослідження. Відзначається високий рівень теоретичної обґрунтованості, поєднаний із практичною реалізацією, що свідчить про володіння автором фаховими знаннями. Застосування сучасних підходів до аналізу, моделювання та оцінювання системи, а також наявність візуалізацій, діаграм і результатів експериментів підвищують наукову і практичну цінність роботи.

Бажано було б надати більше уваги питанням масштабованості та інтеграції системи в реальні умови розробки програмного забезпечення, з урахуванням вимог до безпеки, конфіденційності та доступності..

Водночас це не зменшує загальної якості кваліфікаційної магістерської роботи, яка відзначається високим рівнем теоретичної та практичної реалізації поставлених завдань.

Практична значимість висновків і рекомендацій полягає в можливості їх застосування під час розробки та впровадження інформаційних систем управління ролями й відповідальністю в командній діяльності. Запропоновані підходи можуть бути використані для підвищення ефективності управління проєктами, покращення контролю за виконанням завдань і своєчасного виявлення ризиків порушення відповідальності. Рекомендації щодо використання інтелектуального аналізу логів дій та прогнозування навантаження мають прикладну цінність і здатні сприяти підвищенню продуктивності команд у сфері розробки програмного забезпечення.

Український державний університет
імені Михайла Драгоманова,
завідувач кафедри інформаційних технологій
і програмування факультету математики,
інформатики та фізики УДУ імені Михайла Драгоманова
к.пед.н, доцент Василь ЄФИМЕНКО

ЗМІСТ

ВСТУП

Актуальність проблеми. У сучасних умовах проєктної діяльності, зокрема в сфері розробки програмного забезпечення, зростає потреба в ефективному управлінні командною взаємодією, де чітке визначення ролей, відповідальності та контролю за виконанням завдань набуває особливого значення. Традиційні системи управління не враховують динаміку розподілу навантаження, зміни у поведінці учасників чи ризики порушення відповідальності, що знижує ефективність проєктів та ускладнює оперативне прийняття рішень. Актуальність проблеми зумовлена необхідністю створення інтелектуальної інформаційної системи, здатної не лише документувати ролі та дії, але й виявляти приховані відхилення, прогнозувати критичні ситуації та адаптуватися до змін у структурі команди.

Аналіз останніх досліджень і публікацій. Управління ролями у проєкті розглядається як важливий аспект забезпечення успішної реалізації програмних проєктів, адже ефективна команда формується на основі чіткого розподілу повноважень між учасниками з різними функціональними ролями, зокрема керівниками, розробниками, тестувальниками та іншими спеціалістами (Созинова, 2024). У науковій літературі підкреслюється важливість застосування таких інструментів, як управління відхиленнями (Подзігун, 2023), та проєктного підходу, особливо у сферах з чіткими часовими обмеженнями (Присяжнюк, 2022). Проте дослідники вказують на певні бар’єри щодо ефективної реалізації таких підходів, зокрема недостатній рівень науково обґрунтованих рішень (Присяжнюк, 2022). Водночас у контексті змін, спричинених пандемією COVID-19, управління командою потребує нових підходів до адаптації та забезпечення гнучкості процесів (Шинкарук, 2021). На тлі цього зростає інтерес до створення сучасних інформаційних систем, які б поєднували інструменти управління ролями з інтелектуальними модулями аналізу, як засобу підвищення ефективності прийняття рішень у складних організаційних умовах.

Наукова проблема, що лежить в основі даного дослідження, полягає в необхідності створення адаптивної інформаційної управляючої системи, здатної не лише автоматизувати розподіл ролей і відповідальності у проєктній діяльності, а й виявляти приховані ризики, аномалії та відхилення в реальному часі за допомогою інтелектуального аналізу даних. У сучасних умовах складності командної взаємодії, збільшення кількості децентралізованих процесів і потреби в оперативному управлінському реагуванні, традиційні підходи до контролю відповідальності в межах проєктів виявляються недостатньо ефективними. Це зумовлює потребу в поєднанні формалізованого моделювання (наприклад, на основі методології SysML) з методами машинного навчання, графового аналізу та процесного майнінгу.

Результати дослідження можуть бути використані для вдосконалення процесів управління IT-проєктами, формування інтелектуальних модулів у корпоративних інформаційних системах, побудови навчальних кейсів з управління відповідальністю в командній роботі. Запропонована концепція системи може знайти застосування як в освітньому процесі (для навчання проєктному менеджменту та аналізу ролей), так і в реальних розробницьких середовищах, де важливо не лише фіксувати фактичні дії, а й робити висновки щодо потенційних порушень і перевантажень.

Особистий внесок автора полягає у формуванні архітектури інформаційної системи, виборі методів аналізу поведінкових та структурних даних, розробленні логічної моделі бази даних, побудові алгоритмів виявлення відхилень і аномалій, а також реалізації прототипу інтелектуального модуля в середовищі Python.

Мета дослідження: спроєктувати інформаційну управляючу систему для підтримки процесу розподілу ролей і відповідальності в проєктній діяльності з використанням інтелектуальних методів аналізу.

Завдання: розробити архітектуру системи, спроєктувати базу даних, реалізувати інтелектуальний модуль, провести аналіз і виявлення відхилень у розподілі відповідальності

Об'єкт дослідження: процеси управління ролями та відповідальністю в межах реалізації ІТ-проєктів.

*Предмет дослідження***:** методи та інформаційні технології для підтримки процесів управління проєктними ролями.

*Методи дослідження***:** системний аналіз, моделювання SysML, колаборативна фільтрація, графовий аналіз, Process Mining, машинне навчання.

Теоретична, методична та практична значущість отриманих результатів полягає у формалізації підходу до проєктування інформаційно-управляючої системи управління ролями та відповідальністю в командній діяльності шляхом поєднання методології SysML із сучасними інструментами інтелектуального аналізу даних. Методична значущість полягає у запропонованій структурі побудови ІУС, яка може бути адаптована до інших проєктних середовищ. Практична цінність полягає у можливості застосування створеної системи для підвищення ефективності управління командою, контролю за виконанням обов'язків і своєчасного реагування на ризики у реальних організаційних умовах.

Інформаційна база дослідження охоплює праці вітчизняних і зарубіжних науковців з питань інформаційних систем, управління проєктами, аналізу даних, документи міжнародних стандартів у сфері проєктного менеджменту, статистичні матеріали, офіційні звіти ІТ-компаній, а також результати власного моделювання та експериментального тестування прототипу системи.

*Структура роботи. З*агальний обсяг роботи становить 97 сторінок друкованого тексту, 31 рисунок на 27 сторінках, 15 таблиць на 15 сторінках, 1 додаток на 2 сторінках. Список використаних джерел налічує 50 найменування.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ РОЛЯМИ ТА ВІДПОВІДАЛЬНІСТЮ В ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Аналіз існуючих інформаційних систем управління ролями та відповідальністю при розробці програмного забезпечення

1.1.1 Дослідження предметної області

У сучасних умовах стрімкого розвитку цифрових технологій та поширення гнучких методологій розробки важливу роль відіграє управління ролями та відповідальністю в проєктах створення програмного забезпечення важливим стає. Зростання складності ПЗ, мультидисциплінарність команд, розподіленість учасників по різних часових і географічних зонах вимагають чітко регламентованого розподілу повноважень і зон відповідальності. Без прозорої системи ролей у рази зростають ризики дублювання функцій, втрати інформації, порушення строків і якості.

Особливої актуальності ця проблема набуває в умовах інтенсивної цифровізації бізнесу, коли компанії прагнуть автоматизувати критичні процеси розробки, впровадження та обслуговування IT-рішень. У таких сценаріях необхідна інтеграція між технічними й управлінськими підсистемами, де чітко визначені ролі забезпечують цілісність та безпеку виконання задач. Водночас зростають вимоги до аудиту, контролю змін і відповідності нормативним стандартам – що також підвищує значення системного управління ролями [1].

Реалії сьогодення – зокрема тенденції до масштабування, впровадження DevOps-практик, використання хмарних інфраструктур – вимагають переходу від неформального, ручного управління до централізованих, автоматизованих інформаційних систем. Такі рішення не лише покращують дисципліну виконання, але й формують надійну основу для аналізу, прогнозування і прийняття рішень на основі ролей, прав і відповідальностей у динамічному середовищі розробки ПЗ.

У процесі розробки програмного забезпечення поняття «роль» і «відповідальність» є фундаментальними для побудови ефективної команди та організації управління [2].

Під роллю розуміють узагальнений набір обов'язків, повноважень і очікувань, який визначає, яку функцію виконує учасник у межах проєкту. Наприклад, до типових ролей відносяться [3, 4]:

- аналітик;
- архітектор;
- розробник;
- тестувальник;
- адміністратор баз даних;
- менеджер проєкту;
- DevOps-інженер тощо.

Кожна з них має своє призначення в життєвому циклі ПЗ, що зумовлює потребу у формалізованому описі.

Відповідальність, у свою чергу, охоплює сферу дій, за які конкретна роль або виконавець несе зобов'язання. Це не лише перелік функціональних завдань, але й контроль за їх результатом, відповідність очікуваній якості, безпечність реалізації змін, дотримання строків та стандартів. Саме баланс між призначеною роллю та відповідальністю дозволяє уникнути помилок, зменшити неузгодженості між членами команди та забезпечити прозорість процесу розробки [5,6].

Чітке визначення ролей є особливо актуальним у середовищі з гнучкою структурою управління (Scrum, Kanban, SAFe тощо), де команди мають високий рівень автономії, але повинні діяти в межах узгоджених процесів. Ролі в таких методологіях часто визначають не тільки технічні функції, а й рівень прийняття рішень, комунікаційні потоки, взаємодію з замовником та іншими зацікавленими сторонами. Тому системна модель управління ролями та відповідальністю стає основою узгодженого розвитку проєкту [7].

У сфері розробки програмного забезпечення широке застосування знаходять структуровані моделі управління ролями й відповідальністю, що дозволяють

уникати плутанини у виконанні завдань і прийнятті рішень. Найпоширенішими серед них є моделі RACI, DACI та RAPID, які забезпечують формалізований підхід до визначення ролей учасників проєктів і розмежування відповідальності. Розглянемо ці моделі більш детально.

Модель RACI є однією з найвідоміших і застосовується для визначення ролей у межах конкретного процесу або завдання. Абревіатура RACI розшифровується як: Responsible (той, хто виконує завдання), Accountable (той, хто несе остаточну відповідальність), Consulted (ті, з ким консультуються), Informed (ті, кого інформують про результат). Така матриця дозволяє візуалізувати ролі для кожного етапу проєкту, уникнути дублювання функцій і забезпечити чіткість відповідальності [8].

У сфері розробки програмного забезпечення модель RACI використовується для формального визначення ролей та відповідальності членів команди у виконанні конкретних завдань. Цей підхід дозволяє уникати непорозумінь, дублювання обов’язків або пропущених функцій, що особливо важливо в умовах роботи з міждисциплінарними або розподіленими командами. Модель RACI визначає, хто виконує завдання, хто несе за нього відповідальність, хто залучається до консультацій і кого інформують про результат.

На прикладі типових процесів у команді розробки ПЗ матриця ролей RACI представлена в табл. 1.1.

Таблиця 1.1 – RACI-матриця ролей для команди розробки програмного забезпечення

Завдання / Роль	**Аналітик**	**Розробник**	**Тестувальник**	**Менеджер проєкту**
Збір вимог	R			A
Розробка технічного завдання	C	R		A
Написання коду		R		A
Тестування функціоналу			R	A
Прийняття рішення про реліз			C	A

Джерело: розроблено автором на основі [8]

Позначення в матриці мають такий зміст:

- R (Responsible) – той, хто безпосередньо виконує завдання;
- A (Accountable) – той, хто несе остаточну відповідальність за результат;
- C (Consulted) – особи, яких залучають для консультацій;
- I (Informed) – особи, яких інформують про виконання або результат.

Застосування RACI-матриці дає змогу уникнути типових помилок у координації дій між учасниками проєкту, оскільки кожен член команди чітко розуміє свою роль і очікування від неї. Такий підхід особливо ефективний у ситуаціях, коли кілька осіб можуть впливати на один і той самий процес – наприклад, під час передавання вимог між аналітиком, розробником і тестувальником. Крім того, формалізація ролей створює основу для автоматизації контролю виконання завдань в інформаційній системі управління проєктами.

Модель DACI має схожий принцип, але більше орієнтована на процес прийняття рішень. Вона означає Driver (відповідальний за організацію процесу), Approver (той, хто затверджує остаточне рішення), Contributor (учасники, які надають дані, аналітику, думки), Informed (ті, кого повідомляють про результат). Такий підхід особливо ефективний у багаторівневих структурах управління, де важливо координувати кілька джерел впливу на рішення [9].

Модель DACI використовується в управлінні проєктами розробки програмного забезпечення переважно для структурування процесу ухвалення рішень. Вона дозволяє розподілити обов’язки між учасниками таким чином, щоб уникнути паралельного прийняття суперечливих рішень, мінімізувати затримки та забезпечити прозорість у взаємодії. DACI ефективна у багаторівневих структурах, де необхідно узгодити інтереси кількох підрозділів або команд.

Приклад застосування моделі DACI для ситуації вибору архітектурного рішення подано в табл. 1.2.

Таблиця 1.2 – Розподіл ролей за моделлю DACI для прийняття технічного рішення

Учасник	**Роль у рішенні**
Архітектор	Driver
Технічний директор	Approver
Старший розробник	Contributor
Аналітик	Contributor
QA-команда	Informed
Менеджер проєкту	Informed

Джерело: розроблено автором на основі [9]

Позначення ролей у моделі DACI наступні:

- Driver – ініціатор, який організовує процес прийняття рішення;
- Approver – особа або орган, який ухвалює остаточне рішення;
- Contributor – учасники, які надають інформацію, пропозиції, експертну думку;
- Informed – особи, яких інформують про ухвалене рішення без залучення до процесу обговорення.

Модель DACI дозволяє зменшити навантаження на керівників, адже роль Driver може бути делегована технічним спеціалістам, а остаточне затвердження залишається за ключовою фігурою. Це сприяє швидшому прийняттю рішень, кращому використанню експертизи членів команди та підвищенню рівня відповідальності кожного учасника за свій внесок.

Модель RAPID (Recommend, Agree, Perform, Input, Decide) фокусується на деталізації процесу ухвалення рішень у проєктах зі складною структурою. Вона чітко виділяє учасників, які: готують рекомендації (Recommend), погоджуються з ними (Agree), виконують завдання (Perform), надають вхідні дані (Input), ухвалюють остаточне рішення (Decide). RAPID дозволяє не лише розподілити відповідальність, але й оптимізувати швидкість ухвалення рішень у складних командах [10].

Модель RAPID застосовується в управлінні розробкою програмного забезпечення для формалізації процесів ухвалення складних рішень, де залучено багато учасників. Вона дозволяє розподілити ролі відповідно до етапів вироблення,

узгодження, виконання та затвердження рішень, завдяки чому забезпечується логічна послідовність дій і уникнення конфліктів повноважень.

Приклад розподілу ролей у процесі вибору інструменту CI/CD на основі моделі RAPID наведено в табл. 1.3.

Таблиця 1.3 – Приклад застосування моделі RAPID у процесі ухвалення технічного рішення

Етап	**Учасник**
Recommend	DevOps-інженер
Agree	Архітектор
Perform	DevOps-інженер
Input	Системний адміністратор
Decide	Технічний директор

Джерело: розроблено автором на основі [10]

Позначення етапів у моделі RAPID мають такий зміст:

- Recommend – учасник, який ініціює пропозицію або рекомендацію;
- Agree – особа, яка погоджує або відхиляє запропоноване рішення;
- Perform – виконавець, відповідальний за реалізацію ухваленого рішення;
- Input – особа або підрозділ, який надає вхідну інформацію або експертизу;
- Decide – відповідальний за ухвалення остаточного рішення.

Модель RAPID сприяє впорядкуванню комунікацій у команді, забезпечує логічне проходження рішень через визначені етапи та дозволяє зосередити ключові управлінські зусилля на критичних точках впливу. Такий підхід особливо ефективний у великих командах або при реалізації складних технічних змін, де ризики розмиття відповідальності є високими.

Порівняльний аналіз моделей RACI, DACI та RAPID показано у 1.4, де наведено характеристики кожної з моделей, що демонструють їхнє призначення, сферу застосування та особливості розподілу відповідальності у процесі розробки програмного забезпечення.

Таблиця 1.4 – Порівняльна характеристика моделей RACI, DACI та RAPID

Критерій	RACI	DACI	RAPID
Призначення	Узгодження ролей у виконанні завдань	Організація процесу прийняття рішень	Формалізація логіки ухвалення рішень
Основні ролі	Responsible, Accountable, Consulted, Informed	Driver, Approver, Contributor, Informed	Recommend, Agree, Perform, Input, Decide
Фокус	Хто і як виконує завдання	Хто організовує і затверджує рішення	Який порядок і розподіл етапів ухвалення рішень
Застосування	Процеси з багатьма виконавцями	Колективні технічні або управлінські рішення	Складні та багатоступеневі процеси ухвалення рішень
Переваги	Прозорість виконання, уникнення дублювання	Чітка відповідальність за організацію та затвердження	Оптимізація комунікації та послідовності дій

Джерело: розроблено автором на основі [8-10]

Ці моделі, хоча й різняться за акцентами, мають спільну мету – забезпечити структуровану, зрозумілу й відтворювану схему управління ролями, яка може бути реалізована в інформаційній системі.

У сучасній практиці розробки програмного забезпечення використовуються різноманітні методології, які визначають організацію процесів, взаємодію учасників і логіку управління. Найпоширенішими серед них є Scrum, Kanban, DevOps і Waterfall. Кожна з них по-своєму впливає на підхід до визначення ролей та розподілу відповідальності.

Scrum – одна з найвідоміших Agile-методологій, яка передбачає ітеративну розробку, постійне вдосконалення та командну автономію. У Scrum чітко визначені ролі: Product Owner (відповідає за цінність продукту), Scrum Master (відповідає за дотримання процесу Scrum) і Development Team (самоорганізована команда, яка виконує роботу). Важливо, що команда розробки не має ієрархії всередині – усі несуть колективну відповідальність за виконання Sprint Backlog [11].

Kanban – це метод управління потоком завдань, що ґрунтується на візуалізації процесу, обмеженні кількості одночасних задач і безперервному покращенні. У Kanban відсутня формалізація ролей, але відповідальність розподіляється через встановлення «точок зупинки» (WIP-обмеження), що

зменшує перевантаження та дозволяє фіксувати, хто на якому етапі відповідає за завдання [11, 12].

DevOps – це підхід, що інтегрує розробку (Dev) і операційне обслуговування (Ops), спрямований на скорочення циклу поставки, підвищення якості та безперервну інтеграцію й розгортання. У DevOps рольова структура включає інженерів CI/CD, SRE (Site Reliability Engineers), DevOps-архітекторів та інших фахівців, відповідальних за автоматизацію, безпеку, моніторинг і стабільність інфраструктури. Особливістю є спільна відповідальність за якість і доступність сервісу між усіма командами [13].

Waterfall – традиційна каскадна модель, у якій розробка відбувається послідовно через фази: аналіз вимог, проєктування, реалізація, тестування, супровід. У цій методології ролі найчастіше строго відокремлені: аналітики передають документацію проєктувальникам, ті – розробникам, а потім – тестувальникам. Кожна роль несе відповідальність за свій етап, але з низькою гнучкістю у змінах [11, 14].

Оскільки моделі RACI, DACI та RAPID застосовуються для управління ролями й відповідальністю, доцільно порівняти їхнє використання в межах різних методологій розробки програмного забезпечення. У табл. 1.5 наведено доцільність або поширеність використання кожної з моделей у контексті Scrum, Kanban, DevOps і Waterfall.

Таблиця 1.5 – Застосування моделей RACI, DACI, RAPID у методологіях розробки ПЗ

Методологія	RACI	DACI	RAPID	Коментар
Scrum	✓	✓	△	RACI для визначення відповідальності, DACI для прийняття рішень, RAPID рідше використовується
Kanban	✓	△	△	RACI можливий для візуалізації відповідальності, інші моделі менш типові
DevOps	✓	✓	✓	Усі три моделі можуть застосовуватись, особливо для координації між Dev і Ops
Waterfall	✓	✓	△	RACI і DACI добре підходять через формалізованість, RAPID – у складних рішеннях

Джерело: розроблено автором на основі [11-14]

Умовні позначення:

✓ – модель широко застосовується;

△ – модель може застосовуватись у специфічних випадках або з адаптацією

Окрім практичних методологій розробки програмного забезпечення, розподіл ролей та відповідальності формалізується і на рівні міжнародних стандартів, таких як ISO/IEC 12207 та ISO/IEC 27001. Ці стандарти відіграють важливу роль у забезпеченні якості, безпеки та управлінської прозорості в IT-проєктах [15].

ISO/IEC 12207 – це міжнародний стандарт, що описує процеси життєвого циклу програмного забезпечення. Він детально визначає ролі, відповідальні за кожен із процесів, таких як розробка, верифікація, валідація, технічне супроводження, управління конфігурацією тощо. У межах цього стандарту кожен процес має чітке призначення, вхідні та вихідні артефакти, а також вказано, хто саме несе відповідальність за їх створення чи затвердження. Таким чином, ISO/IEC 12207 сприяє побудові структурованої моделі розподілу обов'язків на всіх етапах життєвого циклу ПЗ [15].

ISO/IEC 27001 – це стандарт управління інформаційною безпекою, який особливо акцентує увагу на ролях і відповідальності в контексті захисту інформаційних активів. У ньому чітко визначено, що всі функції, пов'язані з управлінням ризиками, інцидентами, безпечністю даних, мають бути закріплені за конкретними особами або підрозділами. Стандарт вимагає документування обов'язків у сфері безпеки, що може реалізовуватись через матриці відповідальності (наприклад, на основі RACI), а також вимагає періодичної перевірки того, наскільки ці обов'язки виконуються [15].

У результаті, стандарти не лише задають нормативну рамку, але й прямо впливають на необхідність впровадження формалізованих підходів до управління ролями, що своєю чергою підсилює актуальність застосування спеціалізованих інформаційних систем у цій сфері.

У процесі розробки програмного забезпечення існує низка ключових процесів, які особливо потребують чіткого та формалізованого розподілу ролей і

відповідальності. Їх ігнорування або спрощене трактування часто призводить до конфліктів, дублювання функцій або втрат контролю над якістю продукту.

Перш за все, це управління вимогами, де визначення відповідальних є основою точного формулювання очікувань замовника. Аналітик, замовник, архітектор, Product Owner – кожен із цих учасників має окрему зону впливу, яка має бути чітко окреслена. Без цього можлива неконкретність вимог, постійні зміни та втрати у функціональності [16].

Наступний важливий процес – реалізація (розробка) та тестування програмного забезпечення. Тут особливо важливим є поділ між тими, хто створює код (розробники), перевіряє його на відповідність вимогам (тестувальники, QA), та тими, хто затверджує переходи між етапами (технічні менеджери або лідери команд). Якщо ролі не визначено, виникає ризик несанкціонованих змін, неадекватного покриття тестами та порушення послідовності релізу.

Окремої уваги заслуговує процес релізу та підтримки, де відповідальність часто розподілена між DevOps-інженерами, адміністраторами, представниками підтримки користувачів і менеджерами релізів. Тут необхідна скоординованість, контроль змін і дотримання регламентів, тому формалізація відповідальності в системах реліз-менеджменту є досить важливою [13].

Також до процесів, де важливо визначити відповідальних, належать управління інцидентами, ризиками, змінами, забезпечення інформаційної безпеки, конфігураційне управління, проведення аудитів. Усі ці напрями передбачають оперативну взаємодію між командами, а отже, вимагають прозорого підходу до ролей – як у щоденній практиці, так і на рівні документації, політик чи автоматизованих систем.

Розробка програмного забезпечення охоплює кілька критичних процесів, для яких чітке визначення ролей та відповідальності є запорукою узгодженої командної роботи, контролю якості та дотримання термінів. У табл. 1.6 наведено ключові процеси, що вимагають формального підходу до розподілу обов’язків, із прикладами типових ролей, які беруть участь у кожному з них [17].

Таблиця 1.6 – Основні процеси в розробці ПЗ і приклади відповідальних ролей

Процес	Типові відповідальні ролі
Управління вимогами	Бізнес-аналітик, Product Owner, Замовник, Архітектор
Розробка та тестування	Розробник, Тестувальник, QA-інженер, Тімлід
Реліз та підтримка	DevOps, Release Manager, Служба підтримки, Системний адміністратор
Управління змінами, інцидентами, безпекою	Менеджер змін, Security Officer, Служба підтримки, IT-директор

Джерело: розроблено автором на основі [15-17]

Відсутність чіткого управління ролями та відповідальністю в процесі розробки програмного забезпечення призводить до низки системних проблем, які негативно впливають на ефективність роботи команди, якість кінцевого продукту та загальну керованість проєктом [18, 19].

Однією з найпоширеніших проблем є дублювання завдань або їх залишення без виконавця. У ситуаціях, коли кілька учасників команди не мають чіткого розмежування функцій, одне й те саме завдання може виконуватись паралельно або, навпаки, залишитись без уваги, оскільки всі припускають, що його вже виконує хтось інший.

Другою суттєвою загрозою є порушення строків виконання проєкту. Без чіткого розподілу відповідальності складно контролювати прогрес окремих підзадач, що унеможливлює точне планування релізів, збільшує кількість перенесених дедлайнів і знижує довіру з боку замовників чи керівництва.

Важливим аспектом є також зниження якості продукту через неузгоджену комунікацію. Якщо відповідальність за технічні рішення, рев'ю коду, тестування або безпеку не закріплена за конкретними особами, зростає ймовірність появи критичних помилок, порушення стандартів розробки та неповного тестового покриття.

Ще одна поширена проблема – конфлікти між членами команди, особливо в умовах стресу, дедлайнів чи зміни пріоритетів. Відсутність прозорої структури

ролей спричиняє непорозуміння, коли кілька осіб претендують на прийняття рішень або, навпаки, відмовляються брати на себе відповідальність.

Нарешті, без формалізації ролей неможливо ефективно реалізувати систему контролю доступу, аудиту, відповідності стандартам ISO або вимогам замовника, що стає критичним чинником у середовищах з високими вимогами до безпеки та регуляторного комплаєнсу.

З огляду на розглянуті методології, стандарти та типові проблеми, що виникають у разі відсутності чіткої системи управління ролями, постає об’єктивна необхідність у впровадженні спеціалізованої інформаційної системи. Така система дозволяє формалізувати розподіл обов’язків, забезпечити контроль за виконанням завдань, підвищити прозорість прийняття рішень і уніфікувати підходи до взаємодії між учасниками команди. Крім того, автоматизація процесів управління ролями полегшує інтеграцію з системами контролю версій, CI/CD, моніторингу, звітності та аудиту, що особливо актуально в умовах сучасної цифрової розробки.

1.1.2 Дослідження ринку інформаційних управляючих систем

У сучасному IT-середовищі, де розробка програмного забезпечення відбувається у багаторівневих і міжфункціональних командах, ефективне управління ролями та відповідальністю потребує підтримки з боку спеціалізованих інформаційних систем. Такі системи повинні не лише відображати структуру повноважень, а й забезпечувати автоматизований контроль за виконанням обов’язків, аудит змін, керування доступом, а також інтеграцію з іншими компонентами цифрової інфраструктури.

Метою цього підпункту є аналіз ринку існуючих інформаційних управляючих систем, які реалізують функціональність розподілу ролей і відповідальності в контексті розробки програмного забезпечення. Дослідження проводиться з урахуванням таких критеріїв, як підтримка моделей управління

(RACI, DACI, RAPID), можливість кастомізації, тип ліцензування (open-source або комерційне), наявність механізмів інтеграції (API, плагіни), інструменти контролю безпеки та відповідність міжнародним стандартам.

2. Класифікація систем за типами [20-24]:

- системи управління проєктами (Atlassian Jira, ClickUp, Trello);
- DevOps-платформи (Azure DevOps, GitLab, GitHub Enterprise);
- системи управління доступом (Okta, Keycloak, OneLogin);
- системи класу ITSM (ServiceNow, OTRS);
- корпоративні ERP або ECM з модулями керування ролями (SAP, Oracle Cloud)

Системи управління проєктами.

Системи управління проєктами є одним із найпоширеніших типів інформаційних платформ, що використовуються для організації командної роботи в розробці програмного забезпечення. До таких систем належать Atlassian Jira, ClickUp, Trello, які надають інструменти для постановки задач, відстеження прогресу, комунікації та управління ресурсами [25, 26].

Atlassian Jira є флагманським продуктом для гнучкої розробки, який підтримує Scrum, Kanban та інші Agile-підходи. Вона дозволяє створювати структуру проєкту з визначенням користувачів, ролей і прав доступу на рівні задач, епіків, компонентів і модулів. Система підтримує реалізацію RACI-логіки через кастомні поля, плагіни, схеми відповідальності, дозволяє формувати звіти про завантаження виконавців, ескалації задач і контроль дедлайнів. Інтегрується з Confluence, Bitbucket, Bamboo та іншими сервісами Atlassian-екосистеми.

ClickUp є хмарною платформою з широкими можливостями кастомізації, яка орієнтована як на IT-команди, так і на бізнес-підрозділи. Вона надає можливість налаштовувати рівні доступу, визначати відповідальних за завдання (Assigned, Watchers), використовувати шаблони RACI та автоматизовані сценарії. ClickUp підтримує діаграми Ганта, дошки, часові лінії та має інтеграцію з популярними

системами зберігання даних, пошти, код-репозиторіями, що забезпечує повну видимість процесу виконання задач у розрізі ролей [27].

Trello позиціонується як проста, візуально орієнтована Kanban-система, яка підходить для невеликих команд або допоміжних процесів. Вона дозволяє закріплювати відповідальних за кожну картку, використовувати мітки, чеклісти, коментарі та автоматизацію через Butler. Хоча Trello не підтримує складні ієрархії ролей на рівні організації, її функціональність можна розширити через Power-Ups і інтеграції з іншими системами, такими як Slack, GitHub або Google Workspace [28].

Таким чином, системи управління проєктами дозволяють впроваджувати контроль за виконанням обов'язків, навіть якщо первинно не орієнтовані саме на управління ролями. Їх гнучкість і розширюваність роблять їх ефективними інструментами для команд розробки, які потребують прозорого механізму відповідальності.

DevOps-платформи.

DevOps-платформи – це інтегровані системи, що поєднують розробку програмного забезпечення, автоматизоване тестування, безперервну інтеграцію та розгортання (CI/CD), а також контроль продуктивності, інцидентів і змін. До найпоширеніших представників цього типу належать Azure DevOps, GitLab, GitHub Enterprise [29].

Azure DevOps – це потужна платформа від Microsoft, яка охоплює повний життєвий цикл розробки ПЗ. Вона включає сервіси для управління репозиторіями (Repos), завданнями (Boards), тестуванням (Test Plans), CI/CD (Pipelines) та артефактами (Artifacts). Управління ролями реалізовано через проєктні групи, які мають чіткі рівні доступу (Reader, Contributor, Project Administrator тощо). Система підтримує призначення відповідальних за робочі елементи, контроль затвердження змін, історію дій користувачів та аудит. Є вбудовані механізми для інтеграції з Active Directory та Azure AD, що дозволяє реалізовувати централізоване управління правами доступу на рівні організації.

GitLab – це платформа з відкритим кодом (у базовій версії), яка поєднує репозиторії, задачі, CI/CD та DevSecOps-інструменти в єдиній системі. Вона

дозволяє призначати ролі на рівні груп, проєктів і задач, використовувати approval-процеси (наприклад, для злиття pull-запитів), автоматизувати виконання через пайплайни з контролем прав, а також інтегрувати механізми безпеки й аудитів. GitLab дозволяє вбудовувати логіку RACI або DACI на рівні процесів через політики, групи та механізми взаємодії між командами [30].

GitHub Enterprise – це корпоративна версія добре відомої платформи GitHub, яка зосереджена на керуванні кодом, співпраці між командами та автоматизації. Система підтримує організаційні рівні доступу (Owner, Admin, Member), можливість створення команд і правил для review-запитів, обов'язкових перевірок та підписань. У GitHub Actions можна реалізовувати складну автоматизацію, а також вбудовувати контроль відповідальності через механізми CODEOWNERS, інтеграцію з зовнішніми системами (Jira, Slack, SAML-провайдерами тощо) [31].

Таким чином, DevOps-платформи надають не лише інструменти розробки, а й чіткі механізми закріплення відповідальності, контролю дій, затвердження змін і дотримання процедур безпеки. Їх перевага полягає у високій інтегрованості та гнучкому управлінні повноваженнями, що робить їх придатними як базу для автоматизації управління ролями в командах розробки ПЗ.

Системи управління доступом (IAM)].

Системи управління доступом, або Identity and Access Management (IAM) системи, зосереджені на адмініструванні користувачів, ролей, прав доступу та автентифікації в інформаційних середовищах. Хоча вони не належать до спеціалізованих інструментів розробки ПЗ, IAM-системи є ключовими компонентами в побудові політик відповідальності та безпеки, особливо в середовищах, де діють вимоги відповідності (compliance). Серед найпоширеніших – Okta, Keycloak, OneLogin, Auth0, Microsoft Entra ID (ex-Azure AD) [32-33].

Okta є хмарною платформою IAM, яка забезпечує централізоване керування ідентичностями, ролями, сесіями та багатофакторною автентифікацією. Вона дозволяє створювати рольову модель з деталізованим контролем доступу до застосунків, API та середовищ. Через групи, політики та rule-based provisioning реалізується розмежування прав відповідно до функціональної ролі користувача в

організації. Okta інтегрується з тисячами хмарних сервісів, зокрема Jira, GitHub, Google Workspace, AWS.

Keycloak – це open-source IAM-рішення, яке підтримує автентифікацію, авторизацію на основі ролей (RBAC), єдиний вхід (SSO), федерацію облікових записів та протоколи OpenID Connect, OAuth2, SAML. Keycloak дозволяє задавати структуру ролей, груп, політик доступу та призначати їх на рівні ресурсів або API, що робить його ефективним для розробників, які хочуть гнучко реалізувати контроль відповідальності безпосередньо в інформаційній системі [34].

OneLogin та Auth0 також є популярними платформами, орієнтованими на безпечне централізоване управління користувачами. Вони дозволяють реалізувати адаптивну автентифікацію, правила доступу залежно від контексту, аудит дій користувачів і контроль над ресурсами на основі ролей та політик. Усі ці рішення можуть бути інтегровані як з внутрішніми інформаційними системами, так і з зовнішніми хмарними сервісами.

IAM-системи надають інфраструктурну основу для управління відповідальністю через регламентований контроль доступу. Вони не завжди підтримують моделі типу RACI безпосередньо, проте дозволяють закріплювати повноваження та відповідальність за ресурси, що є важливою частиною загальної системи управління ролями в межах IT-проєктів.

Системи класу ITSM.

Системи класу ITSM (IT Service Management), до яких належать ServiceNow, OTRS, BMC Remedy та інші, орієнтовані на управління IT-послугами, інцидентами, змінами, запитами та конфігурацією. Вони традиційно використовуються в корпоративних IT-відділах, але дедалі частіше інтегруються з розробкою ПЗ у середовищах DevOps та розширеної підтримки життєвого циклу продукту. Ці системи мають розвинуту модель ролей, що дозволяє ефективно реалізувати принципи відповідальності, звітності та контролю [35-37].

ServiceNow – одна з найпотужніших платформ, яка забезпечує повний цикл управління IT-процесами відповідно до стандартів ITIL. У ній реалізовано детальну модель доступу на основі ролей (RBAC), а також підтримку гнучкої організації

потоків задач, розподілу повноважень між аналітиками, виконавцями, кураторами, керівниками змін тощо. Ролі та зони відповідальності можуть визначатися як вручну, так і за допомогою вбудованих правил автоматичного призначення залежно від типу звернення, пріоритету, категорії чи статусу користувача. Завдяки цьому система може реалізовувати логіку, подібну до моделей RACI або RAPID [36].

OTRS (Open Ticket Request System) – це відкрита система управління зверненнями та запитами, яка дозволяє налаштовувати категорії, черги, групи та облікові записи з різними ролями доступу. Система дозволяє фіксувати відповідального за кожну заявку, контролювати зміни статусів, визначати SLA і вести журнал дій. Незважаючи на свою простоту, OTRS добре масштабується в IT-відділах із чітко розподіленими функціями [37].

BMC Remedy – ще один представник корпоративного сегменту, який орієнтований на автоматизацію складних сценаріїв ITSM. Тут реалізовано централізоване управління змінами, інцидентами, проблемами, знаннями, а також функції контролю відповідальності, призначення завдань і створення схем ескалації. BMC підтримує гнучку інтеграцію з DevOps-інструментами, що дозволяє зв’язувати процеси розробки з процесами обслуговування [38].

ITSM-системи мають найбільш формалізовану структуру відповідальності серед усіх типів систем, що розглядаються. Вони дозволяють впроваджувати централізовану модель контролю виконання IT-процесів з розподілом ролей, що є особливо корисним у великих проєктах з високим рівнем регламентації.

Корпоративні ERP або ECM з модулями керування ролями.

Корпоративні ERP (Enterprise Resource Planning) та ECM (Enterprise Content Management) системи часто включають модулі для управління ролями, доступом і відповідальністю, які використовуються в ширшому контексті – не лише в розробці ПЗ, а в управлінні всім підприємством. Попри свою орієнтацію на фінансово-ресурсне або контентне управління, ці системи дедалі частіше інтегруються з інструментами розробки, проєктного менеджменту та сервісного обслуговування, особливо у великих організаціях [39].

SAP (Systems, Applications, and Products) – одна з найвідоміших ERP-систем, яка реалізує складну модель ролей та прав доступу через механізми авторизацій і профілів. SAP дозволяє визначати функціональні ролі для кожного модуля (HR, FI, MM, PP, Project Systems тощо) з подальшою деталізацією доступу до транзакцій, звітів і бізнес-об'єктів. У контексті розробки та проєктного управління (через SAP Project System) можна створювати структуру відповідальних осіб, затверджувачів та виконавців, що відповідає підходам RACI або DACI [40].

Oracle Cloud ERP має подібний рівень деталізації. Усі дії користувачів контролюються через механізми job roles і data access sets, що дозволяє реалізувати чітке розмежування відповідальності в межах різних бізнес-процесів, включно з IT-розробкою, управлінням релізами або контрактами на технічну підтримку. Система підтримує аудит, контроль змін і багаторівневе затвердження (workflow), що дозволяє реалізувати корпоративну політику відповідальності [41].

ECM-системи, такі як M-Files, OpenText або SharePoint, зосереджуються на управлінні доступом до документів, контроль версій і політики затвердження. У них також реалізовано моделі ролей, що регламентують доступ до контенту, журналювання дій, структуру затвердження документів, які можуть містити технічні завдання, архітектурні рішення або документацію проєктів. Це дозволяє відслідковувати, хто відповідав за підготовку, перевірку, погодження або затвердження окремих елементів у рамках процесу розробки.

Хоча ERP та ECM-системи зазвичай не розроблялися як засоби для управління ролями у проєктах з розробки ПЗ, їх гнучкість, масштабованість і глибока деталізація політик доступу дають змогу використовувати їх як інфраструктурну основу для централізованого управління відповідальністю в великих IT-екосистемах.

Для узагальнення результатів дослідження ринку інформаційних управляючих систем доцільно зіставити ключові типи систем за рядом функціональних критеріїв. У табл. 1.7 представлено порівняльну характеристику обраних систем різних класів, зокрема за такими параметрами: підтримка моделей

управління ролями, можливість інтеграції, наявність механізмів аудиту, тип ліцензії та відповідність вимогам до безпеки.

Таблиця 1.7 – Порівняльна характеристика інформаційних управляючих систем

Система	Тип	Моделі ролей (RACI, DACI, RAPID)	Інтеграції / API	Аудит / контроль змін	Ліцензія
Atlassian Jira	Система управління проєктами	✓ (через кастомізацію)	✓	✓	Комерційна
ClickUp	Система управління проєктами	✓ (RACI шаблони)	✓	✓	Комерційна / безкоштовна версія
Azure DevOps	DevOps-платформа	✓ (через політики)	✓	✓	Комерційна
GitLab	DevOps-платформа	✓ (гнучкі правила)	✓	✓	Open-source / Комерційна
Keycloak	IAM-система	△ (RBAC)	✓	✓	Open-source
ServiceNow	ITSM-система	✓ (через процеси)	✓	✓	Комерційна
OTRS	ITSM-система	△ (через черги)	✓	✓	Open-source
SAP	ERP-система	✓ (через авторизації)	✓	✓	Комерційна
SharePoint	ECM-система	△ (на рівні документів)	✓	✓	Комерційна

Джерело: розроблено автором на основі [35-41]

- ✓ – функціональність підтримується повною мірою або офіційно передбачена;
- △– функціональність можлива за допомогою кастомізації, плагінів або додаткових налаштувань, але не є базовою.

Аналіз ринку інформаційних управляючих систем засвідчив, що більшість розглянутих платформ надають базову або розширену підтримку механізмів розподілу ролей, контролю дій користувачів та інтеграції з іншими сервісами. Системи типу DevOps (Azure DevOps, GitLab) і ITSM (ServiceNow) демонструють найвищий рівень деталізації доступу, підтримку затверджень, трасування змін і аудит, що робить їх придатними для складних сценаріїв управління відповідальністю. Проєктні системи (Jira, ClickUp) забезпечують гнучкість у використанні RACI-підходу завдяки кастомізації, проте потребують додаткових налаштувань для реалізації повноцінного процесного контролю. Системи IAM

(Keycloak, Okta) і ERP (SAP) пропонують масштабовані механізми прав доступу, однак не є профільними для щоденного управління розробкою, що обмежує їх самостійне використання без інтеграції. Крім того, комерційні рішення часто мають високу вартість впровадження та ліцензування, а open-source системи потребують технічної експертизи для адаптації під конкретні задачі.

1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи управління ролями при створенні програмного забезпечення

Проєкт створення інформаційної управляючої системи управління ролями та відповідальністю передбачає побудову інтегрованої програмної платформи, яка поєднує традиційні функції управління (розподіл ролей, контроль виконання завдань, фіксація дій) з інтелектуальними аналітичними модулями, здатними виявляти приховані закономірності та потенційні ризики у процесі реалізації проєктів. Основна ідея полягає у створенні такої ІУС, яка б не лише забезпечувала структуроване управління відповідальністю в межах RACI-матриці, але й адаптувалась до динаміки командної взаємодії через впровадження моделей штучного інтелекту.

Для формального опису та структурного представлення концептуальної архітектури системи обрано методологію SysML (Systems Modeling Language). Вона дозволяє формалізувати логіку взаємодії між компонентами системи, відобразити функціональні, структурні та поведінкові аспекти, що є особливо важливим при розробленні комплексних інформаційних систем із декількома рівнями логіки. Використання SysML забезпечує побудову узгоджених моделей, які охоплюють специфікацію ролей, подій, взаємозв’язків між підсистемами, а також інтеграцію інтелектуального модуля на концептуальному рівні.

Застосування SQL Server як системи управління базами даних обґрунтовується її стабільною підтримкою транзакцій, широкими можливостями для забезпечення цілісності даних, ефективною реалізацією індексування та можливістю масштабування в корпоративному середовищі. Логічна структура даних базується на нормалізованій моделі, де окремими таблицями виділено об’єкти проєктів, ролі, функції, користувачів, дії (логи) та зв’язки між ними. Така організація дозволяє гнучко реалізовувати запити до даних та ефективно інтегрувати алгоритмічні аналітичні модулі.

Інтелектуальна складова ІУС включає такі компоненти:

рекомендаційна модель призначення ролей, побудована на основі методів колаборативної фільтрації, забезпечує автоматичне формування RACI-матриці з урахуванням історичних даних, профілів користувачів і типових шаблонів.

- прогнозування ризиків, що пов’язане з виявленням потенційного перевантаження учасників або ймовірності невиконання задач у задані строки, реалізується за допомогою алгоритмів машинного навчання.

- інтелектуальний аналіз логів передбачає застосування методів Process Mining для виявлення порушень у відповідальності та невідповідностей між запланованими і фактичними діями користувачів.

- графовий аналіз соціальної взаємодії (SNA) дозволяє візуалізувати структуру комунікацій, ідентифікувати ключових учасників, зони ізоляції або надмірної централізації у виконанні задач.

Для реалізації аналітичного модуля обрано мову програмування Python через її багатий набір бібліотек для роботи з даними (pandas, numpy), машинного навчання (scikit-learn, xgboost), графового аналізу (networkx) та процесного майнінгу (pm4py). Розробка здійснюється у середовищі Jupyter або Google Colab, що забезпечує інтерактивність аналізу, зручну візуалізацію та гнучкість у тестуванні моделей.

Запропонований підхід дозволяє не лише автоматизувати управління ролями та відповідальністю, але й створює передумови для проактивного виявлення

ризиків та прийняття обґрунтованих управлінських рішень на основі даних. Комплексне поєднання класичних методів управління, формалізованого проєктування та сучасних інтелектуальних технологій формує інноваційну модель ІУС, адаптовану до потреб командної взаємодії в динамічному середовищі розробки програмного забезпечення.

Висновки до розділу 1

У результаті дослідження предметної області встановлено, що ефективне управління ролями та відповідальністю є важливим елементом успішної організації процесів розробки програмного забезпечення. Проаналізовано ключові моделі управління (RACI, DACI, RAPID), їхнє застосування в сучасних методологіях (Scrum, DevOps, Kanban, Waterfall), а також вимоги міжнародних стандартів ISO/IEC 12207 та ISO/IEC 27001 щодо документування та закріплення ролей. Визначено типові процеси, які потребують чіткого розподілу обов’язків, і охарактеризовано проблеми, що виникають за його відсутності. Обґрунтовано необхідність впровадження інформаційної системи для централізованого управління ролями, що дозволить забезпечити контроль, узгодженість, прозорість і відповідність регуляторним вимогам.

У ході дослідження ринку інформаційних управляючих систем проаналізовано дев’ять сучасних рішень різних класів – від систем управління проєктами до IAM- і ERP-платформ. Встановлено, що найбільш повну підтримку ролей і контролю відповідальності забезпечують DevOps- і ITSM-системи, які мають вбудовані механізми призначення відповідальних, журналювання дій, затвердження змін і інтеграції з CI/CD та хмарними сервісами. Проєктні системи, зокрема Jira та ClickUp, забезпечують гнучкість через шаблони та кастомізацію, але вимагають додаткових налаштувань для відповідності стандартам типу RACI чи DACI. Open-source рішення є економічно привабливими, але потребують технічної адаптації. Отримані результати дозволяють виділити функціональні орієнтири для розробки власної інформаційної системи управління ролями.

РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ

2.1 Структура і характеристика системи

Розроблення інформаційної системи управління ролями та відповідальністю передбачає поетапне проєктування її архітектури з урахуванням логічної структури, функціональних складових та принципів взаємодії між компонентами. У цьому підпункті буде сформовано структурну модель системи, визначено ключові підсистеми та їх призначення, а також встановлено межі відповідальності між ними. Такий підхід дозволяє забезпечити системність реалізації, чітке розмежування функціональності та подальшу підтримку масштабованості й супроводу.

Для візуалізації структури системи застосовується методологія системної інженерії на основі нотації SysML, яка дозволяє формалізовано описати компоненти, їхню взаємодію, зовнішні залежності та логіку виконання. Основу архітектури становить поділ на фронтенд, бекенд, базу даних і аналітичний модуль, у межах яких функціонують спеціалізовані сервіси – зокрема, підсистема керування ролями, інтелектуальний модуль перевірки відповідальності, блоки обліку дій та аудиту, модуль взаємодії з користувачем [42].

У межах функціонування інформаційної системи виділяються кілька основних типів користувачів, кожен з яких має власні цілі використання, рівень доступу та роль у процесах керування. Такий поділ дозволяє забезпечити диференційований доступ до функціональності системи, реалізувати механізми безпеки та надати кожному користувачу лише релевантні інструменти.

Першу категорію становлять виконавці, які безпосередньо залучені до виконання завдань. Вони мають доступ до власного переліку задач, можуть бачити призначені їм ролі у рамках конкретного процесу (наприклад, як Responsible чи Consulted згідно з моделлю RACI), переглядати історію змін і залишати коментарі

або запити. Система дозволяє виконавцям швидко орієнтуватися в обов’язках, строках та комунікаційних зв’язках із іншими членами команди.

Другою ключовою категорією є тімлід або менеджер, який відповідає за призначення ролей у рамках процесів, керування відповідальністю та загальну координацію роботи. Саме ця роль взаємодіє з інтелектуальною підсистемою системи – використовує аналітичні звіти, отримує рекомендації щодо оптимального розподілу навантаження, має змогу виявляти ризики перевантаження або конфліктів. Тімлід також ініціює аналіз соціальної взаємодії, вивчає результати процес-майнінгу та ухвалює рішення щодо перегляду ролей або втручання в організацію процесу.

У системі також передбачено роль адміністратора, який не є учасником щоденного керування ролями, але відповідає за налаштування структури доступу, адміністрування облікових записів, підтримку безпеки та технічне забезпечення роботи підсистем (включно з інтерфейсами до сторонніх сервісів, зокрема Jira або GitLab).

Нарешті, концептуальною складовою є віртуальний агент або інтелектуальний модуль, який виконує роль непрямого учасника системи. Його функції включають обробку журналів змін, побудову соціальних графів, виявлення аномалій, формування прогнозів та рекомендацій. Цей модуль діє автономно, ініціюється тімлідом і надає дані у формі звітів, візуалізацій або сповіщень.

Відповідно до окресленої структури користувачів та особливостей їхньої взаємодії з системою, можна сформулювати основні функціональні вимоги, які відображають ключові аспекти функціонування інформаційної системи управління ролями та відповідальністю. Ці вимоги охоплюють як базові сценарії взаємодії користувачів, так і елементи аналітичної та інтелектуальної підтримки.

1. Управління задачами та ролями

1.1. Можливісті створення, редагування та видалення задач із призначенням ролей (Responsible, Accountable, Consulted, Informed).

1.2. Відображення переліку задач з відповідними ролями для кожного користувача.

1.3. Можливість масового або автоматизованого призначення ролей за шаблонами.

2. Керування користувачами та політиками доступу

2.1. Реєстрація та аутентифікація користувачів із поділом на ролі (виконавець, тімлід, адміністратор).

2.2. Призначення рівнів доступу відповідно до типу користувача.

2.3. Контроль за виконанням політик відповідальності (наприклад, одна особа не може бути одночасно R та A).

3. Інтелектуальний аналіз та підтримка прийняття рішень

3.1. Виявлення відхилень у виконанні задач від очікуваних моделей відповідальності.

3.2. Формування рекомендацій щодо призначення ролей на основі історичних даних.

3.3. Побудова соціальних графів комунікації для виявлення ізольованих учасників або перевантажених осіб.

3.4. Прогнозування ризиків невиконання задач у строк залежно від складу команди.

4. Аудит, звітність та візуалізація

4.1. Ведення журналу дій користувачів для цілей аудиту.

4.2. Візуалізація RACI-матриць, ролей у процесах та соціальних мереж взаємодії.

4.3. Генерація звітів щодо виконання задач, відповідності ролей, ефективності розподілу.

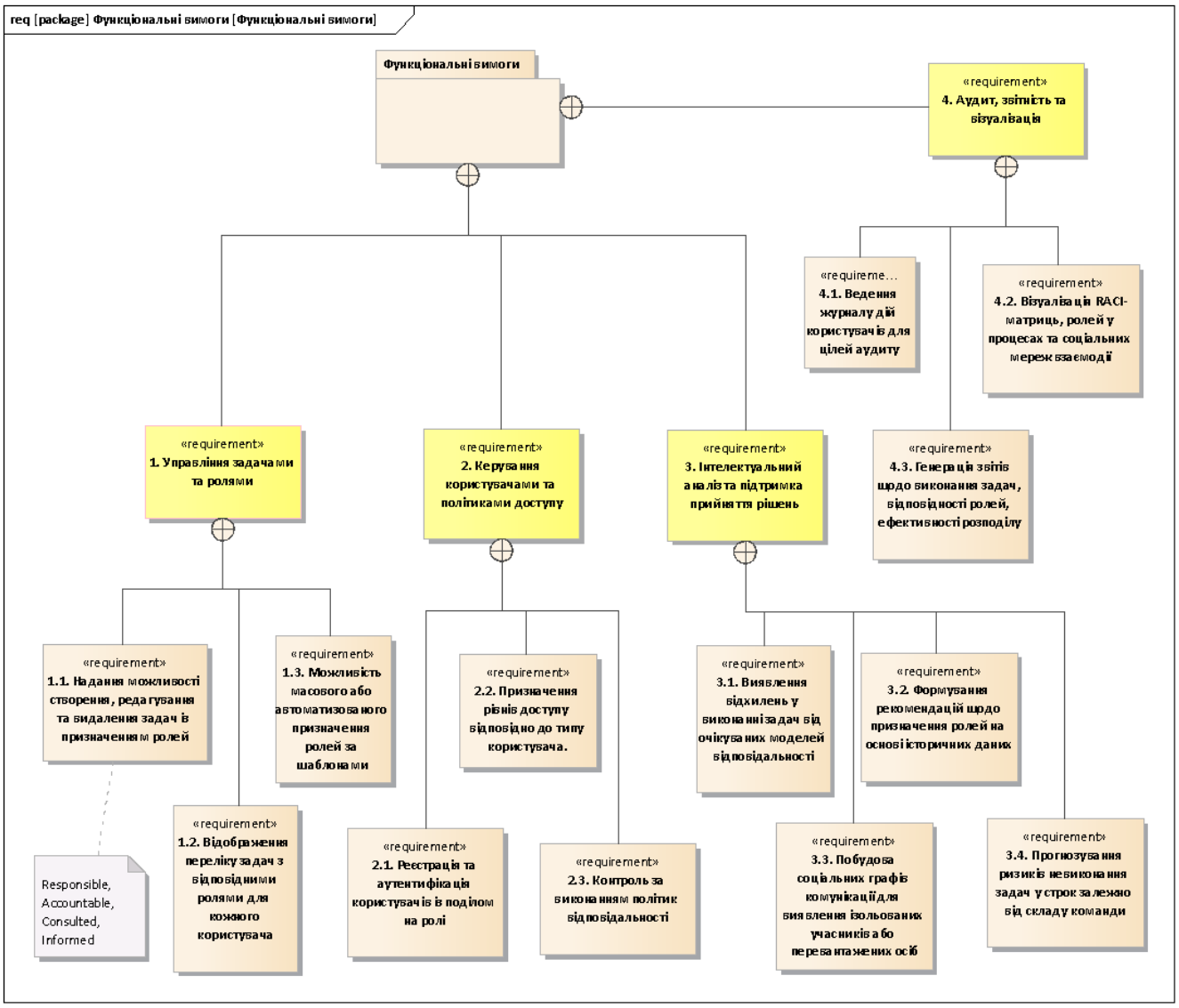


Рисунок 2.1 – Функціональні вимоги до системи

Джерело: розроблено автором самостійно

На основі сформульованих функціональних вимог було побудовано діаграму прецедентів, яка дозволяє наочно представити взаємодію основних користувачів системи з її ключовими функціональними можливостями. Такий підхід дає змогу структурувати поведінку системи з позиції користувачів і візуалізувати межі їх відповідальності, а також ідентифікувати ролі, що потребують підтримки в рамках програмної реалізації.

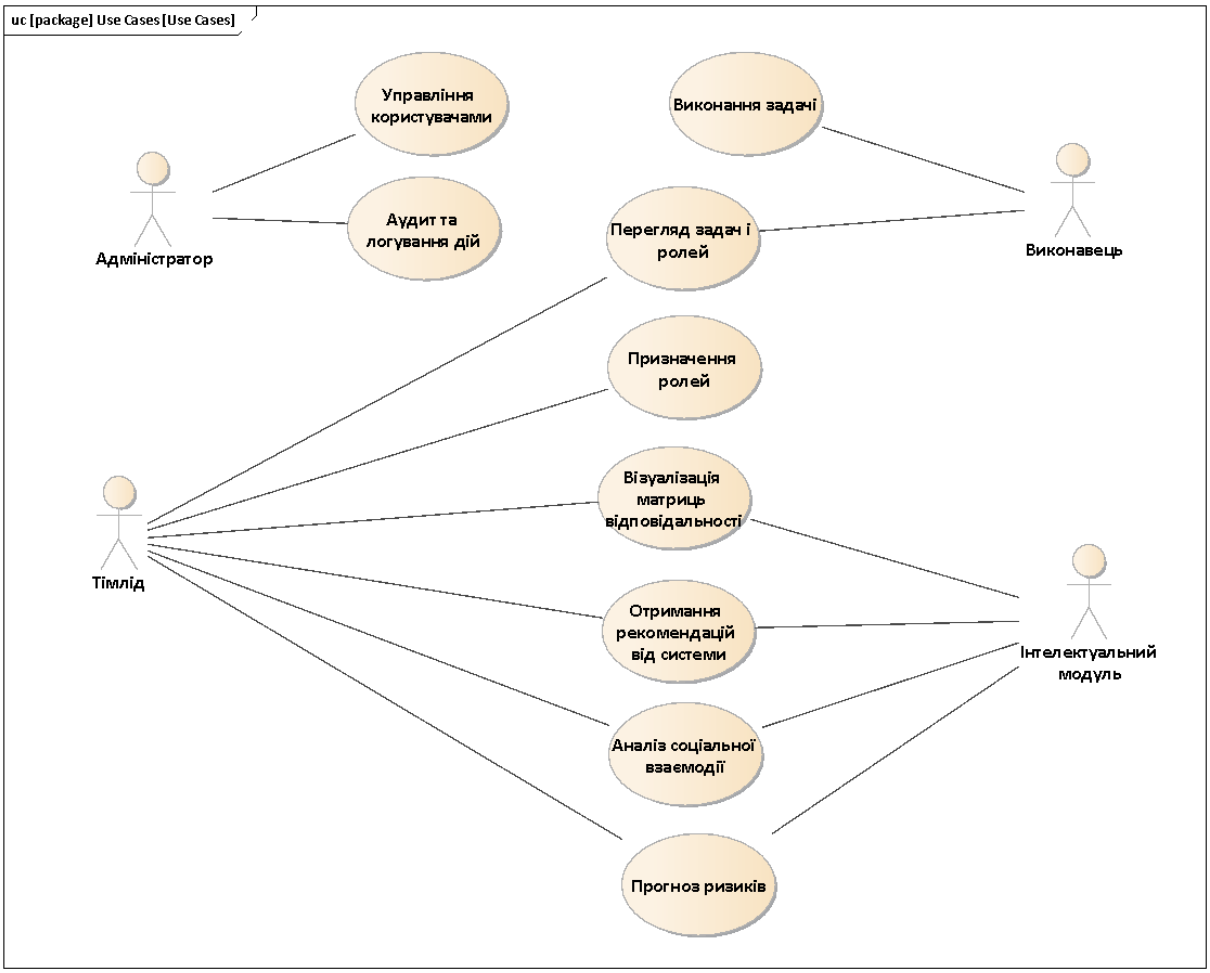


Рисунок 2.2 – Діаграма прецедентів взаємодії користувачів з функціональністю системи

Джерело: розроблено автором самостійно

Діаграма відображає три основні типи користувачів: виконавець, тімлід та адміністратор. Кожен із них взаємодіє з окремим підмноженням функцій – зокрема, тімлід отримує доступ до інтелектуальних функцій (аналіз соціальної взаємодії, прогнозування ризиків, рекомендації системи), тоді як виконавець зосереджений на виконанні задач та перегляді ролей, а адміністратор відповідає за налаштування системи та аудит.

Розподіл прецедентів за ролями дозволяє також виокремити точки взаємодії з інтелектуальним модулем, який діє у фоновому режимі, підтримуючи прийняття рішень на основі аналізу історичних даних, логів та комунікацій.

Для деталізації логіки взаємодії між основними компонентами системи в рамках ключових сценаріїв використання побудовано діаграми послідовності. Вони демонструють динаміку виконання операцій, порядок викликів методів, роль користувачів та підсистем у реалізації функціональних вимог. Діаграми акцентують увагу на важливих моментах: фіксації дій, ініціації аналітики, отриманні рекомендацій та перевірці відповідності дій користувачів політикам відповідальності.

На рисунку 2.3 зображено послідовність дій тімліда під час створення нового завдання та призначення ролей відповідно до моделі RACI. Цей сценарій є одним із центральних, оскільки формує базу для подальшого контролю виконання, аналізу відповідальності та виявлення порушень. Система дозволяє з інтерфейсу тімліда здійснити валідацію ролей, отримати підказки щодо кандидатів і зберегти матрицю у базі даних.

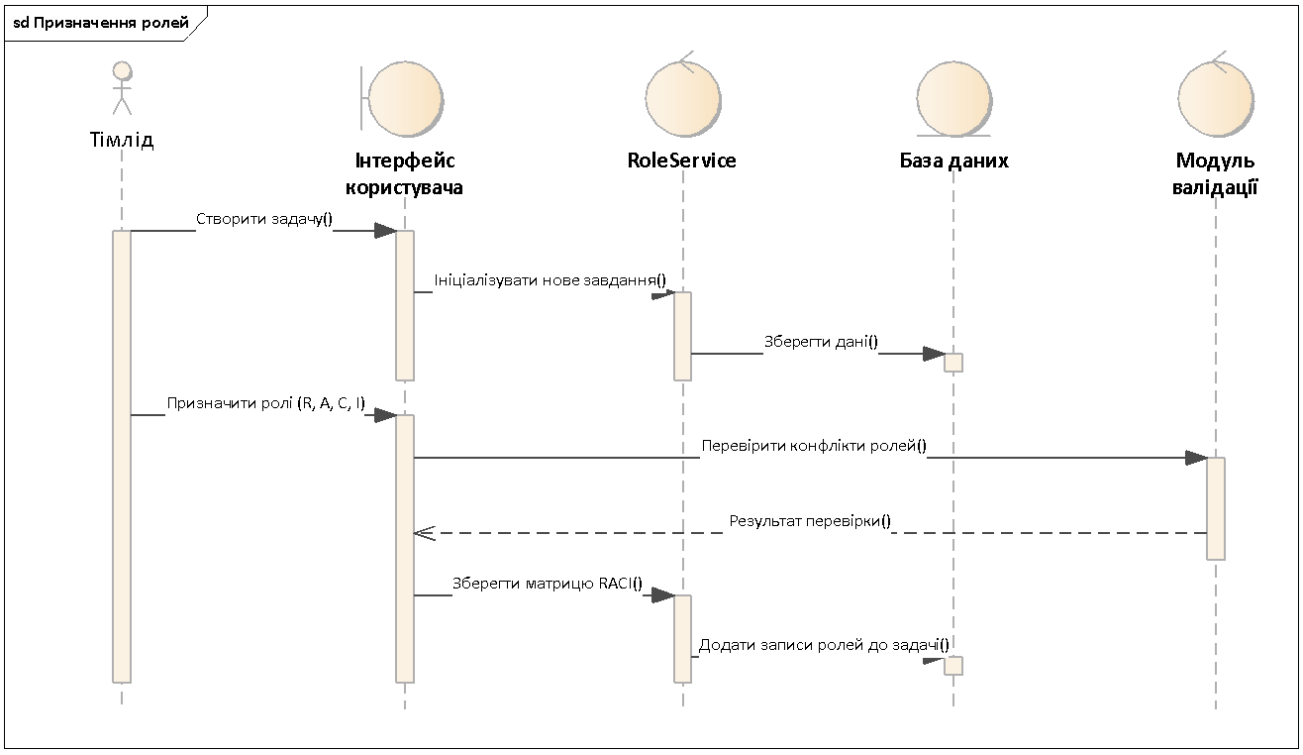


Рисунок 2.3 – Призначення ролей тімлідом у системі методом RACI

Джерело: розроблено автором самостійно

У діаграмі використано умовне позначення RoleService для компонента, що реалізує логіку керування ролями в задачах. Цей модуль відповідає за збереження та обробку інформації про ролі користувачів, формування RACI-матриць та взаємодію з підсистемами перевірки і базою даних.

У процесі призначення ролей у межах задачі тімлід взаємодіє з інтерфейсом користувача, який викликає відповідні функції керувального модуля RoleService. Цей компонент координує збереження даних про задачу, перевірку відповідності призначених ролей політикам та реєстрацію RACI-матриці. Модуль валідації (ValidationModule) виконує перевірки на логічні конфлікти між ролями. Для збереження результатів використовується сутність База даних, яка містить дані про задачі, учасників і ролі.

З метою деталізації архітектури цього сценарію побудовано діаграму класів, яка формалізує структуру граничного, керувального та сутнісних класів, що реалізують відповідну поведінку.

На ній відображено основні атрибути класів, їх функції та взаємозв’язки, які забезпечують виконання прецеденту «Призначення ролей методом RACI».

Це дозволяє чітко простежити розподіл відповідальності між компонентами системи та логіку їхньої інтеграції (рис. 2.4).

Діаграма демонструє структуру граничних, керувальних і сутнісних класів, що забезпечують створення задачі, перевірку ролей і збереження відповідальності в системі.

Наступний сценарій, який ми розглянемо, передбачає використання інтелектуальної складової системи для підтримки процесу прийняття рішень під час призначення ролей. Тімлід може ініціювати запит до модуля рекомендацій, який на основі історичних даних, моделей машинного навчання та мета-інформації про задачі формує список рекомендованих кандидатів на роль Responsible, Accountable тощо. Це дозволяє враховувати попередній досвід, рівень навантаження, успішність виконаних завдань, а також уникати суб’єктивного призначення.

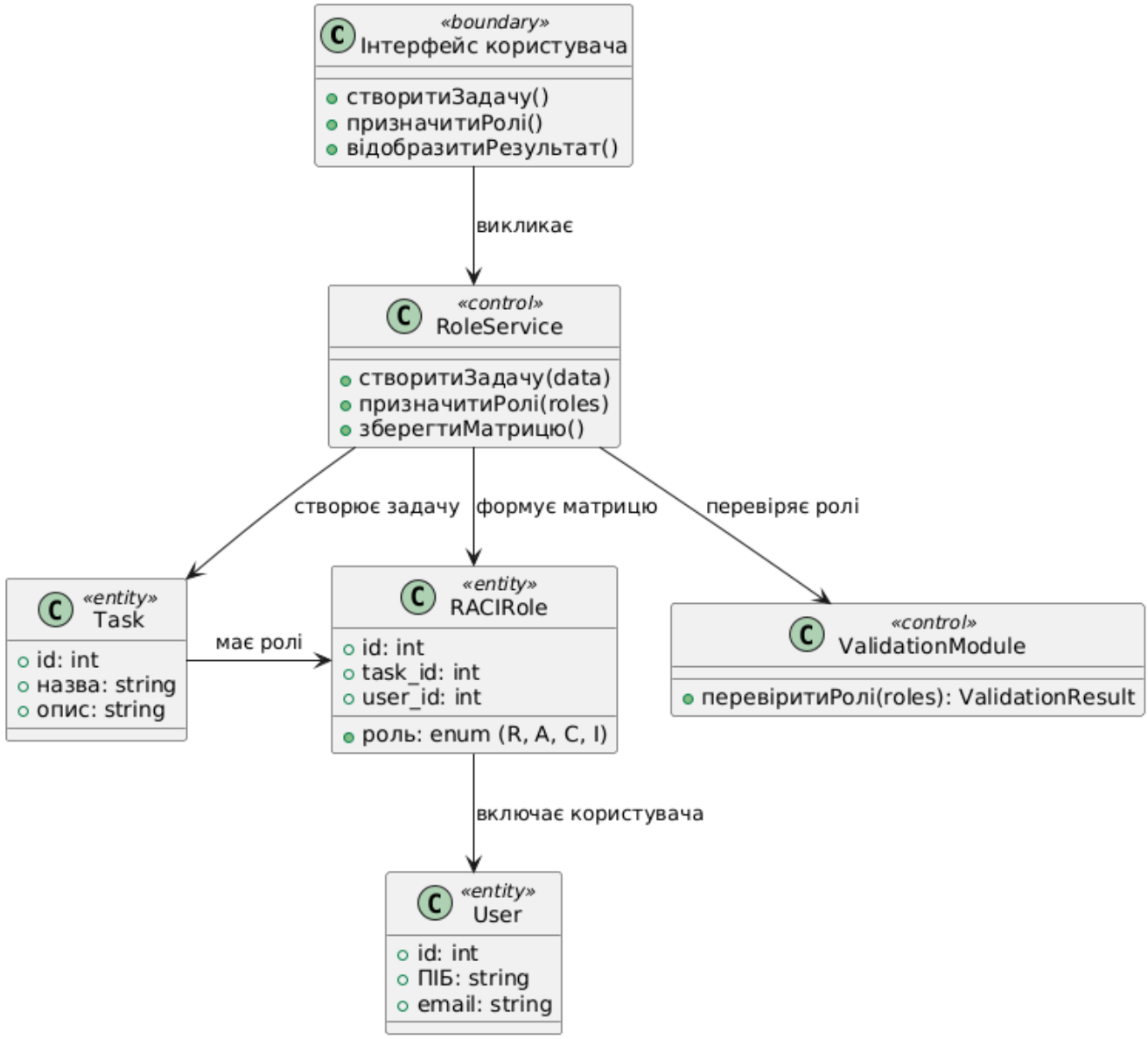


Рисунок 2.4 – Діаграма класів для реалізації прецеденту «Призначення ролей методом RACI»

Джерело: розроблено автором самостійно

На рисунку 2.5 зображено послідовність викликів між тімлідом, інтерфейсом користувача, модулем рекомендацій та аналітичною підсистемою, яка реалізує логіку оцінювання та підбору відповідальних. Результатом цього сценарію є автоматично згенерований список рекомендованих кандидатів для кожної ролі в межах задачі, який тімлід може редагувати або підтверджувати.

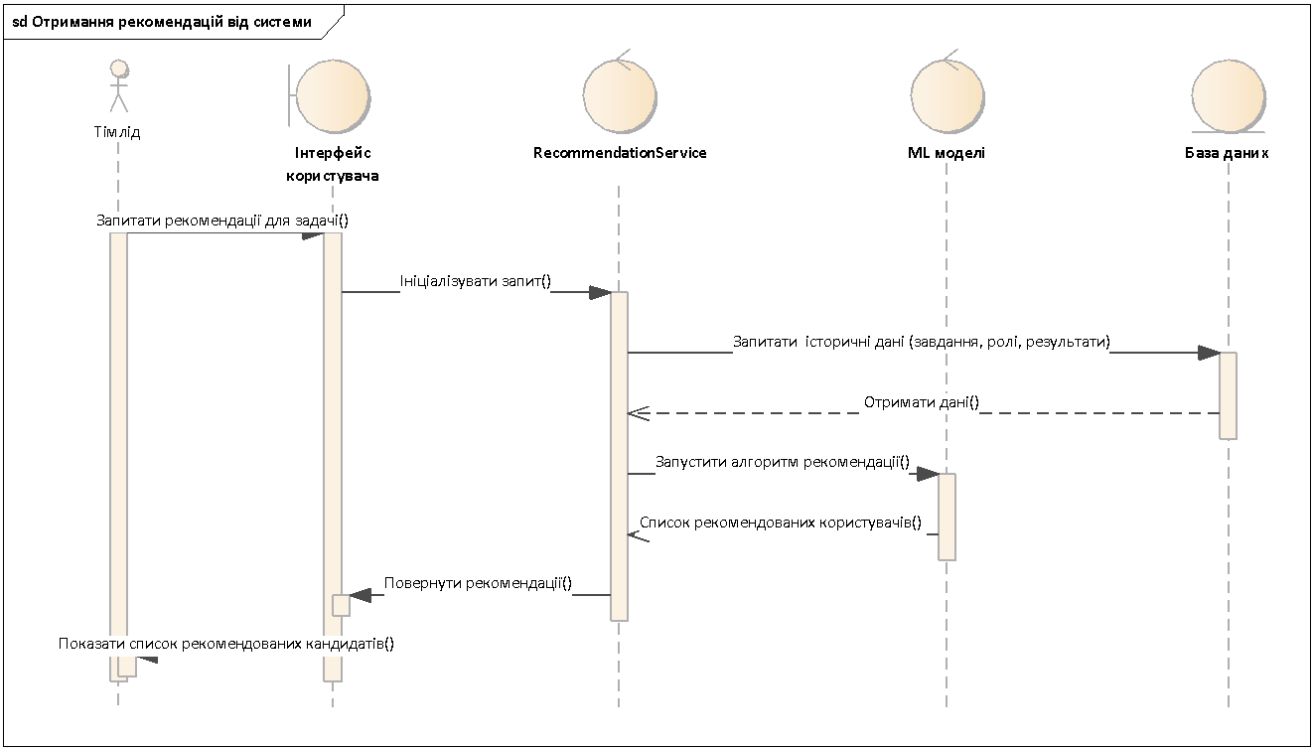


Рисунок 2.5 – Отримання тімлідом рекомендацій щодо призначення ролей

Джерело: розроблено автором самостійно

Компонент RecommendationService відповідає за обробку запитів на формування рекомендацій, використовує аналітичні моделі та взаємодіє з базою даних для аналізу історії задач, ролей і результатів їх виконання.

Для формалізованого подання архітектури цього сценарію побудовано діаграму класів, у якій відображено основні об’єкти взаємодії, їхню роль (граничний, керувальний або сутнісний клас) та залежності між ними (рис. 2.6).

Діаграма демонструє зв’язки між інтерфейсом користувача, керувальним класом RecommendationService, модулем ML-моделей та джерелом історичних даних. Такий підхід дозволяє чітко відокремити логіку управління, обчислень та зберігання, що відповідає принципам архітектури з розподілом обов’язків.

Рисунок 2.6 – Діаграма класів для реалізації прецеденту «Отримання рекомендацій від системи»

Джерело: розроблено автором самостійно

Ще одним важливим сценарієм використання інтелектуальної складової є оцінювання ймовірності невиконання задач у строк, перевантаженості учасників команди та потенційних конфліктів між призначеними ролями. Цей процес активується як за запитом тімліда, так і автоматично під час оновлення задач або формування RACI-матриці.

Модуль прогнозування аналізує історичні дані про виконання задач, обсяг навантаження, типи ролей, рівень взаємодії між користувачами, а також ознаки порушення політик. Застосовуються алгоритми машинного навчання, зокрема XGBoost або дерева рішень, що дозволяє формувати оцінку ризику та надавати зворотний зв’язок тімліду ще до остаточного затвердження ролей.

На рисунку 2.5 представлено послідовність взаємодії між тімлідом, модулем оцінки ризиків, аналітичною моделлю та базою даних. Результатом сценарію є зведений звіт із прогнозами ризику, що може бути врахований у прийнятті управлінських рішень.

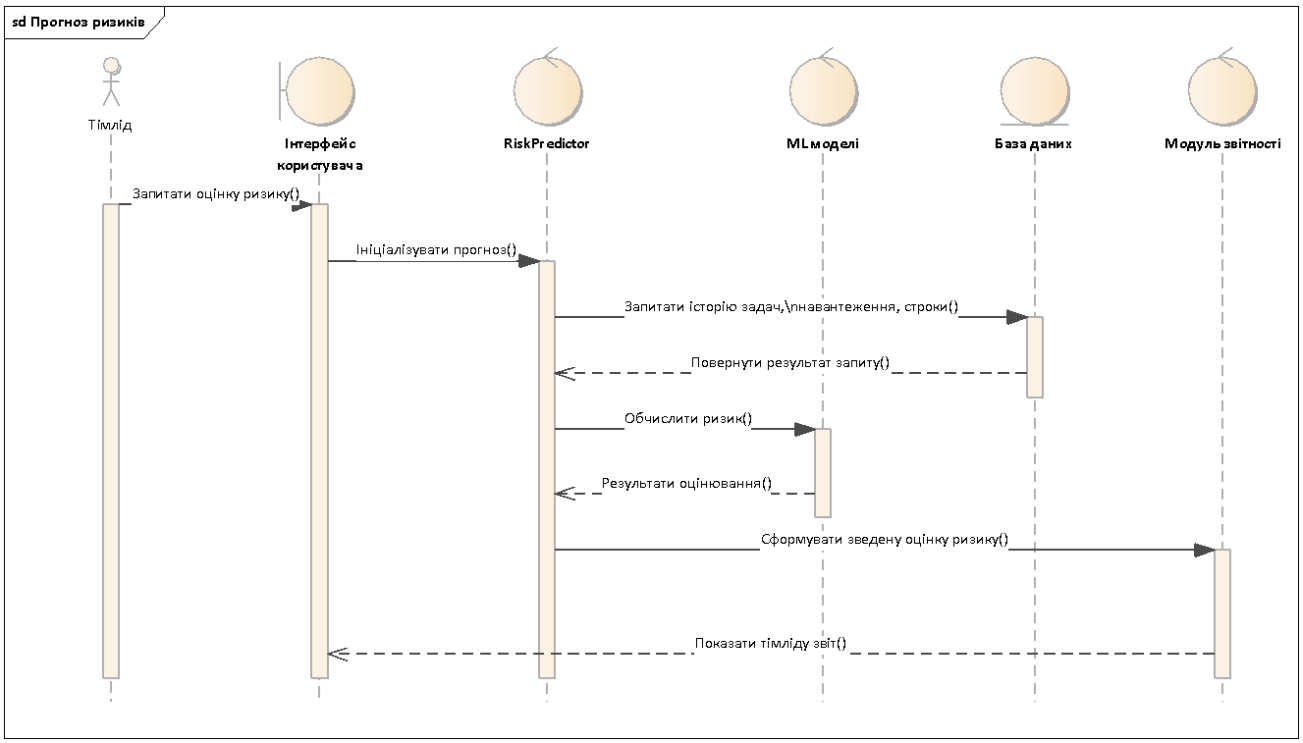


Рисунок 2.7 – Прогнозування ризику невиконання задач і перевантаження учасників

Джерело: розроблено автором самостійно

Компонент RiskPredictor використовує попередньо навчені моделі для оцінки ризиків за обраною RACI-структурою та історичними даними, формуючи інтегровану оцінку навантаження і ймовірності відхилення від плану.

Для забезпечення узгодженого представлення архітектури цього процесу побудовано діаграму класів, яка описує основні об'єкти, задіяні у сценарії прогнозування ризиків.

У моделі виокремлено граничний клас інтерфейсу тімліда, керувальний модуль RiskPredictor, аналітичну модель, а також сутності, що зберігають інформацію про функції, ролі, навантаження та історію виконання.

Ця структура дозволяє описати, як саме система отримує вхідні дані, передає їх на обчислення та повертає результат для ухвалення рішень (рис. 2.8).

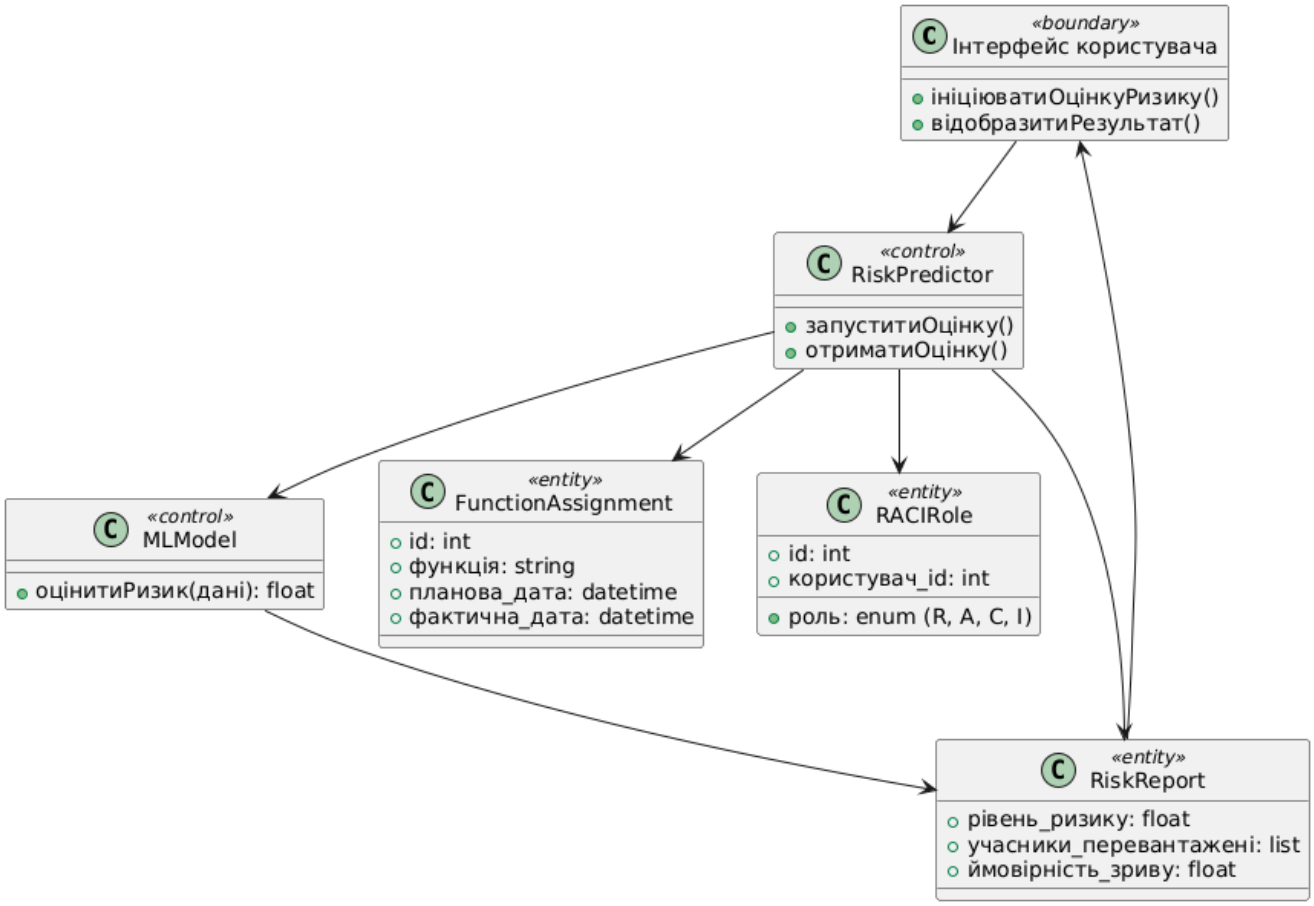


Рисунок 2.8 – Діаграма класів для реалізації прецеденту «Прогнозування ризику»

Джерело: розроблено автором самостійно

Наступний розглянутий сценарій ілюструє застосування інтелектуальної аналітики до журналів подій (логів), сформованих у процесі роботи користувачів. Такий аналіз дозволяє виявити випадки, коли дії виконували особи, не уповноважені відповідно до своєї ролі (наприклад, розробник виконує DevOps-операцію або аналітик змінює фінальні специфікації). Використовуючи підходи Process Mining та порівняння з очікуваними RACI-структурами, система фіксує та повідомляє про відхилення.

На рисунку 2.9 подано послідовність обробки логів – від запиту тімліда до формування звіту про виявлені порушення. У процесі беруть участь модулі

обробки логів, зіставлення з політиками ролей, генератор звітів та база даних, у якій зберігаються як історичні дані, так і результати аналізу.

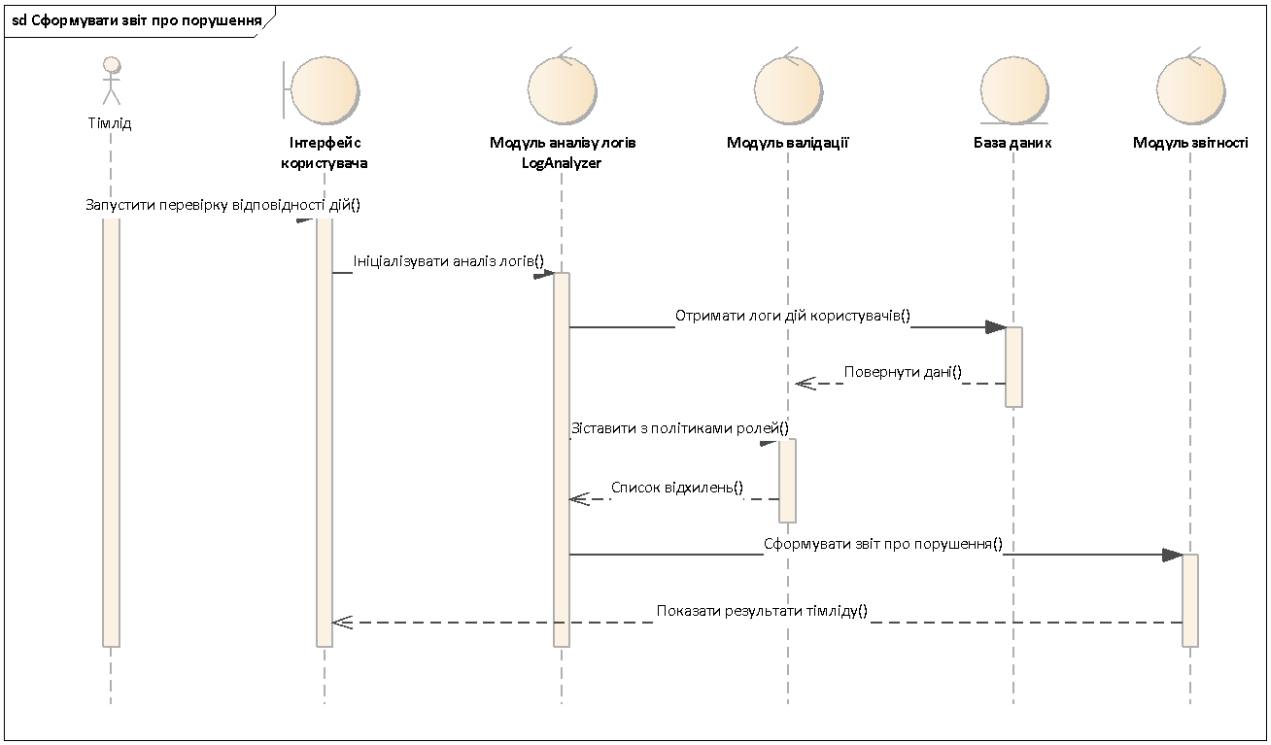


Рисунок 2.9 – Інтелектуальний аналіз логів і виявлення порушень відповідальності

Джерело: розроблено автором самостійно

Модуль LogAnalyzer використовує методи процесного аналізу та політик відповідності ролей для виявлення критичних відхилень від очікуваних моделей поведінки в межах задач або процесів.

Модуль LogAnalyzer використовує методи процесного аналізу та політик відповідності ролей для виявлення критичних відхилень від очікуваних моделей поведінки в межах задач або процесів.

Для відображення логіки реалізації цього сценарію побудовано діаграму класів, у якій змодельовано компоненти, відповідальні за ініціацію перевірки, обробку журналів дій, виявлення відхилень і генерацію звітів.
У діаграмі виокремлено граничний клас взаємодії з тімлідом, керувальний модуль LogAnalyzer, спеціалізований клас DeviationDetector для ідентифікації порушень, а також сутності ActionLog та RACIRole, що є джерелом інформації про реальні та

очікувані дії. У результаті формується звіт, що може бути використаний для внутрішнього аудиту або перегляду призначень ролей.

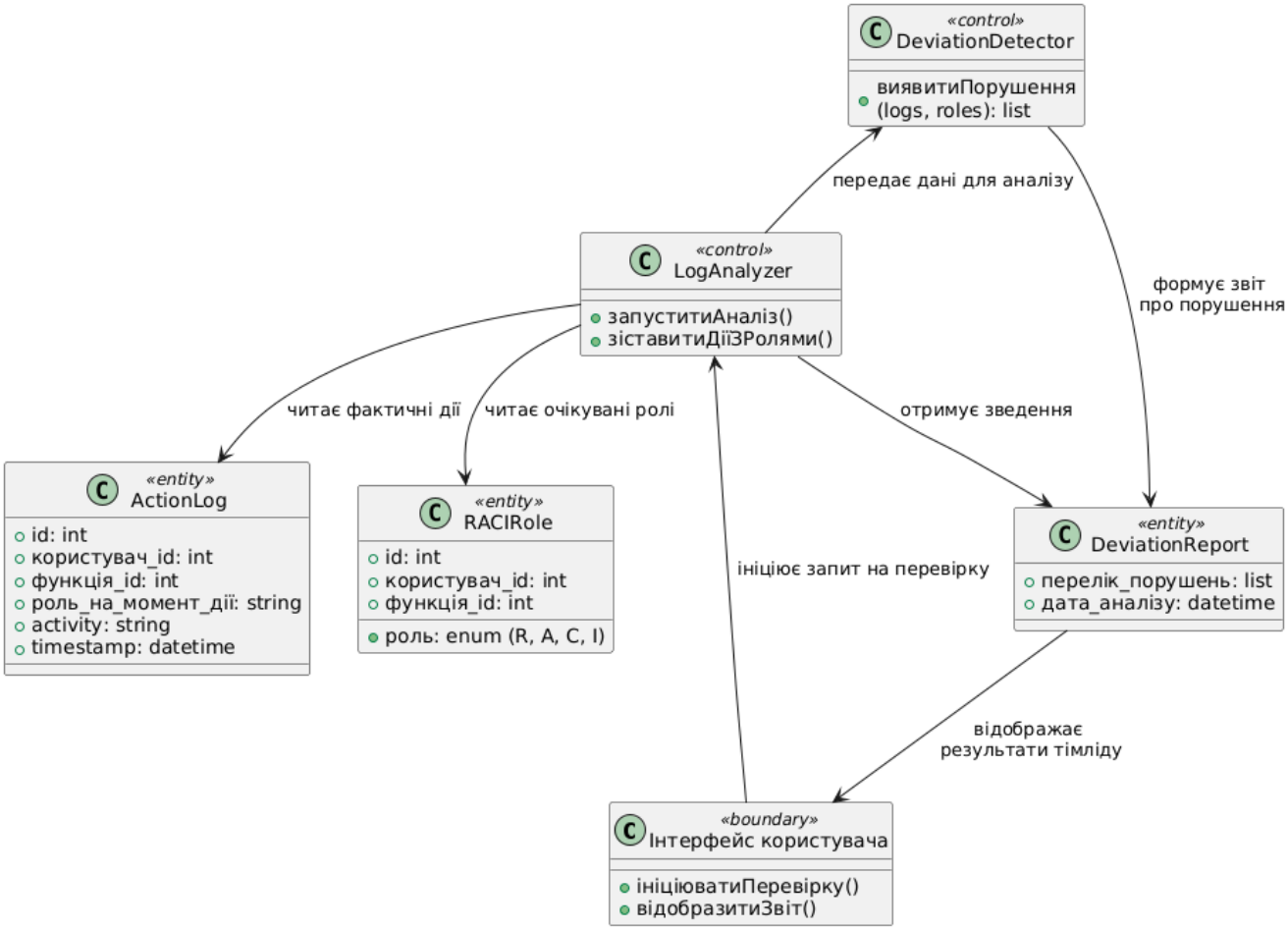


Рисунок 2.10 – Діаграма класів для реалізації прецеденту «Аудит та логування дій»

Джерело: розроблено автором самостійно

Для системного подання логічної структури даних, які використовуються під час виконання функцій управління ролями та відповідальністю, сформовано діаграму класів-сутностей. Вона відображає основні об'єкти предметної області, їхні атрибути та взаємозв’язки, що є важливим при проєктуванні структури бази даних, налаштуванні зв’язків між модулями та формуванні запитів до інтелектуальних моделей.

На рисунку 2.11 представлено ключові сутності системи: учасники, проєкти, функції (згруповані у межах проєкту), ролі RACI, а також журнал дій (логування).

Особливу увагу приділено таблиці Функції_проєкту, яка містить поля як для планових, так і фактичних строків виконання, що дозволяє будувати процесний аналіз і прогнозувати відхилення. Структура таблиці ActionLog підтримує як класичний аудит, так і аналітичні сценарії, включаючи побудову соціальних графів та процесних моделей.

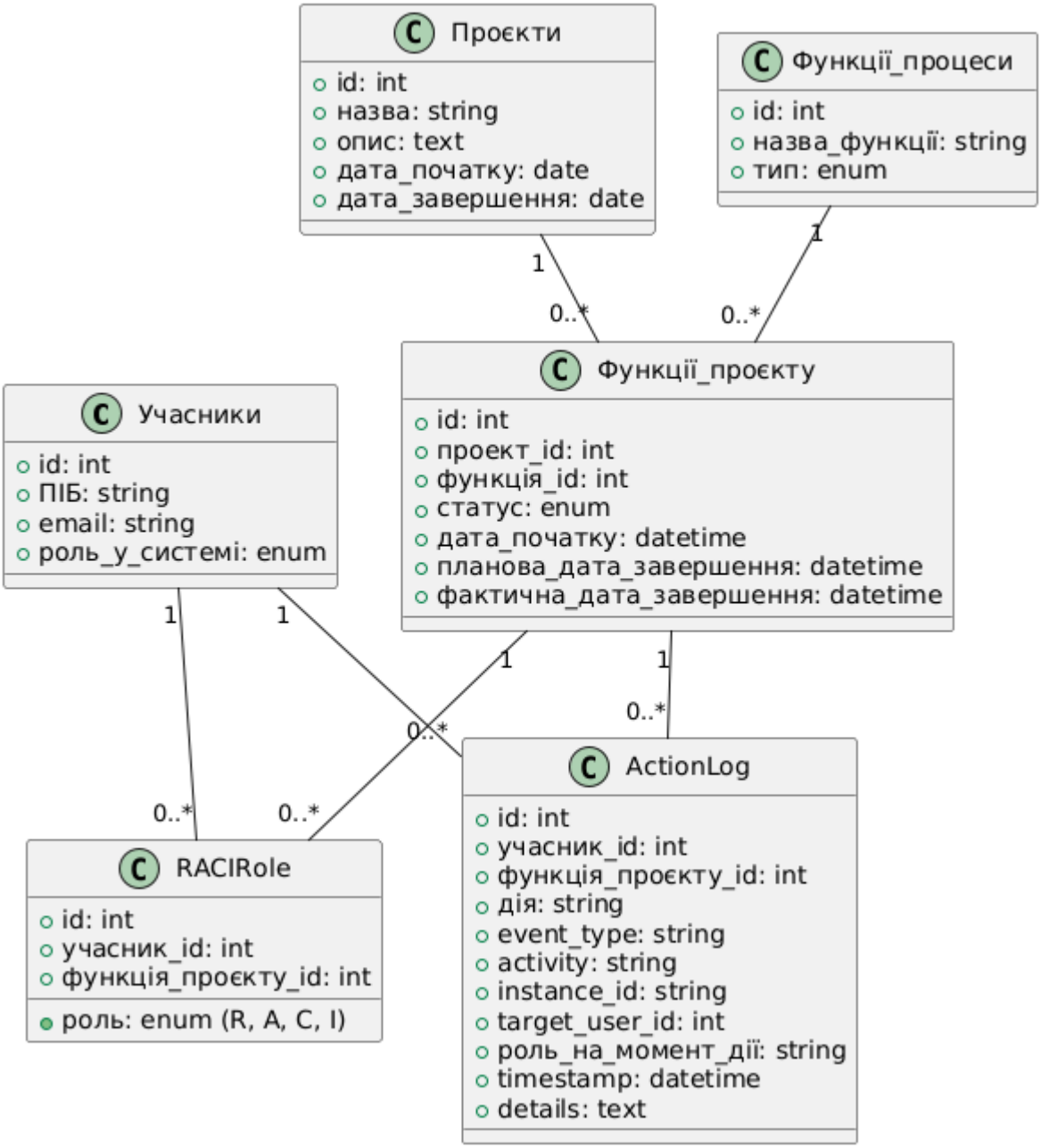


Рисунок 2.11 – Діаграма класів-сутностей системи управління ролями та відповідальністю

Джерело: розроблено автором самостійно

Діаграма моделює логіку зберігання і пов’язаність даних між учасниками, функціями, проєктами, призначеними ролями та подіями у системі. Вона є основою для реалізації бази даних у SQL Server та забезпечення інтелектуального аналізу.

2.2 Методи та моделі в інформаційних управляючих системах управління ролями і відповідальністю при розробці програмного забезпечення

У межах розроблення інформаційної системи управління ролями та відповідальністю важливим аспектом є поєднання традиційних підходів до розподілу обов’язків із інтелектуальними методами аналізу даних. У даному підпункті розглянуто формальні моделі, які лежать в основі управління ролями в задачах, включаючи RACI-структури, матриці відповідальності та механізми перевірки конфліктів. Крім того, представлено математичні моделі та алгоритми, що реалізують функції автоматизованого призначення ролей, прогнозування перевантаження, аналізу соціальної взаємодії та виявлення порушень відповідальності на основі логів. Особливу увагу приділено формалізації цих моделей із використанням відповідних функцій, формул і логічних структур, що забезпечує прозорість, відтворюваність і точність аналітичних рішень у системі.

До переліку моделей, що використовуються для вирішення задач управління ролями та відповідальністю в даній інформаційній системі, належать:

- RACI-модель (Responsible, Accountable, Consulted, Informed)
- Модель рольового доступу RBAC
- Модель доступу на основі атрибутів ABAC
- Формалізована матриця відповідальності
- Алгоритми перевірки конфліктів ролей

- Класифікаційні моделі машинного навчання (Logistic Regression, Decision Tree, Random Forest)
- Алгоритми колаборативної фільтрації
- Моделі градієнтного бустингу (XGBoost)
- Методи аналізу соціальних мереж (SNA)
- Алгоритми процесного майнінгу (Process Mining)
- Методи виявлення аномалій за допомогою відхилень від політик RACI
- RACI-модель.

RACI-модель.

RACI-модель (Responsible – Accountable – Consulted – Informed) є однією з найпоширеніших формалізованих структур, що використовуються в управлінні задачами, процесами та ролями в командній роботі. Її суть полягає в тому, щоб чітко визначити роль кожного учасника в контексті виконання конкретного завдання чи функції. Це дозволяє уникнути конфліктів відповідальності, дублювання дій та прогалин у контролі [43].

У формалізованому вигляді матриця відповідальності RACI для задачі $\boldsymbol{T}$ та множини користувачів $U = \{u_{1,}u_2, \dots, u_n\}$ і множині ролей $R = \{R, A, C, I\}$ може бути представлена як булева матриця $M \in \{0,1\}^{n\,x\,4}$, де

$M_{ij} = \{1,$ якщо користувач u_i має роль R_j у задачі T $0,$ інакше

(2.1)

Для забезпечення логічної узгодженості моделі застосовуються наступні правила цілісності:

- для кожної задачі обов’язково має бути хоча б один користувач із роллю A (Accountable);
- один користувач не може одночасно мати ролі R і A для тієї самої задачі (конфлікт виконавця та відповідального);

- ролі C (Consulted) та I (Informed) можуть бути призначені кільком учасникам.

RACI-модель застосовується не лише на рівні проєктного управління, а й у побудові інформаційних систем – для генерації інтерфейсів відповідальності, контролю політик доступу, перевірки відповідності дій та побудови RACI-матриць у базі даних.

У системах, орієнтованих на контроль доступу до інформаційних ресурсів, широке застосування мають моделі рольового доступу (RBAC – Role-Based Access Control) та моделі доступу на основі атрибутів (ABAC – Attribute-Based Access Control). Вони дозволяють встановлювати політики доступу залежно від ролей користувачів або сукупності атрибутів, що описують як самих користувачів, так і об'єкти доступу. Такі підходи є типовими для розмежування прав у корпоративних IT-системах, де ключовою задачею є захист даних і запобігання несанкціонованим діям.

Натомість у розробленій інформаційній системі фокус зроблено не на контролі доступу, а на організаційному розподілі відповідальності в рамках виконання проєктних функцій. Тому центральне місце займають RACI-матриці та інші моделі, що описують взаємозв’язки між користувачами, задачами та ролями.

Формалізована матриця відповідальності та перевірка конфліктів.

Формалізована матриця відповідальності є розширенням логіки RACI-моделі та слугує основою для автоматизованого аналізу ролей у межах задач [44]. У контексті реалізації системи така матриця подається у вигляді відображення задач (функцій проєкту) на множину користувачів із визначенням конкретної ролі кожного з них.

Формально, для множини задач $T = \{t_1, t_{2,}, \dots t_m\}$, множини користувачів $U = \{u_{1,}u_2, \dots, u_n\}$ і множині ролей $R = \{R, A, C, I\}$ будується тривимірне відображення:

$$M: T \times U \rightarrow R \cup \{\emptyset\}, \quad (2.2)$$

де $M(t_i, u_j) = r_k$, якщо користувач u_j виконує роль r_k у задачі t_i, або $\emptyset$, якщо не призначено.

Для підтримки логічної цілісності застосовується низка формалізованих правил перевірки конфліктів:

1. У кожній задачі має бути рівно один користувач із роллю $\boldsymbol{A}$:

$$\sum_{j=1}^{n} \quad \delta(M(t_i, u_j) = A = 1, \forall t_i \in T \tag{2.3}$$

2. Користувач не може мати ролі $\boldsymbol{R}$ та $\boldsymbol{A}$ одночасно:

$$\neg\exists u_j \in U : M(t_i, u_j) = R \cap M(t_i, u_j) = A \tag{2.4}$$

3. Ролі $\boldsymbol{C}$ і $\boldsymbol{I}$ не виключаються, але можуть бути обмежені в кількості залежно від політик організації.

Валідаційний модуль системи реалізує ці правила у вигляді набору предикатів і виводить звіти про порушення або автоматично блокує некоректне призначення. Таким чином, формалізована матриця відповідальності служить не лише для фіксації розподілу ролей, а й як об'єкт логічної перевірки, аналітики та подальшої інтелектуальної обробки.

Рекомендаційна модель призначення ролей на основі колаборативної фільтрації.

Для автоматизованого формування пропозицій щодо призначення ролей у межах нових функцій проєкту пропонуємо використовувати підхід колаборативної фільтрації. У цьому контексті кожна функція розглядається як об'єкт, а користувач – як потенційний "виконавець" ролі. Історія попередніх призначень створює основу для побудови матриці, аналогічної тій, що застосовується в системах рекомендацій (user-item interaction matrix) [45, 46].

Нехай $U = \{u_{1,} u_2, \dots, u_n\}$ – множина користувачів, $F = \{f_1, f_2, \dots, f_n\}$ - множина функцій (завдань), і $R(u_i, f_j \in \{0,1\}$ – бінарна ознака, що користувач u_i мав певну роль у функції f_j. Тоді матриця призначень $R \in \{0,1\}^{m \times n}$ піддається факторизації:

$$R \approx P \cdot Q^T, \text{ де} \tag{2.5}$$

$P \in R^{m \times k}$ – матриця латентних характеристик користувачів;

$Q \in R^{n \times k}$ – матриця латентних характеристик функцій;

$\boldsymbol{k}$ – кількість латентних ознак (гіперпараметр).

Оцінка відповідності користувача u_i на новому наборі даних визначається як:

$$\hat{R}_{ij} = P_i \cdot Q_j^T \tag{2.6}$$

і може бути використана для ранжування кандидатів на роль у новій функції. Якщо дані є розрідженими, застосовуються регуляризація та стохастичні методи оптимізації, наприклад, Funk SVD або ALS.

Перевага даного підходу – здатність враховувати неявні закономірності у виборі виконавців, навіть якщо функції не ідентичні, але мають спільні патерни ролей і контексту.

Прогнозування ризику перевантаження та зриву задач на основі XGBoost

Інформаційна система повинна не лише рекомендувати відповідальних осіб, а й попереджати ситуації перевантаження або зриву строків виконання задач. Для цього реалізується модель прогнозування ризику на основі алгоритму градієнтного бустингу (XGBoost), який дозволяє будувати точні прогностичні моделі для структурованих даних [47].

Метою моделі є ідентифікація потенційно ризикованих функцій або користувачів, чий рівень навантаження може вплинути на строки виконання завдань у проєкті.

Нехай

- $x_i \in R^d$ – вектор ознак для функції і (кількість учасників, ролі, тип функції, кількість днів до дедлайну тощо);
- $y_i \in \{0,1\}$ – ознака того, чи була функція виконана із запізненням або ризиком (1 – так, 0 – ні).

Алгоритм XGBoost формує прогноз:

$\hat{y}_i = \sum_{k=1}^{K} \quad f_k(x_i), f_k \in F$, де (2.7)

F – простір дерев рішень, а кожне f_k– окреме дерево. Під час навчання XGBoost мінімізує регуляризовану функцію втрат:

$L = \sum_{i=1}^{n} \quad l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \quad \Omega(f_k).$

(2.8)

Результат – це оцінка ймовірності ризику для конкретної задачі або для кожного виконавця в ролі. Якщо ймовірність перевищує порогове значення Θ, система може автоматично позначити задачу як ризиковану.

Для оцінки ризику перевантаження окремого користувача додатково використовується емпіричний індикатор:

$\text{Ризик}_{\text{перевантаження}}(u_j) = \frac{Z_j}{Z_{max}}$ (2.9)

де Z_j – поточна кількість активних ролей у користувача u_j, а Z_{max} – встановлений поріг максимально допустимого навантаження. Така оцінка дозволяє не лише виявити задачі з ризиком зриву, а й попередити ситуації перевтоми та неефективного розподілу обов'язків між членами команди.

Аналіз соціальної взаємодії (SNA) на основі графових моделей.

Аналіз соціальної взаємодії (SNA, Social Network Analysis) виступає важливим інструментом для виявлення структурних характеристик командної роботи, оцінки залученості учасників та діагностики можливих комунікаційних дисфункцій. У межах розроблюваної інформаційної системи реалізовано графову модель соціальної взаємодії, яка базується на даних журналів дій користувачів (ActionLog) і дозволяє формалізувати схеми реального обміну інформацією в команді [48].

Формально модель соціальної взаємодії визначається як граф $G = (V, E)$, де $V = \{u_1, u_2, \dots, u_n\}$ – множина учасників проєкту, а $E \subseteq V \times V$ – множина зв'язків між ними, які утворюються внаслідок спільної участі в задачах, коментування, взаємного призначення ролей тощо. Кожне ребро (u_i, u_j) має вагу $w(u_i, u_j)$, що відповідає кількості зафіксованих взаємодій між користувачами u_i та u_j протягом певного періоду.

Для аналізу графа застосовуються наступні SNA-метрики.

1. Центральність степеня:

$C_D(u_i)=deg(u_i)$

(2.10)

визначає кількість безпосередніх зв'язків користувача та інтерпретується як рівень його загальної залученості у взаємодії.

2. Центральність посередництва (betweenness):

$C_B(u_i) = \sum_{s \neq u_i \neq t} \frac{\sigma_{st}(u_i)}{\sigma_{st}}$,

(2.11)

де σ_{st} – кількість найкоротших шляхів від користувача s до t, а $\sigma_{st}(u_i)$ – число таких шляхів, що проходять через u_i. Ця метрика вказує на посередницьку роль учасника у загальному інформаційному потоці.

3. Виявлення ізольованих учасників:

$des(u_i) = 0 \Rightarrow$ ізоляція (2.12)

дозволяє виявити користувачів, які повністю виключені з активної взаємодії.

Практичне застосування SNA в системі включає:

- ідентифікація неформальних лідерів (найцентраліші учасники),
- виявлення учасників з низькою залученістю, які ризикують бути недоінформованими,
- виявлення вузьких місць – учасників, без яких порушується потік взаємодії,

Результати аналізу використовуються для перегляду призначених ролей, уточнення комунікаційних стратегій та підвищення ефективності командної взаємодії..

Аналіз логів та виявлення порушень відповідальності (Process Mining)

З метою контролю дотримання відповідальності, визначеної за RACI-матрицею, у розробленій системі реалізовано механізм аналізу логів на основі підходів Process Mining. Цей підхід дозволяє виявляти відхилення між формально

призначеними ролями та фактичними діями користувачів, які відображено у журналах подій (ActionLog). Такий аналіз є важливим для підтримання прозорості управлінських процесів і виявлення порушень політик відповідальності [49].

Кожну подію у системі описує розширений кортеж:

$e_i = (a_i, t_i, u, r_i)$,

(2.13)

де a_i – тип дії (наприклад, створення, редагування, зміна статусу),

t_i – час виконання дії,

u_i – користувач, який виконав дію,

r_i – роль користувача в момент виконання дії згідно з RACI.

Кожному типу дії a відповідає множина допустимих ролей $R_{expected}(a)$, яка визначається внутрішніми правилами або політиками системи. Порушення фіксується тоді, коли фактична роль r_i не належить до очікуваної множини:

$\Delta = \{e_i \in E | r_i \notin R_{expected}(a_i)\}$,

де Δ – множина подій, що є порушенням політик відповідальності. Таке порушення може свідчити про потенційну загрозу безпеці, ігнорування процедур або недостатню формалізацію ролей.

До типових прикладів виявлених відхилень належать:

- виконання критичних дій учасниками без призначених ролей;
- зміна статусу задачі особою, що не має ролі Responsible;
- делегування рішень особам, що не є Accountable.

Практичне застосування:

- побудова реальних моделей виконання функцій на основі логів (фреймворки Alpha Miner, Heuristic Miner);
- виявлення відхилень від політик RACI у режимі реального часу або з періодичним аудитом;

- формування зведених звітів для тімлідів або адміністраторів із зазначенням критичних порушень;
- використання даних для перегляду матриць відповідальності та посилення контролю.

Отримані результати аналізу логів інтегруються в звітність та можуть автоматично активувати механізми сповіщення або ініціювати перегляд RACI-структури задачі. Таким чином, Process Mining виступає інструментом верифікації відповідальності та підтримки аудиту виконання проєктних функцій.

Висновки до розділу 2

У процесі аналізу було визначено основні типи користувачів системи, серед яких тімлід, виконавець, адміністратор та інтелектуальний модуль, що виконує фонову аналітику. Для кожного з користувачів сформульовано функціональні вимоги, які стали основою для побудови діаграми прецедентів та подальшого моделювання логіки роботи системи.

Реалізація ключових сценаріїв була представлена у вигляді діаграм послідовності, що відображають динаміку взаємодії між користувачами та функціональними компонентами, зокрема під час призначення ролей, формування рекомендацій, прогнозування ризиків і аналізу логів. Для кожного прецеденту додатково побудовано діаграму класів з виділенням boundary-, control- та entity-класів, що дозволило формалізувати внутрішню архітектуру сценаріїв.

На завершення розроблено узагальнену діаграму класів-сутностей, яка описує логічну структуру предметної області та зберігання даних у системі. Особливу увагу приділено структурі таблиці ActionLog, що є джерелом для інтелектуального аналізу, та уточненню атрибутів таблиці Функції_проєкту для підтримки план-факт оцінювання. Отримані моделі створюють основу для побудови фізичної архітектури та реалізації інтелектуальної функціональності системи.

Було проведено систематизацію методів і моделей, які формують аналітичне та логічне ядро інформаційної системи управління ролями та відповідальністю. Основою є RACI-модель, що формалізує структуру відповідальності в командній роботі та дозволяє забезпечити однозначність розподілу обов’язків. Для її підтримки запропоновано формальну модель матриці відповідальності з наборами перевірок цілісності, яка забезпечує автоматизовану валідацію призначених ролей відповідно до встановлених політик.

Особливу увагу приділено впровадженню інтелектуальних моделей, які на основі історичних даних, поведінкових патернів та взаємодії користувачів дозволяють підвищити якість прийняття рішень. Зокрема, реалізовано колаборативну фільтрацію для рекомендацій призначення ролей, XGBoost-модель для прогнозування ризику перевантаження, графовий аналіз для виявлення неформальних лідерів та ізольованих учасників, а також механізми Process Mining для виявлення порушень відповідальності на основі логів.

Застосування вищенаведених моделей дозволяє не лише формалізувати організаційну структуру в межах проєктної діяльності, а й вивести управління ролями на рівень адаптивної, прогнозної та аналітичної системи. Таким чином, сформовано цілісний підхід, що поєднує логіку управління з інструментами машинного навчання для забезпечення точності, обґрунтованості та прозорості процесу розподілу відповідальності.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

3.1 Проектування бази даних для інформаційної управляючої системи

Проєктування бази даних є важливим етапом створення інформаційної управляючої системи, оскільки саме структура зберігання даних визначає ефективність обробки, цілісність інформації та можливість її подальшого аналізу. На цьому етапі формується логічна модель предметної області, яка відображає взаємозв’язки між сутностями, ролями, функціями й подіями, що виникають у процесі розробки програмного забезпечення. База даних відіграє роль центрального репозитарію, де акумулюється інформація про учасників, проєкти, функції та дії, забезпечуючи цілісну підтримку процесів управління ролями та відповідальністю.

3.1.1 Проєктування інфологічної моделі бази даних

Інфологічне логічне моделювання (ІЛМ) є проміжним етапом між концептуальним описом предметної області та фізичним проєктуванням бази даних. Метою ІЛМ є формалізація структурованої інформації про об’єкти системи, їхні атрибути, типи зв’язків і обмеження, необхідні для подальшої реалізації в реляційній СУБД. У контексті даної інформаційної системи моделювання охоплює наступні сутності:

- проєкти;
- функції проєктів;
- учасники;
- ролі відповідальності;
- логи дій.

Для побудови логічної моделі бази даних було використано спеціалізоване середовище Vertabelo [50], яке забезпечує засоби для створення діаграм сутність-зв’язок, визначення атрибутів, типів даних, ключів і обмежень цілісності. Інструмент дозволяє отримати повноцінне представлення логіки зберігання даних у вигляді наочної схеми, що може бути безпосередньо експортована у вигляді SQL-інструкцій для розгортання структури бази даних у середовищі Microsoft SQL Server.

Застосування Vertabelo дало змогу детально опрацювати структуру інформаційної системи з урахуванням вимог до масштабованості, гнучкості в розширенні та підтримки складних зв’язків між сутностями. Створена логічна модель стала основою для формування фізичної структури бази даних, а також послугувала джерелом для автоматичної генерації документації до проєкту.

На рис. 3.1 показана логічна модель бази даних, спроєктована в середовищі Vertabelo.

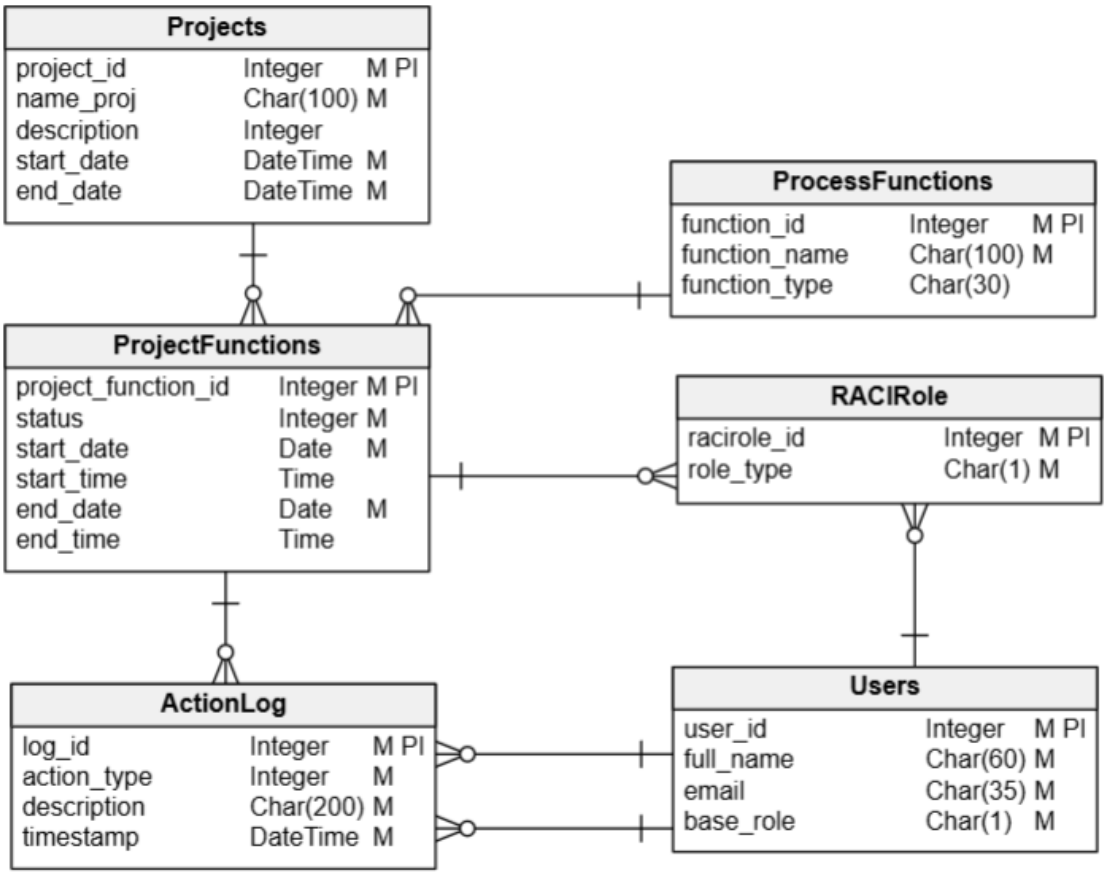


Рисунок 3.1 – Логічна модель бази даних, побудована у Vertabelo

Джерело: розроблено автором самостійно

Логічна модель включає шість основних таблиць, кожна з яких відображає окремий аспект управління процесом розробки програмного забезпечення. Центральне місце займає таблиця «Проєкти», до якої прив’язуються функції проєкту, пов’язані з довідником функцій-процесів.

Учасники, які беруть участь у реалізації функцій, мають відповідні ролі згідно з моделлю RACI, що реалізується через зв’язувальну таблицю. Для забезпечення прозорості та контролю системи передбачено окрему таблицю для реєстрації дій користувачів.

Кожна сутність відображає конкретний об’єкт предметної області, має власний набір атрибутів і бере участь у зв’язках, які забезпечують цілісність і узгодженість даних. Надамо опис таблиць логічної моделі бази даних, зокрема:

- сутність Projects зберігає основні відомості про проєкти розробки програмного забезпечення, вона містить унікальний ідентифікатор проєкту, його назву, опис, дату початку та завершення;

- сутність ProcessFunctions є довідником типових функцій, які можуть бути використані в межах різних проєктів, включає атрибути, що описують назву функції, її тип та класифікацію (наприклад, бізнес-процес чи технічна задача);

- сутність ProjectFunctions реалізує зв’язок між функціями та конкретним проєктом і зберігає атрибути, що описують контекст використання функції у проєкті (статус, часові межі реалізації тощо);

- сутність Users містить відомості про всіх учасників системи, включаючи їхні ідентифікаційні дані, ім’я, електронну адресу та базову роль у проєкті;

- сутність RACIRole реалізує модель відповідальності RACI, описуючи, яку саме роль (Responsible, Accountable, Consulted, Informed) виконує конкретний користувач стосовно певної функції в межах проєкту;

- сутність ActionLog відповідає за зберігання інформації про події, що відбуваються в системі, до неї входять тип події, опис дії, дата та час, учасник, який ініціював подію, і, за потреби, цільовий користувач.

3.1.2 Проєктування фізичної моделі БД в середовищі Microsoft SQL Server

На основі логічної моделі, розробленої в середовищі Vertabelo, було згенеровано фізичну модель бази даних, яка конкретизує типи даних атрибутів, обмеження цілісності, первинні та зовнішні ключі, а також nullable-поля. Ця модель відповідає специфікації платформи Microsoft SQL Server і є підготовленою до реалізації на фізичному рівні.

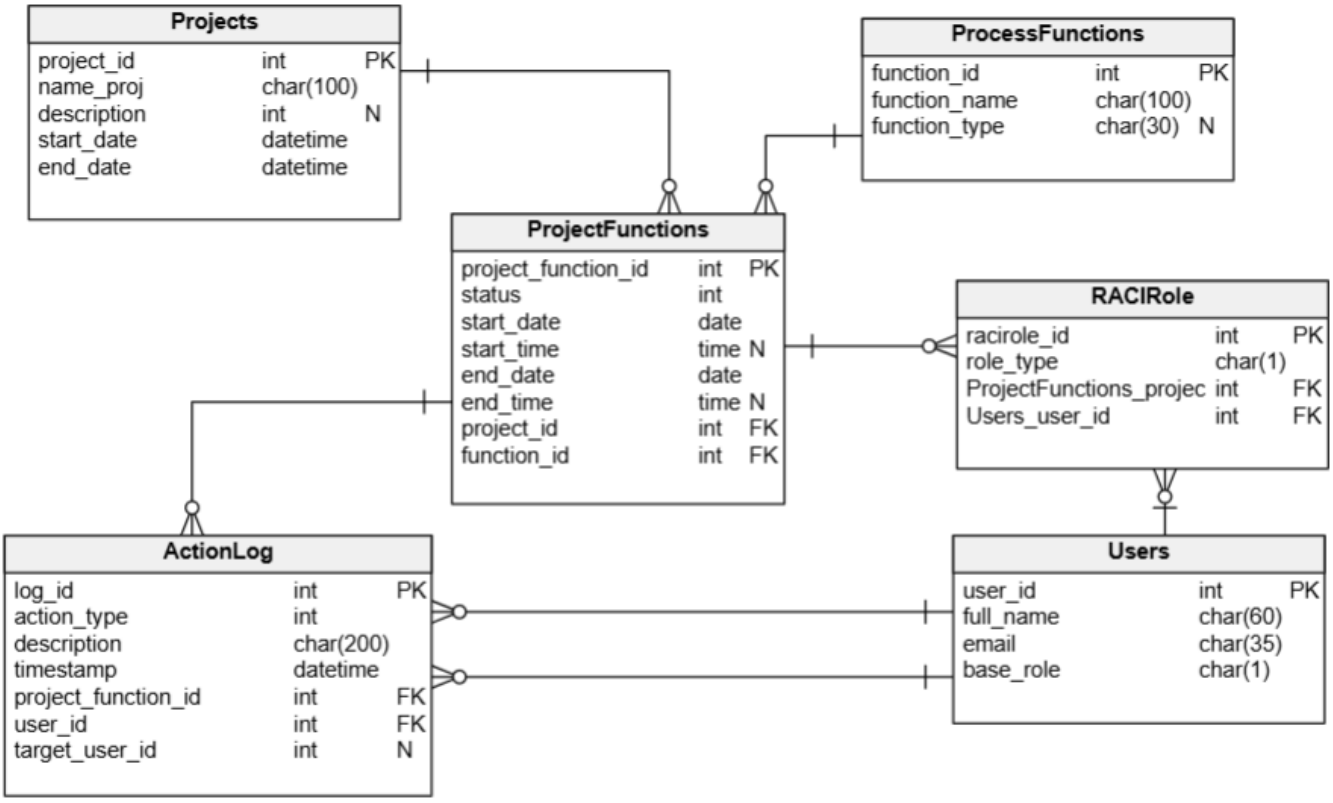


Рисунок 3.2 – Фізична модель бази даних, згенерована у Vertabelo

Джерело: розроблено автором самостійно

Фізична модель забезпечує технічну основу для створення таблиць у СУБД, з урахуванням вимог до формату полів (наприклад, char, int, datetime), а також вказує, які атрибути допускають відсутність значення (позначені як N – nullable), і як реалізовані зв’язки між таблицями.

Для забезпечення роботи системи обрано реляційну модель даних, яка реалізується за допомогою платформи Microsoft SQL Server. Ця СУБД забезпечує високий рівень масштабованості, підтримку транзакцій, засоби контролю доступу й вбудовані інструменти для забезпечення безпеки та продуктивності.

Фізична модель була згенерована автоматично за допомогою середовища Vertabelo, яке на основі побудованої логічної структури дозволяє отримати повний SQL-скрипт для створення таблиць, ключових обмежень та зв'язків у форматі, сумісному з Microsoft SQL Server. Згенерований скрипт показаний у додатку А. Такий підхід значно скорочує час розробки, мінімізує ризики помилок при ручному кодуванні та забезпечує узгодженість між логічною та фізичною моделями.

Окрім візуального подання структури бази даних, Vertabelo дозволяє автоматично експортувати створену схему у вигляді DDL-інструкцій, які можуть бути безпосередньо використані для розгортання бази у СУБД. Це особливо важливо для командної розробки та подальшого супроводу інформаційної системи.

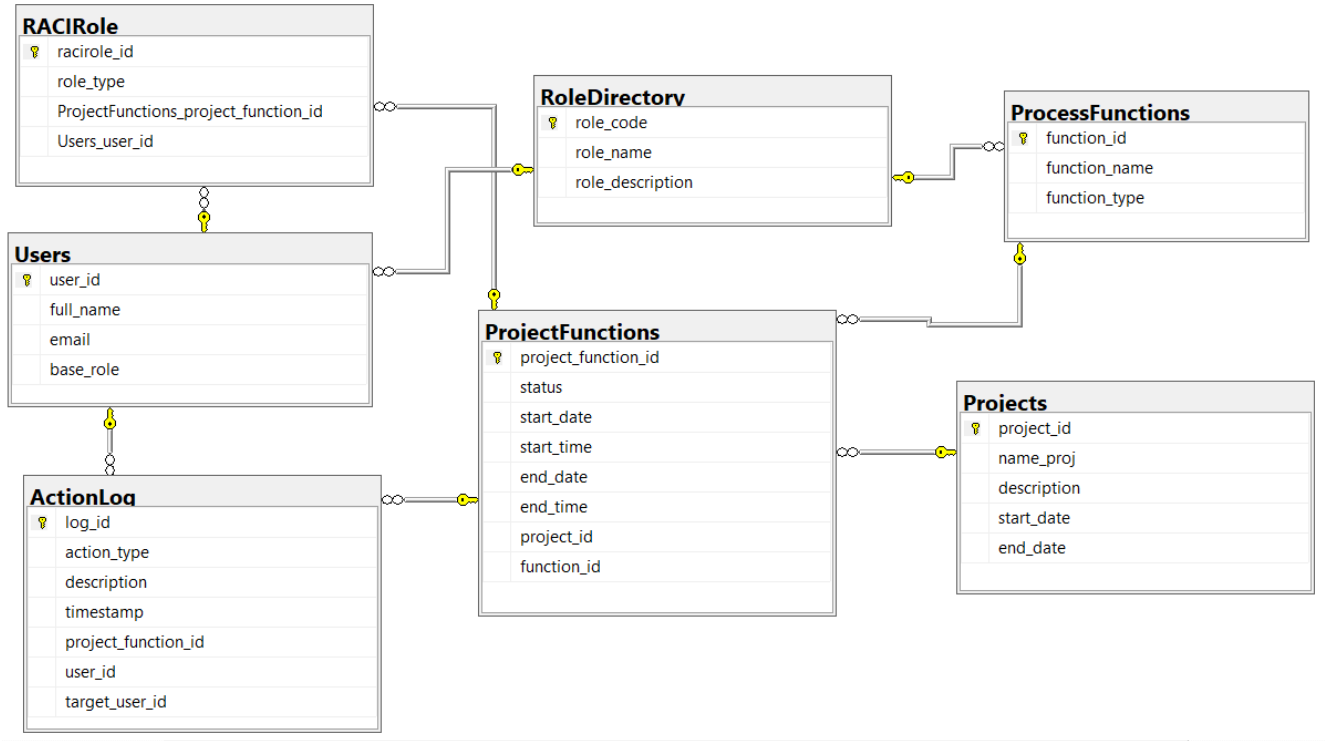


Рисунок 3.3 – Модель бази даних в середовищі Microsoft SQL Server

Джерело: розроблено автором самостійно

Структура таблиць бази даних показана в таблицях 3.1 – 3.7.

Таблиця "Журнал дій" зберігає інформацію про всі дії користувачів у системі, включаючи тип дії, опис, мітку часу, функцію проєкту, яка виконувалась, а також користувача, що здійснив дію, і за потреби – користувача, до якого ця дія була застосована. Опис таблиці показаний в табл. 3.1.

Таблиця 3.1 –Структура таблиці "Журнал дій"

Ідентифікатор таблиці – ActionLog

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов'язкове поле	Зв'язки
1	Ідентифікатор запису	log_id	int	PK	Так	-
2	Тип дії	action_type	int	-	Так	-
3	Опис дії	description	char(200)	-	Так	-
4	Мітка часу	timestamp	datetime	-	Так	-
5	Функція проєкту	project_function_id	int	FK	Так	ProjectFunctions
6	Користувач	user_id	int	FK	Так	Users
7	Цільовий користувач	target_user_id	int	-	Ні	-

Джерело: розроблено автором самостійно

Таблиця "Довідник функцій і процесів" містить перелік усіх функцій, які можуть бути виконані в межах проєктів. Кожна функція має унікальний ідентифікатор, назву та тип. Цей довідник є базою для призначення функцій у конкретних проєктах (табл. 3.2).

Таблиця 3.2 - Опис таблиці "Довідник функцій і процесів"

Ідентифікатор – ProcessFunctions

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов'язкове поле	Зв'язки
1	Ідентифікатор функції	function_id	int	PK	Так	ProjectFunctions
2	Назва функції	function_name	char(100)	-	Так	-
3	Тип функції	function_type	char(30)	FK	Так	RoleDirectory

Джерело: розроблено автором самостійно

Таблиця "Функції у межах проєкту" описує функціональні дії, які виконуються в рамках конкретного проєкту. Вона містить інформацію про статус

виконання функції, часові межі реалізації, а також зв’язок із довідником функцій і відповідним проєктом. Опис таблиці в табл. 3.3.

Таблиця 3.3 - Опис таблиці "Функції у межах проєкту"

Ідентифікатор – ProjectFunctions

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов’язкове поле	Зв’язки
1	Ідентифікатор функції проєкту	project_function_id	int	PK	Так	ActionLog, RACIRole
2	Статус виконання	status	int	-	Так	-
3	Дата початку	start_date	date	-	Так	-
4	Час початку	start_time	time	-	Ні	-
5	Дата завершення	end_date	date	-	Так	-
6	Час завершення	end_time	time	-	Ні	-
7	Ідентифікатор проєкту	project_id	int	FK	Так	Projects
8	Ідентифікатор функції	function_id	int	FK	Так	ProcessFunctions

Джерело: розроблено автором самостійно

Таблиця "Проєкти" (табл. 3.4) містить основну інформацію про всі проєкти, які реалізуються в системі. Вона включає ідентифікатор, назву, опис, а також часові межі реалізації проєкту.

Таблиця 3.4 - Опис таблиці "Проєкти"

Ідентифікатор – Projects

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов’язкове поле	Зв’язки
1	Ідентифікатор проєкту	project_id	int	PK	Так	ProjectFunctions
2	Назва проєкту	name_proj	char(100)	-	Так	-
3	Опис проєкту	description	int	-	Ні	-
4	Дата початку	start_date	datetime	-	Так	-
5	Дата завершення	end_date	datetime	-	Так	-

Джерело: розроблено автором самостійно

Таблиця "Ролі відповідальності у проєктах" відображає призначення ролей типу Responsible, Accountable, Consulted, Informed для користувачів щодо конкретних функцій у проєкті. Вона дозволяє встановлювати, хто саме несе ту чи іншу відповідальність за виконання завдань у рамках функцій проєкту.

Таблиця 3.5 - Опис таблиці "Ролі відповідальності у проєктах"

Ідентифікатор – RACIRole

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов’язкове поле	Зв’язки
1	Ідентифікатор ролі	racirole_id	int	PK	Так	-
2	Тип ролі (R, A, C, I)	role_type	char(1)	-	Так	-
3	Функція проєкту	project_function_id	int	FK	Так	ProjectFunctions
4	Користувач	Users_user_id	int	FK	Так	Users

Джерело: розроблено автором самостійно

Таблиця "Користувачі" містить базову інформацію про всіх користувачів, що беруть участь у проєктах. Вона включає ідентифікатор, повне ім’я, електронну пошту та базову роль користувача в системі.

Таблиця 3.6 - Опис таблиці "Користувачі"

Ідентифікатор – Users

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов’язкове поле	Зв’язки
1	Ідентифікатор користувача	user_id	int	PK	Так	ActionLog, RACIRole
2	Повне ім’я	full_name	char(60)	-	Так	-
3	Електронна пошта	email	char(35)	-	Так	-
4	Базова роль	base_role	char(1)	FK	Так	RoleDirectory

Джерело: розроблено автором самостійно

Таблиця "Довідник ролей" містить перелік типових ролей учасників команди розробки програмного забезпечення. Кожен запис включає код ролі, назву та опис основних функцій, які виконує користувач у відповідній ролі.

Таблиця 3.7 - Опис таблиці "Довідник ролей"

Ідентифікатор – RoleDirectory

№	Назва поля	Ідентифікатор	Тип даних	Ключ	Обов'язкове поле	Зв'язки
1	Код ролі	role_code	char(1)	PK	Так	Users, ProcessFunctions
2	Назва ролі	role_name	varchar(50)	-	Так	-
3	Опис функцій	role_description	varchar(200)	-	Так	-

Джерело: розроблено автором самостійно

3.1.3. Контрольний приклад та обрахунок прогнозних обсягів бази даних

Розроблену базу даних було заповнено даними, результати показані на рис. 3.4-3.7.

На рис. 3.4 показано код заповнення і результат виконання запиту для таблиці «Довідник ролей».

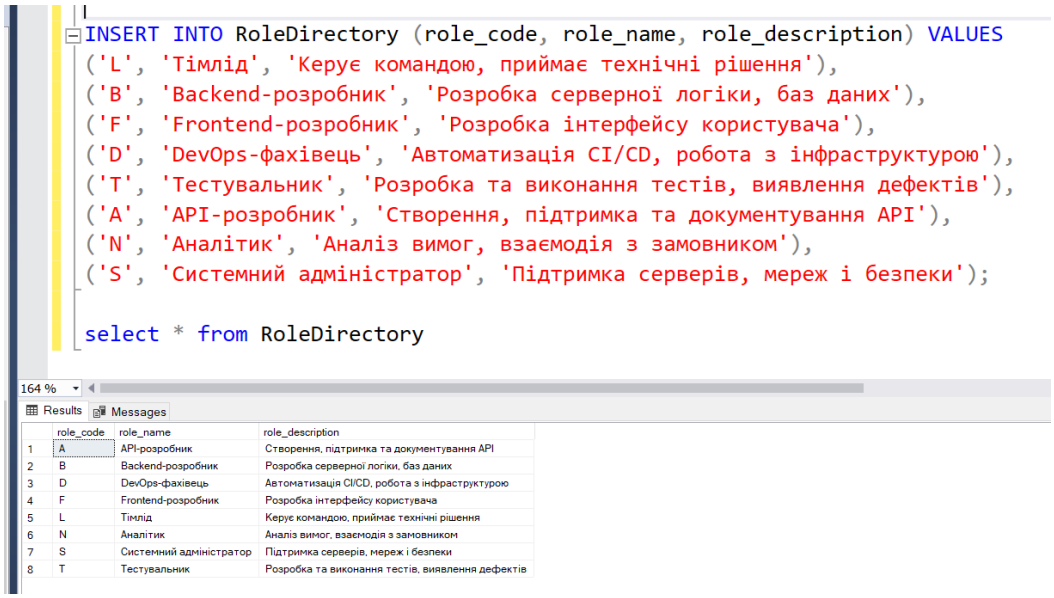


Рисунок 3.4 – Код заповнення і результат для таблиці «Довідник ролей»

Джерело: розроблено автором самостійно

На рис. 3.5 показаний фрагмент вмісту таблиці «Довідник функцій і процесів», на рис. 3.6 – вміст таблиці «Журнал дій», на рис.37 – таблиця «Функції

у межах проєкту», а на рис. 3.8 – вміст таблиць «Проєкти», «Ролі відповідальності у проєктах», «Користувачі».

```
SELECT * FROM ProcessFunctions;
```

164 %

Results | Messages

	function_id	function_name	function_type
1	1	Координація команди	L
2	2	Перевірка статусу виконання	L
3	3	Оцінка ризиків	L
4	4	Ухвалення технічних рішень	L

Рисунок 3.5 – Вміст таблиці «Довідник функцій і процесів»

Джерело: розроблено автором самостійно

```
SELECT * FROM ActionLog
```

164 %

Results | Messages

	log_id	action_type	description	timestamp	project_function_id	user_id	target_user_id
1	9	3	Призначено відповідального користувачем ID 1	2024-02-24 11:15:00.000	191	1	100
2	17	3	Призначено відповідального користувачем ID 23	2024-05-21 16:15:00.000	195	23	40
3	20	1	Створено завдання користувачем ID 74	2024-05-28 10:30:00.000	192	74	53
4	40	1	Створено завдання користувачем ID 9	2024-06-09 17:45:00.000	194	9	14
5	48	5	Функцію позначено як виконану користувачем ID 69	2024-03-14 10:15:00.000	195	69	66
6	80	2	Оновлено опис функції користувачем ID 37	2024-04-25 09:00:00.000	189	37	60

Рисунок 3.6 – Вміст таблиці «Журнал дій»

Джерело: розроблено автором самостійно

```
SELECT * FROM ProjectFunctions;
```

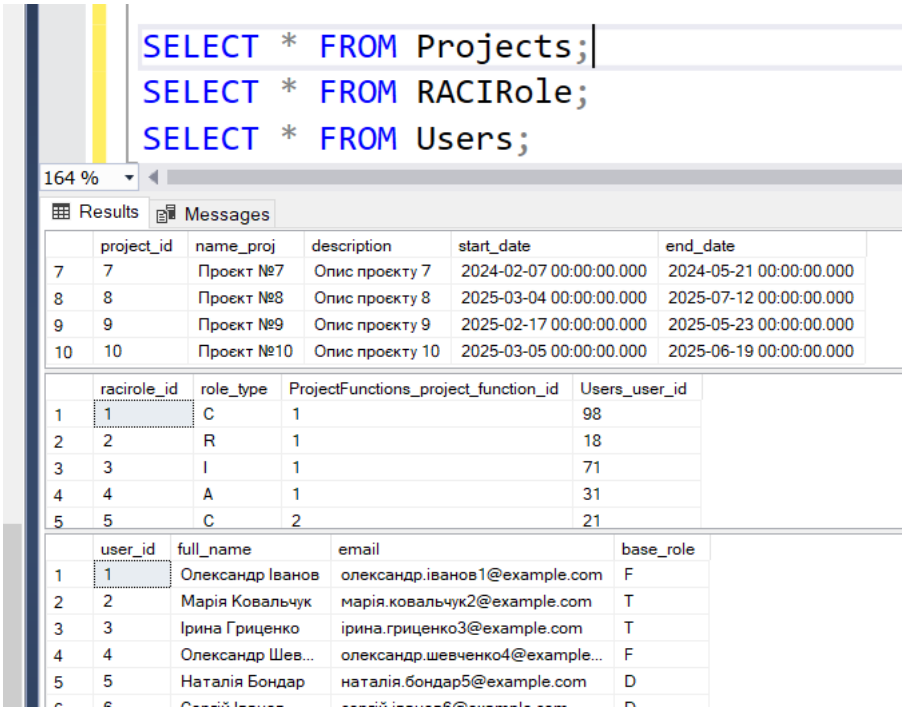
164 %

Results | Messages

	project_function_id	status	start_date	start_time	end_date	end_time	project_id	function_id
170	170	0	2024-02-26	10:15:00.0000000	2024-04-17	14:45:00.0000000	1	3
171	171	2	2025-01-13	09:15:00.0000000	2025-01-24	16:45:00.0000000	8	17
172	172	1	2025-02-03	11:00:00.0000000	2025-02-16	17:45:00.0000000	9	26
173	173	1	2025-01-22	08:45:00.0000000	2025-03-02	15:00:00.0000000	8	17
174	174	2	2024-01-25	09:30:00.0000000	2024-03-22	15:00:00.0000000	3	30
175	175	0	2024-01-28	08:00:00.0000000	2024-02-23	17:45:00.0000000	5	19
176	176	2	2025-05-14	10:45:00.0000000	2025-07-02	18:30:00.0000000	9	22
177	177	1	2025-05-03	10:30:00.0000000	2025-06-06	18:15:00.0000000	9	12

Рисунок 3.7 – Вміст таблиці «Функції у межах проєкту»

Джерело: розроблено автором самостійно



	project_id	name_proj	description	start_date	end_date
7	7	Проєкт №7	Опис проєкту 7	2024-02-07 00:00:00.000	2024-05-21 00:00:00.000
8	8	Проєкт №8	Опис проєкту 8	2025-03-04 00:00:00.000	2025-07-12 00:00:00.000
9	9	Проєкт №9	Опис проєкту 9	2025-02-17 00:00:00.000	2025-05-23 00:00:00.000
10	10	Проєкт №10	Опис проєкту 10	2025-03-05 00:00:00.000	2025-06-19 00:00:00.000

	racirole_id	role_type	ProjectFunctions_project_function_id	Users_user_id
1	1	C	1	98
2	2	R	1	18
3	3	I	1	71
4	4	A	1	31
5	5	C	2	21

	user_id	full_name	email	base_role
1	1	Олександр Іванов	олександр.іванов1@example.com	F
2	2	Марія Ковальчук	марія.ковальчук2@example.com	T
3	3	Ірина Гриценко	ірина.гриценко3@example.com	T
4	4	Олександр Шев...	олександр.шевченко4@example...	F
5	5	Наталія Бондар	наталія.бондар5@example.com	D

Рисунок 3.8 – Вміст таблиць «Проєкти», «Ролі відповідальності у проєктах», «Користувачі»

Джерело: розроблено автором самостійно

Для оцінки розміру майбутньої бази даних інформаційної управляючої системи було здійснено розрахунок прогнозованого обсягу таблиць із урахуванням логічної структури моделі, обраної СУБД Microsoft SQL Server та часовим горизонтом у 6 місяців. Обрахунки виконано вручну з використанням типових розмірів атрибутів відповідно до типів даних SQL Server.

Для кожної сутності визначено середній розмір одного запису в байтах, початкову кількість записів, очікувану кількість нових записів щомісяця, а також орієнтовний обсяг, що буде займати таблиця з урахуванням приросту. Додатково оцінено розмір індексів як 20% від загального обсягу таблиці. Результати показано у таблиці 3.8.

Аналіз прогнозованих обсягів таблиць бази даних інформаційної управляючої системи протягом 6 місяців свідчить про її помірну ресурсну вимогливість. Найбільш значущим джерелом зростання є таблиця ActionLog, яка через високу частоту фіксації подій досягає понад 1 МБ за пів року. Це обґрунтовує

потребу у відповідному моніторингу продуктивності, можливому архівуванні історичних логів або впровадженні механізмів їх агрегації.

Таблиця 3.8 – Прогнозований обсяг БД за 6 місяців

Назва таблиця БД	Розмір запису (байт)	Початковий обсяг (КБ)	Зростання/міс (КБ)	Обсяг через 6 міс (КБ)	Індекси (КБ)	Загальний обсяг (КБ)
Projects	124	6.05	0.61	9.69	1.94	11.62
ProcessFunctions	134	13.09	1.31	20.94	4.19	25.12
ProjectFunctions	44	12.89	2.58	28.36	5.67	34.03
Users	100	9.77	0.98	15.62	3.12	18.75
RACIRole	13	3.81	0.63	7.62	1.52	9.14
ActionLog	228	222.66	111.33	890.62	178.12	1068.75
RoleDirectory	255	1.99	0	1.99	0.40	2.39

Джерело: розроблено автором самостійно

Інші таблиці – такі як Projects, Users, RACIRole – характеризуються повільним і передбачуваним зростанням, що не створює критичного навантаження на систему. Таблиця RoleDirectory, як довідкова, має незначний розмір і не зазнає змін у часі, тому не потребує окремих заходів оптимізації.

Загалом, сумарний прогнозований обсяг усіх таблиць разом з індексами становить 1169.8 КБ, що є незначним для сучасних СУБД і дозволяє зробити висновок про ефективну масштабованість розробленої логічної структури та її придатність до подальшого розширення.

3.2 Засоби інтелектуального аналізу даних

3.2.1 Підготовка даних до інтелектуального аналізу

Для модуля інтелектуальної складової інформаційної системи управління ролями та відповідальністю пропонується реалізація таких задач:

- побудова рекомендаційної моделі для призначення ролей,
- прогнозування ризиків перевантаження учасників команди,

- виявлення порушень політик відповідальності на основі аналізу логів дій,
- аналіз соціальної взаємодії у проєктах засобами графового моделювання.

Для реалізації цих задач необхідно сформувати інформаційну базу, що забезпечить повноту, структурованість і зв'язність даних.

Підготовка даних для моделювання передбачає попередню інтеграцію таблиць із бази даних, зокрема: Users, Projects, ProcessFunctions, ProjectFunctions, RACIRole та ActionLog, які містять відомості про склад учасників, структуру проєктів, призначення функцій і розподіл відповідальності, а також зафіксовані дії користувачів у межах конкретних проєктів.

Основним джерелом подій слугує таблиця `ActionLog`, яка містить інформацію про дії користувачів у контексті виконання функцій у межах проєктів. Щоб доповнити кожен запис логів необхідними атрибутами – даними про користувача, його призначену функцію, тип ролі за RACI, а також інформацією про відповідний проєкт, – було здійснено послідовне об'єднання таблиць через SQL-запити з використанням ключових зв'язків.

Для побудови представлення подій користувачів у контексті призначених функцій, ролей і проєктів, сформовано наступний SQL-запит, фрагмент і результат виконання якого показаний на рис. 3.9. В додатку В показаний повний текст запиту.

У запиті JOIN-об'єднання використовуються для послідовного злиття інформації:

- з таблиці `Users` — щодо особи, яка здійснила дію;
- з таблиці `ProjectFunctions` — щодо функції в межах певного проєкту;
- з таблиці `Projects` — щодо контексту проєкту;
- з таблиці `ProcessFunctions` — для ідентифікації типу функції;

- з таблиці `RACIRole` — для визначення ролі користувача у відповідному функціональному контексті (за моделлю RACI).

Використання оператора LEFT JOIN для таблиці `RACIRole` дає змогу включати до результатів дії, які були виконані навіть без офіційно закріпленої ролі — що важливо для виявлення потенційних порушень.

```
SELECT
    al.log_id, al.timestamp, al.action_type, al.description,
    u.user_id, u.full_name, u.email, u.base_role,
    pf.project_function_id, pf.status, pf.start_date AS pf_start,
    pf.end_date AS pf_end,
    p.project_id, p.name_proj, p.description AS project_description,
    p.start_date AS project_start, p.end_date AS project_end,
    f.function_id, f.function_name, f.function_type,
    rr.role_type AS raci_role
FROM ActionLog al
JOIN Users u ON al.user_id = u.user_id
JOIN ProjectFunctions pf ON al.project_function_id = pf.project_function_id
JOIN Projects p ON pf.project_id = p.project_id
JOIN ProcessFunctions f ON pf.function_id = f.function_id
LEFT JOIN RACIRole rr
```

164 %

Results | Messages

	log_id	timestamp	action...	description	user_id	full_name	email	base_role	project_function_id	status	pf_start	pf_end	pro...	name_proj	project_description
1	1	2024-05-06 10:30:00.000	3	Призначено відповідального користувачем ID 60	60	Марія Ковальчук	марія.ковальчук60@example.com	A	14	0	2024-04-04	2024-05-08	5	Проект №5	Опис проєкту 5
2	2	2024-06-20 09:45:00.000	4	Залишено коментар до завдання користувачем ID 38	38	Ігор Гриценко	ігор.гриценко38@example.com	A	165	2	2024-01-13	2024-02-01	5	Проект №5	Опис проєкту 5
3	3	2024-03-04 13:00:00.000	4	Залишено коментар до завдання користувачем ID 29	29	Олена Козак	олена.козак29@example.com	N	121	1	2024-02-10	2024-02-20	2	Проект №2	Опис проєкту 2
4	4	2024-03-12 17:45:00.000	4	Залишено коментар до завдання користувачем ID 14	14	Сергій Козак	сергій.козак14@example.com	B	157	2	2024-03-12	2024-04-17	1	Проект №1	Опис проєкту 1
5	5	2024-02-01 13:15:00.000	3	Призначено відповідального користувачем ID 63	63	Віктор Гриценко	віктор.гриценко63@example.com	S	108	0	2025-02-02	2025-03-07	10	Проект №10	Опис проєкту 10
6	6	2024-03-12 09:45:00.000	2	Оновлено опис функції користувачем ID 96	96	Ірина Ткаченко	ірина.ткаченко96@example.com	F	14	0	2024-04-04	2024-05-08	5	Проект №5	Опис проєкту 5
7	7	2024-05-08 10:15:00.000	1	Створено завдання користувачем ID 82	82	Марія Козак	марія.козак82@example.com	A	41	1	2024-04-09	2024-05-16	3	Проект №3	Опис проєкту 3
8	8	2024-06-07 16:15:00.000	3	Призначено відповідального користувачем ID 73	73	Олександр Петренко	олександр.петренко73@example.com	A	160	2	2025-05-06	2025-06-16	8	Проект №8	Опис проєкту 8

Рисунок 3.9 – SQL-запит для формування набору даних для інтелектуального аналізу

Джерело: розроблено автором самостійно

Результатом виконання запиту є повноцінний плоский датасет, що містить усі необхідні поля для реалізації подальших етапів інтелектуального аналізу.

Одним із головних напрямів первинного аналізу є визначення інтенсивності участі користувачів у реалізації функцій, що здійснюється через розрахунок частоти виконання дій за типами (`action_type`) у розрізі проєктів, функцій або часових інтервалів. Такий підхід дозволяє отримати кількісну характеристику активності користувачів, зіставити її з формально призначеною роллю та виявити потенційні відхилення між нормативною та фактичною поведінкою.

Для виконання цієї задачі формується відповідний SQL-запит, який агрегує дані за параметрами `user_id`, `project_function_id` та `action_type`, підраховуючи кількість виконаних дій кожного типу (рис. 3.10, додаток В)

```sql
SELECT
    al.user_id,
    pf.project_function_id,
    f.function_name,
    al.action_type,
    COUNT(*) AS action_count
FROM ActionLog al
JOIN ProjectFunctions pf ON al.project_function_id = pf.project_function_id
JOIN ProcessFunctions f ON pf.function_id = f.function_id
GROUP BY al.user_id, pf.project_function_id, f.function_name, al.action_type
ORDER BY al.user_id, pf.project_function_id, action_type;
```

164 %

Results | Messages

	user_id	project_function_id	function_name	action_type	action_count
620	63	48	Розробка користувацького інтерфейсу	3	1
621	63	62	Перевірка статусу виконання	2	1
622	63	69	Забезпечення безпеки API	3	2
623	63	73	Аналіз вимог	4	1
624	63	92	Аналіз вимог	3	1

Рисунок 3.10 – SQL-запит для формування набуру даних для інтелектуального аналізу

Джерело: розроблено автором самостійно

Отримані дані дають змогу виявити осіб із підвищеним рівнем активності у межах функцій, де їхня формальна роль не передбачає виконавчих повноважень, або навпаки — зафіксувати відсутність дій з боку тих, хто має бути відповідальним згідно з RACI. Такий аналіз слугує основою для формування висновків щодо ефективності розподілу відповідальності, потенційного перевантаження окремих учасників або недоопрацювання в структурі ролей. У подальшому ці результати можуть бути використані як вхідні дані для побудови моделей класифікації, оцінки ризиків або формування рекомендацій.

На рис. 3.11 – результат завантаженого набору даних в середовищі Python.

```
df.head(15)
```

	log_id	timestamp	action_type	action_type_name	description	user_id	full_name	target_user_id	project_id	project_name	function_id	function_name	function_type	raci_role
0	1	2024-04-30 00:00:00	1	Створення завдання	Створення завдання користувачем Олена Кравець	8	Олена Кравець	8	1	Проєкт №1	20	Автоматизація тестів	T	A
1	2	2024-03-20 00:00:00	4	Коментування	Коментування користувачем Марина Шевченко	4	Марина Шевченко	3	1	Проєкт №1	10	Адаптивна верстка	F	R
2	3	2024-02-05 00:00:00	1	Створення завдання	Створення завдання користувачем Марина Шевченко	4	Марина Шевченко	8	1	Проєкт №1	20	Автоматизація тестів	T	C
3	4	2024-01-24 00:00:00	3	Призначення відповідального	Призначення відповідального користувачем Натал...	6	Наталія Білик	1	1	Проєкт №1	10	Адаптивна верстка	F	A
4	5	2024-05-06 00:00:00	1	Створення завдання	Створення завдання користувачем Андрій Коваль	1	Андрій Коваль	3	1	Проєкт №1	10	Адаптивна верстка	F	C

Рисунок 3.11 – завантажений у програму набір даних

Джерело: розроблено автором самостійно

3.2.2 Проєктування та реалізація інтелектуального аналізу даних

Після формування єдиного узгодженого датасету, що інтегрує інформацію про користувачів, функціональні ролі, проєкти та зафіксовані дії, було здійснено перехід до етапу моделювання. На цьому етапі застосовуються інструменти інтелектуального аналізу даних, які дозволяють виявляти приховані закономірності, оцінювати відповідність між фактичною та формальною поведінкою учасників проєктів, а також прогнозувати можливі ризики, пов’язані з перевантаженням, конфліктами відповідальності або неефективним розподілом ролей.

Залежно від поставлених цілей, моделювання передбачає використання різних аналітичних підходів, зокрема: побудови графа взаємодії (SNA), виявлення відхилень між запланованим і фактичним перебігом процесів (Process Mining), формування рекомендаційної системи для автоматизованого призначення ролей або виконавців. Основою для реалізації кожного з цих підходів є одна й та сама сукупність атрибутів, але застосованих у відповідних контекстах і перетворених згідно з логікою задачі.

Прогнозування ризиків перевантаження учасників команди.

Одним із завдань інтелектуального аналізу в системі управління ролями та відповідальністю є виявлення ознак перевантаження окремих учасників команди. Надмірна концентрація функціонального навантаження на одному користувачі може свідчити як про порушення принципів рівномірного розподілу

відповідальності, так і про ризики зниження ефективності команди або появи критичних точок відмови.

Для візуального представлення інтенсивності виконання дій кожним учасником було побудовано теплову карту, у якій по горизонтальній осі відображено типи дій, по вертикальній — користувачів, а кольорова шкала відповідає кількості відповідних дій, зафіксованих у журналі подій. Такий підхід дає змогу швидко ідентифікувати найбільш активних користувачів, визначити типи дій, на які припадає основне навантаження, а також виявити нерівномірність розподілу функціональної активності.

На рис. 3.12 показано код для створення теплової карти, а візуалізація результатів наведена на рисунку 3.13.

```
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=.5, annot=True, fmt='d', cbar_kws={'label': 'Кількість дій'})
plt.title("Теплова карта активності користувачів за типами дій")
plt.xlabel("Тип дії")
plt.ylabel("Користувач")
plt.tight_layout()
plt.show()
```

Рисунок 3.12 – Код побудови теплової карти

Джерело: розроблено автором самостійно

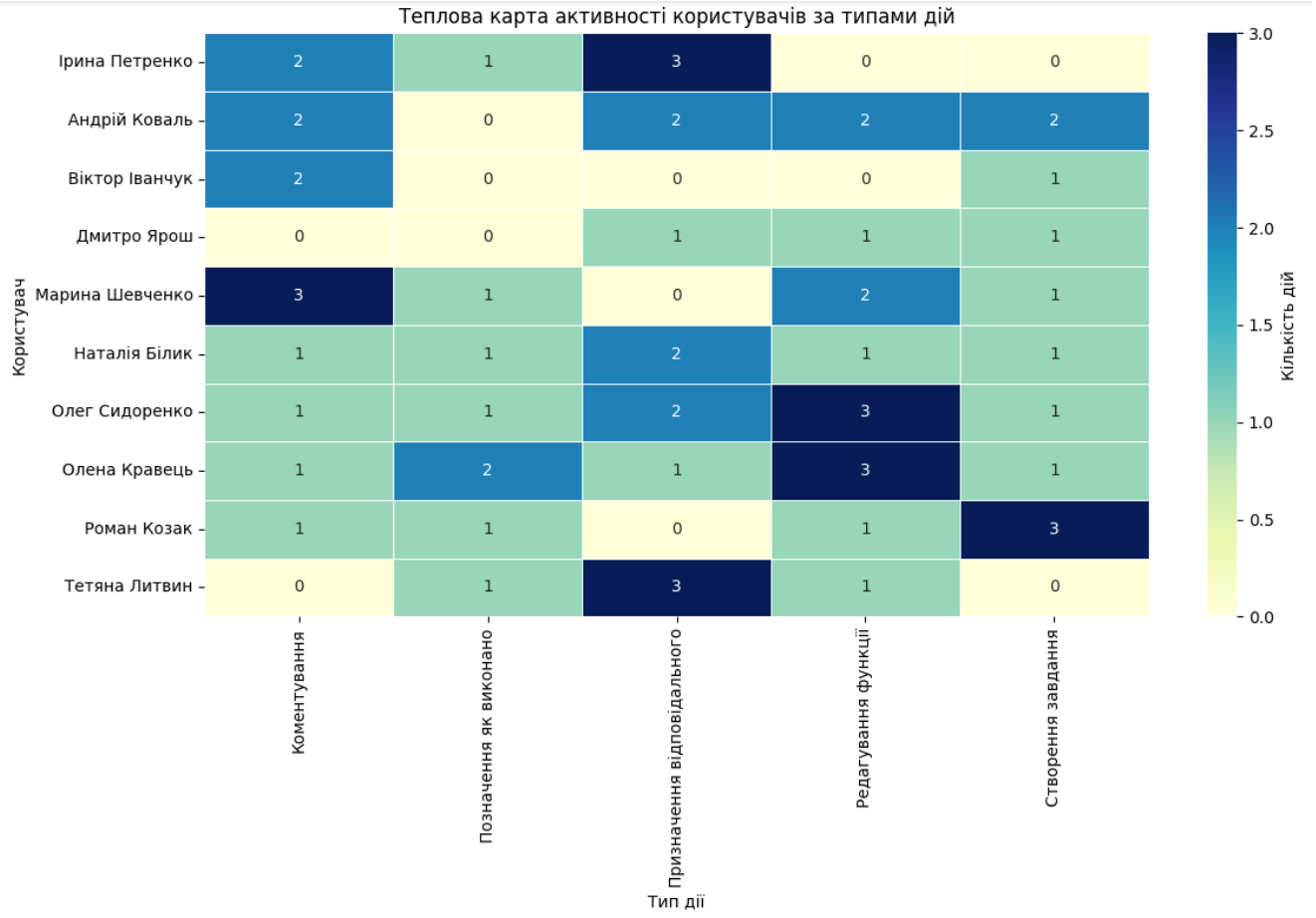


Рисунок 3.13 – Теплова карта активності користувачів

Джерело: розроблено автором самостійно

Яскраві ділянки карти свідчать про високу інтенсивність взаємодії, що, у поєднанні з формально призначеною роллю користувача, дозволяє виявляти розбіжності між декларованими обов'язками та реальною участю в проєкті. Наприклад, користувач з роллю I (Informed) не повинен активно втручатися у виконання завдань, а його інтенсивна активність у категоріях типу create_task або complete_task може вказувати на потенційне порушення політик відповідальності.

Таким чином, теплова карта є ефективним інструментом для первинного моніторингу перевантаження та аномалій у структурі командної взаємодії, а її результати можуть бути використані для ініціювання перегляду розподілу ролей або автоматичного формування рекомендацій системою.

Аналіз соціальної взаємодії.

Аналіз соціальної взаємодії між учасниками проєктів реалізується шляхом побудови орієнтованого або неорієнтованого графа, у якому вузли відповідають окремим користувачам, а ребра позначають факт спільної участі у виконанні однієї функції в межах певного проєкту. Для формування такого графа було використано агрегований описаний вище датасет, що містить інформацію про ідентифікатори користувачів, функціональні блоки, проєкти, а також тип виконаних дій.

Оскільки структура бази даних не містить уніфікованого ідентифікатора функціонального блоку (project_function_id), для побудови графа було сформовано допоміжне поле `function_block`, яке являє собою конкатенацію `project_id` та `function_id`. Це дозволяє однозначно ідентифікувати контекст спільної роботи над окремим функціональним завданням у межах конкретного проєкту.

Візуалізація графа реалізована на основі бібліотек `NetworkX` і `Matplotlib`. Код побудови графа наведено на рисунку 3.14.

У рамках побудови графа реалізовано такий підхід:

- для кожної унікальної функції визначається множина користувачів, які брали в ній участь.
- для кожної пари користувачів, що працювали над цією функцією, у графі додається ребро.
- якщо така пара вже має зв’язок, до ваги ребра додається одиниця, що відображає інтенсивність взаємодії. Таким чином, вага ребра відображає кількість випадків спільної участі користувачів у функціях.

```
G = nx.Graph()

for block in df['function_block'].unique():
    participants = df[df['function_block'] == block]['user_id'].unique()
    for i in range(len(participants)):
        for j in range(i + 1, len(participants)):
            u1, u2 = participants[i], participants[j]
            if G.has_edge(u1, u2):
                G[u1][u2]['weight'] += 1
            else:
                G.add_edge(u1, u2, weight=1)
```

Рисунок 3.14 – Код програми для побудови графа взаємодії.

Джерело: розроблено автором самостійно

Побудований граф дає змогу виконати подальший аналіз соціальної структури команди, розрахувати метрики центральності, виявити неформальні центри впливу та оцінити ефективність розподілу відповідальності відповідно до моделі RACI. Графічна інтерпретація результатів подана на рисунку 3.15.

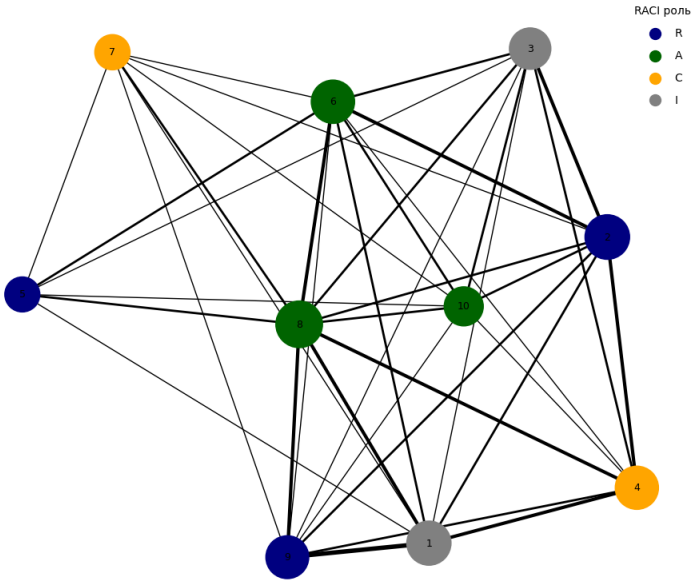


Рисунок 3.15 – Граф взаємодії між учасниками в межах трьох проєктів

Джерело: розроблено автором самостійно

У побудованому графі кольори вузлів відповідають типу ролі користувача згідно з моделлю RACI: темно-синій для відповідальних (R), темно-зелений для підзвітних (A), помаранчевий для консультованих (C) та сірий для поінформованих

(I). Розміри вузлів масштабуються відповідно до метрики Eigenvector Centrality, що дає змогу виявити користувачів, які мають найбільший вплив у структурі командної взаємодії.

На основі побудованого графа соціальної взаємодії користувачів відкриваються можливості для глибшого аналізу структури командної роботи в межах проєкту. Зокрема, можуть бути реалізовані наступні напрямки:

4. Розрахунок метрик центральності: ідентифікація ключових користувачів за впливовістю.

5. Порівняння формальної ролі з фактичною активністю: зіставлення ролей RACI з позицією у графі.

6. Виявлення структурних аномалій: пошук ізольованих або надмірно навантажених учасників.

7. Аналіз спільнот (кластерів): виділення підгруп, що формуються природним шляхом через співпрацю.

8. Побудова динаміки змін взаємодій у часі: спостереження за еволюцією структури зв’язків.

9. Мережевий аналіз функцій замість користувачів: побудова графа перетину функціонального навантаження.

10. Виявлення ролей-“вузлів ризику”: виявлення критичних точок, через які проходить більшість шляхів.

Наприклад, розглянемо аналіз кластера взаємодії за алгоритмом modularity. Для виявлення природних підгруп (кластерів) у командній взаємодії використано алгоритм Louvain, що дозволяє виділити спільноти у неорієнтованому зваженому графі (рис. 3.16). Такий підхід дає змогу:

- визначити, як фактично формуються робочі групи незалежно від формальних призначень;
- порівняти розподіл ролей у межах кожного кластеру;
- виявити надмірну концентрацію зв’язків у певних підсистемах.

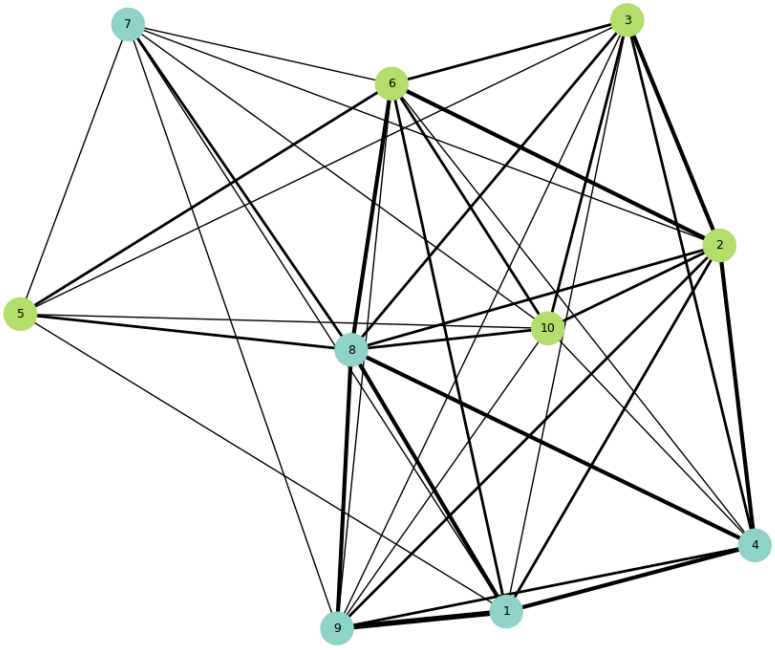


Рисунок 3.16 – Граф взаємодії між учасниками в межах трьох проєктів

Джерело: розроблено автором самостійно

Кожен колір позначає окрему спільноту користувачів, які інтенсивно взаємодіють у межах одного або кількох функціональних блоків. Виявлення таких груп дозволяє менеджеру проєкту краще зрозуміти реальну структуру комунікацій і, за потреби, скоригувати призначення відповідальності, уникнувши дублювання або конфлікту функцій.

- один кластер позначено світло-зеленим кольором — наприклад, вузли 2, 3, 5, 6, 10;
- другий кластер — блідо-блакитний — містить вузли 1, 4, 7, 8, 9.

Учасники в межах одного кластеру мають інтенсивну взаємодію між собою – вони часто працювали над спільними функціями в рамках проєкту. Кількість та товщина ребер всередині кластерів значно вища, ніж міжкласові зв’язки. Це не обов’язково збігається з формальними ролями (RACI), що дозволяє:

- перевірити ефективність формальної структури команди;
- скоригувати розподіл функцій або відповідальності;

- визначити неформальні робочі підгрупи.

3.3. Програмне забезпечення

3.3.1 Проєктування програмного забезпечення

Проєктування програмного забезпечення інформаційної системи управління ролями та відповідальністю здійснювалося з урахуванням вимог до інтеграції аналітичних модулів з інтерфейсом користувача, системою зберігання даних та механізмами збору логів. Основна увага була зосереджена на модульному підході, який забезпечує незалежність компонентів системи, їхню гнучкість, повторне використання та масштабованість.

До складу системи входить кілька функціональних модулів:

- модуль збору логів користувачів (ActionLog),
- процесор попередньої обробки даних,
- інтелектуальний модуль виявлення відхилень,
- модуль аналізу взаємодій (SNA), п
- ідсистема збереження та запитів до бази даних (на базі SQL Server),
- інтерфейс користувача.

Функціональність інтелектуального модуля реалізована за допомогою мови Python із використанням бібліотек PM4Py, NetworkX, Pandas та Scikit-learn у середовищі Jupyter Notebook або Google Colab, що дозволяє ефективно експериментувати з алгоритмами Process Mining, побудовою графів та обробкою журналів подій.

Логіка взаємодії між модулями реалізована у вигляді скриптів, які можуть бути викликані вручну або автоматизовано через вебінтерфейс. Виведення результатів аналізу здійснюється у вигляді графіків, таблиць та інтерактивних

візуалізацій, що надаються адміністраторам для подальшої оцінки відповідності дій користувачів RACI-моделі.

3.3.2. Вибір архітектури програмного забезпечення

Архітектура програмного забезпечення інформаційної системи управління ролями та відповідальністю побудована за принципом модульної децентралізації із чітким розмежуванням функціональних зон відповідальності. Такий підхід забезпечує гнучкість, масштабованість і сумісність між окремими компонентами системи, а також дозволяє ефективно розподіляти обчислювальне навантаження між клієнтською та серверною частинами.

На рівні логічної організації система складається з трьох основних груп компонентів: модулів збору та зберігання даних, аналітичного ядра, а також модуля взаємодії з користувачем. Клієнтський рівень реалізовано у вигляді вебінтерфейсу адміністратора, через який користувачі отримують доступ до звітів, аналітичної інформації та сповіщень. Він побудований як фронтенд-застосунок, що взаємодіє з серверною частиною через API.

Серверна частина системи включає модуль автентифікації та управління доступом, модуль збору логів дій користувачів, модуль генерації звітів та інтелектуальний модуль виявлення відхилень, який виконує аналіз поведінки відповідно до RACI-моделі. Усі ці компоненти звертаються до централізованої бази даних, реалізованої на платформі Microsoft SQL Server, де зберігаються журнали подій, ролі, матриці відповідальності та результати аналізу.

Аналітичне ядро системи розроблено з використанням мови Python, у середовищі Jupyter Notebook або Google Colab, проте у фінальній архітектурі інтегрується як серверний сервіс, що працює незалежно від розробницького середовища. Цей сервіс періодично або за запитом адміністратора отримує дані з бази, обробляє їх за допомогою методів Process Mining (PM4Py) та графового аналізу (NetworkX), і формує аналітичні висновки щодо відповідності дій користувачів встановленим ролям.

Таким чином, архітектура реалізована за моделлю «тонкий клієнт – потужний сервер», що дозволяє користувачам працювати через звичайний браузер без необхідності інсталяції спеціального програмного забезпечення. Усі аналітичні й обчислювальні процеси виконуються на серверному рівні, що гарантує безперервність і керованість системи.

Загальна структура програмного забезпечення подана на рисунку 3.17.

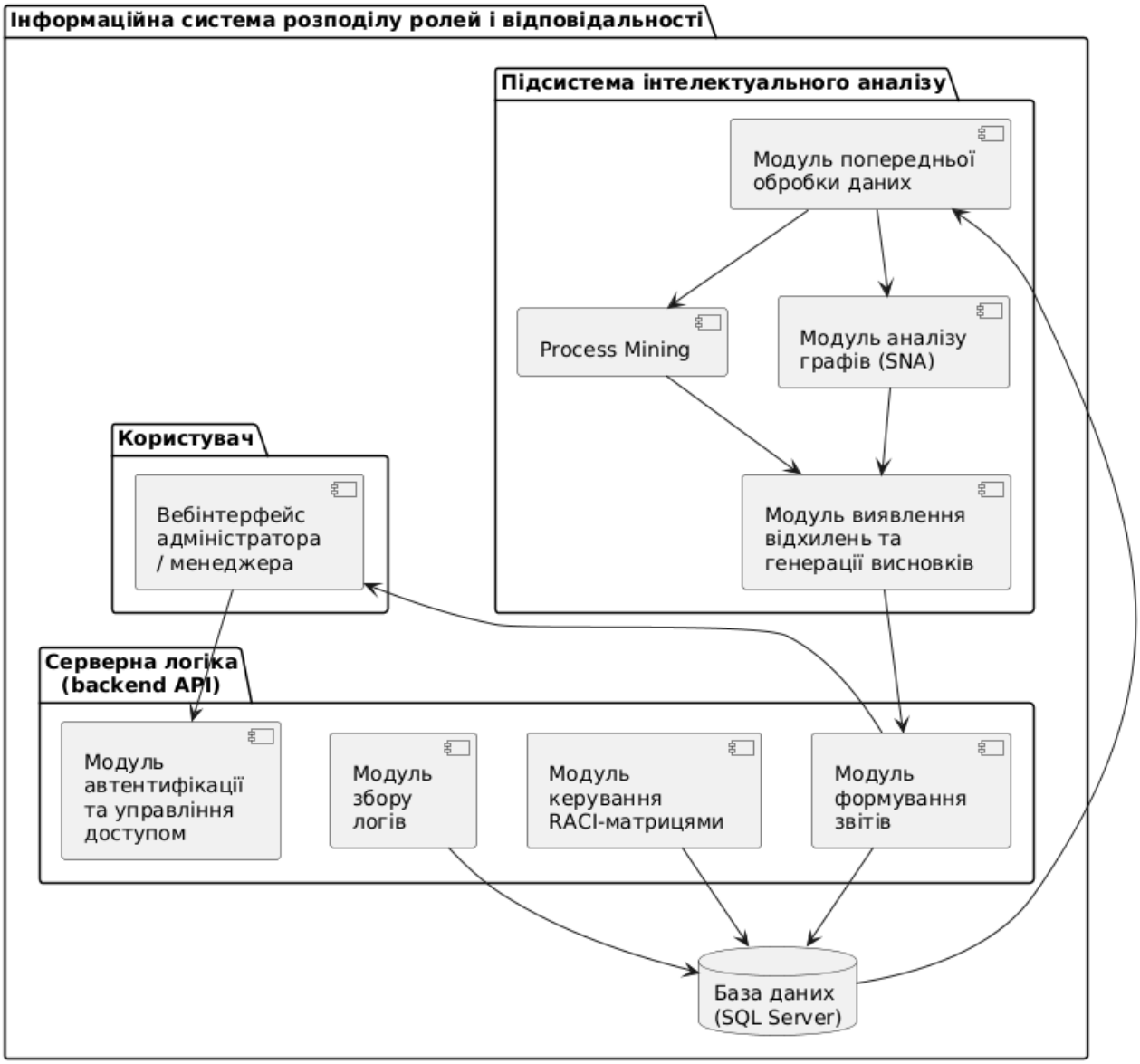


Рисунок 3.17 – Архітектура програмного забезпечення ІУС управління ролями та відповідальністю

3.3.3 Інструментальні засоби розробки для створення прикладного програмного забезпечення інформаційно управляючої системи

Розробка прикладного програмного забезпечення інформаційної системи управління ролями та відповідальністю здійснювалася із застосуванням сучасних інструментальних засобів, які забезпечують підтримку повного життєвого циклу створення ІУС: від проєктування логіки до реалізації модулів, налагодження та тестування.

Для реалізації інтерфейсу користувача використано вебтехнології, зокрема HTML5, CSS3 та JavaScript з фреймворком React. Цей підхід забезпечив створення адаптивного і зручного вебінтерфейсу, який не потребує встановлення додаткового програмного забезпечення на стороні користувача.

Серверна логіка реалізована за допомогою мови програмування Python, з використанням фреймворку FastAPI, що дозволило реалізувати RESTful API для взаємодії між фронтендом, аналітичними модулями та базою даних. FastAPI було обрано через його продуктивність, підтримку асинхронних запитів та вбудовану систему документації інтерфейсів (OpenAPI).

Для зберігання даних та обробки запитів застосовано Microsoft SQL Server, який забезпечує підтримку транзакцій, цілісності даних та високий рівень сумісності з іншими інструментами системи.

Реалізація інтелектуального модуля здійснювалася в середовищі Jupyter Notebook або Google Colab на мові Python. При цьому використовувалися такі бібліотеки:

- PM4Py – для реалізації методів Process Mining, зокрема побудови моделей процесів на основі логів дій користувачів;
- NetworkX – для побудови соціальних графів взаємодії користувачів;
- Pandas, NumPy, Matplotlib, Seaborn – для обробки табличних даних, візуалізації та агрегування інформації.

Проміжні обчислювальні блоки аналітичного модуля можуть бути інтегровані у серверну частину системи через REST API або запускатися як

окремий асинхронний сервіс, що звертається до централізованого сховища за даними.

Для керування залежностями, структурування проєкту та автоматизації виконання окремих задач використовувалися засоби середовища Visual Studio Code у поєднанні з інструментами командного рядка Python (venv, pip, pytest, flake8) та системою контролю версій Git.

Таким чином, обрані інструментальні засоби дозволили реалізувати модульну, масштабовану та легко адаптовану систему, придатну як для локального розгортання, так і для перенесення у хмарне середовище.

3.3.4 Тестування програмного забезпечення

Тестування програмного забезпечення інформаційної системи управління ролями та відповідальністю проводилося з метою перевірки правильності реалізації функціоналу, відповідності очікуваним сценаріям використання, виявлення помилок і забезпечення стабільності системи в умовах реального навантаження.

Процес тестування охоплював кілька рівнів:

1. Модульне тестування здійснювалося для перевірки окремих функцій та компонентів серверної логіки та інтелектуального модуля. Для цього застосовувалися засоби pytest, які дозволили верифікувати правильність обробки логів, обчислення ролей, запуск аналітичних обчислень (Process Mining, SNA) та формування висновків. Також було протестовано обробку нештатних ситуацій: відсутність подій у логах, порожні матриці ролей, конфлікти у відповідальності.

2. Інтеграційне тестування проводилося для перевірки взаємодії між модулями: зокрема, передача даних з бази до модуля попередньої обробки, подальший запуск алгоритмів виявлення відхилень, збереження результатів і передача їх у звітну систему. У цьому етапі перевірялася коректність форматів даних, стабільність API-зв’язків і обробка типових запитів від інтерфейсу користувача.

3. Системне тестування охоплювало повну перевірку функціонування всієї системи в тестовому середовищі. Було змодельовано типові сценарії: вхід користувача, виконання дій у системі, генерація логів, формування звітів для адміністратора. Особливу увагу приділено перевірці наявності реакцій на порушення відповідальності та своєчасності формування відповідних повідомлень.

4. Тестування інтерфейсу охоплювало юзабіліті-тестування вебінтерфейсу адміністратора. Перевірялася зручність навігації, доступність основних функцій, логічність відображення аналітичної інформації, коректність відображення графіків і таблиць результатів аналізу.

5. Навантажувальне тестування проводилося із застосуванням симульованих сесій роботи кількох користувачів, що паралельно здійснюють дії у системі. Було оцінено час виконання запитів, швидкість обробки журналів та час відповіді аналітичного модуля. Система зберігала працездатність за умов перевищення 10 одночасних сесій з обробкою понад 100 тис. подій.

Усі виявлені помилки були задокументовані та усунуті. Результати тестування свідчать про відповідність реалізованого функціоналу заявленим вимогам, стабільність роботи системи в рамках очікуваного навантаження та готовність до подальшого впровадження.

3.4. Технічне забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Для забезпечення надійного функціонування інформаційної системи управління ролями та відповідальністю в процесі розробки програмного забезпечення необхідно сформувати відповідне технічне середовище. Воно включає серверну частину або центральний обчислювальний вузол, мережеву інфраструктуру, засоби резервного зберігання, а також робочі місця користувачів. Обрані технічні рішення повинні відповідати вимогам до продуктивності, надійності, сумісності з програмним забезпеченням і можливості масштабування.

Інформаційна система управління ролями та відповідальністю реалізує низку функціональних компонентів, які потребують обчислювальних ресурсів, зберігання та захищеної передачі даних. Зокрема, до таких компонентів належать: підсистема збору логів дій користувачів, модулі інтелектуального аналізу (процесного та графового), система зберігання даних на базі SQL Server, а також інтерфейс адміністратора для перегляду звітів і виявлення порушень. У сукупності ці модулі формують навантаження на центральний обчислювальний вузол системи та визначають необхідність побудови внутрішньої мережі, яка забезпечує взаємодію між клієнтськими робочими місцями та сервером.

З урахуванням кількості користувачів (до 10 одночасно активних у пілотному режимі), обсягу згенерованих логів (сотні тисяч записів щомісяця) та специфіки виконання аналітичних операцій (обробка логів, побудова графів взаємодій, аналіз відхилень), система потребує апаратної платформи, здатної забезпечити:

- стійке зберігання даних з підтримкою резервного копіювання;
- паралельну обробку транзакцій і аналітичних запитів;
- локальну мережу для обміну інформацією;
- швидкий доступ до системи з клієнтських робочих місць.

Для розгортання інформаційної системи управління ролями та відповідальністю обрано рішення, що відповідає запланованому рівню навантаження та забезпечує базову масштабованість. Замість використання серверів промислового класу, які потребують значних фінансових витрат, доцільним є використання стандартної робочої станції або сервера початкового рівня, яка має достатній обсяг оперативної пам’яті, підтримує сучасні операційні системи та сумісна з обраною системою управління базами даних.

Це рішення дозволяє оптимізувати витрати без втрати продуктивності, зберігаючи здатність обробляти журнали подій, виконувати аналітичні процеси (таких як побудова процесних моделей або графів взаємодії) та обслуговувати паралельні запити від декількох користувачів. Обране обчислювальне ядро

підтримує реалізацію інформаційної системи в режимі локального або гібридного (із частковою хмарною підтримкою) розгортання.

У разі зростання навантаження або переходу до комерційної експлуатації система може бути перенесена на потужніший сервер або віртуалізоване середовище, зберігаючи архітектуру та логіку взаємодії компонентів.

Окрім центрального обчислювального вузла, для забезпечення стабільної роботи інформаційної системи необхідна наявність додаткового технічного забезпечення, яке забезпечує взаємодію між користувачами, обмін даними, резервне копіювання та енергетичну стабільність. Це обладнання несе допоміжну, але досить важливу функцію для підтримки цілісності та безперервності роботи системи.

Насамперед, передбачається створення внутрішньої комп’ютерної мережі, яка об’єднує сервер із клієнтськими робочими станціями, адміністративним інтерфейсом та іншими користувачами системи. Для цього використовується базове мережеве обладнання, таке як маршрутизатор або комутатор (switch), що забезпечує гігабітну передачу даних без затримок. Наявність швидкісного дротового з’єднання дозволяє уникнути втрат при передачі логів у реальному часі.

Для захисту даних від втрати внаслідок перебоїв з живленням необхідне використання джерела безперебійного живлення (ІБП), яке дозволить завершити активні сесії, зберегти дані в базі та забезпечити контрольоване завершення роботи сервера. Такий компонент особливо важливий у системах, які ведуть постійний запис логів і працюють у режимі близькому до реального часу.

Ще одним елементом є засоби резервного копіювання. Вони можуть бути реалізовані через зовнішні накопичувачі (зовнішній HDD/SSD або NAS-пристрій), які зберігають копії баз даних, логів подій та звітної інформації. Це створює додатковий рівень безпеки на випадок пошкодження основного сховища.

Також у складі системи враховуються клієнтські робочі місця, з яких здійснюється доступ до інтерфейсу ІУС, аналіз результатів, перегляд відхилень та формування звітів. Робочі місця мають бути сумісні з вебінтерфейсом системи або з відповідним клієнтським ПЗ, мати стабільне мережеве з’єднання та належну

продуктивність для обробки візуалізованих результатів аналізу (наприклад, графів або логів).

У разі, якщо система передбачає створення друкованих звітів, додатково використовується периферійне обладнання – принтер або багатофункціональний пристрій. Це дозволяє реалізувати супровідну документацію у фізичному вигляді, що особливо актуально у середовищах із вимогами до архівного зберігання звітів про відповідальність та рольові дії.

Таким чином, уся система складається не лише з програмного та обчислювального ядра, а й із комплексу допоміжних технічних засобів, які забезпечують її цілісне, безпечне й ефективне функціонування.

Обране технічне забезпечення забезпечує повну сумісність із архітектурою інформаційної системи та використовуваним програмним середовищем. Операційна система сервера (Windows Server 2022 або Windows 11 Pro) та СУБД (Microsoft SQL Server 2019 Express або Standard) безперешкодно взаємодіють із програмними модулями, реалізованими на сучасних мовах програмування з підтримкою бібліотек для обробки логів, аналізу графів та візуалізації даних.

Рішення щодо використання економного, але функціонального обчислювального вузла є виправданим з точки зору вартості володіння: для початкового впровадження системи немає потреби у промислових серверах із надлишковою продуктивністю. Такий підхід дозволяє уникнути необґрунтованих капітальних витрат і зосередити ресурси на реалізації програмної частини, тестуванні та адаптації системи до реального процесу розподілу ролей та відповідальності.

Окрему увагу приділено питанню масштабованості. У разі збільшення навантаження на систему (зростання обсягів логів, кількості користувачів, складності процесів) технічна архітектура дозволяє перейти до продуктивнішого рішення без зміни логіки роботи ІУС. Це може бути реалізовано як через оновлення апаратної конфігурації, так і через перенесення окремих сервісів (зокрема аналітичних) до хмарного середовища. Така гнучкість гарантує, що інвестиції в

початкову інфраструктуру не будуть втрачені, а система зможе розвиватися разом з організаційними потребами.

Запропонований підхід до формування технічного забезпечення ґрунтується на балансі між доступністю ресурсів, вимогами до стабільності функціонування та потенціалом для розвитку. Це дозволяє впровадити інформаційну систему у типовому середовищі без залучення спеціалізованого обладнання промислового класу, зберігаючи водночас високу якість роботи її ключових компонентів.

3.4.2 Ресурсні вимоги до технічного забезпечення

На основі вибору технічного забезпечення встановлено мінімальні та рекомендовані апаратні параметри, необхідні для забезпечення стабільної та ефективної роботи інформаційної системи управління ролями та відповідальністю. У системі реалізовано функції збору подій, зберігання та аналізу логів, виявлення відхилень, побудови графів взаємодії користувачів та формування звітності, що зумовлює потребу в достатніх обчислювальних та мережевих ресурсах.

Вимоги до серверної частини (обчислювального ядра ІУС):

- багатоядерний процесор, не нижче Intel Core i7 10-го покоління або Xeon E-2224G;
- оперативна пам’ять не менше 32 ГБ DDR4 (рекомендовано 64 ГБ при одночасному використанні Process Mining і SNA);
- накопичувачі:
    - твердотілий диск (SSD) обсягом 1 ТБ – для розміщення ОС, бази даних та поточних логів;
    - жорсткий диск (HDD) обсягом 2–4 ТБ – для архівування подій, резервного копіювання та збереження звітів;
- вережева інфраструктура: гігабітна мережева карта (1 GbE) з можливістю підключення до внутрішнього комутатора/маршрутизатора;

- операційна система Windows Server 2022 або Windows 11 Pro (у разі розгортання в малому середовищі);
- СУБД Microsoft SQL Server 2019 Express (з обмеженням до 10 ГБ на БД) або Standard (залежно від обсягів даних);
- засоби захисту: джерело безперебійного живлення потужністю не менше 1000 ВА для захисту від перебоїв електроживлення;
- засоби резервного копіювання: зовнішній SSD/HDD на 2 ТБ або NAS-пристрій.

Вимоги до клієнтських робочих місць:

- процесор не нижче Intel Core i5 або AMD Ryzen 5;
- оперативна пам’ять: 8–16 ГБ (залежно від навантаження на аналіз та візуалізацію даних);
- накопичувач SSD від 256 ГБ;
- графічний інтерфейс: монітор з роздільною здатністю Full HD (1920×1080) або вище;
- операційна система Windows 10/11 Pro;
- програмне забезпечення:
  - браузер з підтримкою JavaScript та WebSocket,
  - Microsoft Excel (за потреби експорту логів або звітів),
  - додаткові візуалізаційні інструменти (наприклад, Power BI Viewer).

Вказані вимоги орієнтовані на роботу системи в умовах помірного навантаження (до 10 одночасно активних користувачів, до 1 млн логів у базі). За потреби масштабування – нарощується обсяг оперативної пам’яті, накопичувачі та/або здійснюється перенесення компонентів системи в хмарне середовище або на більш потужну серверну платформу.

Зазначені апаратні параметри є достатніми для розгортання повнофункціональної версії інформаційної системи в базовому середовищі, з урахуванням майбутньої можливості масштабування. Це дозволяє забезпечити надійність, доступність і продуктивність роботи всіх ключових модулів системи.

3.5 Реалізація інформаційної управляючої системи

Інформаційна система передбачає наявність зручного вебінтерфейсу для адміністратора або менеджера проєкту, який дозволяє керувати проєктами, переглядати результати інтелектуального аналізу та оперативно реагувати на виявлені порушення.

Робота користувача з системою здійснюється через авторизований доступ. Після введення облікових даних (логіну й пароля) (рис. 3.18)

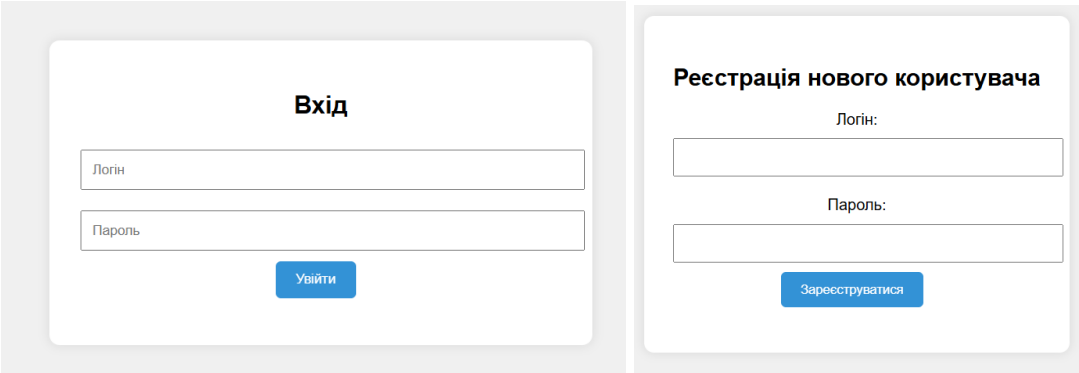


Рисунок 3.18 – Вхід у систему

Джерело: розроблено автором самостійно

Користувач потрапляє до особистого кабінету адміністратора, де доступні такі основні функції:

- створення нового проєкту, де користувач заповнює базову інформацію про проєкт, призначає учасників та ролі відповідно до RACI-моделі (рис. 3.19).
- перегляд і редагування проєктів: доступна таблиця всіх проєктів із можливістю фільтрації, відкриття детальної інформації, внесення змін до структури ролей та відповідальності (рис. 3.20).

Детальна інформація про поєкт на рис. 3.21.

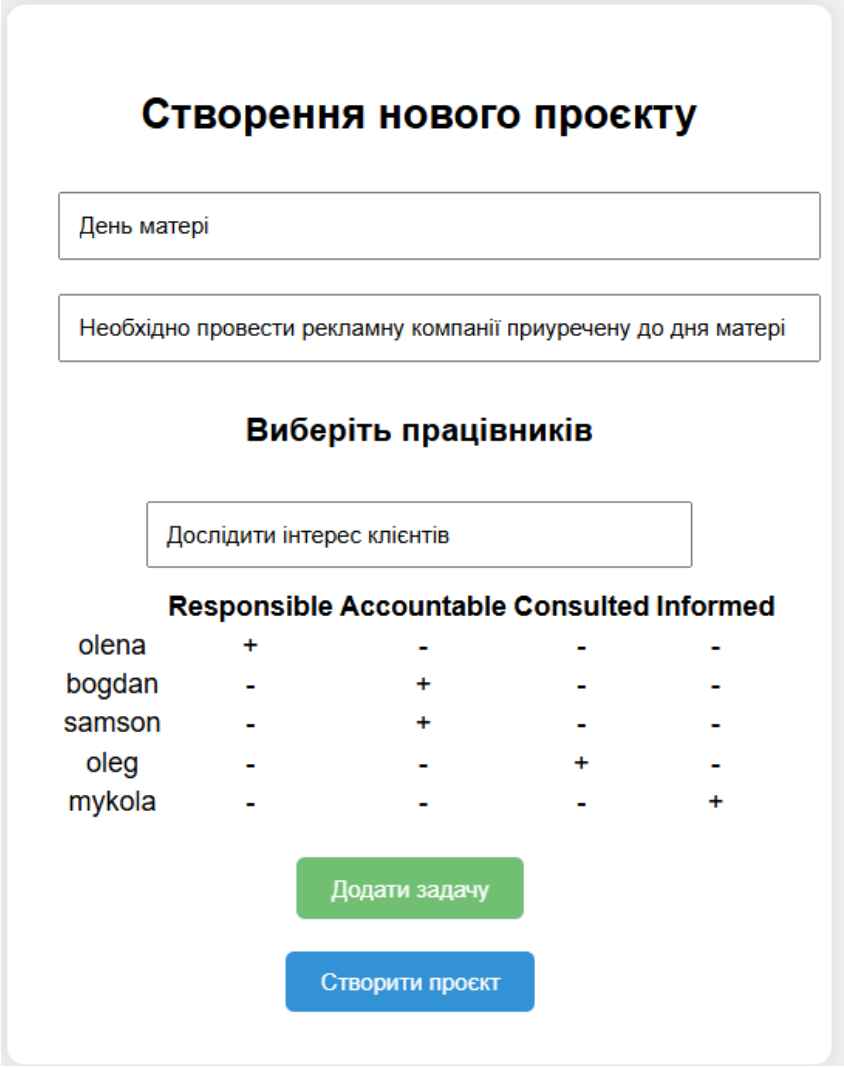


Рисунок 3.19 – Екранна форма створення нового проєкту

Джерело: розроблено автором самостійно

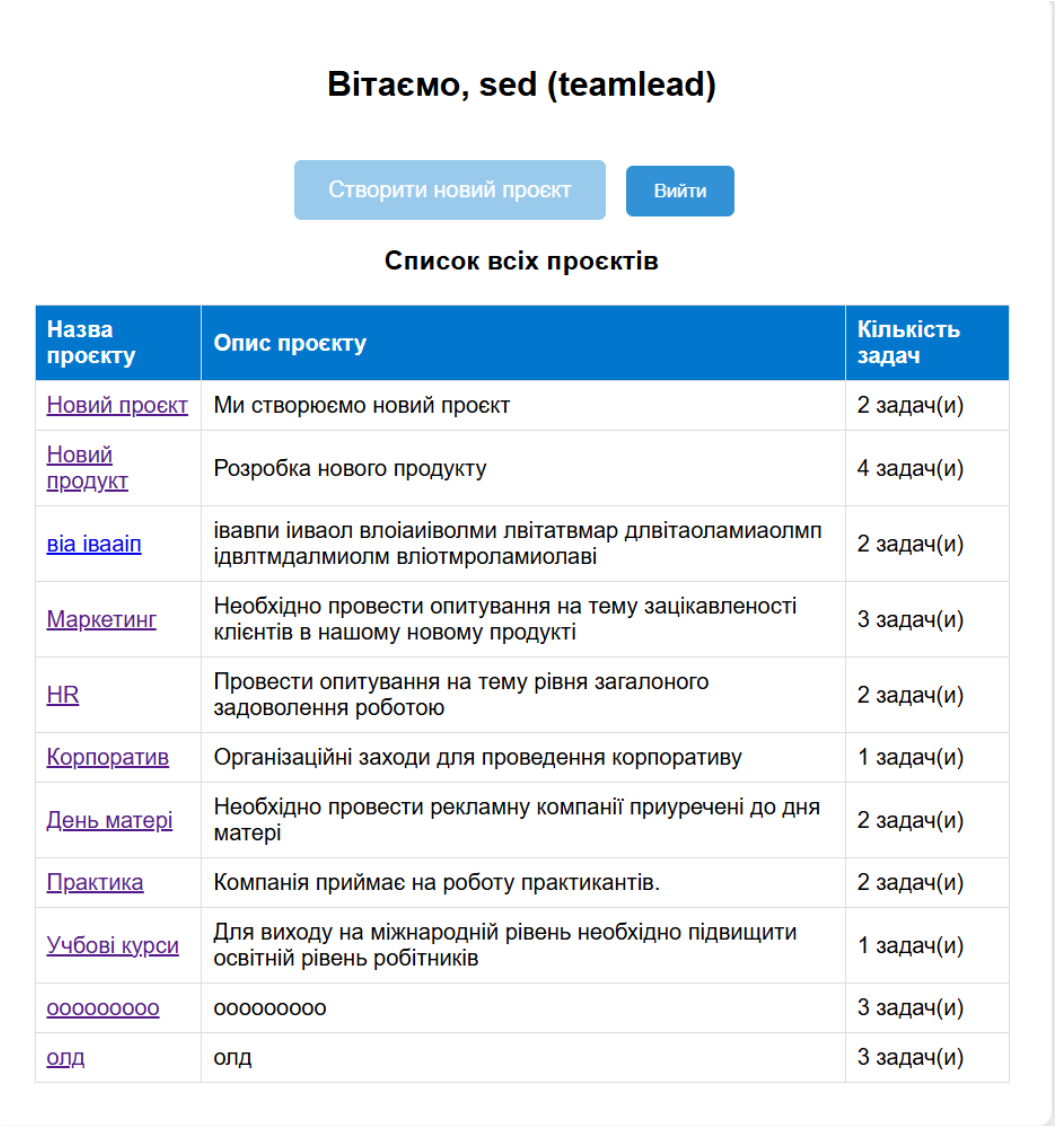


Рисунок 3.20 – Екранна форма списку проєктів

Джерело: розроблено автором самостійно

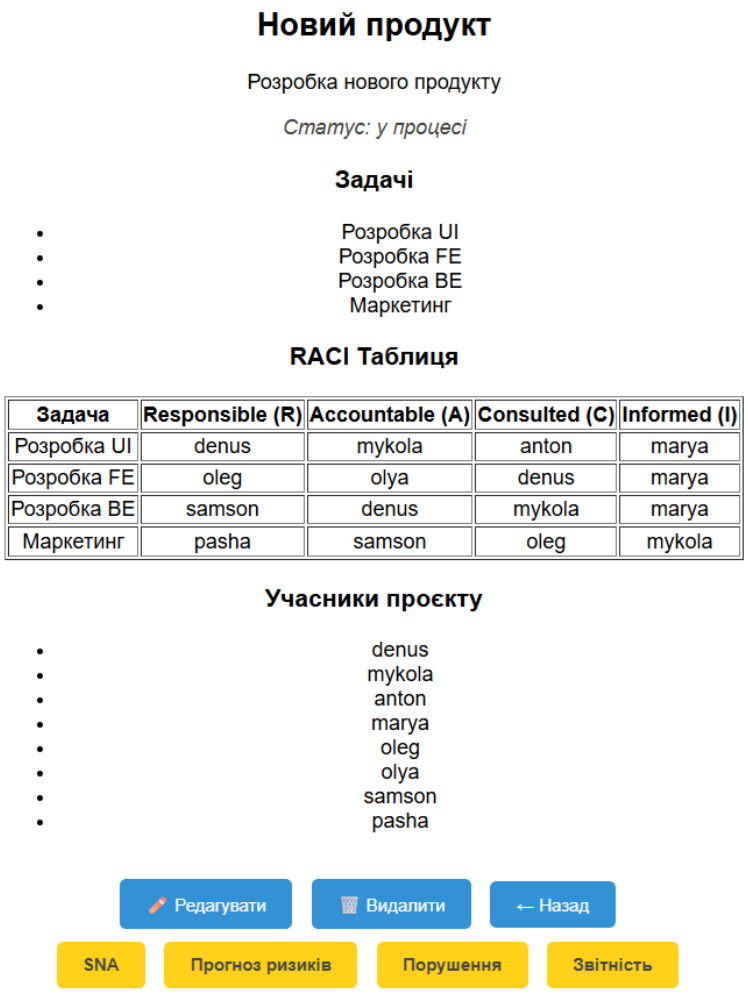


Рисунок 3.20 – Екранна форма інформації про обраний проєкт

Джерело: розроблено автором самостійно

В системі передбачений модуль інтелектуального аналізу, а саме: система проводить автоматичний аналіз логів подій користувачів у рамках вибраного проєкту та виводить результати в окремих розділах:

Користувач має змогу переглядати, фільтрувати та експортувати звіти у форматах PDF або CSV. Усі дані відображаються в інтерактивному вигляді – таблиці, графіки, діаграми залежностей. Система автоматично сповіщає адміністратора про нові інциденти через візуальні маркери у кабінеті.

Інтерфейс орієнтований на користувачів без спеціальної технічної підготовки, тому реалізовано підказки, логічну навігацію, структуру з вкладками та швидкий доступ до кожного розділу.

ВИСНОВКИ

Аналіз предметної області показав, що чіткий і формалізований розподіл ролей та відповідальності є обов’язковою умовою для забезпечення узгодженості командної роботи, уникнення дублювання функцій і підвищення якості розроблюваного програмного забезпечення. Встановлено, що моделі RACI, DACI та RAPID ефективно застосовуються для побудови структур управління, особливо в умовах гнучких методологій (Scrum, DevOps) і відповідно до міжнародних стандартів ISO/IEC 12207 та ISO/IEC 27001. Ідентифіковано типові процеси, що вимагають чіткого розмежування обов’язків, і доведено, що без такої формалізації виникають суттєві управлінські ризики, які негативно впливають на якість і строки виконання проєктів. Отримані результати підтверджують необхідність впровадження інформаційної системи, яка дозволить автоматизувати управління ролями з урахуванням вимог безперервного розвитку IT-продуктів.

За результатами аналізу сучасного ринку інформаційних систем виявлено, що більшість платформ підтримують базові або розширені механізми управління ролями, проте не завжди орієнтовані на гнучке використання моделей відповідальності в середовищі розробки програмного забезпечення. Найбільш функціонально відповідними виявились DevOps- та ITSM-рішення, тоді як системи класу IAM та ERP потребують додаткової адаптації. Це підтверджує доцільність розробки спеціалізованої інформаційної системи, яка забезпечуватиме автоматизоване управління ролями з урахуванням гнучких методологій, нормативних вимог і потреб конкретного проєктного середовища.

У результаті дослідження було сформовано структурну модель інформаційної системи управління ролями та відповідальністю, виділено основні типи користувачів, функціональні вимоги, сценарії використання та відповідні діаграми послідовності. Побудовано діаграми класів для ключових прецедентів, що дозволяє формалізувати архітектуру системи з урахуванням логіки управління, обробки даних та інтеграції інтелектуальної складової.

Описано типи користувачів, сценарії взаємодії, а також класи, які реалізують логіку призначення ролей, формування рекомендацій, прогнозування ризиків та виявлення порушень, що заклало основу для подальшої розробки функціональних компонентів системи.

За результатами опрацювання методів і моделей, що лежать в основі інформаційної системи управління ролями та відповідальністю, сформовано комплексний підхід, який поєднує формалізовану логіку RACI-моделі з інтелектуальними алгоритмами аналізу даних. Реалізація механізмів рекомендацій, прогнозування ризиків, соціальної аналітики та виявлення порушень забезпечує не лише структурне, а й адаптивне управління проєктними функціями, що підвищує ефективність прийняття рішень та якість взаємодії в команді.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Типи ролей у команді та їх значення [Електронний ресурс]. – Режим доступу: https://kadrovik.isu.net.ua/news/544359-typy-roley-u-komandi-ta-yikh-znachennya (дата звернення: 19.04.2025).

2. Ролі в команді [Електронний ресурс]. – Режим доступу: https://stud.com.ua/32301/menedzhment/roli_komandi (дата звернення: 19.04.2025).

3. Деренська Я. М. Команда проєктна [Електронний ресурс]. – Режим доступу: https://www.pharmencyclopedia.com.ua/article/3621/komanda-proektna (дата звернення: 18.04.2025).

4. IT професії [Електронний ресурс]. – Режим доступу: https://stsaltiv.gov.ua/useful-info/it-profesii (дата звернення: 19.04.2025).

5. Project team roles and responsibilities (with examples) [Електронний ресурс]. – Режим доступу: https://resourceguruapp.com/blog/project-team-roles-and-responsibilities (дата звернення: 19.04.2025).

6. Pratt Lile S. Defining a project team: what are different team roles in a typical work project? [Електронний ресурс]. – Режим доступу: https://www.beautiful.ai/blog/defining-a-project-team-what-are-different-team-roles-in-a-typical-work-project (дата звернення: 19.04.2025).

7. Landau P. 15 key project roles & their responsibilities [Електронний ресурс]. – Режим доступу: https://www.projectmanager.com/blog/project-roles-responsibilities (дата звернення: 19.04.2025).

8. Talas J. Roles and responsibilities in project team [Електронний ресурс]. – Режим доступу: http://wiki.doing-projects.org/index.php/Roles_and_responsibilities_in_project_team#cite_ref-20 (дата звернення: 19.04.2025).

9. Project team roles and responsibilities explained [Електронний ресурс]. – Режим доступу: https://www.villanovau.com/articles/project-management/project-team-roles-and-responsibilities/ (дата звернення: 19.04.2025).

10. Project team roles and responsibilities in project management [Електронний ресурс]. – Режим доступу: https://www.indeed.com/career-advice/career-development/project-team (дата звернення: 19.04.2025).

11. Project team member roles and responsibilities [Електронний ресурс]. – Режим доступу: https://hubstaff.com/tasks/project-team-roles-responsibilities (дата звернення: 19.04.2025).

12. Sommerville I. Software Engineering. London : Pearson, 2015. 816 p.

13. Балабанова Л. В., Сардак О. В. Управління персоналом : підручник. Київ : Центр учбової літератури, 2011. 468 с.

14. Бала О. І., Мукан О. В., Бала Р. Д. Принципи корпоративної культури підприємства: сутність та види. Вісник Національного університету «Львівська політехніка». 2010. № 682. С. 11–15.

15. Wrike – офіційний сайт продажу продуктів компанії. URL: https://www.wrike.com (дата звернення: 18.04.2025).

16. Wiegers K. Software Requirements (Developer Best Practices). London : Pearson, 2013. 672 p.

17. Созинова І., Ярмолюк О., Соболєва А. Специфіка ролей учасників проєктної команди та їх внесоку реалізацію інтернет-проєкту. Наукові перспективи. 2024. №. 3(45). URL: https://doi.org/10.52058/2708-7530-2024-3(45)-744-757.

18. Martin R.C. The Clean Coder: A Code of Conduct for Professional Programmers. London : Pearson, 2011. 256 p.

19. Fowler M. Refactoring: Improving the Design of Existing Code. Boston : Addison-Wesley Professional, 2018. 448 p.

20. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 432 p.

21. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Boston : Addison-Wesley Professional, 2018. 191 p.

22. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. London : Pearson, 2004. 736 p.

23. Rumbaugh J., Blaha M. UML 2.0. Object-Oriented Modeling and Design with UML. London : Pearson, 2012. 496 p.

24. Beck K. Test Driven Development: By Example. Boston : Addison-Wesley Professional, 2002. 240 p.

25. Axelrod A. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. New York : Apress, 2018. 588 p.

26. IBM. What is Identity and Access Management (IAM)? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/identity-access-management (дата звернення: 18.04.2025).

27. Okta. Identity and Access Management (IAM) overview [Електронний ресурс]. – Режим доступу: https://developer.okta.com/docs/concepts/iam-overview/ (дата звернення: 18.05.2025).developer.okta.com+1developer.okta.com+1

28. **2.** Shekhar G. KeyCloak: An Open Source for Identity and Access Management [Електронний ресурс]. – Режим доступу: https://medium.com/@gauravshekharster/keycloak-an-open-source-for-identity-and-access-management-0ac85b3b9eaa (дата звернення: 18.04.2025)

29. Tamres L. Introducing Software Testing. Boston : Addison-Wesley Professional, 2002. 304 p.

30. Макаровських Т. Документування програмного забезпечення. На допомогу технічному письменнику. М. : Ленанд, 2016. 264 с.

31. Шинкарук, Л.В., Сергій Кубіцький, Марина Дєліні. Особливості управління персоналом в проєктній діяльності в сучасних умовах. 2021.

32. Artificial intelligence as an enabler for achieving human resource resiliency: past literature, present debate and future research directions / G. Panda, M. Dash, A. Samadhiya, A. Kumar, E. Mulat-weldemeskel // International Journal of Industrial Engineering and Operations Management. – 2023. – https://doi.org/10.1108/ijieom-05-2023-0047.

33. Шульга, О. (2021). Сучасні підходи до розроблення й упровадження інформаційних систем управління підприємством: ТЕОРЕТИКО-

МЕТОДОЛОГІЧНИЙ АСПЕКТ . Підприємництво та інновації, (18), 62-66. https://doi.org/10.37320/2415-3583/18.11

34. Карпенко М.Ю., Манакова Н.О., Гавриленко І.О. Технології створення програмних продуктів та інформаційних систем. Харків : ХНУМГ ім. О.М. Бекетова, 2017. 93 с

35. Киричук, В. (2024). СУТНІСТЬ ТА ТИПОЛОГІЯ КОМАНД . Економіка та суспільство, (62). https://doi.org/10.32782/2524-0072/2024-62-67

36. САКУН, Л. ., ВЄДЄНІНА, Ю. ., & ШИШЛОВА, Ю. . (2023). ВПЛИВ СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА КОМАНДНОЇ РОБОТИ НА СИСТЕМУ МЕНЕДЖМЕНТУ ОРГАНІЗАЦІЙ В УМОВАХ ГЛОБАЛІЗАЦІЙНИХ ВИКЛИКІВ. MODELING THE DEVELOPMENT OF THE ECONOMIC SYSTEMS, (1), 85–92. https://doi.org/10.31891/mdes/2023-7-12

37. Аналіз сучасних корпоративних інформаційних систем, які пропонуються на ринку програмного забезпечення. URL: http://ism.flybb.ru/topic198.html (дата звернення: 13.04.2025)

38. Delligatti, Lenny. SysML Distilled: A Brief Guide to the Systems Modeling Language. – Addison-Wesley Professional, 2013.

39. Holt, Jon. SysML for Systems Engineering. – The Institution of Engineering and Technology, 2008. – ISBN 978-0-86341-825-9.

40. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. Sebastopol: O'Reilly Media, 2019. 819 p.

41. Mohri M., Rostamizadeh A., Talwalkar A. Foundations of Machine Learning. 2nd ed. Cambridge: MIT Press, 2018. 480 p.

42. Blokdyk G. RACI Matrix: A Complete Guide – 2021 Edition. [S.l.]: The Art of Service, 2020. 304 p.

43. Martin P.K. Matrix Management 2.0™ Quick Guide. Portland: Matrix Management Institute, 2014. 49 p.

44. Majumdar A. Collaborative Filtering: Recommender Systems. Boca Raton: CRC Press, 2024. 250 p.

45. Aggarwal C.C. Recommender Systems: The Textbook. Cham: Springer, 2016. 471 p.

46. Brownlee J. XGBoost for Regression Predictive Modeling and Time Series Analysis. [S.l.]: Machine Learning Mastery, 2018. 250 p.

47. Wang Y., Ni X.S. A XGBoost Risk Model via Feature Selection and Bayesian Hyper-Parameter Optimization. arXiv preprint arXiv:1901.08433, 2019. 15 p.

48. Lusher D., Koskinen J., Robins G. Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications. New York: Cambridge University Press, 2013. 360 p.

49. Knoke D., Yang S. Social Network Analysis. 3rd ed. Thousand Oaks: SAGE Publications, 2020. 136 p.

50. Vertabelo Database Modeler [Електронний ресурс]. – Режим доступу: https://vertabelo.com/ (дата звернення: 18.04.2025).

ДОДАТКИ

Додаток А. Згенерований скрипт для фізичної моделі БД

```
-- Created by Vertabelo (http://vertabelo.com)
-- Last modification date: 2025-05-12 23:37:26.717

-- tables
-- Table: ActionLog
CREATE TABLE ActionLog (
    log_id int  NOT NULL,
    action_type int  NOT NULL,
    description char(200)  NOT NULL,
    timestamp datetime  NOT NULL,
    project_function_id int  NOT NULL,
    user_id int  NOT NULL,
    target_user_id int  NULL,
    CONSTRAINT ActionLog_pk PRIMARY KEY  (log_id)
);

-- Table: ProcessFunctions
CREATE TABLE ProcessFunctions (
    function_id int  NOT NULL,
    function_name char(100)  NOT NULL,
    function_type char(30)  NULL,
    CONSTRAINT ProcessFunctions_pk PRIMARY KEY  (function_id)
);

-- Table: ProjectFunctions
CREATE TABLE ProjectFunctions (
    project_function_id int  NOT NULL,
    status int  NOT NULL,
    start_date date  NOT NULL,
    start_time time  NULL,
    end_date date  NOT NULL,
    end_time time  NULL,
    project_id int  NOT NULL,
    function_id int  NOT NULL,
    CONSTRAINT ProjectFunctions_pk PRIMARY KEY  (project_function_id)
);

-- Table: Projects
CREATE TABLE Projects (
    project_id int  NOT NULL,
    name_proj char(100)  NOT NULL,
    description int  NULL,
    start_date datetime  NOT NULL,
    end_date datetime  NOT NULL,
    CONSTRAINT Projects_pk PRIMARY KEY  (project_id)
);

-- Table: RACIRole
CREATE TABLE RACIRole (
    racirole_id int  NOT NULL,
    role_type char(1)  NOT NULL,
    ProjectFunctions_project_function_id int  NOT NULL,
    Users_user_id int  NOT NULL,
    CONSTRAINT RACIRole_pk PRIMARY KEY  (racirole_id)
);

-- Table: Users
CREATE TABLE Users (
```

```
    user_id int  NOT NULL,
    full_name char(60)  NOT NULL,
    email char(35)  NOT NULL,
    base_role char(1)  NOT NULL,
    CONSTRAINT Users_pk PRIMARY KEY  (user_id)
);

-- foreign keys
-- Reference: ActionLog_ProjectFunctions (table: ActionLog)
ALTER TABLE ActionLog ADD CONSTRAINT ActionLog_ProjectFunctions
    FOREIGN KEY (project_function_id)
    REFERENCES ProjectFunctions (project_function_id);

-- Reference: ActionLog_Users (table: ActionLog)
ALTER TABLE ActionLog ADD CONSTRAINT ActionLog_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id)
    ON DELETE  CASCADE
    ON UPDATE  CASCADE;

-- Reference: ActionLog_Users (table: ActionLog)
ALTER TABLE ActionLog ADD CONSTRAINT ActionLog_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id)
    ON DELETE  SET NULL
    ON UPDATE  SET NULL;

-- Reference: ProjectFunctions_ProcessFunctions (table: ProjectFunctions)
ALTER TABLE ProjectFunctions ADD CONSTRAINT ProjectFunctions_ProcessFunctions
    FOREIGN KEY (function_id)
    REFERENCES ProcessFunctions (function_id);

-- Reference: ProjectFunctions_Projects (table: ProjectFunctions)
ALTER TABLE ProjectFunctions ADD CONSTRAINT ProjectFunctions_Projects
    FOREIGN KEY (project_id)
    REFERENCES Projects (project_id);

-- Reference: RACIRole_ProjectFunctions (table: RACIRole)
ALTER TABLE RACIRole ADD CONSTRAINT RACIRole_ProjectFunctions
    FOREIGN KEY (ProjectFunctions_project_function_id)
    REFERENCES ProjectFunctions (project_function_id);

-- Reference: RACIRole_Users (table: RACIRole)
ALTER TABLE RACIRole ADD CONSTRAINT RACIRole_Users
    FOREIGN KEY (Users_user_id)
    REFERENCES Users (user_id);

-- sequences
-- Sequence: ActionLog_seq
CREATE SEQUENCE ActionLog_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;

-- Sequence: ProcessFunctions_seq
CREATE SEQUENCE ProcessFunctions_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;
```

```
-- Sequence: ProjectFunctions_seq
CREATE SEQUENCE ProjectFunctions_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;

-- Sequence: Projects_seq
CREATE SEQUENCE Projects_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;

-- Sequence: RACIRole_seq
CREATE SEQUENCE RACIRole_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;

-- Sequence: Users_seq
CREATE SEQUENCE Users_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    NO CYCLE
    NO CACHE;

-- End of file.
```

Додаток Б

Б.1. Код моделі стурктури підсистеми інтелектуального аналізу

```
@startuml
package "Інформаційна система розподілу ролей і відповідальності" {
database "База даних журналів подій" as db
[Модуль збору даних] --> db
package "Підсистема інтелектуального аналізу" {
  db --> [Модуль попередньої обробки даних]
  [Модуль попередньої обробки даних] --> [Process Mining Engine\n(Heuristics Miner)]
  [Модуль попередньої обробки даних] --> [Модуль аналізу графа взаємодій\n(SNA)]
  [Process Mining Engine\n(Heuristics Miner)] --> [Модуль виявлення відхилень\nта формування звітів]
  [Модуль аналізу графа взаємодій\n(SNA)] --> [Модуль виявлення відхилень\nта формування звітів]
  [Модуль виявлення відхилень\nта формування звітів] --> [Інтерфейс адміністратора / менеджера]
  db --> [Process Mining Engine\n(Heuristics Miner)]
  db --> [Модуль виявлення відхилень\nта формування звітів]
}
}
@enduml

@startuml


package "Інформаційна система розподілу ролей і відповідальності" {

  ' Централізоване сховище
  database "База даних\n(SQL Server)" as DB

  ' Інтерфейс користувача
  package "Користувач" {
    [Вебінтерфейс\nадміністратора \n/ менеджера] as UI
  }

  ' Серверна логіка / бізнес-рівень
  package "Серверна логіка\n(backend API)" {
    [Модуль \nавтентифікації\nта управління \nдоступом] as Auth
    [Модуль \nзбору \nлогів] as Log
    [Модуль \nкерування\nRACI-матрицями] as RACI
    [Модуль \nформування \nзвітів] as Reports
```

```
  UI --> Auth
  Log --> DB
  RACI --> DB
  Reports --> DB
  Reports --> UI
 }

 ' Інтелектуальна аналітика
 package "Підсистема інтелектуального аналізу" {
  [Модуль попередньої\nобробки даних] as Preproc
  [Process Mining] as Miner
  [Модуль аналізу \nграфів (SNA)] as SNA
  [Модуль виявлення\nвідхилень та\nгенерації висновків] as Deviation

  DB --> Preproc
  Preproc --> Miner
  Preproc --> SNA
  Miner --> Deviation
  SNA --> Deviation
  Deviation --> Reports
 }

}

@enduml
```

Додаток В. SQL-запит для формування датасету

/* Запит, що формує набір даних для інтелектуального аналізу*/

```
SELECT
    al.log_id, al.timestamp, al.action_type, al.description,
    u.user_id, u.full_name, u.email, u.base_role,
    pf.project_function_id, pf.status, pf.start_date AS pf_start,
    pf.end_date AS pf_end,
    p.project_id, p.name_proj, p.description AS project_description,
    p.start_date AS project_start, p.end_date AS project_end,
    f.function_id, f.function_name, f.function_type,
    rr.role_type AS raci_role
FROM ActionLog al
JOIN Users u ON al.user_id = u.user_id
JOIN ProjectFunctions pf ON al.project_function_id = pf.project_function_id
JOIN Projects p ON pf.project_id = p.project_id
JOIN ProcessFunctions f ON pf.function_id = f.function_id
LEFT JOIN RACIRole rr
    ON rr.ProjectFunctions_project_function_id = pf.project_function_id
    AND rr.Users_user_id = u.user_id
```

/* Запит, що підраховує кількість дій, що виконав користувач*/

```
SELECT
    al.user_id,
    pf.project_function_id,
    f.function_name,
    al.action_type,
    COUNT(*) AS action_count
FROM ActionLog al
JOIN ProjectFunctions pf ON al.project_function_id = pf.project_function_id
JOIN ProcessFunctions f ON pf.function_id = f.function_id
GROUP BY al.user_id, pf.project_function_id, f.function_name, al.action_type
ORDER BY al.user_id, pf.project_function_id, action_type;
```

```
!pip install networkx matplotlib pandas


#  Імпорт бібліотек
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/Colab Notebooks/Дані і
датасети/Role/flat_dataset_full.csv'
df = pd.read_csv(file_path)

# Завантаження даних
df = pd.read_csv(file_path)

# Перевірка структури
df.head(15)

# Побудова графа співпраці (SNA)
G = nx.Graph()

# Створимо ключ як комбінацію project_id + function_id
df['function_block'] = df['project_id'].astype(str) + "_" + df['function_id'].astype(str)

# Додаємо зв’язки між усіма користувачами, які працювали над однією функцією в межах проєкту
for key in df['function_block'].unique():
    participants = df[df['function_block'] == key]['user_id'].unique()
    for i in range(len(participants)):
        for j in range(i + 1, len(participants)):
            u1, u2 = participants[i], participants[j]
            if G.has_edge(u1, u2):
                G[u1][u2]['weight'] += 1
            else:
```

```
                G.add_edge(u1, u2, weight=1)

# · Візуалізація графа
plt.figure(figsize=(10, 8))
pos = nx.spring_layout(G, seed=42)
nx.draw_networkx_nodes(G, pos, node_color='skyblue', node_size=600)
nx.draw_networkx_edges(G, pos, width=[G[u][v]['weight'] for u, v in G.edges()])
nx.draw_networkx_labels(G, pos)
plt.title("Граф взаємодії користувачів (SNA)")
plt.axis('off')
plt.show()

# · Розрахунок метрик
degree_centrality = nx.degree_centrality(G)
betweenness_centrality = nx.betweenness_centrality(G, weight='weight')
eigenvector_centrality = nx.eigenvector_centrality(G, weight='weight')

# · Формування таблиці результатів
centrality_df = pd.DataFrame({
    'user_id': list(degree_centrality.keys()),
    'degree_centrality': list(degree_centrality.values()),
    'betweenness_centrality': list(betweenness_centrality.values()),
    'eigenvector_centrality': list(eigenvector_centrality.values())
})

# · Виведення 10 найвпливовіших користувачів
centrality_df.sort_values('eigenvector_centrality', ascending=False).head(10)

# □ Визначаємо основну роль кожного користувача (найчастішу)
main_roles = (
    df.groupby(['user_id', 'raci_role'])
    .size()
    .reset_index(name='count')
    .sort_values(['user_id', 'count'], ascending=[True, False])
    .drop_duplicates(subset=['user_id'])
    .set_index('user_id')['raci_role']
```

```
)

# Створюємо унікальний ключ для кожної функції в межах проєкту
df['function_block'] = df['project_id'].astype(str) + "_" + df['function_id'].astype(str)

# Додаємо ребра між користувачами, які працювали над тією самою функцією в межах проєкту
for block in df['function_block'].unique():
    participants = df[df['function_block'] == block]['user_id'].unique()
    for i in range(len(participants)):
        for j in range(i + 1, len(participants)):
            u1, u2 = participants[i], participants[j]
            if G.has_edge(u1, u2):
                G[u1][u2]['weight'] += 1
            else:
                G.add_edge(u1, u2, weight=1)

# · Кольори вузлів за RACI
role_colors = {'R': 'navy', 'A': 'darkgreen', 'C': 'orange', 'I': 'gray'}
node_colors = [role_colors.get(main_roles.get(n), 'black') for n in G.nodes()]

# · Розрахунок центральностей
eigen = nx.eigenvector_centrality(G, weight='weight')

# · Візуалізація
plt.figure(figsize=(12, 10))
pos = nx.spring_layout(G, seed=42)

nx.draw_networkx_nodes(G, pos, node_color=node_colors, node_size=[500 + 3000 * eigen[n] for n
in G.nodes()])
nx.draw_networkx_edges(G, pos, width=[G[u][v]['weight'] for u, v in G.edges()])
nx.draw_networkx_labels(G, pos, font_size=9)

# Легенда
for role, color in role_colors.items():
    plt.scatter([], [], c=color, label=role, s=100)
plt.legend(scatterpoints=1, frameon=False, labelspacing=1, title='RACI роль')
plt.title("Граф взаємодії користувачів з розфарбуванням за RACI-роллю")
```

```
plt.axis('off')
plt.show()

import community as community_louvain  # алгоритм Louvain

# Створено раніше: G = граф взаємодії користувачів

# Розрахунок кластерів за алгоритмом Louvain
partition = community_louvain.best_partition(G, weight='weight')

# Призначення кольору для кожної спільноти
unique_clusters = list(set(partition.values()))
cluster_colors = {cl: plt.cm.Set3(i / len(unique_clusters)) for i, cl in
enumerate(unique_clusters)}
node_colors = [cluster_colors[partition[n]] for n in G.nodes()]

# Візуалізація з підписами user_id
plt.figure(figsize=(12, 10))
pos = nx.spring_layout(G, seed=42)
nx.draw_networkx_nodes(G, pos, node_color=node_colors, node_size=600)
nx.draw_networkx_edges(G, pos, width=[G[u][v]['weight'] for u, v in G.edges()])
nx.draw_networkx_labels(G, pos, font_size=9)

plt.title("Кластери взаємодії користувачів за алгоритмом Louvain")
plt.axis('off')
plt.show()

import seaborn as sns

# Побудова зведеної таблиці
pivot_table = df.pivot_table(
    index='full_name',
    columns='action_type_name',
    aggfunc='size',
    fill_value=0
)
```

```
# Побудова теплової карти

plt.figure(figsize=(12, 8))

sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=.5, annot=True, fmt='d', cbar_kws={'label':
'Кількість дій'})

plt.title("Теплова карта активності користувачів за типами дій")

plt.xlabel("Тип дії")

plt.ylabel("Користувач")

plt.tight_layout()

plt.show()
```

Звіт подібності

метадані

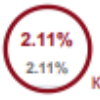
Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU
Заголовок
Розроблення інформаційної системи для управління ролями та відповідальністю в процесі розробки програмного забезпечення

Автор
Ляскович Ангеліна Олександрівна
Науковий керівник / Експерт
Шевченко К.Л.

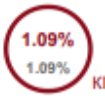
підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

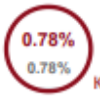
Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2



КЦ

25
Довжина фрази для коефіцієнта подібності 2

16101
Кількість слів

129217
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		1
Інтервали		0
Мікропробіли		3
Білі знаки		0
Парафрази (SmartMarks)	a	28

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розроблення інформаційної управляючої системи для оптимізації діяльності цифрового інноваційного хабу [redacted] 5/16/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	104 0.65 %
2	https://ir.kneu.edu.ua/bitstreams/725b64da-359e-479b-b2e4-50ed2d83d86c/download	43 0.27 %
3	https://ua-referat.com/uploaded/protokol--vid-2020-r-golova-nmr-a-m-kolot/index9.html	28 0.17 %