МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему «Система логічної інтеграції та очищення даних з неоднорідних джерел системи управління проектами»

«System of data logical integration and purification from heterogeneous sources of the project management system»

Виконав: студент денної форми навчання

спеціальності 123 – Комп'ютерна інженерія .

(шифр і назва напряму підготовки, спеціальності)

Шлемко Олександр Віталійович

(прізвище, ім'я, по-батькові)

Керівник д-р техн. наук, проф. Малахов Є. В.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент ст. викл. Максимов О. С. _____

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2021 р.

Завідувач кафедри

_____ Є.В. Малахов

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «_ _» ____ 2021 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Н.Ф. Казакова

(підпис) (прізвище, ініціали)

Одеса – 2021

АНОТАЦІЯ

В даній дипломній роботі розробляється інформаційна система управління проектами з вбудованою системою логічної інтеграції та очищення даних.

Мета роботи – проектування та побудова інформаційної системи управління проектами з вбудованою системою логічної інтеграції та очищення даних. Особливістю даної системи є можливість будувати процеси за допомогою методології OKR, а також можливість встановлювати динамічні зв’язки між об’єктами. В інформаційній системі реалізований захист від несанкціонованого доступу та здійснено розмежування повноважень різних категорій користувачів.

Для розробки інформаційної системи використані СУБД PostgreSQL та Neo4j, а також мова програмування JavaScript (фреймворк Express). Клієнтська частина реалізована за допомогою фреймворка Vue JS.

Результатом дипломної роботи є інформаційна система управління проектами, яка дозволяє користувачам ефективно планувати розподіл ресурсів, підвищувати якість комунікації між підрозділами, зв’язувати цілі бізнесу з цілями співробітників і, як наслідок, покращувати процеси управління проектами.

ABSTRACT

In this graduate work, a project management information system with a built-in system of data logical integration and purification is developed.

The purpose of the work is to design and build a project management information system with a built-in system of logical integration and data purification. A feature of this system is the ability to build processes using the OKR methodology, as well as the ability to establish dynamic relationships between objects. The information system provides protection against unauthorized access and differentiates the powers of different categories of users.

PostgreSQL and Neo4j databases, as well as JavaScript programming language (Express framework) were used to develop the information system. The client part is implemented using the Vue JS framework.

The thesis results in a project management information system that allows users to effectively plan the allocation of resources, improve the quality of communication between departments, link business goals to employee goals and, as a result, improve project management processes.

АННОТАЦИЯ

В данной дипломной работе разрабатывается информационная система управления проектами со встроенной системой логической интеграции и очистки данных.

Цель работы – проектирование и построение информационной системы управления проектами со встроенной системой логической интеграции и очистки данных. Особенностью данной системы является возможность строить процессы с помощью методологии OKR, а также возможность устанавливать динамические связи между объектами. В информационной системе реализована защита от несанкционированного доступа и осуществлено разграничение полномочий различных категорий пользователей.

Для разработки информационной системы использованы СУБД PostgreSQL и Neo4j, а также язык программирования JavaScript (фреймворк Express). Клиентская часть реализована с помощью фреймворка Vue JS.

Результатом работы является информационная система управления проектами, которая позволяет пользователям эффективно планировать распределение ресурсов, повышать качество коммуникации между подразделениями, связывать цели бизнеса с целями сотрудников и, как следствие, улучшать процессы управления проектами.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ПрО – предметна область.

AJAX – Asynchronous JavaScript and XML.

API (Application Program Interface) – прикладний програмний інтерфейс.

JWT – JSON Web Token.

ORM (Object-Relational Mapping) – об’єктно-реляційне відображення.

SPA (Single Page Application) – односторінковий застосунок.

ВСТУП

На сьогоднішній день з підвищенням темпів промисловості та зростанням вибагливості ринку все важливішу роль в процесах виробництва товарів та надання послуг займає управлінська сфера. Проекти стають масштабнішими і такими, що потребують більшого професіоналізму в управлінні. В той самий час діяльність менеджерів, пов’язаних з виконанням проектів, вимагає спеціальних навичок, організаційної структури та інструментів.

Завдання управління проектами – досягти встановлених цілей за показниками обсягів, часу, витрат та якості. Основні цілі проекту досягаються за допомогою певних процесів управління. І одне з головних завдань сучасних систем управління проектами полягає у забезпеченні планування, аналізу і управління всіма етапами проекту, пов'язавши ресурси і завдання зі стратегією бізнесу і бізнес-цілей.

Порівнюючи між собою існуючі системи управління проектами можна легко дійти висновку, що більшість з них надмірно великі, створюють багато проблем під час налаштування і мають заплутаний та не завжди інтуїтивний інтерфейс. Саме тому для пересічного представника проектного менеджменту іноді буває доволі складно якісно побудувати у них зручні процеси розробки продуктів.

Така система повинна:

1) підтримувати механізм побудови розгалужених зв’язків між цілями, ключовими результатами та задачами;

2) надати користувачам можливість створювати корпоративні та особисті цілі та вимірювати їх кількісно за допомогою ключових результатів;

3) спростити процес відстеження навантаження на співробітників;

4) надати користувачам можливість створювати команди співробітників та призначати їм задачі.

Крім того, в таких системах є необхідність встановлювати багатомірні зв'язки між сутностями та побудову складних багаторівневих ієрархій. Звісно, більшість понять предметної області можна розкласти у реляційну модель даних, але під час побудови зв’язків між ними з’являється проблеми. Стандартні методи виявляються доволі неефективними, так як структура бази даних буде ставати дедалі складнішою з кожною появою нового типу зв’язків між сутностями. Одним з рішень проблем такого типу є застосування інших моделей даних для зв’язування об’єктів одне з одним. У якості прикладу можна обрати графову модель даних. Якщо умовно прийняти вершини графа за об’єкти, то усі зв’язки можна представити у вигляді ребер графа між його вершинами. Такий підхід значно спрощує процес побудови ієрархічних структур та дозволяє гнучко маніпулювати зв’язками не змінюючи при цьому структуру бази даних.

Отже, метою даного проекту є проектування та побудова інформаційної системи управління проектами з вбудованою системою логічної інтеграції та очищення даних. Для досягнення цієї мети необхідно вирішити наступні завдання:

1) проаналізувати процеси управління проектами та існуючі системи їх автоматизації;

2) визначити існуючі проблеми управління проектами та можливі способи їх вирішення;

3) вибрати архітектуру системи і засоби реалізації;

4) побудувати систему, яка дасть змогу особам, залученим до управління проектами контролювати процеси всередині компанії;

5) забезпечити захист системи від несанкціонованого доступу і запровадити розмежування доступу з боку різних категорій користувачів.

1 ЗАГАЛЬНА ІНФОРМАЦІЯ ТА СИСТЕМИ УПРАВЛІННЯ ПРОЕКТАМИ

1.1 Постановка задачі

Під час аналізу предметної області виявлені три основні типи користувачів інформаційної системи:

1) співробітник – може створювати персональні цілі та ключові результати, вести облік власних задач, відстежувати прогрес компанії у досягненні корпоративних цілей;

2) менеджер – має можливість створювати задачі для співробітників, оновлювати корпоративні цілі та результати, групувати співробітників у команди, створювати персональні цілі та ключові результати;

3) адміністратор – може переглядати, змінювати і видаляти дані предметної області, а також керувати обліковими записами користувачів ІС.

Більш детально всі задачі користувачів описані в додатку А.

1.2 Загальні відомості про системи управління проектами.

Системи управління проектами – це програмні системи, що дозволяють автоматизувати одну або декілька складових управління проектами: складання календарного плану робіт, управління ресурсами, витратами, ризиками, якістю тощо.

Будь-яка система управління проектами передбачає наявність певних цілей і завдань. Цілями систем управління проектами є:

1) підвищення ефективності співробітників організації в роботі з проектами;

2) покращення якості проект-менеджменту керівників проектів;

3) підвищення ефективності управління загальним портфелем організації (за конкретний термін реалізується більше проектів при менших витратах ресурсів).

А серед завдань систем управління проектами можна виділити, головним чином, наступні:

1) надання учасникам проектів простих і ефективних інструментів для виконання завдань і доступу до будь-якої інформації, яка може знадобитися;

2) забезпечення керівників проектів інструментами для планування проектів і контролю їх виконання;

3) надання керівникам підрозділів інструментів з контролю обсягів робіт співробітників за завданнями в рамках і поза рамками проектів;

4) надання керівникам інформації для прийняття рішень про закріплення за співробітниками нових проектів і перерозподілу обсягів робіт між ними;

5) забезпечення керівників проектів єдиної моніторингової панеллю за всіма проектами, що включає можливість аналітики відхилень і прийняття управлінських рішень.

Найчастіше системи управління проектами складаються з наступних структурних елементів:

1) засоби для календарного планування;

2) засоби для вирішення окремих задач (проектний аналіз, розробка бізнес-планів, аналіз ризиків, управління контрактами, часом, бюджетом);

3) засоби для організації комунікацій між виконавцями проекту.

Переваги застосування систем управління проектами, в першу чергу, передбачають заощадження коштів. Витрати на впровадження та використання систем управління проектами окупаються завдяки тому, що проектна діяльність стає більш ефективною – підвищується ефективність операцій з фінансами, ресурсами і строками.

1.3 Існуючі системи управління проектами

Так як в даний час попит на системи управління проектами є вкрай великим, представлена на ньому лінійка систем є доволі різноманітною. Для порівняльного аналізу в рамках даної роботи обрані найпопулярніші системи управління проектами, а саме:

1) Jira;
2) Asana;
3) YouTrack.

Всі вищезгадані системи знаходяться на ринку доволі довго і вже встигли себе добре зарекомендувати. Однак кожна з цих систем має ряд суттєвих недоліків, які проявляються під час користування і негативно позначаються на комфорті користувачів. Узагальнений список проблем більшості продуктів на ринку:

1) перше налаштування більшості систем управління проектами - довгий та складний процес;

2) велика кількість систем не дає користувачам можливості комбінувати декілька підходів до управління проектами;

3) часто у користувача немає доступу до необхідних функцій одразу після налаштування. Для вирішення своїх задач необхідно встановлювати додаткові плагіни (найчастіше платні);

4) через велику кількість інформації у інтерфейсі важко виділити дійсно важливі моменти.

Також варто відзначити, що окрім основних вищезгаданих проблем також є ряд більш специфічних недоліків, притаманних кожній системі окремо. Детальний порівняльний аналіз популярних систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих систем управління проектами

	Agile	MindMap	Діаграми Ганта	Чат	Документ ообіг	Логування часу	Звіти	Зручний інтерфейс
Jira	Так	Так	Так	Ні	Ні	Так	Так	Ні
Asana	Так	Ні	Так	Так	Так	Ні	Так	Ні
YouTrack	Так	Ні	Так	Ні	Так	Так	Так	Так

Як можна помітити, усі наведені в таблиці 1.1 системи пропонують використання принципів Agile під час роботи, надають можливість календарного планування за допомогою діаграм Ганта та можливість налаштувань системи звітів. До недоліків, пов’язаних з процесами управління, можна віднести складність інтерфейсу, відсутність у деяких систем вбудованого чату, документообігу та логування часу у задачах.

1.4 Методологія OKR

OKR (Objectives and Key Results) – одна з методологій управління проектами, яка використовується в сучасному менеджменті. Своїй появі OKR завдячує Джону Дорру, який запропонував введення даної методології під час роботи у компанії Intel у 1952 році [1]. Вважається, що OKR є продовженням і розвитком методології MBO (Management by objectives)[2] і була покликана вирішити її стандартні проблеми:

1) спуск цілей згори донизу;
2) виконання конкретних завдань без розуміння процесу;
3) централізація прийняття рішень;
4) обмеження інформації для співробітників;
5) довгий бюрократизований процес постановки цілей;
6) пряма прив'язка цілей до бонусів;
7) річний цикл планування.

OKR – це система постановки цілей, що дозволяє побудувати дерево цілей всієї організації зі стратегічного рівня і до конкретного виконавця. Цілі визначають цінність для бізнесу або клієнта, а ключові результати встановлюють вимірні і однозначні ознаки досягнення цілей. OKR дуже проста і гнучка система, в ній немає чітко описаних правил і алгоритмів. Але в той же час є набір базових принципів, які дозволяють отримувати значні вигоди, наприклад:

1) побудувати ієрархію цілей всередині організації;

2) спрямовувати зусилля команди на вирішення дійсно важливих проблем;

3) підвищити прозорість і якість комунікації між підрозділами;

4) підвищити гнучкість і свободу прийняття рішень для «осередків» всередині організації;

5) дати менеджерам інструмент повноцінного делегування повноважень і відповідальності.

Серед основних відмінностей OKR від інших методологій управління проектами можна виділити декілька найголовніших:

1) амбіційність цілей – цілі мають бути складними, але досяжними, успішним вважається досягнення цілі на більше ніж 70 – 75%;

2) відв'язка від бонусів і іншої матеріальної мотивації;

3) вертикальна зв'язаність цілей – дерево цілей будується таким чином, щоб можна було піднятися від конкретної людини (або команди) до цілей компанії та виявити там чіткий зв'язок;

4) публічність – будь-який користувач повинен мати можливість простежити зв’язок від його мети до мети компанії;

5) вимірність результатів – кожен ключовий результат має обов’язково вимірюватись чисельно.

Для повного розуміння побудови процесів за допомогою OKR розглянемо наступну ситуацію для прикладу: компанія поставила перед собою ціль потрапити в трійку найбільш завантажуваних застосунків в App Store. Для цього визначили три ключових результати:

1) збільшити число користувачів застосунка на 50%;
2) підвищити рейтинг застосунка до 4,5 зірок;
3) отримати 100 позитивних відгуків від користувачів.

На рівні команди мобільної розробки корпоративній ціль можна визначити наступним чином: збільшити задоволеність користувачів інтерфейсом на 50%. Ключові результати командного рівня:

1) випустити дві версії нового інтерфейсу;
2) отримати як мінімум 50 відгуків про нові інтерфейси;
3) додати мінімум три мови для перекладів застосунку;
4) опитати 1000 користувачів та виявити головні скарги і запитувану функціональність.

В той самий час на індивідуальному рівні працівник може поставити собі ціль збільшити задоволеність користувачів інтерфейсом на 50% і визначити кількісні оцінки свого прогресу:

1) реалізувати дві нові сторінки в новому інтерфейсі;
2) отримати як мінімум 50 відгуків про новий інтерфейс.

Звісно, OKR сама по собі не є абсолютною гарантією побудови якісних процесів, але її використання у поєднанні з іншими методами управління проектами здатно суттєво полегшити рутинні задачі та безумовно варто уваги. Зважаючи на ефективність даної методології, було вирішено використати концепції даної методології під час проектування розроблюваної інформаційної системи.

1.5 Онтологічний підхід до створення моделі ПрО

Під онтологіями [3] будемо розуміти спільно використовувані, формальні класифікації предметної області (ПрО). Формально, онтологія - універсальна модель знання, яка являє собою систему понять (об'єктів), властивостей цих об'єктів і відносин між цими об'єктами. Об'єктами предметної онтології є об'єкти предметної області, виявлені та вибрані під час аналізу ПрО.

Онтології є моделями даних, що володіють двома специфічними особливостями:

1) онтології будуються на основі спільного розуміння предметної області в рамках спільноти;

2) онтології використовують спосіб представлення, який може оброблятися комп'ютерними програмами (тобто записуються з використанням формальних мов), що дає можливість комп'ютерам працювати з онтологіями, до таких дій відносяться передача онтологій між комп'ютерами, зберігання онтологій, перевірка узгодженості онтологій, виконання логічних висновків на онтологіях.

Процес створення онтології – це процес побудови системи логічних співвідношень, кожне з яких має зміст, з яким згідна певна спільнота, а вся система є явним поданням знань предметної області.

Визначимо основні поняття, що використовуються для побудови концептуальної моделі:

1) Об'єкт – абстрактна множина предметів, в якому всі предмети-екземпляри мають одні і ті ж характеристики. Усі екземпляри підпорядковані і узгоджені з одним і тим самим набором правил та кожен об'єкт в інформаційній моделі повинен мати свій унікальний ідентифікатор.

2) Атрибут – кожна окрема характеристика, яка є спільною для всіх можливих екземплярів об'єкта. Кожен атрибут визначається ім'ям, унікальним в межах об'єкта. Множина атрибутів може об'єднуватися в групу атрибутів і мати власний ідентифікатор групи атрибутів.

3) Представлення – об'єкт в інформаційній моделі разом зі своїми атрибутами, представлений в графічному або текстовому вигляді.

4) Зв’язок – це абстракція набору відносин, які систематично виникають між різними видами предметів в реальному світі. Реальні предмети повинні бути самі абстраговані як і об'єкти. Кожен зв'язок повинен мати унікальний ідентифікатор.

Для фіксації значущих відносин між об’єктами виділяються основні зв'язки предметної області, які можна відобразити графічно. Такі зв'язки надалі стають основою моделі інформаційної системи.

1.6 Висновок до розділу 1

Проведений аналіз подібних існуючих на ринку систем для управління проектами, виявив ряд проблем, пов’язаних з використанням даних систем. До широко розповсюджених недоліків можна віднести складність інтерфейсу користувача, довгий та складний процес первісного налаштування та практична неможливість комбінування декількох методологій управління проектами. Доцільність реалізації розроблюваної системи ґрунтується на можливості вирішення вищезазначених проблем завдяки використанню онтологічного підходу до проектування, вибору гнучкої методології управління проектами та використання допоміжної моделі даних.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Інформаційне моделювання предметної області

Згідно онтологічного підходу [4] під час процесу моделювання складноструктурованої предметної області [5], спершу необхідно визначити основні сутності, спираючись на бізнес-задачі користувачів, наведені в додатку А.

В інформаційній системі управління проектами виділені наступні сутності:

1) користувачі – містить у собі інформацію про користувачів інформаційної системи;

2) проекти – містять у собі інформацію про проекти, створені у даній інформаційній системі;

3) цілі – містять у собі інформацію про поставлені корпоративні або особисті цілі;

4) ключові результати – є кількісним виміром прогресу виконання цілі, визначаються поточним значенням та кінцевим значенням, виходячи з якого обчислюється об’єм виконаної роботи;

5) задачі – представляють собою точно визначене завдання певному виконавцю, містить інформацію про хід виконання, та заплановані витрати часу;

6) команди – групи виконавців, об'єднані схожими/перехресними обов’язками, або які мають однаковий фах;

7) стан задачі – являє собою перелік станів, які проходить задача від моменту її створення і до моменту її прийняття/відхилення, може використовуватися для групування задач (наприклад, на канбан-дошці);

8) пріоритет задачі – містить відомості про терміновість задачі, використовується під час планування робочого навантаження;

9) зв’язки між задачами – є одним з найважливіших понять предметної області, використовуються під час зв’язування інших елементів предметної області, можуть мати зворотній напрям (такі зв’язки називаються реверсивними);

10) типи зв’язків між задачами – поняття, яке регламентує, які саме зв’язки можна встановлювати між об’єктами;

11) об’єкти для зв’язків – містить перелік комбінацій об’єктів, між якими можливо встановлювати зв’язки.

Після визначення основних сутностей предметної області необхідно формалізувати зв’язки між ними. Варто зауважити, що під час проектування зв’язки у предметній області були розділені на ті, які доцільно відобразити за допомогою реляційної моделі та ті, які варто відображати за допомогою графової моделі. Основним критерієм приналежності певного типу зв’язків до графової моделі є відсутність чіткого визначення типу об’єктів, які можуть мати цей зв’язок між собою. Розглянемо основні типи зв’язків між сутностями, починаючи зі зв’язків, які відображаються графовою моделлю.

Почнемо із зв’язків, які описують породжуючі відношення між сутностями. Наприклад, об’єкт проекту може мати зв’язок типу «IS_PARENT_OF», по відношенню до об’єкту цілі, так як такий зв'язок є розповсюдженим при використанні методології OKR. Так само такий вид зв’язку може існувати між об’єктами цілей та ключових результатів, там між іншими об’єктами, якщо один із них породжує інший. Зворотним (реверсивним) зв’язком до даного буде зв’язок типу «IS_CHILD_OF». Варто окремо зазначити, що реверсивні зв’язки не потребують визначення на рівні бази даних і є суто логічною одиницею.

Далі розглянемо тип зв’язку «RELATES_TO», який дозволяє логічно поєднувати об’єкти за обраними ознаками. Наприклад, можна поєднати задачі, які відносяться до різних ключових результатів, але є подібними за призначенням. Варто виділити той факт, що даний тип зв’язку не має окремого зворотного зв’язку – він відображається сам на себе.

Наступними розглянемо тип зв’язку, який дозволяє будувати залежності між об’єктами – «DEPENDS_ON». Цей тип зв’язків характерний для ситуацій, коли взаємодія з одним об’єктом є неможливою без іншого. Як приклад можна навести ситуацію з задачами різних рівнів, коли невиконання однієї задачі блокує виконання іншої. Для реверсивного зв’язку в даному випадку використовується «IS_REQUIRED_FOR».

Останнім типом зв’язків у графовій моделі є тип зв’язку, який також доволі часто використовується для опису зв’язків між задачами. Зв’язок «DUPLICATES» та зворотний для нього зв’язок «IS_DUPLICATED_BY» в управлінні проектами використовуються для маркування задач та цілей, які повністю, або частково дублюють одне одного.

Приклад структури графової бази даних з використанням вищезгаданих зв’язків наведено на рисунку 2.1.

Далі розглянемо зв’язки між сутностями, описаними в рамках реляційної моделі.

Розглянемо зв’язок між задачами та їх станами. У кожної задачі обов’язково має бути стан, у якому вона зараз перебуває, маємо безумовний зв’язок типу один-до-багатьох. Для формалізації даного зв’язку додамо ідентифікатор стану задачі до таблиці задач в якості зовнішнього ключа.

Також розглянемо зв’язок між задачами та їх пріоритетами. Кожна задача має свій пріоритет, маємо безумовний зв’язок типу один-до-багатьох. Для формалізації даного зв’язку додамо ідентифікатор пріоритету до таблиці задач в якості зовнішнього ключа.

Далі розглянемо зв’язок між задачами та їх виконавцями. Кожна задача може мати одного виконавця, маємо умовний зв’язок типу один-до-одного. Для формалізації даного зв’язку додамо ідентифікатор користувача до таблиці задач в якості зовнішнього ключа.

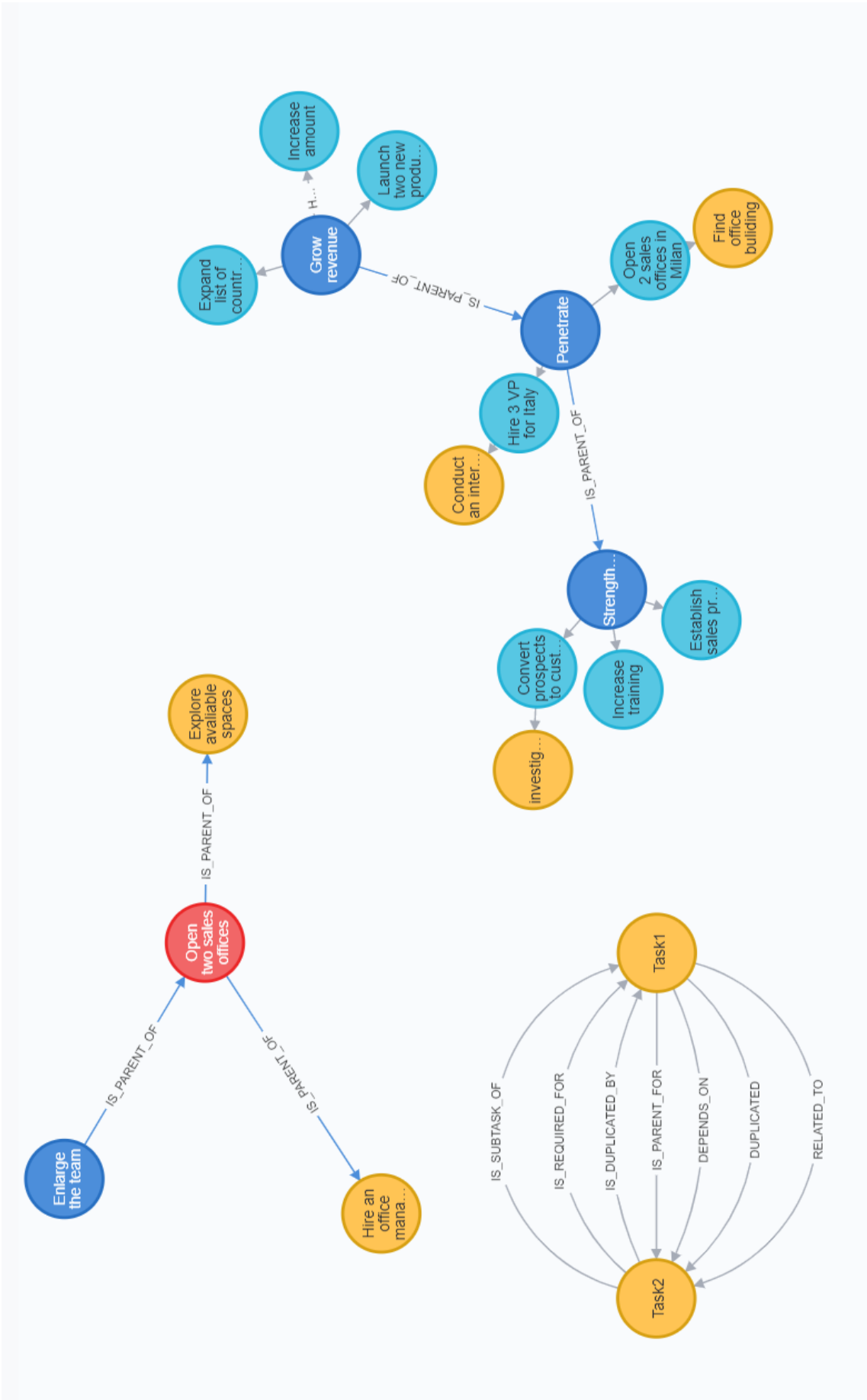

Рисунок 2.1 – Структура графової БД

Наступним розглянемо зв’язок між користувачами та командами. Кожен користувач може належати до декількох груп, маємо умовний зв’язок типу багато-до-багатьох. Отже, для формалізації даного зв’язку створимо нову сутність, у яку додамо ідентифікатори користувача та команди у якості зовнішніх ключів.

Розглянемо зв’язок між цілями та їх виконавцями. Кожна ціль може мати виконавця, маємо умовний зв’язок типу один-до-одного. Для формалізації даного зв’язку додамо ідентифікатор користувача до таблиці цілей у якості зовнішнього ключа.

Далі розглянемо зв’язок між цілями та командами. Ціль може мати команду, якій вона належить, маємо умовний зв’язок типу один-до-одного. Для формалізації даного зв’язку додамо ідентифікатор команди до таблиці цілей у якості зовнішнього ключа.

Останніми розглянемо з’єднання об’єктів для зв’язків з типами самих зв’язків. Очевидно, що один і той самий вид зв’язку може з’єднувати багато різних об’єктів. Отже, маємо зв’язок типу багато-до-багатьох і для його формалізації створимо нову сутність, у яку додамо ідентифікатори об’єкту для зв’язків і типу зв’язків у якості зовнішніх ключів.

ER-діаграма, отримана внаслідок формалізації зв’язків наведена на рисунку 2.2. Для побудови діаграми використана програмна система DB Diagram.

2.2 Архітектура системи

Для виконання поставлених завдань в рамках ІС обрана триланкова архітектура.

Триланкова (трирівнева) архітектура (рис. 2.3) передбачає наявність таких компонентів програми:

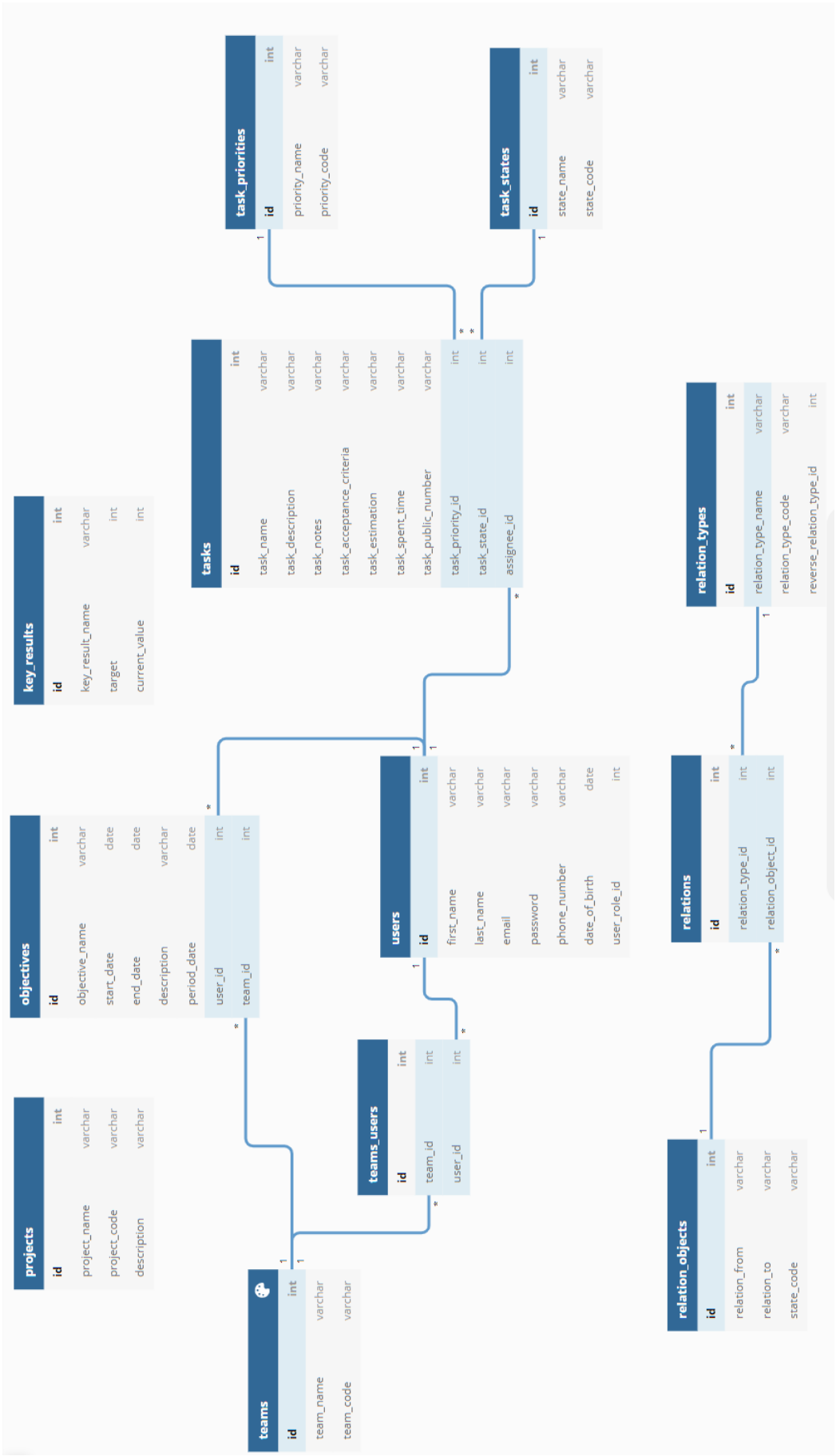

Рисунок 2.2 – ER - діаграма реляційної БД

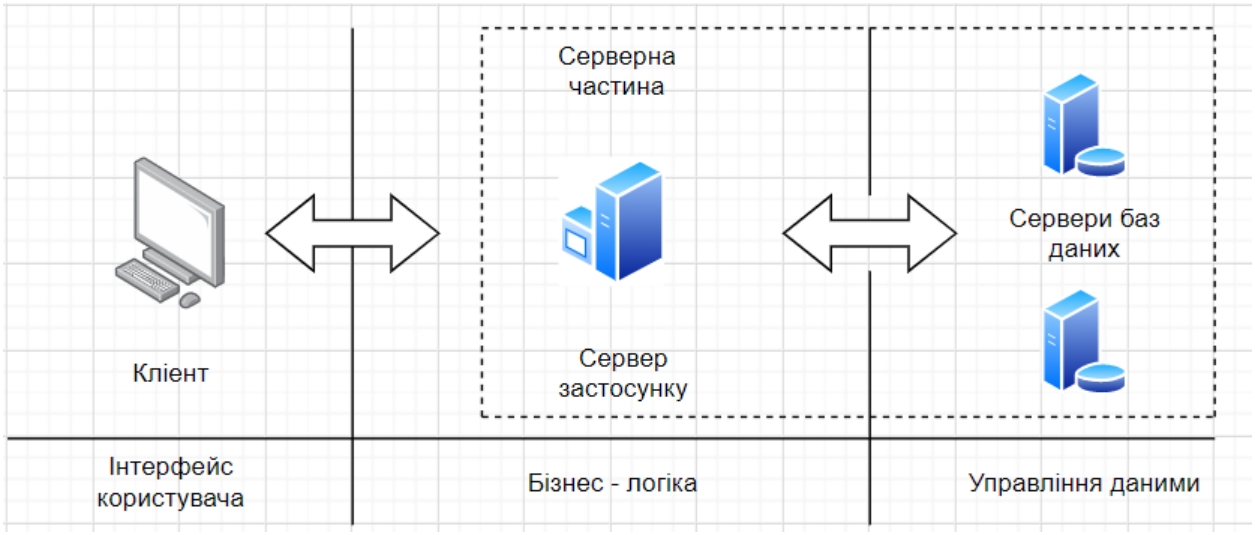

Рисунок 2.3 – Трирівнева архітектура «клієнт-сервер»

1) клієнтська частина – інтерфейсний шар, головне завдання якого – надати користувачеві дані в зручній для нього формі, а також дати йому можливість взаємодіяти з ними;

2) сервер застосунку – другий шар, на якому зосереджена бізнес-логіка додатка, виконує сполучну функцію між клієнтом і базою даних, поза ним залишаються лише фрагменти, винесені на клієнтську частину, а також елементи, винесені безпосередньо на сторону сервера бази даних (тригери і процедури);

3) сервер бази даних – забезпечує зберігання даних, зазвичай виноситься на окремий рівень і реалізується засобами СУБД.

Основною перевагою триланкової архітектури є її підвищена безпека і надійність, так як клієнтові не дозволено звертатися до сервера бази даних безпосередньо. У порівнянні з дворівневою клієнт-серверною або файл-серверною архітектурою, трирівнева архітектура забезпечує більшу конфігурованість. Також варто відзначити гнучкість даної архітектури та гарну здатність до масштабування.

Окремо варто зауважити, що в рамках даної роботи шар баз даних представлений одразу двома серверами – сервером реляційної та графової баз даних. Такий вибір пояснюється специфікою обраної предметної області, яка

полягає у необхідності зберігати складні багаторівневі структури у базі даних.

Після вибору архітектури необхідно визначитися з шаблоном проектування, який буде використаний в рамках розробки веб-програми. Для реалізації коректного взаємодії між усіма частинами обраний патерн MVC.

Шаблон проектування MVC (рис. 2.4) передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: Модель, Представлення і Контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно.

Зупинимося більш детально на кожному з елементів патерну:

1. Модель – цей компонент відповідає за дані, а також визначає структуру програми.

2. Представлення – цей компонент відповідає за взаємодію з користувачем. Тобто код компонента визначає зовнішній вигляд програми і способи його використання.

3. Контролер – цей компонент відповідає за зв'язок між моделлю і представленням. Код контролера визначає, як додаток реагує на дії користувача.

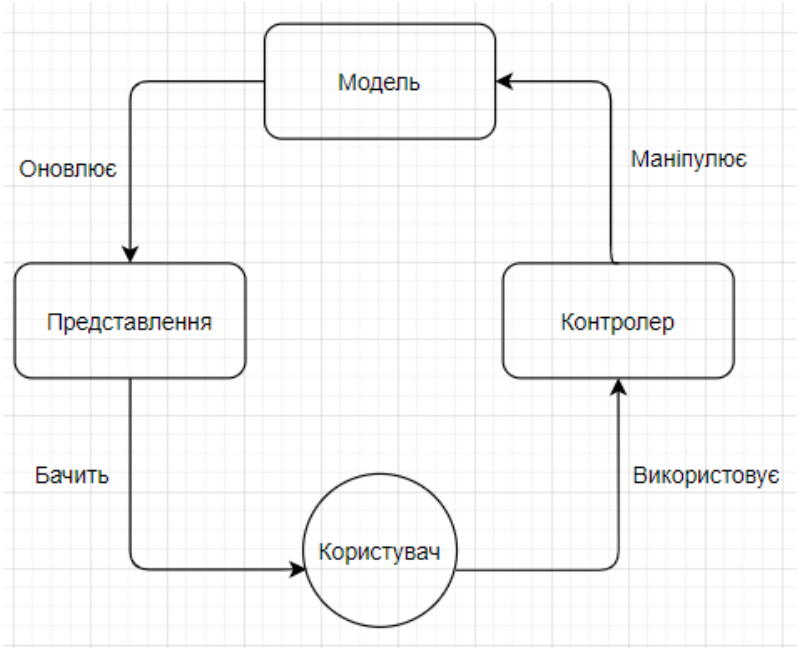

Рисунок 2.4 – Шаблон проектування MVC

Даний патерн зручний у використанні під час розробки додатків з триланковою архітектурою, так як можна провести паралелі між компонентами патерну і шарами архітектури. Так, наприклад, уявлення (клієнт) надає дані користувачеві. Модель описує об'єкти БД, створює підключення до сервера бази даних і виконує необхідні операції над даними. Також вона інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних. Контролер ж є сполучною компонентом між моделлю і представленням, в ньому описані всі бізнес-процеси, що характеризують логіку роботи програми в цілому.

Варто окремо зауважити, що окрім вибору відповідного шаблону проектування слід також дотримуватись основних принципів програмування, таких як SOLID, KISS, DRY і тд. Саме дотримання цих основних принципів разом з відповідними шаблонами проектування дає змогу побудувати стабільно працюючу, відкриту до масштабування та розширення систему.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір програмного забезпечення

Дана інформаційна система розроблена у вигляді веб-застосунку. В якості СУБД обрані PostgreSQL та Ne04j.

PostgreSQL – це популярна безкоштовна об'єктно-реляційна система управління базами даних. PostgreSQL базується на мові SQL і підтримує численні можливості. Можна виділити наступні переваги використання даної СУБД:

1) PostgreSQL є безкоштовним ПЗ з відкритим вихідним кодом;
2) потужні і надійні механізми транзакцій і реплікації;
3) існує можливість розширення функціоналу за рахунок збережених процедур, тригерів, уявлень, складних типів даних;

Ne04j – це високопродуктивна, NoSQL база даних побудована з використанням графової моделі. У ній немає такого поняття як таблиці із чітко заданими полями, вона оперує гнучкою структурою у вигляді вузлів і зв'язків між ними.

Графові бази даних, в першу чергу, призначені для вирішення тих завдань, де дані тісно пов'язані між собою у відносинах, які можуть заглиблюватися в кілька рівнів. Розгалужені зв'язки особливо актуальні в різних соціальних сферах застосування, в задачах пошуку маршрутів і т.п. Графові бази даних покликані вирішити проблеми, коли дані можуть бути віддалені один від одного на два і більше рівні. Вони вирішуються при використанні моделі даних як вершин графа, а зв'язків - як ребер графа між цими вузлами та обходів графа за допомогою давно відомих і ефективних алгоритмів.

Серверна частина програми написана на програмній платформі Node.js з використанням фреймворку Express.js. Node.js – програмна платформа, що перетворює JavaScript з вузькоспеціалізованої мови в мову загального

призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення, підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript коду. В основі Node.js лежить подієво-орієнтоване і асинхронне програмування з неблокуючим вводом / виводом.

Неблокуючий ввід / вивід та асинхронна обробка запитів зробили Node.js здатним обробляти запити без будь-яких затримок. В контексті серверної частини синхронна обробка передбачає, що код виконується послідовно. Таким чином, кожен запит блокує потік, змушуючи інші запити чекати його завершення. Асинхронна обробка дозволяє обробляти запити без блокування потоку. Таким чином, після обробки запиту він може відправити зворотний виклик і продовжити обслуговування запитів. Це дозволяє Node.js максимально використовувати однопоточність, що призводить до короткого часу відгуку і паралельній обробці.

Інший аспект – це подієва модель. При використанні спільної мови на стороні клієнта і сервера синхронізація відбувається швидко, що особливо корисно для застосунків реального часу, заснованих на подіях. Завдяки своїй асинхронній, неблокуючій і однопоточній природі Node.js є популярним вибором для онлайн-ігр, чатів, відеоконференцій або будь-якого іншого рішення, яке вимагає постійно оновлюваних даних.

Використання Express.js як серверного фреймворка дає всі переваги повнофункціональної розробки на JavaScript, такі як:

1) велика ефективність і загальна продуктивність розробника;
2) спільне використання та повторне використання коду;
3) швидкість і продуктивність;
4) легкий обмін знаннями в команді;
5) значну кількість безкоштовних інструментів.

У якості ORM-системи для серверу застосунку була обрана Sequelize ORM. Sequelize – базована на промісах [7] ORM-система для баз СУБД

Postgres, MySQL, MariaDB, SQLite, and Microsoft SQL Server. Серед основних функціональних особливостей можна виділити наступні:

1) Sequelize створює абстрактний шар над безпосередньою реалізацією бази даних, тим самим забезпечуючи можливість швидкого переходу до іншої СУБД;

2) Sequelize має вбудоване проміжне програмне забезпечення, яке дає змогу мутувати помилки від бази даних та визначати методи їх обробки;

3) Sequelize забезпечує API на основі промісів, яке допомагає керувати асинхроними операціями;

4) має добре задокументовану систему міграцій, за допомогою якої зручно підтримувати базу даних в актуальному стані та легко її масштабувати.

Також Sequelize забезпечує розробника надійною транзакційною системою та дає можливість виконувати заплановані реплікації без прямого звернення до бази даних. Завдяки всім вищезгаданим перевагам, Sequelize займає важливу роль в екосистемі Node.js серед бібліотек, призначених для роботи з реляційними базами даних.

Для реалізації клієнтської частини програми використаний Vue.js. Vue.js – це прогресивний фреймворк для JavaScript, що використовується для створення веб-інтерфейсів і односторінкових застосунків. Vue.js також використовується як для настільних, так і для мобільних додатків за допомогою платформи Electron.

Головні переваги Vue.js:

1) невеликий розмір вихідного коду (~18кб);
2) віртуальний рендеринг і продуктивність DOM-дерева;
3) реактивна двостороння прив'язка даних;

3.2 Система логічної інтеграції даних

Розглянемо детально принцип роботи побудованої підсистеми логічної інтеграції даних. Головне її завдання – забезпечити синхронізацію між двома моделями баз даних – реляційною та графовою. Також необхідно зберегти

консистентність(узгодженість) даних між базами даних. Узгодженість має на увазі, що в будь-який момент часу отримувачі даних можуть бути впевнені, що працюють з коректною, технічно актуальною версією даних, і можуть розраховувати на неї при прийнятті рішень, виконанні запитів і тд. Один зі способів узгодити інформацію між собою в такого роду системах – використання транзакційних можливостей обраних СУБД. Так, наприклад, під час оновлення чи видалення інформації з двох баз даних одночасно, запит до реляційної бази обов'язково має здійснюватися в контексті транзакції. Так, у випадку невиконання запиту до графової моделі, транзакція не буде завершена і консистентність даних не буде порушена.

Розглянемо також принцип побудови даної підсистеми. Головна ідея полягає в тому, що всі сутності предметної області створюються і зберігаються у реляційній базі даних, а у графовій БД, у разі необхідності побудови зв'язків між об'єктами, будуть створені відповідні вузли. Важливо розуміти, що інформація про об'єкти не дублюється до графової моделі, у вузлах зберігається мінімальний набір атрибутів (див. рисунок 3.1), необхідний для синхронізації :

1) тип реляційної сутності об'єкта;
2) ідентифікатор об'єкта з реляційної бази даних;
3) найменування об'єкта для випадків, коли важливий сам факт наявності зв'язків, а не конкретної інформації про них (дозволяє уникнути зайвого запиту до реляційної бази даних).

Комбінація типу реляційної сутності та ідентифікатору дає можливість максимально однозначно визначити необхідні параметри для запиту до реляційної БД. Фактично, знаючи тип об'єкту та його ідентифікатор, задача знаходження відповідного запису у реляційній БД спрощується до задачі знаходження запису за ідентифікатором.

Окремо зауважимо, що використання ідентифікаторів об'єктів графової бази даних під час синхронізації неприпустимо. Все через те, що для оптимізації внутрішніх процесів Neo4j після видалення вузлів, або зв'язків

використовує їх ідентифікатори з новими доданими об’єктами. Окрім цього, ідентифікатори зв’язків мають одну і ту саму наскрізну нумерацію з вузлами, що унеможливлює використання даних ідентифікаторів для синхронізації з іншими базами даних. Актуальність роботи обговорена під час вісімнадцятої Всеукраїнської конференції студентів та молодих науковців, тези доповіді опубліковано [7].

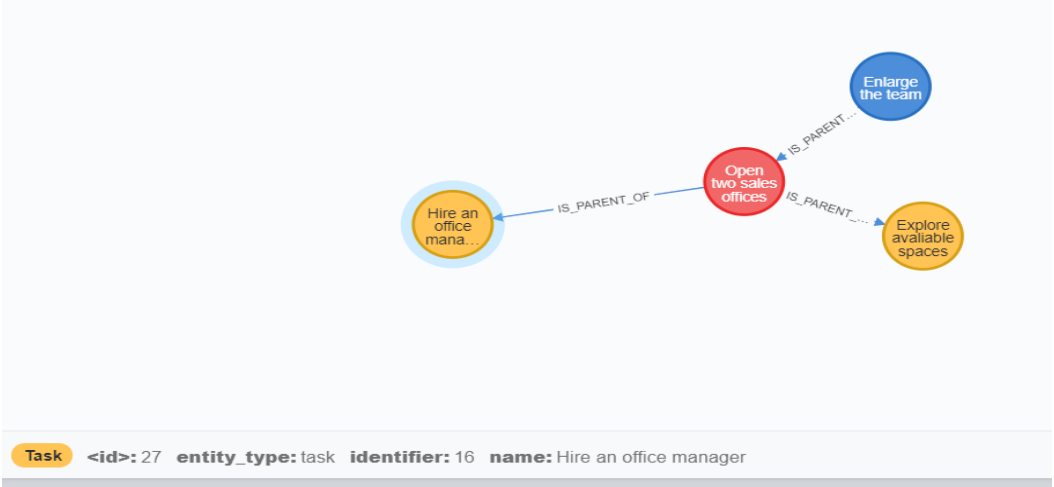

Рисунок 3.1 – Атрибути об’єктів у вузлі графу

3.3 Реалізація базових класів

У якості прикладу розглянемо структуру типового контролера побудованої системи. Контролер ObjectivesController є контролером для сутності цілей в предметній області, тому саме на його прикладі добре видно особливості реалізації даної інформаційної системи. Розглянемо детальніше структуру контролера на прикладі методів, які виконують CRUD-операції для даної сутності. Метод getObjectives, наведений у лістингу 3.1 по своїй суті є методом-обробником для HTTP запиту користувача на отримання переліку цілей.

В якості параметрів метода виступають об’єкти request та response, які надаються екосистемою Express.js та відповідають об’єктам запиту та відповіді на нього відповідно. Усі інструкції всередині метода загорнуті в

конструкцію try-catch, що дозволяє коректно обробляти помилки, логувати їх та сповіщати про них користувача.

Після отримання запиту від користувача, в контролері відбувається визначення параметрів, необхідних для звернення до бази даних. В даному методі визначається, чи присутній параметр, необхідний для фільтрації особистих цілей для певного користувача. У разі його наявності, він буде динамічно підставлений у параметри запиту до бази даних. Звернення до бази даних здійснюється асинхронно, про що свідчить ключове слово «await» перед викликом. Основна перевага у використанні асинхронних функцій – відсутність блокування потоку одним запитом, тобто поки сервер застосунку чекає на відповідь від серверу бази даних, він може паралельно приймати і обробляти інші запити користувачів. Після отримання відповіді від бази даних, до результату запиту додається опис атрибутів отриманих об’єктів для виводу в інтерфейсі користувача. Після цього відбувається відповідь на запит клієнта та HTTP-з’єднання закривається. У разі виникнення виключень під час опрацювання запиту до клієнта повернеться відповідь на запит з відповідним HTTP статус-кодом.

```
exports.getObjectives = async function (request, response) {
    try {
        const instance = global.currentUser;
        const filter = request.query.filter;
        const filterCondition = filter === "personal"?
{user_id:request.user.id} : {}
        const res = await Objective(instance).findAll({
            order: [['id', 'ASC']],
            include:[
                {
                    model: User(instance),
                    attributes: ['id','first_name','last_name'],
                },
            ],
            attributes: ["id", "objective_name", "start_date",
"end_date", "description", ['user_id', "assignee_id"]],
            where:filterCondition
        });
```

```
        var headers = [
            {
                value: 'objective_name',
                text: 'Name',
                sortable: true,
                align: 'start',
                type: "text"
            },
            {
                value: 'description',
                text: 'Description',
                sortable: false,
                align: 'start',
                type: "text"
            },
            {
                value: 'start_date',
                text: 'Start date',
                sortable: true,
                align: 'start',
                type: "date"
            },
            {
                value: 'end_date',
                text: 'End date',
                sortable: true,
                align: 'start',
                type: "date"
            },
]
response.send({
  model: res.map(item => {
       return {...item.dataValues}
         }),

            form_parameters: {
                form_title: 'Objectives',
               headers
            }
        });
    } catch (e) {
        console.log(e)
    }
};
```

Лістинг 3.1 – Метод getObjectives

Далі розглянемо метод delete (лістинг 3.2), який відповідає за видалення цілей. Його особливістю є наявність звернення одразу до двох баз даних під час процесу видалення запису. За аналогією з попереднім розглянутим методом, цей також приймає два аргументи, та загорнутий в конструкцію try/catch. Для видалення відповідного об’єкту вузла з графової бази даних використовується підвиклик допоміжного метода deleteCorrespondingNode (див. лістинг 3.3). Цей метод має аналогічну структуру побудови, відрізняється він лише зверненням до іншого драйверу бази даних. Також, для збереження консистентності даних запит на видалення запису з реляційної бази даних знаходиться всередині створеної транзакції. У випадку помилки під час видалення вузла з графової бази, транзакція не буде завершена і клієнт буде сповіщений, що операція пройшла невдало.

```
exports.delete = async function (request, response) {
    try {
        var t = await instance.transaction();
        var id = request.body.id;
        var instance = global.currentUser;
        const res = await Objective(instance).destroy({
            where: {id: id}
        })
 const deletedNode = await
deleteCorrespondingNode(id,'Objective');
t.commit();
        response.send({
            res: res,
            message: 'Successfully deleted'
        })

    } catch (error) {
          t.rollback();
        response.status(500).send({error: error, message:
error.message})
    }
};
```

Лістинг 3.2 – Метод delete

```
      exports.deleteCorrespondingNode = async function (node_id,
entity_type) {
         let session;
         try {
             session = ne04j_driver.session();
             return await session.run(`
                match (n:${entity_type} {identifier:${node_id}})
                detach delete n
             `);
         } catch (error) {
             return new Error('Unable to delete corresponding
node');
         } finally {
             await session.close();
         }
     }
```

Лістинг 3.3 – Допоміжний метод deleteCorrespondingNode

Також окремо варто розглянути контролер, який є складовою частиною підсистеми логічної інтеграції даних. Його основне завдання – забезпечити можливість маніпулювати зв'язками між вузлами у графовій базі даних, тим самим будуючи зв’язки між відповідними об’єктами, створеними у реляційній базі даних.

Для прикладу, розглянемо метод linkObjects (див. лістинг 3.4), за допомогою якого можна створювати нові зв’язки між вузлами у графовій базі даних. Всередині об’єкта request в даному запиті присутні параметри для формування звернення до бази даних, а саме ідентифікатори об’єктів, між якими будується зв’язок і обраний користувачем тип зв’язку. Після визначення параметрів відкривається нове з’єднання з графовою базою даних, в рамках якого буде виконаний запит на створення нового зв’язку.

```
exports.linkObjects = async function (request, response) {
   let session;
     try {
        session = ne04j_driver.session();
        const instance = global.currentUser;
        const {object, relation, rel_obj} = request.body;
        const res = await session.run(`
```

```
       merge ( a:${capitalize(rel_obj.relation_from)},
{identifier:${object.id},
name:'${object[convertEntityNameToModel(rel_obj.relation_from).n
ame_key]}', entity_type:'${rel_obj.relation_from}'})
       with a
       merge
(b:${capitalize(rel_obj.relation_to)}{identifier:${relation.rela
tion_target.id},name:'${relation.relation_target.name}',
entity_type:'${rel_obj.relation_to}'})
       with a,b
       merge (a) -
[r:${relation.relation_type.relation_type_code}]->(b)
      return a,r,b
      `);
            response.send({
                items: res
            });
         } catch (e) {
            response.status(400).send({error: "error", message:
'Unable to link objects'})
         } finally {
            await session.close();
     };
```

Лістинг 3.4 – Метод linkObjects

Аналогічним чином реалізований і метод видалення існуючого зв'язку між вузлами (див. лістинг 3.5).

```
exports.deleteRelation = async function (request, response) {
         let session;
         try {
            session = ne04j_driver.session();
            const {from, to, from_id, to_id, relation_code} =
request.body;
            const {records} = await session.run(`

match(n:${capitalize(from)}{identifier:${from_id}})-
[relation:${relation_code}]-
>(m:${capitalize(to)}{identifier:${to_id}})
                delete  relation
            `);
            response.send({res: true});
         } catch (error) {
```

```
            response.status(500).send({error: error, message:
error.message})
        } finally {
            await session.close();
        }
    }
```

Лістинг 3.5 – Метод deleteRelation

3.4 Створення бази даних

У цьому розділі наведені приклади створення необхідних для роботи з предметною областю елементів реляційної бази даних.

Розглянемо процес створення таблиці objectives. Для її створення використовується запит наступного виду:

```
create table objectives
(
id    serial  primary key,
objective_name varchar(255) not null,
start_date     date not null,
end_date       date not null check (end_date > start_date),
description    text,
user_id        integer
                 references users(id)
                 on update cascade
                 on delete cascade
team_id        integer
                 references teams
                 on update cascade
                 on delete cascade
    );
```

Лістинг 3.6 – Створення таблиці objectives

Створення таблиці виконується за допомогою команди create table, далі вказується найменування таблиці. Після найменування поля вказується тип даних обраної СУБД (PostgreSQL) та обмеження цілісності. Поле id є первинним ключем там має тип serial (один з вбудованих типів PostgreSQL,

після створення таблиці створюється послідовність з автоінкрементом, значення з якої будуть вноситися до таблиці у якості первинних ключів нових записів).

Конструкція `check(end_date > start_date)` при визначенні поля end_date накладає додаткову умову на його значення, в даному випадку - дата кінцевої дати має бути більше дати початку.

Поле user_id є зовнішнім ключем для зв'язку з таблицею users. Конструкція `on delete cascade` означає, що при зміні первинного ключа запису з таблиці users даний зовнішній ключ здобуде нове значення, рівне новому значенню первинного ключа запису з таблиці users. Конструкція `on update cascade` означає, що при видаленні запису з таблиці users всі записи, які на нього посилались також будуть видалені.

Всі інші таблиці бази даних, створені з використанням відповідних запитів CREATE TABLE, наведених в додатку Б.

3.5 Запити до бази даних

Платформа Express, за допомогою якої реалізовано застосунок, пропонує вбудовану ORM Sequelize, яка пропонує широкий набір інструментів для роботи з базою даних. Розглянемо запити до реляційної бази даних виконані за допомогою структур ORM-системи Sequelize.

Так, для вирішення задач користувачів (наприклад, задачі С8, М9, А11), пов'язаних з переглядом цілей, необхідний запит наступного виду (лістинг 3.7):

```
const res = await Objective(instance).findAll({
                order: [['id', 'ASC']],
                include:[
                    {
                        model: User(instance),
```

```
                        attributes: ['id','first_name',
'last_name'],
                    },
                     {
                        model: Team(instance),
                        attributes: ['id','team_name'],
                    },

                ],
                attributes: ["id", "objective_name",
"start_date", "end_date", "description", ['user_id',
"assignee_id"]],
                where:filterCondition
            });
```

Лістинг 3.7 – Запит на отримання цілей, виконаний за допомогою Sequelize

Даний запит, реалізований засобами ORM, буде транслюється у SQL-запит наступного виду (лістинг 3.8):

```
SELECT "objectives"."id",
       "objectives"."objective_name",
       "objectives"."start_date",
       "objectives"."end_date",
       "objectives"."description",
       "objectives"."user_id" AS "assignee_id",
       "user"."id"            AS "user.id",
       "user"."first_name"    AS "user.first_name",
       "user"."last_name"     AS "user.last_name",
       "team"."id"            AS "team.id",
       "team"."team_name"     AS "team.team_name"
FROM "objectives"
         LEFT  JOIN "users" AS "user" ON "objectives"."user_id"
= "user"."id"
         LEFT  JOIN "teams" AS "team" ON "objectives"."team_id"
= "team"."id"
ORDER BY "objectives"."id" ASC;
```

Лістинг 3.8 – SQL - запит для отримання цілей

Задачі користувачів, в результаті яких не тільки створюються об’єкти предметної області, а і створюються зв’язки між ними (наприклад, задачі А3, А5, М1), потребують як мінімум три запити до баз даних - до реляційної, та до графової. Розглянемо випадок застосування такого запиту на прикладі створення ключового результату, та створення зв’язку до існуючої цілі.

Спочатку виконується типовий insert-запит до реляційної бази даних (лістинг 3.9):

```
INSERT INTO "key_results" ("id", "key_result_name", "target",
"current_value")
     VALUES (DEFAULT, 'Key result name', 100, 12)
     RETURNING *;
```

Лістинг 3.9 – SQL - запит на створення ключового результату

Далі користувач має змогу вибрати тип зв'язку, який буде створено між об’єктами. Для вибору списку можливих зв’язків між двома сутностями необхідний запит наступного виду (лістинг 3.10):

```
     select relation_type_name,
            relation_type_code,
            relation_from,
            relation_to
     from relations
              join relation_types rt on
relations.relation_type_id = rt.id
              join relation_objects ro on ro.id =
relations.relation_object_id
     where ro.relation_from = 'key_result';
```

Лістинг 3.10 – SQL - запит на вибірку доступних зв’язків за участю ключових результатів

Після вибору типа зв’язку, який буде створений, необхідно знайти об’єкт, до якого буде будуватися зв’язок. Параметрами запиту в даному випадку будуть виступати тип кінцевого об’єкта у визначеному зв’язку, та введена користувачем назва (або її частина) об’єкта (див. лістинг 3.11).

```
SELECT "id", "objective_name" AS "name"
FROM "objectives" AS "objectives"
WHERE "objectives"."objective_name" ILIKE '%Enlarge%'
ORDER BY "objectives"."id" ASC;
```

Лістинг 3.11 – SQL - запит на вибірку цілі за найменуванням

Після отримання усіх необхідних даних про об’єкти та зв’язок між ними, виконується запит на створення всіх необхідних вузлів та ребер у графовій базі даних (див. лістинг 3.12). Запит до графової бази виконаний за допомогою мови запитів Cypher. Варто зазначити, що у наведеному запиті для створення об’єктів використаний метод «м’якого створення» (реалізація за допомогою команди merge). Його суть полягає у зберіганні вузла, якщо він вже існує і додаванні зв’язку до нього без перезапису вузла. Такий підхід дозволяє не порушити вже створені зв’язки між вузлами графа.

```
     merge(
          a:${capitalize(rel_obj.relation_from)}
          {identifier:${object.id},
          name
:'${object[convertEntityNameToModel(rel_obj.relation_from).name_
key]}',
          entity_type:'${rel_obj.relation_from}'})
     with a
     merge (
          b:${capitalize(rel_obj.relation_to)}
               {identifier:${relation.relation_target.id},
               name:'${relation.relation_target.name}',
               entity_type:'${rel_obj.relation_to}'
          })
          with a, b
              merge (a) -
[r:${relation.relation_type.relation_type_code}]->(b)
     return a, r, b
```

Лістинг 3.12 – Cypher - запит на створення зв’язку між ціллю та ключовим результатом

3.6 Безпека інформаційної системи

З боку баз даних безпека забезпечується поділом користувачів на ролі і наданням їм різних привілеїв на доступи до об'єктів. В рамках даної інформаційної системи створено три ролі:

1) співробітник;
2) менеджер;
3) адміністратор.

Запити на створення ролей докладно описані в додатку В.

Процес визначення ролі відбувається під час авторизації користувача. Після введення логіна і пароля відбувається пошук по таблиці користувачів, в разі успіху встановлюється підключення до бази даних під роллю, що належить користувачеві. Варто зауважити, що пароль користувача в базі даних зберігається в зашифрованому вигляді, для хешування використовувалася крипто-бібліотека bcrypt. При збігу введеного логіна з логіном в базі даних, введений пароль хешується і порівнюється з паролем з бази даних. У разі збігу користувач переходить до наступного етапу аутентифікації – отримання JWT- токена [8] .

JWT-токен створюється після входу користувача в систему і зберігає у собі хешовану інформацію про користувача. Сервер програми, що працює з JWT-токеном, при кожному запиті користувача звіряє токен, щоб переконатися, що саме зареєстрований користувач здійснює запит. Найчастіше токен зберігається в LocalStorage браузера і передається на сервер в якості заголовка HTTP-запиту. Також можлива передача токену з HTTP-only куками. Даний тип токенов є тимчасовим і має потребу в заміні через невеликий проміжок часу (~ 30 хвилин). При виході клієнта з облікового запису токен скидається і при наступній авторизації користувачеві буде згенеровано новий токен.

3.7 Інструкція користувача

Перед входом до застосунку, користувачу необхідно створити обліковий запис (рис. 3.2), або увійти до нього, якщо він вже зареєстрований (рис 3.3).

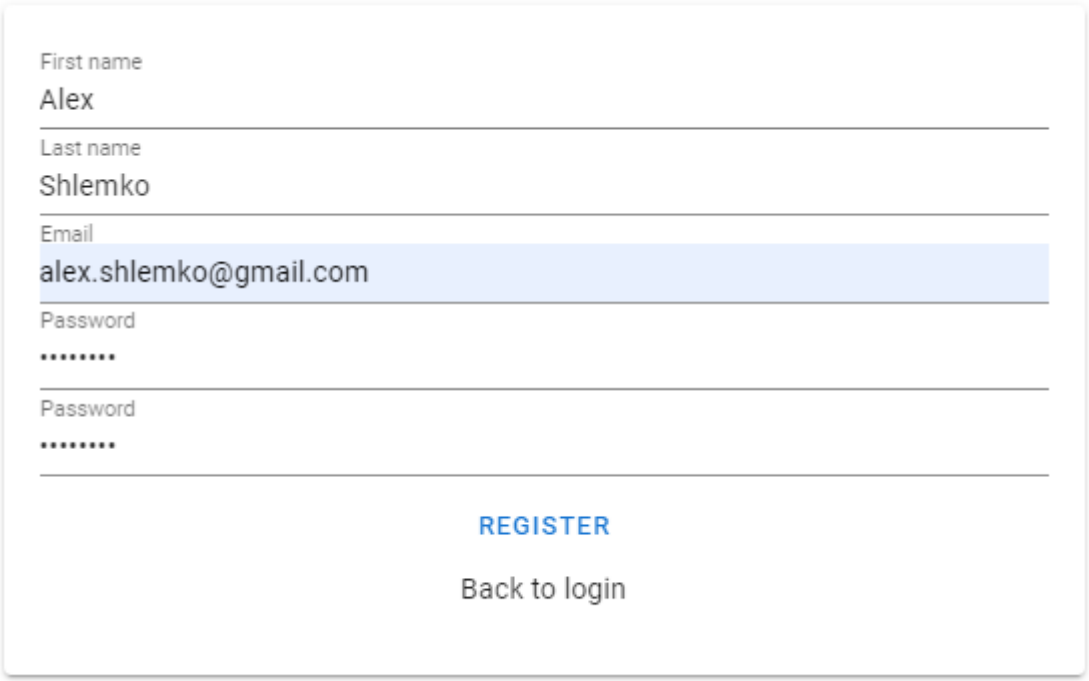

Рисунок 3.2 – Форма реєстрації

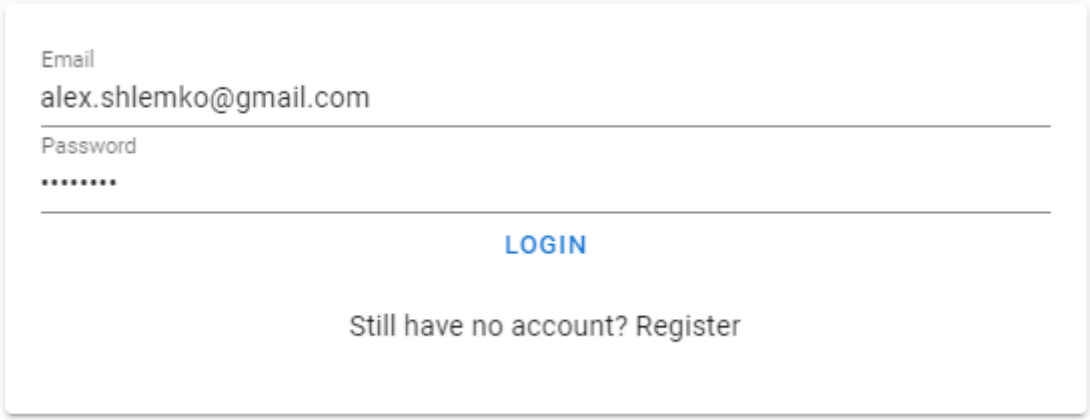

Рисунок 3.3 – Форма авторизації

Після входу до застосунку, користувачу стають доступні домашня сторінка, та навігаційна панель у верхній частині екрану (рис. 3.4). На домашній сторінці користувач може обрати проект і перейти до канбан-дошки для перегляду поточного стану задач, які є підпорядковані обраному проекту (рис 3.5).

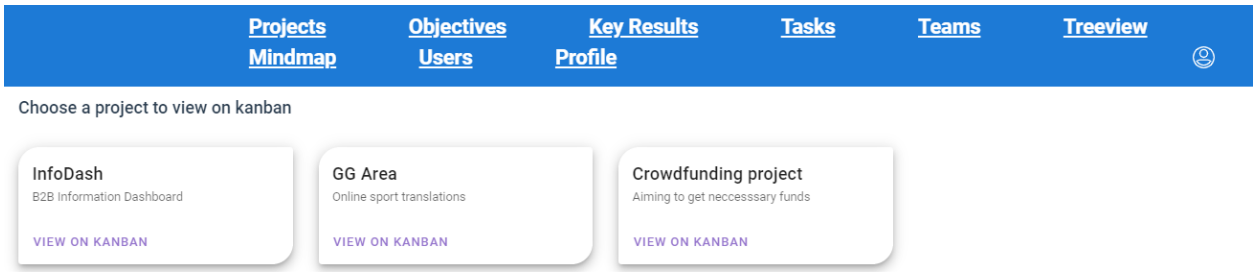

Рисунок 3.4 – Домашня сторінка та навігаційна панель

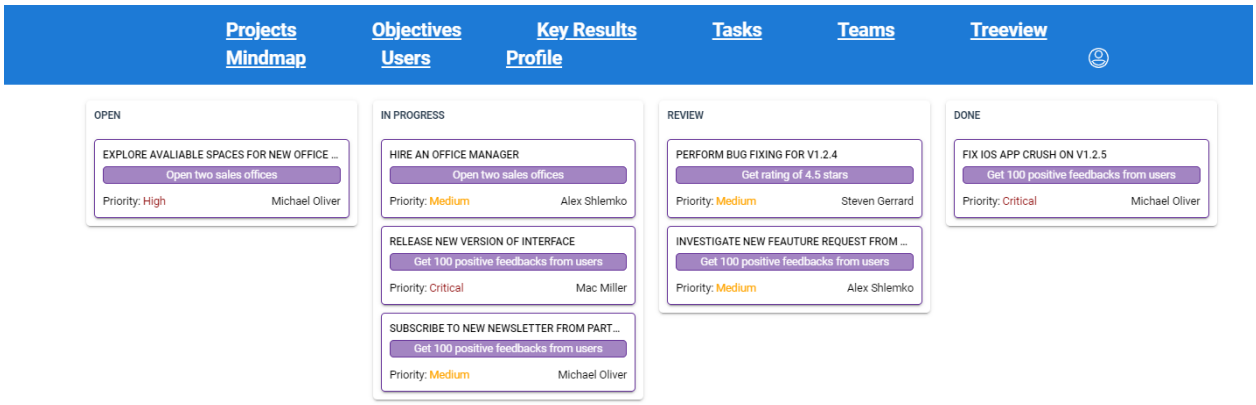

Рисунок 3.5 – Канбан-дошка за обраним проектом

Головне завдання канбан-дошки – дати користувачу змогу відстежувати поточний стан задач та при необхідності змінювати їх статус. Окрім цього, з канбан дошки є можливість перейти одразу на сторінку перегляду/редагування задачі (рис 3.6). Варто зазначити, що усі сторінки перегляду/редагування об’єктів схожі між собою, відмінності полягають лише в наборі специфічних полів.

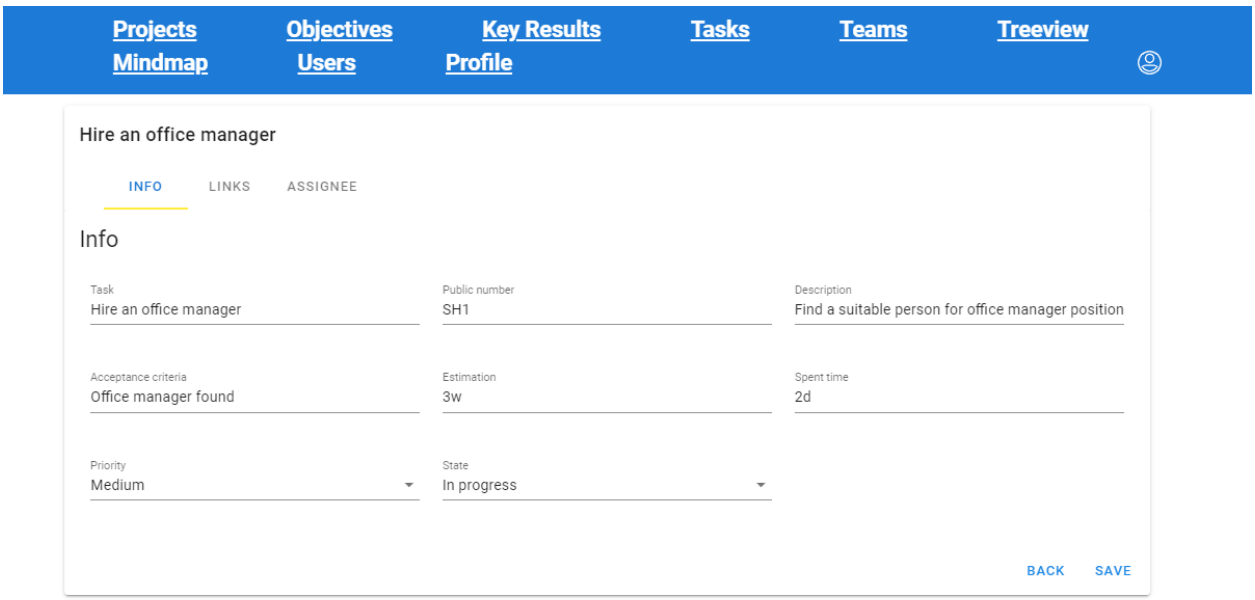

Рисунок 3.6 – Сторінка перегляду/редагування задачі

Далі розглянемо вкладку «Links» на сторінці редагування об’єкта (рис. 3.7), за допомогою якої можливо додавати зв’язки до об’єкта. На даній сторінці у користувача є можливість подивитись існуючі зв’язки з поточним об’єктом, а також додати нові.

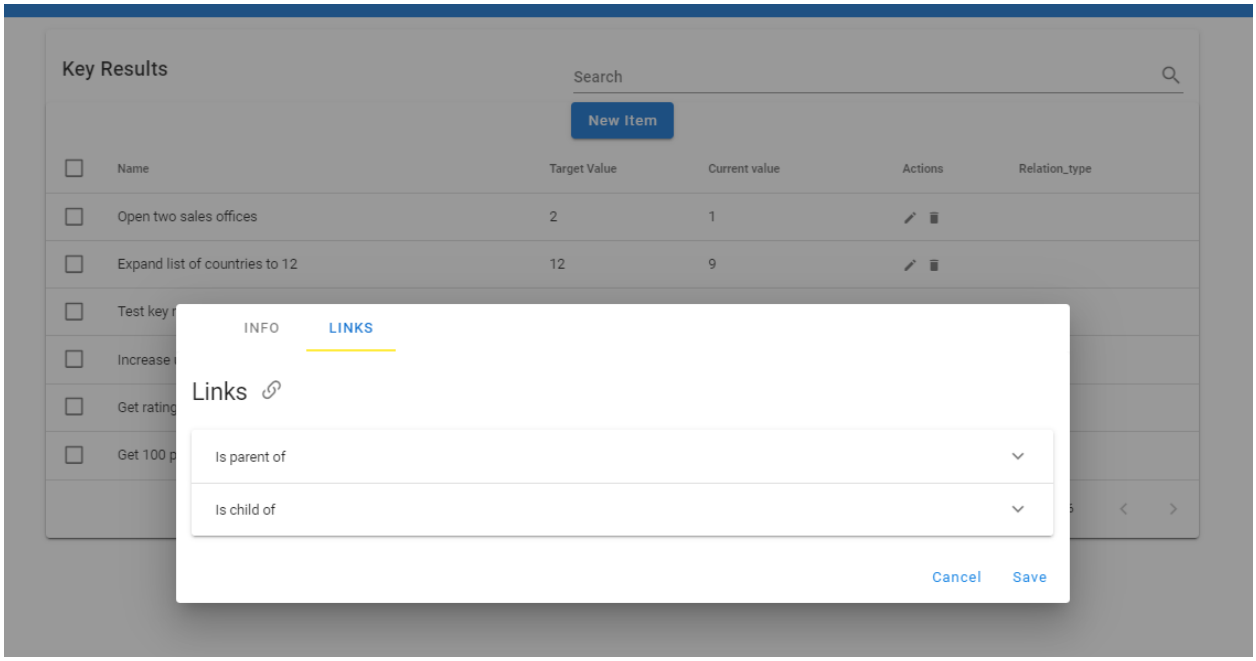

Рисунок 3.7 – Сторінка перегляду/редагування задачі

Натиснувши іконку поряд з заголовком, можна відкрити випадаючий список для вибору типу зв’язку, який буде створений (рис 3.8).

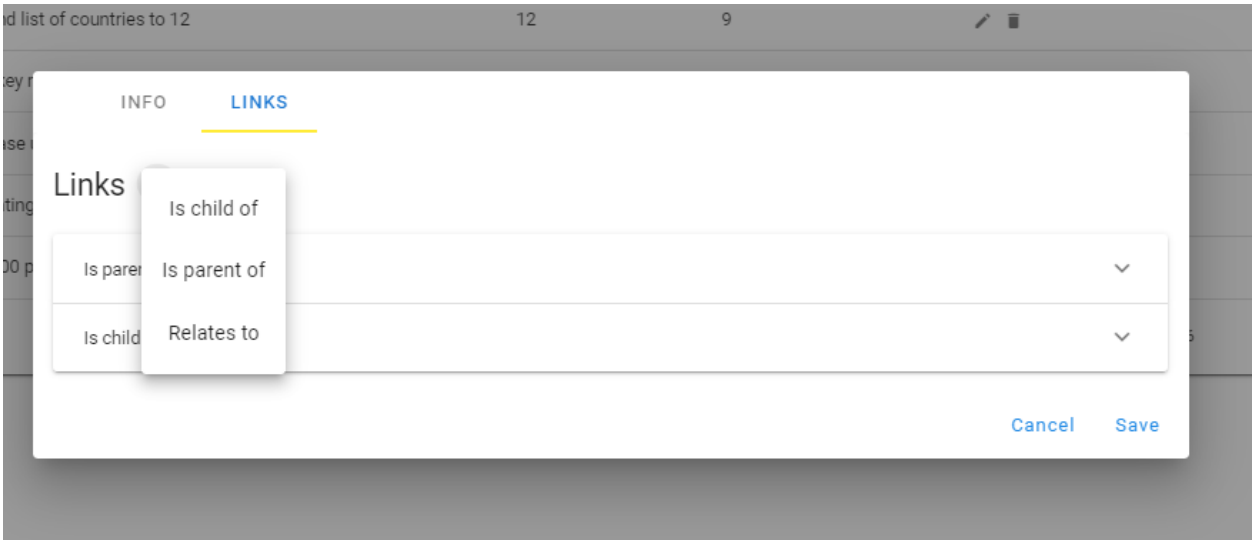

Рисунок 3.8 – Випадаючий список для вибору типу зв’язку

Після вибору типу зв’язка, з’являється модальне вікно з текстовим полем для пошуку кінцевого об’єкта зв’язка за його назвою (рис 3.9).

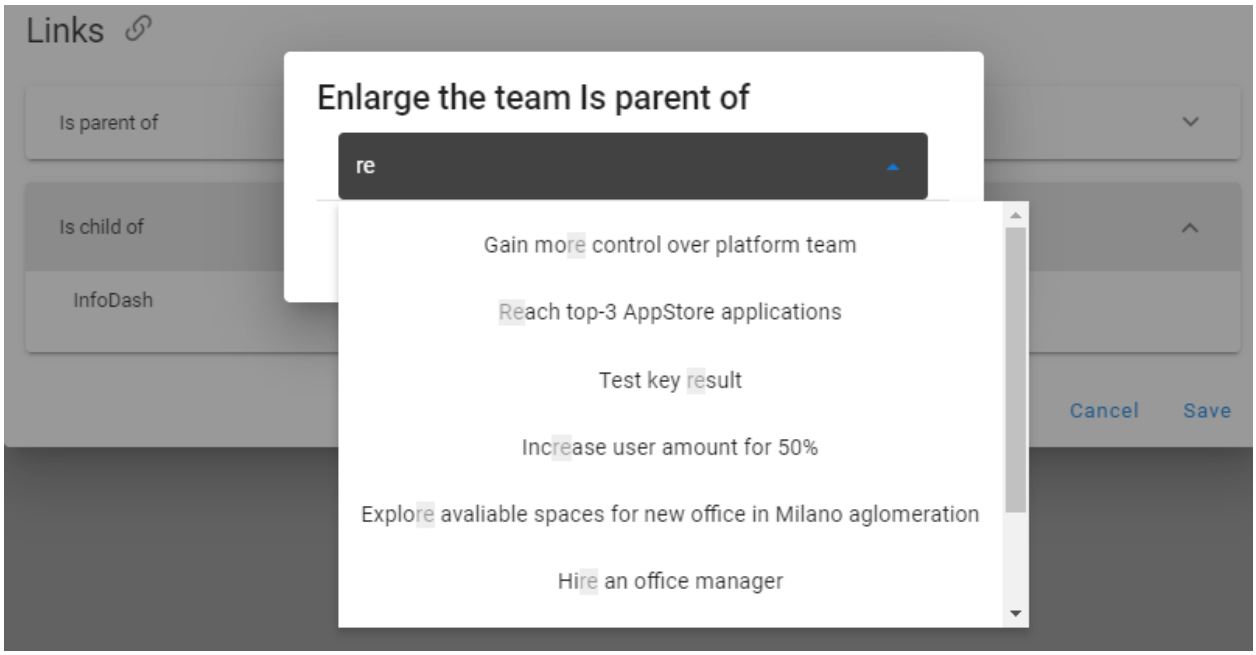

Рисунок 3.9 – Модальне вікно для пошуку кінцевого об’єкта зв’язка у зв’язку

Після вибору знайденого об’єкта, новий зв’язок з’являється у списку побудованих зв’язків (рис 3.10).

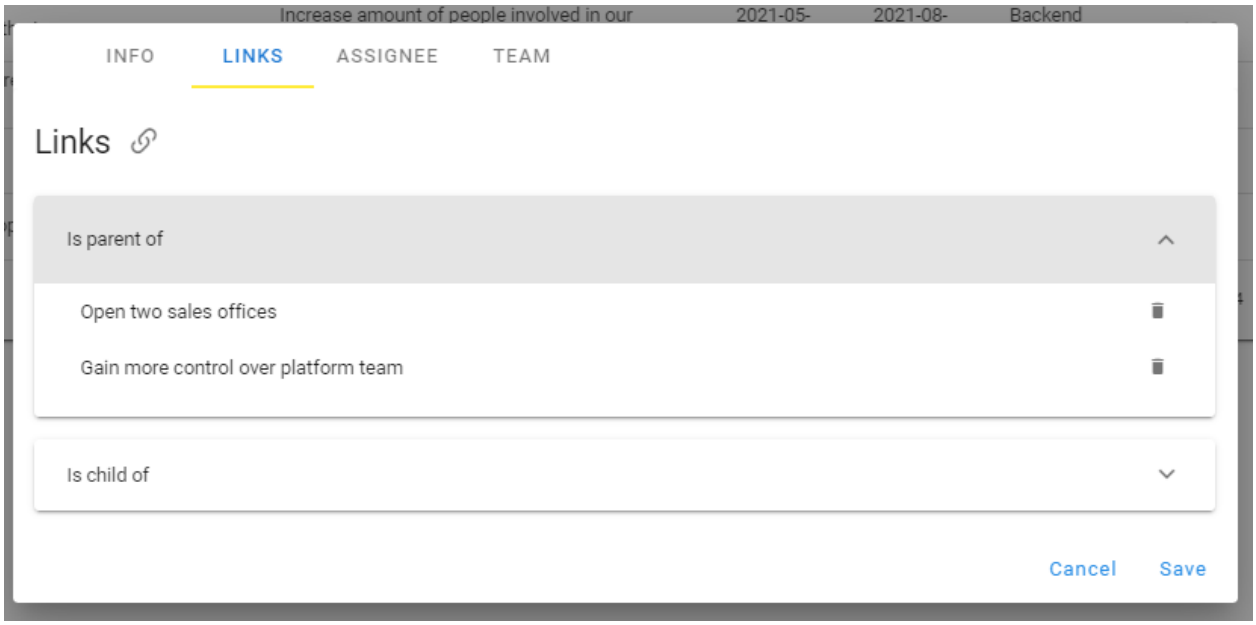

Рисунок 3.10 – Результат додавання зв’язка

Окремо варто розглянути сторінку з відображенням структури проекту. На даній сторінці можна побачити ієрархію об’єктів в межах проекту/цілі, базуючись на зв’язках типу «IS_PARENT_OF» (рис. 3.11).

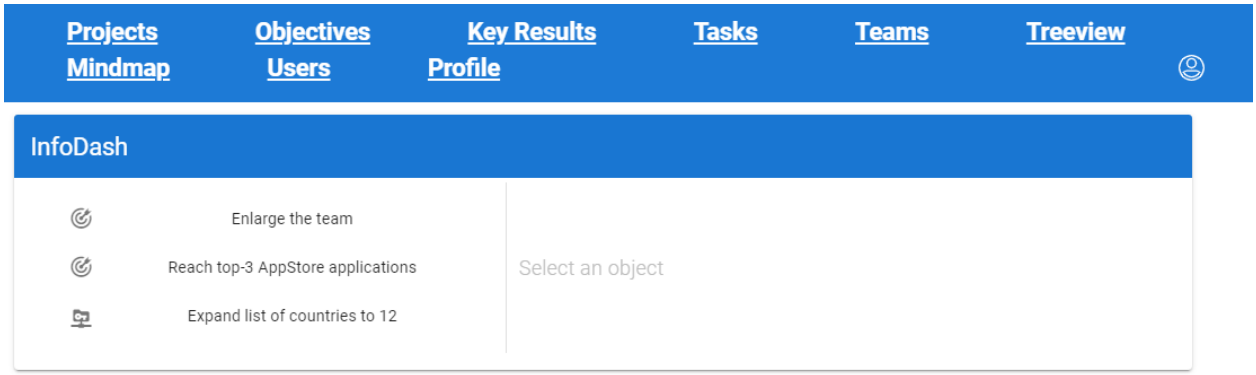

Рисунок 3.11 – Ієрархічна структура проекту

При натисканні на елементи, які являють собою підрівні проекту/цілі, можна розгорнути структуру до елемента найнижчого рівня (рис. 3.12). Як можна помітити у правій частині блоку виводиться загальна інформація про

показаний об’єкт, також є можливість перейти на сторінку вибраного об’єкту, натиснувши кнопку «Go to page».

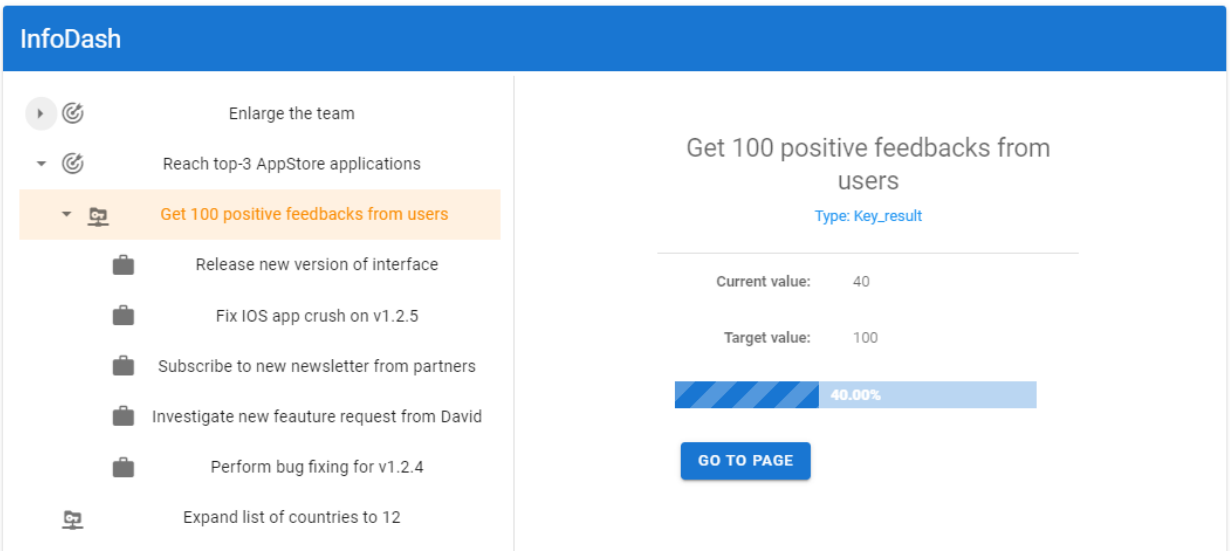

Рисунок 3.12 – Розгорнута структура проекту

Також розглянемо можливість побудови діаграм структури проекту, так званий «Mindmap». Для цього користувачу достатньо обрати у навігаційній панелі відповідне посилання, та обрати проект, діаграму якого буде побудовано (рис 3.13).

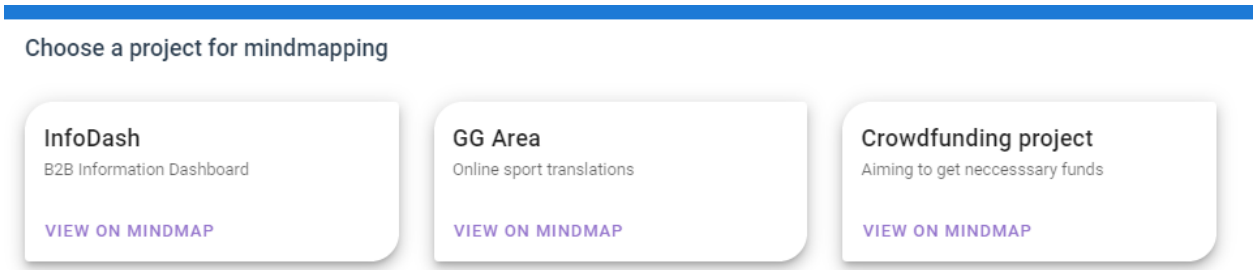

Рисунок 3.13 – Вибір проекту для побудови «Mindmap»

Після вибору проекту, на новій сторінці буде відображена структурна діаграма проекту (рис. 3.14). Варто зауважити, що при натисканні на вузол можна перейти до сторінки редагування відповідного об’єкта. Дана функція

особливо корисна під час управління великими проектами, так як завдяки графічному відображенню можна скорегувати структуру проекту.

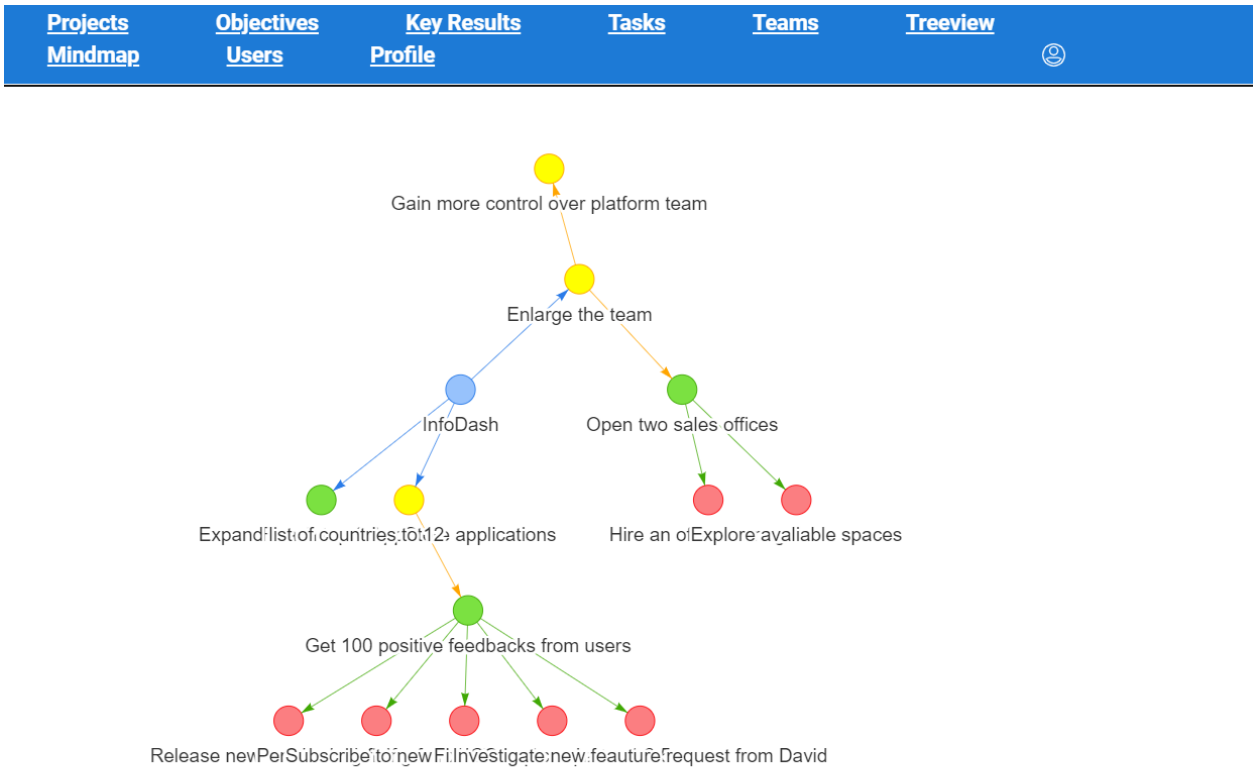

Рисунок 3.14 – «Mindmap» - діаграма

ВИСНОВОК

В ході виконання дипломної роботи створено робочу систему управління проектами з вбудованою підсистемою логічної інтеграції та очищення даних. Для цього проведений огляд існуючих систем управління проектами та проаналізовані їх основні недоліки. Виходячи з проблем систем, представлених на ринку, зроблений висновок про доцільність виконання даного проекту з метою вирішення виявлених проблем. При створенні системи використані принципи і методи структурування предметної області на основі онтологічного підходу. Так як стандартні методи побудови багатовимірних зв'язків між сутностями та створення складних багаторівневих ієрархій виявилися неефективними, вирішено взяти до використання графову модель даних для спрощення процесу побудови зв’язків між сутностями предметної області. Шляхом аналізу особливостей побудованої онтології була виведена модель інформаційної системи, у якій основні сутності предметної області представлені у вигляді реляційної моделі, а зв’язки між ними – у вигляді графової моделі даних.

Розроблена інформаційна система придатна для використання в управлінні проектами у компаніях, які потребують автоматизації управлінських процесів та прив’язки конкретних задач до бізнес-цілей. Актуальність роботи обговорена під час вісімнадцятої Всеукраїнської конференції студентів та молодих науковців, тези доповіді опубліковано.

Подальший розвиток даного проекту можливий в декількох напрямках. По-перше, можливе розширення функціоналу, пов’язане з впровадженням нових засобів планування та відстеження робіт (діаграми Ганта, система бюджетування, тощо). По-друге, корисною функцією є вбудований в систему документообіг, який дозволить спростити процеси, пов’язані, наприклад, з бюджетуванням. Також ефективність даної системи можна підвищити за допомогою інтеграцій зі сторонніми сервісами для командної роботи (системи контролю версій, месенджери, тощо).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Doerr J. Measure What Matters. / L.: P.Publishing Group, 2018. - С. 28-46.
2. Thomson, Thomas M. «Management by objectives». / L.: Butterworth-Heinemann, 2007. - С 4-17.
3. Тельнов Ю.Ф. Принципы и методы семантического структурирования информационно-образовательного пространства на основе реализации онтологического подхода // Вестник УМО. Экономика, статистика, информатика, 2014, № 1. – С. 187–191.
4. Тельнов Ю.Ф., Данилов А.В., Казаков В.А. Программная реализация информационно-образовательного пространства на основе многоагентной технологии и онтологического подхода. 2015; с.73-82.
5. Малахов Е. В. Отображение специфики сложноструктурированных предметных областей в системах организационного управления / Е. В. Малахов // Електротехнічні та комп’ютерні системи. - 2010. - № 1. - С. 122-127.
6. Dr. Axel Rauschmayer. Exploring ES6. / O.: Leanpub, 2018. - С. 104 - 136.
7. Шлемко О. В., Малахов Є. В. Система логічної інтеграції та очищення даних з неоднорідних джерел системи управління проектами/ Інформатика, інформаційні системи та технології: тези доповідей вісімнадцятої Всеукраїнської конференції студентів і молодих науковців. Одеса, 23 квітня 2021 р. – Одеса, 2021.– с.196-198
8. Matthias Biehl. OpenID Connect and JWT. / NY.: API-University Press, -2019, - С. 12 – 45.

ДОДАТОК А

Задачі користувачів інформаційної системи

Таблиця А.1 – Задачі користувачів інформаційної системи

№	Задача	Вхідні дані	Вихідні дані
	Адміністратор		
А1	Створення / редагування цілей на всіх рівнях	Найменування цілі, опис цілі, терміни, об'єкт призначення	Найменування цілі, опис цілі, терміни, об'єкт призначення
А2	Видалення цілей на всіх рівнях	Ціль, найменування цілі	-
А3	Створення / редагування ключових результатів до певної цілі	Найменування КР, опис КР, виміри	Найменування КР, опис КР, виміри
А4	Видалення ключових результатів до певної цілі	Найменування КР, опис КР	-
А5	Створення / редагування / видалення задач до ключових результатів	Найменування задачі, зміст задачі, терміни, відповідальний співробітник, відповідний КР (опціонально)	Найменування задачі, зміст задачі, терміни, відповідальний співробітник, відповідний КР (опціонально)
А6	Видалення задач до ключових результатів	Найменування задачі, опис задачі	-
А7	Створення / редагування команд	Найменування команди, список учасників	Найменування команди, список учасників
А8	Видалення команд	Найменування команди, список учасників	-
А9	Додавання співробітників до команди	Ім'я співробітника	Ім'я співробітника
А10	Видалення співробітників всередині команди	Ім'я співробітника	-
А11	Перегляд цілей	-	Найменування цілі, опис цілі, терміни, об'єкт призначення

Продовження таблиці А.1

№	Задача	Вхідні дані	Вихідні дані
А12	Перегляд / редагування ключових результатів до певної цілі	Ціль	Найменування ключових результатів, опис, одиниці виміру
А13	Перегляд / редагування задач до ключових результатів	Ключовий результат	Найменування задачі, опис, виконавець
А14	Перегляд команд	-	Найменування команди, учасники
А15	Редагування користувачів	Ім'я, логін, приналежність команді, роль	Ім'я, логін, приналежність команді, роль
А16	Видалення користувачів	Ім'я користувача, логін	-
А17	Перегляд профілю користувача	Ім'я користувача	Ім'я користувача, логін, персональна інформація
	Менеджер		
М1	Створення / редагування цілей на всіх рівнях (крім рівня компанії)	Найменування цілі, опис цілі, терміни, об'єкт призначення	Найменування цілі, опис цілі, терміни, об'єкт призначення
М2	Видалення цілей на всіх рівнях (крім рівня компанії)	Ціль, найменування цілі	-
М3	Створення / редагування ключових результатів до певної цілі (крім рівня компанії)	Найменування КР, опис КР, виміри	Найменування КР, опис КР, виміри
М4	Видалення ключових результатів до певної цілі (крім рівня компанії)	Найменування КР, опис КР	-
М5	Створення / редагування / видалення задач до ключових результатів	Найменування задачі, зміст задачі, терміни, відповідальний співробітник, відповідний КР (опціонально)	Найменування задачі, зміст задачі, терміни, відповідальний співробітник, відповідний КР (опціонально)

Продовження таблиці А.1

№	Задача	Вхідні дані	Вихідні дані
М6	Видалення задач до ключових результатів	Найменування задачі, опис задачі	-
М7	Створення / редагування команд	Найменування команди, список учасників	Найменування команди, список учасників
М8	Додавання співробітників до команди	Ім'я співробітника	Ім'я співробітника
М9	Перегляд цілей	-	Найменування цілі, опис цілі, терміни, об'єкт призначення
М10	Перегляд / редагування ключових результатів до певної цілі	Ціль	Найменування ключових результатів, опис, одиниці виміру
М11	Перегляд / редагування задач до ключових результатів	Ключовий результат	Найменування задачі, опис, виконавець
М12	Перегляд команд	-	Найменування команди, учасники
М13	Редагування користувачів	Ім'я, логін, приналежність команді, роль	Ім'я, логін, приналежність команді, роль
М14	Видалення користувачів	Ім'я користувача, логін	-
М15	Перегляд профілю користувача	Ім'я користувача	Ім'я користувача, логін, персональна інформація
	Співробітник		
С1	Створення / редагування особистих цілей	Найменування цілі, опис цілі, терміни, об'єкт призначення	Найменування цілі, опис цілі, терміни, об'єкт призначення
С2	Видалення особистих цілей	Ціль, найменування цілі	-
С3	Створення / редагування ключових результатів до особистих цілей	Найменування КР, опис КР, виміри	Найменування КР, опис КР, виміри
С4	Видалення ключових результатів до особистих цілей	Найменування КР, опис КР	-

Продовження таблиці А.1

№	Задача	Вхідні дані	Вихідні дані
С5	Створення / редагування / видалення задач до особистих ключових результатів	Найменування задачі, зміст задачі, терміни, відповідальний співробітник, відповідний КР	Найменування задачі, зміст задачі, терміни, відповідальний співробітник
С6	Видалення задач до особистих ключових результатів	Найменування задачі, опис задачі	Видалення задач до ключових результатів
С7	Перегляд цілей	-	Найменування цілі, опис цілі, терміни, об'єкт призначення
С8	Перегляд / редагування задач до ключових результатів	Ключовий результат	Найменування задачі, опис, виконавець
С9	Перегляд команд	-	Найменування команди, учасники
С10	Перегляд профілю користувача	Ім'я користувача	Ім'я користувача, логін, персональна інформація
С11	Редагування особистих завдань	Найменування задачі, опис задачі	Найменування задачі, опис задачі

ДОДАТОК Б

Запити для створення реляційної бази даних

```
create table key_results
(
    id serial primary key,
    key_result_name varchar(255) not null,
    target numeric not null,
    current_value   numeric   not null,
 );

create table projects
(
    id serial primary key,
    project_name varchar(255) not null,
    project_public_code varchar(255) not null,
    project_description varchar(255)
);

create table tasks
(
    id serial primary key,
    task_name  varchar(255)  not null,
    task_description         varchar(255),
    task_notes               varchar(255),
    task_acceptance_criteria varchar(255),
    task_estimation          varchar(255),
    task_spent_time          varchar(255),
    task_public_number       varchar(255),
    task_priority_id integer not null
                            references task_priorities(id)
                                       on update cascade
                                       on delete cascade,
    task_state_id integer not null references task_states(id)
                                       on update cascade
                                       on delete cascade,
    assignee_id   integer references users(id)
                                       on update cascade
                                       on delete cascade
);
```

```
create table task_states
(
    id serial primary key,
    state_name varchar(255)  not null,
    state_code varchar(255)  not null
);

create table task_priorities
    id serial primary key,
    priority_name varchar(255)  not null,
    priority_code varchar(255)  not null
);

create table relation_objects
    id serial primary key,
    relation_from varchar(255)  not null,
    relation_to varchar(255)  not null
);

create table relation_types
    id serial primary key,
    relation_type_code varchar(255)  not null,
    relation_type_name varchar(255)  not null,
    reverse_relation_type_id   integer references
                                    relation_types(id)
                                         on update cascade
                                         on delete cascade
);

create table relations
    id serial primary key,
    relation_type_id   integer not null
                references relation_types(id)
                                         on update cascade
                                         on delete cascade,
    relation_object_id   integer not null
                references relation_objects(id)
                                         on update cascade
                                         on delete cascade
);




create table users
(
```

```
    id            serial    primary key,
    first_name    varchar(255) not null,
    last_name     varchar(255) not null,
    email         varchar(255) not null unique,
    role_id       integer not null
            references user_roles(id),
    password      varchar      not null,
);


create table user_roles
(
    id serial primary key,
    role_name varchar(255)  not null,
    role_code varchar(255)  not null
);

create table teams
(
    id serial primary key,
    team_name varchar(255)  not null,
    team_code varchar(255)  not null
);

create table teams_users
    id serial primary key,
    team_id   integer not null
                references teams(id)
                              on update cascade
                              on delete cascade,
    user_id   integer not null
                references users(id)
                              on update cascade
                              on delete cascade
);
```

ДОДАТОК В

Запити для створення ролей та надання привілеїв

Створення ролей та надання привілеїв для реляційної бази даних.

```
create role employee with login password '11111111';
grant select on all tables in schema public to employee;
grant update on users, key_results, objectives, tasks to employee;
grant insert, delete on key_results, objectives, tasks to employee;
grant usage, select on sequence key_results_id_seq, objectives_id_seq, tasks_id_seq to employee;


create role manager with login password '22222222';
grant select, insert, update, delete on all tables in schema public to manager;
grant all on all sequences in schema public to manager;

create role admin with login password '33333333';
grant select, insert, update, delete on all tables in schema public to admin;
grant all on all sequences in schema public to admin;
```

Створення ролей та надання привілеїв для графової бази даних.

```
CREATE ROLE employee;
create user employee set password '11111111';
GRANT MATCH {*} ON GRAPH neo4j NODES * TO employee;
GRANT CREATE ON GRAPH * NODES * TO employee;
DENY CREATE ON GRAPH * NODES Project TO employee;
GRANT DELETE ON GRAPH * NODES * TO employee;
DENY DELETE ON GRAPH * NODES Project TO employee;
GRANT SET PROPERTY {*} ON GRAPH * NODES * TO employee;
DENY SET PROPERTY {*} ON GRAPH * NODES Project TO employee;
GRANT MERGE ON GRAPH * NODES * TO employee;

CREATE ROLE manager;
create user manager set password '22222222';
GRANT MATCH {*} ON GRAPH neo4j NODES * TO manager;
GRANT CREATE ON GRAPH * NODES * TO manager;
GRANT DELETE ON GRAPH * NODES * TO manager;
GRANT SET PROPERTY {*} ON GRAPH * NODES * TO manager;
GRANT MERGE ON GRAPH * NODES * TO manager;

CREATE ROLE administrator;
create user manager set password '33333333';
GRANT MATCH {*} ON GRAPH neo4j NODES * TO administrator;
GRANT CREATE ON GRAPH * NODES * TO administrator;
GRANT DELETE ON GRAPH * NODES * TO administrator;
GRANT SET PROPERTY {*} ON GRAPH * NODES * TO administrator;
GRANT MERGE ON GRAPH * NODES * TO administrator;
```