Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Удосконалення методів і засобів управління знаннями в IT-проєкті»

Виконала: студентка 2-го курсу, групи ПІ-22мз
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Зацепіна Л.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Коваленко О.О
«07» червня 2024 р.

Опонент: к.т.н., доц. каф. ЗІ Куперштейн Л. М.
«07» червня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
(прізвище та ініціали)
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.
_____
«_12_» ____березня_____ 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ

Зацепіній Любові Володимирівні

1. Тема роботи – Удосконалення методів і засобів управління знаннями в ІТ-проєкті.

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «_11_» __березня__ 2024 року № __81__.

2. Строк подання студентом роботи

__03__червня__2024 року__

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мови розробки – JavaScript, TypeScript, фронтенд фреймворк Angular, бекенд – Node.js і Express, база даних – MongoDB, операційна система – Windows 10, методи управління знаннями, засоби управління знаннями в ІТ-проєкті.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; аналіз методів управління знаннями; методи та моделі управління знаннями в ІТ-проєкті; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: блок-схеми алгоритмів роботи додатку; тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О.О., к.т.н., доцент. кафедри ПЗ	12.03.24	03.06.24
5	Кавецький В.Д. к.е.н., доцент кафедри ЕПВМ	26.04.24	06.06.24

7. Дата видачі завдання _____12 березня 2024 р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Обгрунтування доцільності удосконалення методів та засобів управління знаннями	12.03.2024 – 26.03.2024	
2	Аналіз відомих інформаційних систем та застосунків управління знаннями	27.03.2024 – 05.04.2024	
3	Моделювання та вдосконалення методів та засобів управління знаннями в ІТ-проєкті	06.04.2024 – 19.04.2024	
4	Реалізація та тестування програмних модулів	20.04.2024 – 31.05.2024	
5	Економічна частина	26.04.2024 – 06.06.2024	

Студент _____ Зацепіна Л.В.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Коваленко О.О.
(підпис) (прізвище та ініціали)

Анотація

УДК 004. 005:96

Зацепіна Л.В. Удосконалення методів та засобів управління знаннями в ІТ-проєкті. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 175 с.

На укр. мові. Бібліогр.: 38 назв; рисунків: 43; таблиць 12.

Магістерська робота містить аналіз відомих платформ управління знаннями, визначення та сутність моделей управління знаннями. У роботі обгрунтовано доцільність удосконалення методів та засобів управління знаннями в ІТ-проєкті.

У магістерській кваліфікаційній роботі подано результати дослідження методів та моделей управління знаннями. Запропоновано метод автоматизованого управління знаннями, який базується на принципах інформаційних екосистем, враховує методологію створення програмного продукту та реалізується на основі тріади елементність, зв’язність та цілісність, що дозволяє комплексно залучати всіх учасників ІТ-проєктів до процесу управління знаннями та покращувати якість створюваних програмних продуктів.

У роботі виконано моделювання загального функціонування системи, модулів створення нотаток та завдань. Для удосконалення методів та засобів управління знаннями В межах магістерської роботи запропоновано модель, яка, акцентує увагу на вимогах до програмного продукту та враховує модель методології створення програмного продукту, що дозволяє формалізувати знання проєкту, механізми їх генерування, зберігання та обміну для підвищення рівня ефективності реалізації поставлених цілей проєкту.

Отримані в магістерській кваліфікаційній роботі результати можна впровадити у ІТ компаніях для оптимізації управління знаннями на проєктах.

Ключові слова: управління знаннями, розповсюдження знань, методи управління знаннями, інформаціна екосистема, корпоративний портал.

Abstract

Zatsepina L.V. Improvement of Knowledge Management Methods and Tools in IT Projects. Master's Qualification Thesis in the specialty 121 - Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2024. 175 pages.

In Ukrainian. Bibliography: 38 titles; figures: 43; tables: 12.

The master's thesis includes an analysis of well-known knowledge management platforms, the definition and essence of knowledge management models. The thesis substantiates the feasibility of improving knowledge management methods and tools in IT projects.

The master's qualification thesis presents the results of research on knowledge management methods and models. A method of automated knowledge management is proposed, based on the principles of information ecosystems, taking into account the methodology of software product development and implemented on the basis of the triad of elementarity, connectivity, and integrity, which allows for the comprehensive involvement of all IT project participants in the knowledge management process and improves the quality of the developed software products.

The thesis models the general functioning of the system, modules for creating notes and tasks. To improve knowledge management methods and tools, a model is proposed within the framework of the master's thesis that focuses on the requirements for the software product and takes into account the methodology of software product development, allowing for the formalization of project knowledge, mechanisms for their generation, storage, and exchange to increase the efficiency of achieving the project's goals.

The results obtained in the master's qualification thesis can be implemented in IT companies to optimize knowledge management in projects.

Keywords: knowledge management, knowledge dissemination, knowledge management methods, information ecosystem, corporate portal.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Актуальність питання вдосконалення методів та засобів управління знаннями в ІТ-проєктах не втрачає своєї значущості з часом. Незважаючи на те, що системи управління знаннями вже тривалий час використовуються в різних галузях, зокрема в ІТ, постійні зміни у технологіях, методах аналізу даних, а також цифровізації бізнес-процесів, створюють передумови для подальшого розвитку цих систем. Розробка нових методів і засобів управління знаннями базується на сучасних трендах інформаційного менеджменту, що включають удосконалення методів візуалізації даних, оптимізацію процесів обміну інформацією та впровадження нових технологій аналітики. Одним із ключових аспектів є створення інформаційних екосистем, які об’єднують різні джерела знань, дозволяючи їх ефективно використовувати для підвищення продуктивності, ухвалення рішень та інновацій. Вітчизняні та міжнародні дослідження свідчать про необхідність розвитку інструментів управління знаннями для покращення конкурентоспроможності підприємств. Використання розвинутих баз даних і знань, запровадження аналітичних модулів для оцінки продуктивності та залучення персоналу є сучасними підходами, які сприяють підвищенню ефективності ІТ-проєктів.

Метою роботи є розробка програмного забезпечення управління знаннями ІТ-проєкту для підвищення ефективності створення та запровадження програмних продуктів.

Основними завданнями дослідження є:

- Провести аналіз існуючих методів і засобів автоматизації та цифровізації управління знаннями в ІТ-проєктах;

- Виконати аналіз існуючих інформаційних систем управління знаннями в ІТ-проєктах;

- Розробити нові методи удосконалення процесів автоматизованого управління знаннями;

- Розробити моделі програмних модулів для управління знаннями;

- Розробити програмні компоненти для інтеграції в корпоративні портали;

- Провести експериментальні дослідження розроблених методів та засобів.

Об’єкт дослідження – процеси створення системи управління знаннями в ІТ-проєктах.

Предмет дослідження – методи та засоби, програмне забезпечення інформаційних систем управління знаннями.

Методи дослідження. У процесі досліджень використовувались:

- Методи автоматизації та цифровізації процесів управління знаннями;
- Теорія множин;
- Математичне та комп’ютерне моделювання процесів автоматизації управління знаннями;
- Методи та технології розробки програмного забезпечення для інформаційних систем.

Наукова новизна отриманих результатів:

1. Запропоновано метод автоматизованого управління знаннями, який, на відміну від існуючих, базується на принципах інформаційних екосистем, враховує методологію створення програмного продукту та реалізується на основі тріади елементність, зв’язність та цілісність, що дозволяє комплексно залучати всіх учасників ІТ-проєктів до процесу управління знаннями та покращувати якість створюваних програмних продуктів.

2. Удосконалено модель, яка, на відміну від існуючих, акцентує увагу на вимогах до програмного продукту та враховує модель методології створення програмного продукту, що дозволяє формалізувати знання проєкту, механіми їх генерування, зберегіння та обміну для підвищення рівня ефективності реалізації поставлених цілей проєкту.

Практична цінність результатів полягає у запропонованих програмних модулях управління знаннями, які можуть бути впроваджені для управління

проєктів.

Основні результати досліджень опубліковано у матеріалах конференції [1].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 38 найменувань, 5 додатків. Робота містить 43 ілюстрації, 12 таблиць.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА СУТНІСТЬ УПРАВЛІННЯ ЗНАННЯМИ

1.1 Основні поняття управління знаннями

Знання є життєво важливою складовою кожної компанії. Без збереження знань буде складно приймати рішення, вчитися на власних помилках та розробляти нові продукти. Управління знаннями – це підхід, який можна використовувати для збереження, організації та обміну знаннями компанії як з внутрішніми, так і зовнішніми зацікавленими сторонами.

Створення успішної системи управління знаннями не є легкою задачею і вона повинна бути адаптована до власних унікальних потреб та обставин. Необхідно отримати розуміння того, як структурувати систему, потенційні перешкоди на шляху роботи компанії та найефективніші інструменти, які дозволять реалізувати систему.

Розглянемо центральні поняття та аспекти управління знаннями, які є ключовими для ефективної діяльності організацій у сучасному бізнес-середовищі.

Управління знаннями – це комплексний підхід до збору, створення, організації, зберігання, розповсюдження та використання знань у рамках організації з метою досягнення стратегічних цілей та підвищення конкурентоспроможності. У сучасному інформаційному середовищі, особливо в галузі ІТ, управління знаннями стає ключовим фактором успіху, оскільки швидкі темпи змін та інновацій вимагають ефективного використання наявних ресурсів та досвіду. Том Девенпорт визначає управління знаннями як процес здобуття, розподілу та ефективного використання знань [2].

Знання – сукупність інформації, досвіду, інтуїції та інших уявлень, які мають стратегічне значення для організації та можуть бути використані для досягнення її цілей та вирішення проблем. Знання розглядається як актив, оскільки воно має економічну цінність та впливає на продуктивність та

конкурентоспроможність організації.

У сфері бізнесу, поняття "знання" охоплює різні типи інформації, включаючи експліцитні, які виражені чітко, і імпліцитні, які не виражені явно.

Ці знання можна розділити на стратегічні, які визначаються як ключові для досягнення стратегічних цілей, і оперативні, які необхідні для щоденної роботи та функціонування організації. Важливо також враховувати, що знання може бути як індивідуальним, що належить конкретній особі, так і колективним – знаходиться у власності організації та представлене в її базі даних [3].

Управління знаннями – процес, що охоплює збір, створення, організацію, розповсюдження та використання знань для покращення діяльності організації та досягнення її стратегічних цілей. Це важлива практика для забезпечення ефективного функціонування та розвитку організації в умовах зростаючої складності та конкуренції.

Контекстуалізація знань є важливим аспектом управління знаннями, оскільки знання має бути вбудоване у конкретний контекст ситуації або завдання для ефективного використання. Цей процес включає в себе розуміння конкретних потреб, цілей та контексту, в якому буде використовуватися знання. Це означає, що накопичені знання мають бути адаптовані до конкретної ситуації або проблеми, з урахуванням специфіки та умов відповідного контексту. Наприклад, знання про технології виробництва може бути корисним для розв'язання проблем у виробничій галузі, але його застосування може відрізнятися залежно від типу виробництва, обладнання та процесів, які використовуються. Таким чином, контекстуалізація знань допомагає забезпечити їх ефективне використання в конкретних ситуаціях та сприяє досягненню бажаних результатів [4].

Процес передачі знань – це фундаментальний систематичний процес ефективного обміну знань між учасниками організації з метою збагачення досвіду та підвищення ефективності вирішення завдань. У рамках процесу передачі знань можуть використовуватися різноманітні методи та інструменти, такі як наставництво, тренінги, семінари, робочі групи, внутрішні комунікаційні

системи тощо. Ці методи сприяють активному взаємовчителюванню, співпраці та розвитку особистих та професійних навичок учасників організації.

Капіталізація знань – це процес перетворення неструктурованих або імпліцитних знань у структуровані форми, які можуть бути легко доступні та використані в організаційній діяльності. Це включає процеси документування, формалізації та систематизації знань для подальшого використання та обміну.

Культура знань – це середовище, що сприяє виникненню, обміну та використанню знань серед працівників організації. Культура знань є системою цінностей, переконань, норм та практик, які сприяють створенню середовища, що сприяє розвитку та обміну знаннями в організації. Це включає в себе підтримку відкритості, співпраці, навчання та інновацій у всіх рівнях управління.

Технології управління знаннями охоплюють широкий спектр інструментів, засобів та систем, які допомагають організаціям збирати, організовувати, аналізувати та передавати знання. Вони можуть включати в себе системи управління документами, бази даних, інформаційні портали, системи електронного навчання та інші.

Стратегічне управління знаннями – це системний підхід до визначення, розробки та реалізації стратегій та ініціатив, спрямованих на ефективне використання знань для досягнення стратегічних цілей організації. Це включає в себе аналіз потреб, визначення цілей, розробку планів дій та моніторинг їх виконання для забезпечення успішної імплементації стратегічних ініціатив.

Процес управління знаннями можна розбити на 5 кроків:

Створення. Організації повинні ідентифікувати та записувати будь-які знання, які вони хочуть поширити по всій компанії. Знання повинно бути записано та адаптовано у форму, яка буде придатна для цільової аудиторії. Необхідно співпрацювати з експертами у відповідній галузі, щоб отримати знання, якими вони володіють, та перетворити їх в контент, який може бути доступний для будь-кого в компанії.

Організація. Після того, як необхідну інформацію зібрано, потрібно зберегти її таким чином, що буде логічним для обраної ІТ-системи. Знання може

потребувати форматування, щоб відповідати вимогам системи. На цьому етапі контент завантажується до системи управління знаннями та відбувається його організація в категорії для подальшого використання.

Спільне використання. У разі, коли система управління знаннями невідома користувачам та недоступна для них, її цінність мінімальна. З метою забезпечення доступності та широкого поширення знань, необхідно здійснювати їх розповсюдження за допомогою різноманітних інструментів, таких як електронна пошта, системи співпраці, внутрішні мережі тощо. Крім того, коли виникають питання, на які можна знайти відповідь у системі управління знаннями, варто наголосити на використанні відповідного спільнодоступного матеріалу для вирішення даного питання.

Аналіз. У процесі діяльності необхідно проводити аналіз ефективності контенту. Для цього можна здійснити перегляд результатів пошуку, виявити терміни, які не знаходять відповідних записів, та розробити відповідний контент для заповнення цих прогалин. Також доцільно ідентифікувати статті, які не користуються популярністю, та видалити їх, а також врахувати будь-які коментарі користувачів для оновлення матеріалів.

Оптимізація. Оптимізація є етапом, на якому особа діє на основі аналізу даних. Важливо підтримувати актуальність записів та систематично їх оновлювати у відповідь на нові тенденції та проблеми, що виникають у цільовій аудиторії. Крім того, доцільно розглядати різноманітні формати контенту, такі як відео, які можуть сприяти кращому розумінню складних концепцій.

Представлення знань – це процес формального відображення знань про певну область у комп’ютерно-інтерпретованій формі. Для забезпечення цього процесу використовуються моделі представлення знань, які структурують інформацію та надають процедури для пошуку рішень. Одним із важливих аспектів є ефективність моделі, яка повинна мати достатню виразність для передачі всієї необхідної інформації та ефективність у пошуку рішень. Розробка оптимального співвідношення між виразністю моделі та її ефективністю – ключове завдання для розробників систем штучного інтелекту.

На сьогоднішній день існує різноманіття моделей представлення знань, кожна з яких має свої переваги та недоліки. Вибір конкретної моделі залежить від завдання, що перед вами, і може визначити можливість успішного вирішення проблеми. Такі моделі можна умовно розділити на декларативні, де знання подаються у вигляді описів об'єктів та їх відносин, та процедурні, де значення знань представлені алгоритмами. Найпоширеніші серед них – логічні, продукційні, мережеві та фреймові моделі.

Ці поняття становлять основу для розуміння сутності управління знаннями та їхнього застосування в ІТ-проєктах. Детальне вивчення цих понять допоможе в розробці та впровадженні ефективних стратегій управління знаннями, що сприятимуть успішному виконанню проєктів у сфері інформаційних технологій.

1.2 Особливості управління знаннями ІТ-проєкту

Управління знаннями в інформаційно-технологічних (ІТ) проєктах має свої власні особливості, що відрізняються від управління знаннями в інших галузях. Серед них можна виокремити низку ключових аспектів, що впливають на ефективність управління знаннями в ІТ-середовищі.

Управління знаннями в ІТ-проєктах часто пов’язане з використанням спеціалізованих ІТ-інструментів та платформ. Воно може включати системи управління документами, бази даних, колаборативні платформи, системи електронного документообігу та інші технології, що сприяють збору, організації та обміну знаннями в рамках проєкту.

Додатково, важливо зазначити, що інтеграція цих технологій у процес управління знаннями дозволяє забезпечити ефективність та структурованість взаємодії між учасниками проєкту. Спеціалізовані ІТ-інструменти дозволяють не лише зберігати та організовувати інформацію, але й швидко знаходити необхідні дані, вести спільну роботу над документами та забезпечувати відстеження змін.

Застосування колаборативних платформ сприяє підвищенню комунікації між учасниками проєкту, спрощує процес спільного створення та редагування

документів, а також сприяє розвитку командної роботи та обміну ідеями. Такий підхід допомагає уникнути дублювання роботи, підвищує ефективність вирішення завдань та сприяє швидкому розгортанню нових ініціатив в рамках ІТ-проєкту.

Також важливо враховувати можливість інтеграції цих технологій з іншими системами управління, наприклад, системами управління проєктами чи системами моніторингу та аналізу даних. Це дозволяє забезпечити цілісний підхід до управління ІТ-проєктом та забезпечити максимальну ефективність використання знань та ресурсів.

ІТ-проєкти часто характеризуються стрімким розвитком технологій, постійними змінами стратегій, вимог клієнтів та ринкових умов. Управління знаннями повинно бути гнучким та швидкозмінним, щоб ефективно відповідати на нові виклики та можливості.

Відповідно до досліджень, ІТ-проєкти можуть зазнавати значних змін навіть протягом короткого періоду часу. Крім того, зазначена динамічність потребує постійного моніторингу та оновлення знань, які використовуються в рамках проєкту. Необхідно оновлювати та поповнювати знання з технологій, вивчати нові методи роботи та впроваджувати їх у практику проєкту. Це означає, що управління знаннями повинно бути гнучким і відкритим до внесення змін, щоб забезпечити ефективну реакцію на будь-які зміни в умовах проєкту та вирішення поставлених завдань у найкоротший час.

Ефективне управління знаннями в таких умовах передбачає наявність системи, яка дозволяє оперативно враховувати нові виклики та можливості, адаптуватися до нових умов, а також швидко передавати отриману інформацію всім зацікавленим сторонам проєкту.

Управління знаннями в ІТ-проєктах часто потребує розуміння технічних аспектів розробки програмного забезпечення, інфраструктури та інших ІТ-рішень. Це включає в себе розуміння архітектурних рішень, методів розробки, використання мов програмування та інших технічних аспектів, які впливають на процеси управління знаннями.

Врахування технічної складності проєкту дозволяє ефективно створювати та впроваджувати системи управління знаннями, що відповідають специфіці і потребам конкретного ІТ-проєкту. Такий підхід сприяє збереженню, обміну та ефективному використанню знань в рамках проекту, що забезпечує успішне виконання завдань та досягнення бізнес-цілей.

ІТ-проєкти зазвичай збирають великі обсяги даних, які потребують ефективного управління та аналізу. Це можуть бути дані про користувачів, транзакції, вимірювання продуктивності, аналіз раніше отриманих даних тощо. Управління знаннями в таких проєктах повинне включати методи та інструменти для збору та обробки великих обсягів даних з метою їх подальшого аналізу та інтерпретації.

Наприклад, у проєкті розробки програмного забезпечення можуть збиратися великі обсяги даних про користувачів, їх взаємодію з програмою, витрати ресурсів системи тощо. Ці дані можуть бути використані для аналізу ефективності програми, виявлення можливих проблем та вдосконалення функціоналу.

Управління знаннями допомагає забезпечити, збереження усіх даних в систематизованому та доступному форматі, що дозволяє зробити висновки та приймати рішення на основі об'єктивної інформації. Такий підхід до управління даними дозволяє підвищити ефективність роботи команди проєкту та досягти поставлених цілей [5].

В умовах сучасного розвитку технологій та ринкової конкуренції, реалізація ІТ-проєктів зазвичай вимагає співпраці між різними фахівцями з різних галузей, такими як програмісти, інженери, аналітики, дизайнери та інші, що мають різні компетенції та професійний досвід. Управління знаннями в контексті ІТ-проєктів повинно сприяти ефективній комунікації та обміну знаннями між усіма учасниками команди.

Це означає, що необхідно створити механізми, які сприяють вільному потоку інформації та знань між різними спеціалістами. Наприклад, це може включати в себе створення спеціалізованих баз даних або порталів, де кожен

учасник команди може ділитися своїми знаннями, досвідом та рекомендаціями.

Крім того, важливою є розробка чітких процедур та стандартів, які визначають, як саме проводиться обмін знаннями та які ресурси доступні для всіх учасників команди. Наприклад, можна встановити регулярні зустрічі або тренінги, де фахівці можуть обговорювати актуальні питання та ділитися своїми найкращими практиками. Загалом, ефективне управління знаннями в IT-проєктах сприяє підвищенню продуктивності команди.

Необхідно створити команду, яка буде відповідальна за оновлення бази знань або делегуватиме цю роботу співробітникам та експертам з певних питань за потреби. Необхідні ролі включають:

Менеджерів проєктів, які будуть відслідковувати операції та гарантувати, що зусилля з управління знаннями працюють ефективно.

Менеджерів контенту, які створюватимуть, керуватимуть та розповсюджуватимуть цікавий та інформативний контент.

Технічних письменників, які створюватимуть чіткий, зрозумілий та цікавий матеріал з складних тем.

Ця команда також повинна забезпечити відповідність контенту, який доступний для клієнтів через ваші центри підтримки, чат-ботів та інші системи управління знаннями, відповідно стилю та бренду компанії.

Розуміння цих особливостей є ключовим для успішного управління знаннями в інформаційно-технологічних проєктах і сприяє досягненню їх стратегічних цілей.

1.3 Відомі інформаційні системи та застосунки управління знаннями

У сучасному IT-середовищі ефективне управління знаннями є ключовим фактором успіху для будь-якого проєкту. Знання та інформація є найціннішими активами для будь-якої організації, особливо в галузі технологій, де швидкість змін та інновацій на порядок вища, ніж в інших сферах.

У цьому контексті вибір правильної інформаційної системи або застосунку для управління знаннями стає важливим завданням для керівників проєктів та

інформаційних технологій. Доцільний вибір інструменту може вирішити проблеми з комунікацією, збереженням знань та забезпечити потрібний рівень продуктивності та інноваційності.

Програмне забезпечення для керування знаннями – це платформа, призначена для збору, організації, зберігання та полегшення обміну знаннями та інформацією в організації. Це допомагає підприємствам ефективно керувати та використовувати їхні колективні знання, такі як документи, файли, бази даних і досвід.

Під час пошуку ідеального програмного забезпечення для керування знаннями важливо переконатися, що такі фактори відповідають потребам організації:

Наявністть простого у користуванні інтерфейсу – команда повинна мати можливість легко прийняти рішення для управління знаннями.

Централізоване сховище знань повинно бути організовано так, щоб дозволяти організації зберігати та систематизувати колективні знання, щоб отримати до них легкий доступ і виконувати пошук. Це поняття включає в себе документи, файли, обговорення та інші важливі ресурси, доступ до яких можна отримати швидко та ефективно. Централізований центр знань робить процес пошуку та доступу до інформації простішим для всіх працівників організації.

Ефективні функції пошуку є важливою характеристикою систем керування знаннями. Функція пошуку дозволяє команді швидко знаходити необхідну інформацію, ефективно керувати часом та підвищувати продуктивність.

При виборі або створенні програмного забезпечення слід звертати увагу на можливості, які сприяють ефективній співпраці між її учасниками. Такі функції, як редагування в реальному часі, коментування та контроль версій, дозволяють багатьом користувачам одночасно працювати над внутрішньою базою знань, що покращує обмін знаннями та сприяє спільній роботі у вашій організації.

Програмне забезпечення для керування знаннями має доповнювати уже

існуючі системи в організації, наприклад інструмент проектного менеджменту, рішення для керування та інше програмне забезпечення. Програмні рішення на базі знань із потужними можливостями інтеграції додатково оптимізують управління процесами та уникають дублювання зусиль.

Розглянемо кілька відомих інформаційних систем та застосунків для управління знаннями в IT-проєктах. Для кожного інструменту виконаємо аналіз його функціональності, переваг та недоліків, а також проаналізуємо область використання та доступність демоверсій для оцінки його ефективності та придатності для конкретного проєкту.

1. Document360

Це система управління знаннями, яка відзначається у створенні та управлінні контентом, пропонуючи інтуїтивні інструменти для створення та організації статей, часто заданих питань та документації проєктів [6].

Document360 підходить для різних типів бізнесу, включаючи стартапи, підприємства та організації, що розширюються, яким потрібен швидкий доступ до відповідної інформації. Він задовольняє потреби команд з підтримки клієнтів, менеджерів продукту та бізнесів, що обробляють великі обсяги даних та інформації.

Основні функції Document360:

- Виділяються можливості пошуку завдяки розширеним фільтрам, пошуку за ключовими словами та рекомендаціям, що працюють на основі штучного інтелекту;

- Інтуїтивний та зручний інтерфейс;

- Надійні функції співпраці, що дозволяють командам працювати разом над створенням та оновленням статей бази знань;

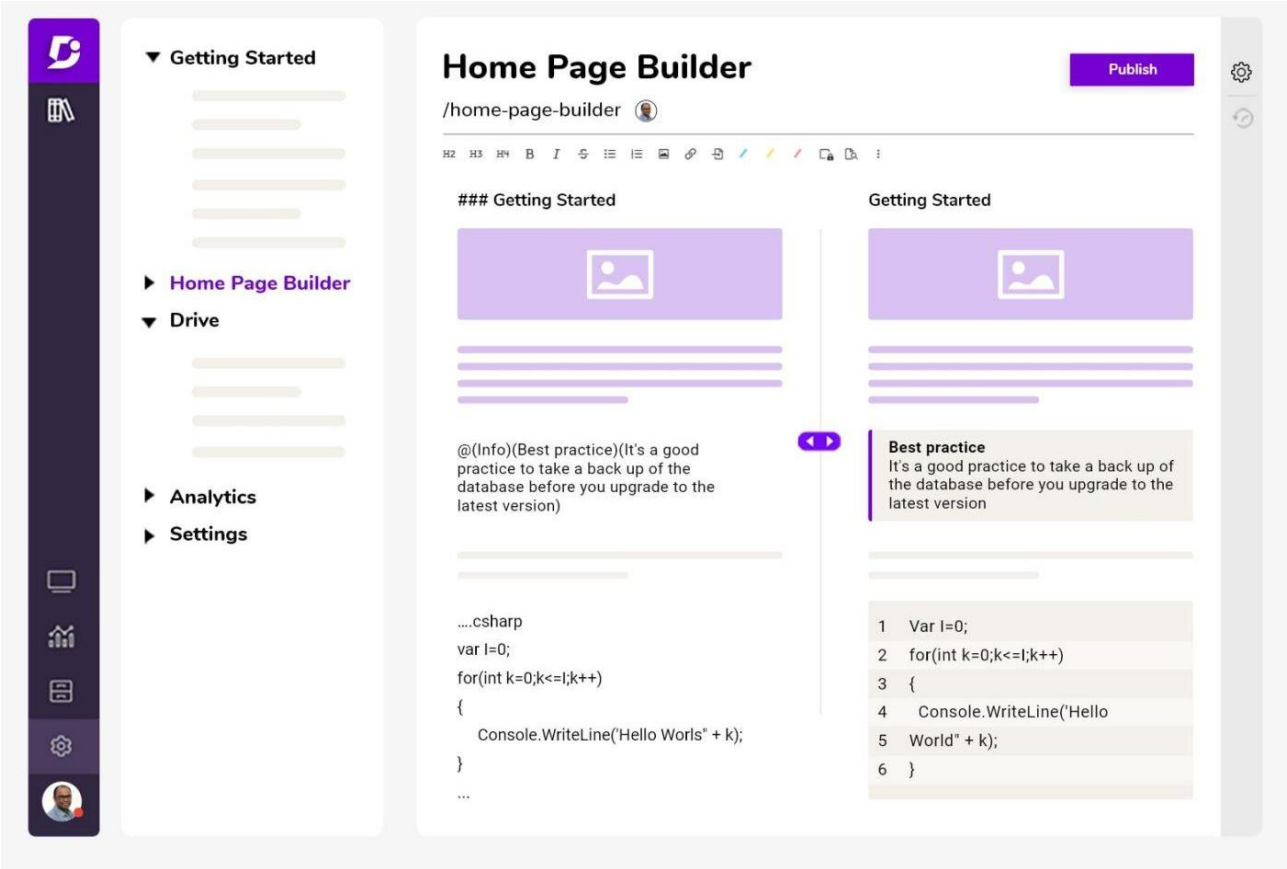


Рисунок 1.1 – Система управління знаннями Document360

Недоліки Document360:

- Висока цінова політика;

- Деякі користувачі вважають, що управління великими та складними базами знань у Document360 може бути складним;

- Деякі користувачі вважають, що ключові функції управління робочим процесом у Document360 є базовими і недостатньо розвинутими.

2. Confluence

Це частина пакету SaaS від Atlassian, є потужним вікі програмним забезпеченням, що дозволяє командам співпрацювати та безперешкодно обмінюватися інформацією, забезпечуючи єдність у роботі [7].

Підходить для широкого спектру бізнесів, але особливо корисний для кросфункціональних команд, керівників проектів та відділів з високим рівнем знань. Він сприяє безперешкодному обміну знаннями, підвищує продуктивність

і сприяє ефективній співпраці в усій компанії.

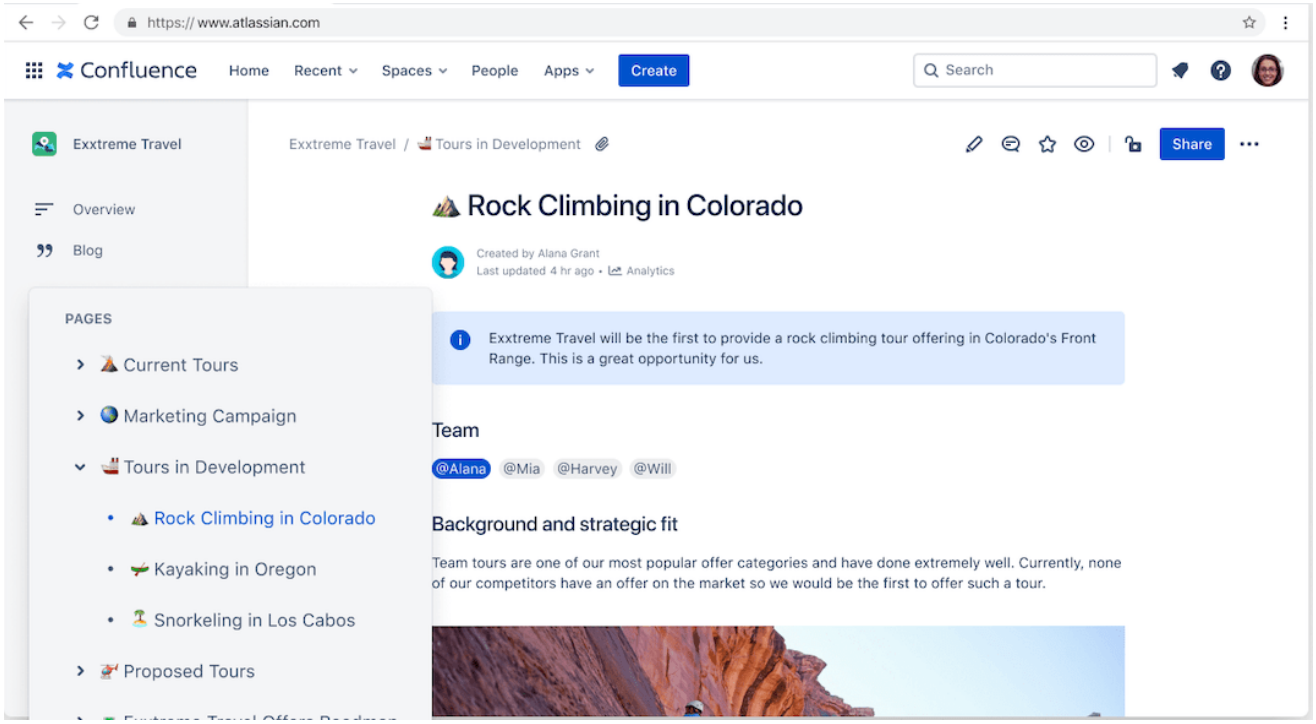


Рисунок 1.2 – Система управління знаннями Confluence

Основні переваги Confluence:

- Створення, редагування та організація контенту в реальному часі між командами та співробітниками;

- Гнучкі та налаштовувані простори дозволяють створювати відведені області для конкретних проектів, відділів чи областей знань;

- Розширені можливості пошуку;

- Надійні можливості налаштування макетів, шаблонів та тем.

Недоліки Confluence:

- Може знадобитися час і зусилля, щоб вивчити і використати функції Confluence ефективно;

- Користувачі зазначили, що варіанти форматування в Confluence, зокрема стосовно оформлення та дизайну документів, можуть бути обмеженими порівняно з спеціалізованим програмним забезпеченням для обробки текстів;

- Хоча Confluence призначений для роботи з великими обсягами контенту,

деякі користувачі стикалися з проблемами продуктивності при роботі з обширними базами знань або складними сторінками.

3. SharePoint

Це інтегрована платформа для спільної роботи та управління документами, розроблена компанією Microsoft. Починаючи зі спільного доступу до документів та календарів і закінчуючи спільними списками завдань та проектів, SharePoint пропонує широкий спектр функцій, які полегшують комунікацію та співпрацю в організації. Крім того, SharePoint може бути використаний як система керування знаннями, де він дозволяє зберігати, організовувати та шукати інформацію, сприяючи ефективному обміну знаннями в межах команди або підприємства [8].

Переваги SharePoint:

- SharePoint ідеально поєднується з іншими інструментами Microsoft, такими як Office 365, Outlook та Teams, що спрощує спільну роботу та обмін інформацією;

- SharePoint дозволяє налаштовувати сайти, списки та бібліотеки документів відповідно до потреб команди або організації, що забезпечує високу гнучкість у використанні;

- SharePoint надає зручні засоби для спільної роботи над документами, включаючи можливість одночасного редагування документів та коментування;

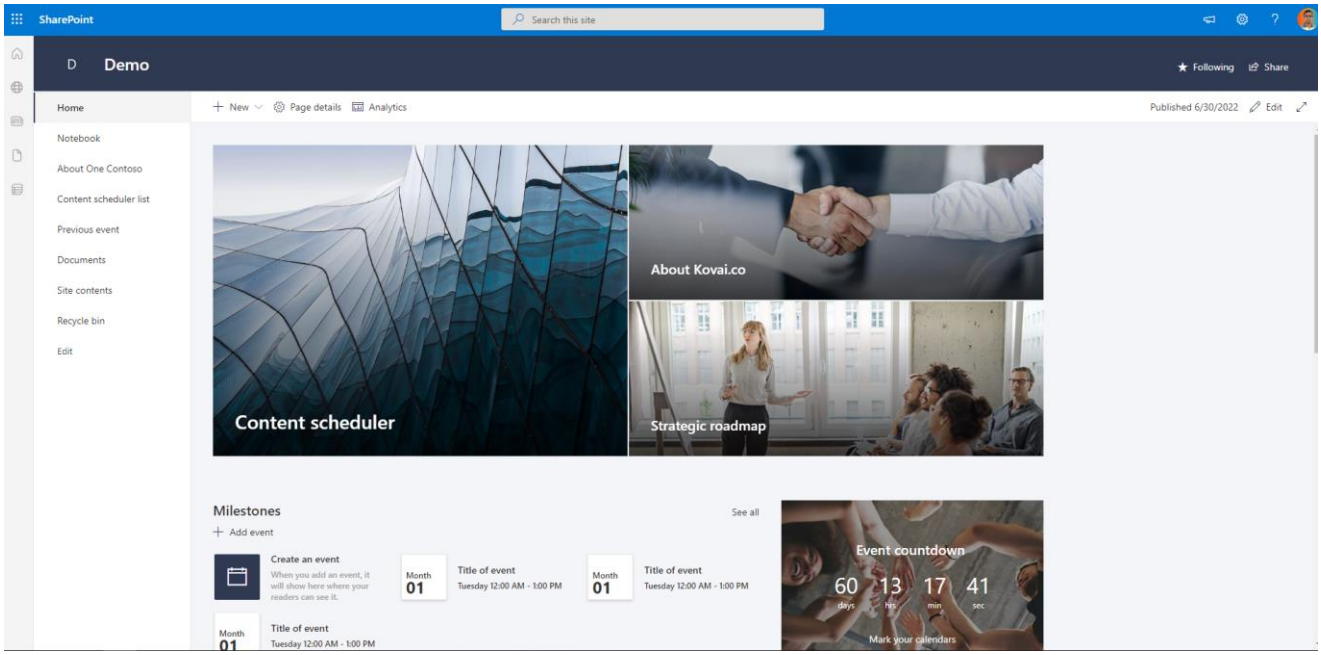


Рисунок 1.3 – Система управління знаннями SharePoint

- Система керування доступом до ресурсів SharePoint дозволяє точно налаштовувати права доступу до документів та інших ресурсів в залежності від ролі користувача.

Недоліки SharePoint:

- Для багатьох користувачів SharePoint може здаватися складною у використанні та освоєнні, особливо для новачків або нетехнічних користувачів;

- У порівнянні з іншими системами, SharePoint може мати обмежені можливості налаштування та адаптації під конкретні потреби організації;

- Впровадження та підтримка SharePoint може бути дорогими для невеликих підприємств або організацій з обмеженими бюджетами;

- При роботі з великим обсягом даних або складними структурами, користувачі можуть зіткнутися з проблемами продуктивності та швидкодії SharePoint.

4. Notion

Це універсальний робочий простір, де команди можуть створювати та організовувати внутрішні бази знань, проектні вікі та спільні документи в одній злагодженій платформі [9].

Система управління знаннями та інструмент для управління завданнями підходить для широкого кола осіб та команд, включаючи керівників проектів, творців контенту та віддалені команди. Вона слугує як комплексне рішення для організації та доступу до інформації, оптимізації робочих процесів та підвищення продуктивності команди.

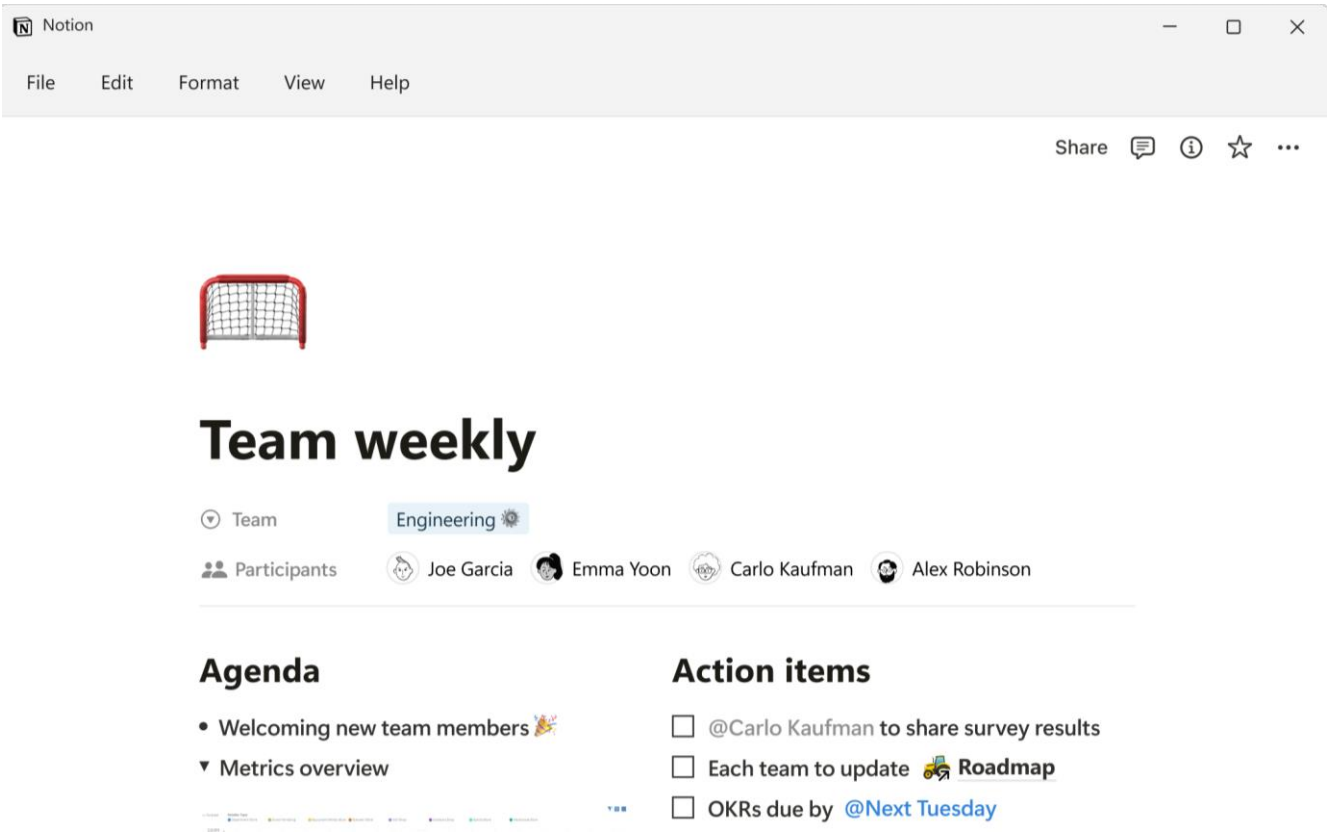


Рисунок 1.4 – Система управління знаннями Notion

Основні функції Notion:

- Гнучкий та налаштовуваний робочий простір дозволяє створювати власні макети, шаблони та бази даних, щоб відповідати вашим конкретним робочим потребам та вподобанням;

- Команди можуть співпрацювати в реальному часі, залишати коментарі та відстежувати зміни;

- Легка вставка та інтеграція різноманітних медіа-типів, включаючи зображення та відео;

- Функціонал баз даних Notion дозволяє створювати структурований

контент, організовувати дані та будувати динамічні перегляди в межах вашої бази знань.

Недоліки Notion:

- Зі збільшенням складності та обсягу інформації користувачі відмічають, що продуктивність Notion може бути порушена;

- Користувачі вказали на труднощі у керуванні контролем доступу до конкретних розділів або чутливої інформації в межах бази знань, якщо вони не знаходяться на рівні Enterprise.

5. Trello

Trello – це інструмент для керування завданнями та проектами, який дозволяє користувачам організовувати робочі процеси за допомогою дошок, списків та карток. Крім того, він може використовуватися як ефективний інструмент для керування знаннями в організації. Завдяки своїм гнучким можливостям конфігурації, Trello може бути використаний для створення централізованого сховища знань, де можна зберігати, організовувати і ділитися інформацією з різних галузей та проектів. Кожна дошка може виконувати роль тематичного розділу або проекту, а списки і картки можуть використовуватися для структурування та організації знань. Крім того, завдяки можливості коментування, обговорення та спільної роботи над картками, учасники команди можуть ділитися знаннями, доповнювати один одного та швидко знаходити необхідну інформацію [10].

Проте варто врахувати, що Trello має свої обмеження як система керування знаннями. Він призначений переважно для керування завданнями і проектами, тому його можливості зі структурування та організації великого обсягу інформації можуть бути обмеженими. Для деяких розширених функцій, які можуть бути потрібні для повноцінного управління знаннями великої компанії, може знадобитися використання додаткових доповнень або інтеграцій.

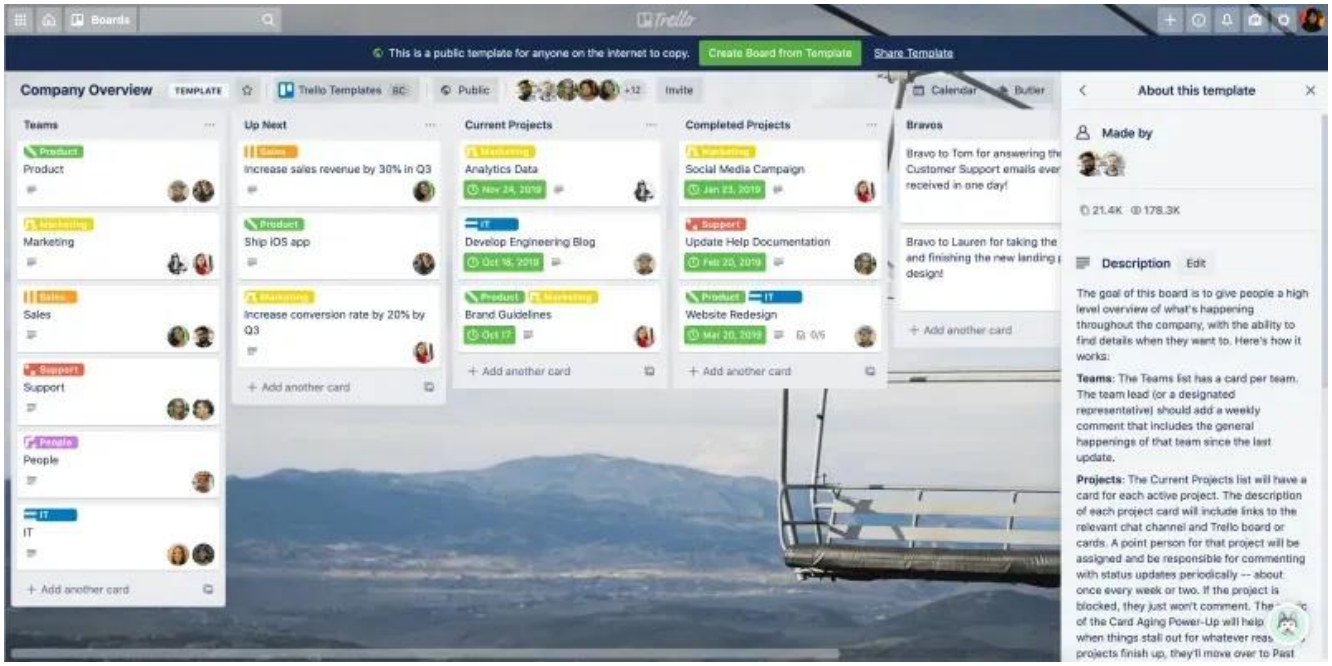


Рисунок 1.5 – Інформаційна система Trello

Переваги Trello:

- Простий та інтуїтивний інтерфейс, який дозволяє швидко розпочати роботу без спеціальної підготовки;

- Гнучкість у створенні різних типів дошок та списків, що дозволяє адаптувати інструмент до потреб будь-якого проєкту чи команди;

- Можливість розподілення завдань між учасниками команди, надання прав доступу та відстеження прогресу виконання завдань;

- Інтеграція з різноманітними іншими інструментами, такими як Google Drive, Dropbox, Slack тощо, для забезпечення зручності та продуктивності роботи.

Недоліки Trello:

- Обмежена функціональність управління завданнями, що може стати недостатньою для складних проектів або великих команд;

- Необхідність використання додаткових доповнень або інтеграцій для деяких розширених функцій, що може збільшити складність використання;

- Відсутність засобів для структурування та організації інформації великого обсягу, що може призвести до розплутування та втрати ефективності

робочого процесу.

Для ретельного аналізу і порівняння різних систем управління знаннями варто скласти порівняльну таблицю 1.1, що відображатиме основні характеристики та функціональні можливості кожної системи.

В процесі аналізу відомих інформаційних систем та застосунків управління знаннями було виявлено різноманітні рішення, які можуть відповідати потребам різних організацій. Кожна з цих систем має свої переваги та недоліки, що необхідно враховувати при виборі оптимального рішення для конкретної ситуації.

Таблиця 1.1 – Порівняння інформаційних систем управління знаннями

Назва системи/ застосунку	Опис	Переваги	Недоліки	Сфера використання	Демо-версія
Document 360	Потужний інструмент управління знаннями, який дозволяє створювати, організовувати та ділитися знаннями всередині організації. Має інтуїтивно зрозумілий інтерфейс, розширені можливості пошуку та категоризації інформації, а також підтримку аналітики використання знань.	- Інтуїтивний інтерфейс - Розширені можливості пошуку і категоризації - Підтримка аналітики	- Вартість може бути високою для деяких організацій	IT, бізнес, освіта	Так
Confluence	Інструмент управління знаннями, розроблений компанією Atlassian. Він надає можливість створювати, спільно редагувати та обмінюватися документацією в реальному часі. Інтегрується з іншими продуктами компанії Atlassian, такими як Jira, для забезпечення повного циклу розробки ПЗ	- Спільне редагування в реальному часі - Інтеграція з іншими продуктами Atlassian	- Вимагає певного часу для освоєння - Може бути вартістю для деяких організацій	IT, розробка, проектний менедж-мент	Так

Продовження таблиці 1.1

SharePoint	Платформа для спільної роботи та управління документами, розроблена Microsoft. Вона озволяє створювати сайти, обмінюватися документами, організовувати проекти та зберігати знання в централізованому місці. Інтегрується з іншими продуктами Microsoft, такими як Office 365.	- Інтеграція з іншими продуктами Microsoft - Централі-зоване зберігання документів та знань	- Вимагає ліцензування Microsoft - Може бути складно для користувачів з мінімальним досвідом	Бізнес, корпоративні структури	Так
Notion	Цифрова робоча платформа, яка поєднує в собі елементи заміток, баз даних, календаря, списків завдань та інших інструментів.	- Простий та інтуїтивний інтерфейс - Широкі можливості налаштування - Зручний спосіб організації інформації	- Обмежені можливості в планах з безкоштовним доступом - Вартість для команд із багатьох користувачів	Бізнес, освіта, особисте використання	Так
Trello	Інструмент для керування проєктами, який використовує карточки та дошки для організації завдань та співпраці між командами.	- Простий та інтуїтивний інтерфейс - Створення та налаштування власних дошок - Інтеграція з іншими інструментами	- Обмежені можливості аналітики - Відсутність багаторівневої організації завдань	Проєкт. менедж-мент, спільна робота, особисте використання	Так

Порівняльний аналіз показав, що кращим варіантом може виявитися система, яка найбільш точно відповідає потребам та можливостям конкретної організації, а також має оптимальне співвідношення між функціональністю та вартістю. На основі проведеного аналізу можна зробити висновок про необхідність дослідження методів та моделей управління знаннями ІТ-проєкту, що має на меті визначення оптимальних підходів до організації, зберігання, пошуку та використання знань у контексті сучасних вимог та технологічних можливостей та програмної реалізації окремих модулів для покращення функціонування системи управління знаннями в ІТ-проєктах.

1.4 Постановка задачі наукових досліджень створення ІС управління знаннями ІТ-проєкту

Постановка задачі наукових досліджень створення інформаційної системи управління знаннями в ІТ-проєкті має на меті визначення конкретних цілей та завдань, які будуть вирішуватися в рамках даного дослідження. Основні завдання включають:

1. Аналіз сучасних підходів до управління знаннями в ІТ-проєктах: Проведення огляду існуючих методів та моделей управління знаннями в ІТ-проєктах для виявлення їх переваг, недоліків та потреб у вдосконаленні;

2. Розробка вимог до системи управління знаннями: Визначення функціональних та нефункціональних вимог до інформаційної системи управління знаннями, враховуючи потреби користувачів та сучасні технологічні можливості;

3. Створення концепції системи: Розробка загальної концепції системи управління знаннями, включаючи структуру даних, інтерфейс користувача та основні функціональні можливості;

4. Вибір технологічних засобів: Визначення найбільш підходящих технологій та інструментів для реалізації інформаційної системи управління знаннями відповідно до встановлених вимог;

5. Розробка прототипу системи: Створення прототипу інформаційної системи управління знаннями для демонстрації основних функціональних можливостей та отримання відгуків від користувачів;

6. Оцінка ефективності системи: Проведення тестування та оцінка ефективності розробленої системи управління знаннями з урахуванням її відповідності поставленим цілям та завданням.

Відповідно до результатів аналізу відомих систем управління знаннями були визначені основні задачі магістерської роботи:

- виконати аналіз моделей та модулів інформаційних систем управління знаннми та сформувати рекомендації для удосконалення;

- розробити програмний застосунок для управління знаннями в проєкті з можливістю роботи з базами даних, додавання нотаток із зображеннями та для реалізації обговорення, додавання коментарів до нотаток, роботи в проєктах, додавання завдань до виконання на проєкті тощо;
- розробити бекенд частину та реалізувати модулі для інформаційної системи управління знаннями;
- розробити фронтенд частину інформаційної системи управління знаннями;
- виконати експериментальні дослідження та тестування розроблених модулів.

Ці задачі спрямовані на створення ефективної інформаційної системи управління знаннями в ІТ-проєкті, що відповідає сучасним вимогам та сприяє підвищенню продуктивності та успішності проєктів.

1.5 Висновки

У першому розділі було зроблено огляд основних понять управління знаннями. Основні поняття в цьому контексті включають нагромадження, організацію, розповсюдження та застосування знань усередині організації для досягнення її цілей та забезпечення конкурентоспроможності. Виконано аналіз ключових аспектів цих понять та їх значення для процесу управління знаннями. Встановлено, що управління знаннями є ключовим аспектом сучасного бізнесу та розвитку організацій.

Проаналізовано особливості управління знаннями в контексті ІТ-проєктів, враховуючи специфіку інформаційних технологій та потреби команди проекту. Визначено ключові аспекти, які необхідно враховувати при впровадженні систем управління знаннями в ІТ-проєктах.

Було проведено огляд відомих інформаційних систем та застосунків для управління знаннями, зокрема Document360, Confluence, Notion та Trello. Проаналізовано їх переваги та недоліки з метою вибору найбільш підходящого

рішення для конкретного ІТ-проєкту.

Деталізовано мету та виконано постановку задачі для розробки методів та моделей управління знаннями для покращення системи управління знаннями в ІТ-проєктах, реалізації програмного забезпечення. Задачі досліджень включають аналіз існуючих підходів, розробку нових методик та інструментів, впровадження та оцінку їх ефективності в практиці ІТ-проєктів.

2 МЕТОДИ ТА МОДЕЛІ УПРАВЛІННЯ ЗНАННЯМИ ІТ-ПРОЄКТУ

2.1 Метод управління знаннями відповідно до методології створення програмного продукту

Управління знаннями в ІТ-проєктах може бути визначено як сукупність процесів планування, збору, зберігання, обробки, передачі та використання знань для досягнення мети проєкту.

Наведемо деякі інструменти, які можна використовувати для ефективного управління знаннями в ІТ-проєктах:

Створення централізованих баз даних або внутрішніх порталів, де члени команди можуть зберігати та шукати інформацію, досвід та ресурси.

Програмні модулі управління документацією проєкту. Ретельна документація кожного етапу проєкту дозволяє зберегти знання про його виконання, проблеми та рішення, що виникають під час розробки.

Визначення та документування місії, цінностей та цілей проєкту. Важливо, щоб усі члени команди розуміли мету та завдання проєкту. Це допомагає забезпечити однакове тлумачення та спрямованість зусиль.

Платформа для комунікацій та спільного навчання для команди проєкту. Спільні сесії навчання, ретельний аналіз помилок та успіхів, обмін кращими практиками – все це сприяє накопиченню та поширенню знань в команді.

Використання спеціалізованих програм для спільної роботи, таких як системи керування проектами, чати, форуми, дозволяє швидко обмінюватися інформацією та співпрацювати. Створення системи менторства, де досвідчені члени команди допомагають новачкам освоюватися в проєкті та передають їм свої знання та навички буде ефективнним при умові використання сучасних інформаційних технологій. Комунікаційні інструменти дозволяють здійснювати ефективний обмін інформацією, генерувати нові знання.

Регулярне оцінювання ефективності процесів управління знаннями, виявлення слабких місць та розробка стратегій і тактик для реалізації та

удосконалення процесів управління знаннями дозволить сформувати спеціальне інформаційне середовище управління проєктом, генерації, зберігання та передавання знань.

Розглянемо методи управління знаннями в контексті існуючих методологій створення програмного продукту.

У методології Agile управління знаннями відбувається через постійну комунікацію та співпрацю між учасниками команди. Одною з реалізацій методології Agile є Scrum.

- Стендапи (Stand-ups) – це щоденні короткі зустрічі команди, де кожен учасник розповідає про свої досягнення, проблеми та плани на день, що сприяє активному обміну знаннями та ідентифікації проблем на ранніх етапах.

- Спринти (Sprints) – це періоди розробки, під час яких команда фокусується на конкретних завданнях. Після завершення кожного спринту проводяться демонстрації працюючого програмного продукту. .

У Scrum управління знаннями є ключовою частиною процесу. До знань проєкту відносять інформацію, яка перетворюється в знання після обробки агентами системи управління проєкту та управління знаннями в цій системі. Так наприклад, Product Backlog – список усіх завдань проєкту – підтримується та оновлюється всією командою. Кожен елемент Product Backlog повинен бути ретельно описаний та бути зрозумілим для всіх учасників проєкту. Документально, знання фікусуються під час зборів перед початком кожного спрінта. Такі збори можна розділити на частини.

Демонстрація продукту – результати роботи під час виконання завдань спрінта.

Sprint Planning – частина зборів перед початком кожного спрінта, коли команда обговорює та отримує завдання з Product Backlog.

Модератором щодо управління знаннями є скрам-майстер.

У методології Kanban управління завданнями зазвичай здійснюється за допомогою візуальних дошок, які відображають статус роботи над завданнями. Основні аспекти включають:

- Формування дошки, що відображає особливості проєкту. Класичні стовпці такої дошки – це може бути «To Do», «In Progress», «Review», «Done» тощо. Заповнена дошка може бути збережена як шаблон і представляє собою знаннєвий актив проєкту.

- Використання метрик, таких як час циклу та кількість завдань, що вирішуються за певний період часу, допомагає команді аналізувати та вдосконалювати свої процеси.

У DevOps управління знаннями зазвичай орієнтоване на автоматизацію процесів та спільний доступ до інструментів і знань. Деякі підходи включають управління версіями та інфраструктурою, автоматизація процесів інтеграції та використання спеціальних платформ.

У кожній з цих методологій управління знаннями відіграє важливу роль у забезпеченні ефективного спілкування, співпраці та навчання всієї команди для досягнення успішного результату в розробці програмного продукту.

Розглянемо більш детально деякі ключові методи управління знаннями.

1. Однією з головних переваг управління документами є підвищення ефективності. Ще однією ключовою перевагою є покращена безпека. Багато організацій мають справу з конфіденційною інформацією, яку необхідно зберігати в таємниці. Системи документообігу можуть гарантувати, що лише уповноважений персонал має доступ до певних документів, зменшуючи ризик витоку даних і несанкціонованого доступу [11].

Завдяки єдиному розташуванню для всіх документів співробітникам легше обмінюватися інформацією та співпрацювати над проєктами. Існує багато різних типів систем керування документами, починаючи від простих рішень для зберігання файлів і закінчуючи складними системами корпоративного рівня. Правильна система для конкретної організації залежатиме від таких факторів, як розмір організації, тип документів, з якими необхідно працювати, і передбачений бюджет [12].

При впровадженні системи документообігу важливо застосовувати комплексний підхід. Це передбачає розгляд усього життєвого циклу документа,

від створення до утилізації. Ще одним важливим фактором є інтерфейс користувача. Система управління документами повинна бути простою у використанні та інтуїтивно зрозумілою [13].

2. Обмін знаннями – це процес обміну інформацією, навичками та досвідом між окремими особами чи організаціями. Існують різні способи обміну знаннями, включаючи тренінги, наставництво, коучинг, спільноти практиків та соціальні мережі. В ІТ-проєктах сюди ще можна додати парне програмування та код рев'ю.

Однією з переваг обміну знаннями є те, що він сприяє формуванню культури навчання в організації. Коли люди діляться своїми знаннями, вони також вчаться в інших і отримують нові ідеї та перспективи, підвищують ефективність особисту та проєкту в цілому, сприяють впровадженню інновацій. Ще одна перевага обміну знаннями полягає в тому, що він може покращити вирішення складних проблем, призвести до більш ефективного прийняття рішень [14].

Однак існують також виклики з обміном знаннями. Однією із головних проблем є те, що люди можуть неохоче ділитися своїми знаннями – через страх втратити свою конкурентну перевагу або відсутність стимулів ділитися. Для подолання цієї проблеми, організації можуть створити культуру, яка цінує обмін знаннями та забезпечує стимул та визнання для людей, які діляться своїми знаннями. Інший виклик полягає в тому, що обмін знаннями (створення навчальних матеріалів або участь у навчальних сесіях) може займати багато часу та може потребувати додаткових ресурсів. Щоб вирішити цю проблему, організації можуть визначити пріоритети для обміну знаннями та відповідно розподілити ресурси [15].

3. Навчання та розвиток є важливим аспектом будь-якої організації, яка прагне максимізувати продуктивність та ефективність своїх співробітників. Навчання та розвиток можуть набувати різних форм, включаючи навчання на робочому місці, навчання в групі, наставництво, коучинг та електронне навчання.

Однією з головних переваг навчання та розвитку є те, що воно допомагає співробітникам набувати нових навичок і знань, які можуть підвищити їхню продуктивність роботи, а це допомагає організаціям покращити свої фінансові результати. Крім того, навчання та розвиток можуть допомогти організаціям зменшити витрати, завдяки зниженні необхідності в нагляді та мінімізувавши помилки.

Ще одна перевага навчання та розвитку полягає в тому, що воно може покращити рівень утримання співробітників, знизити рівень плинності кадрів, який може бути дорогим і руйнівним. Коли працівники відчувають, що роботодавець інвестує в їхній професійний розвиток, вони більш охоче залишаються в організації. Однак впровадження ефективної програми навчання та розвитку може бути складним завданням. Важливо адаптувати програму навчання та розвитку відповідно до потреб усіх працівників [14].

Ще одним викликом є забезпечення відповідності програми навчання цілям та завданням організації. Це вимагає ретельного планування та координації між командою навчання та розвитку і командою керівництва організації. Крім того, програми навчання та розвитку можуть бути дорогими та трудомісткими. Сюди необхідно віднести інвестиції в навчальні матеріали, наймання кваліфікованих тренерів і надання співробітникам достатнього часу для відвідування тренінгів.

Метод управління знаннями в проєкті полягає у використанні моделі визначеної методології управління проєктом створення програмного продукту і дозволяє визначити форми та процеси генерації, зберігання та передачі знань в межах самого проєкту та використання для подальших проєктів [11].

Метод управління знаннями відповідно до методології управління проєктами, на відміну від існуючих, включає в себе модель методології створення програмного продукту, визначені види знань для генерації, зберігання та передачі як в межах проєкту, так і для всіх стейкхолдерів проєкту, формування бази знань та рекомендаційних систем для використання в інших проєктах, що

дозволяє зберігати знання, які були згенеровані досвідом команди та сформовані у вигляді бази знань й систем рекомендацій.

2.2 Моделі управління знаннями IT-проєкту

Існує кілька моделей управління знаннями в IT-проєктах, кожна з яких має свої особливості та принципи. Ось декілька з них:

- Модель SECI;
- Модель Nonaka-Takeuchi;
- Модель KM-CMM;
- Модель OMGEssence.

Модель SECI (з англ. Socialization, Externalization, Combination, Internalization) була розроблена Ічиро Нонака та Хіроюке Такеучі у 1995 році. Вона базується на теорії вироблення знань та розвитку організаційної культури.

Етапи перетворення знань:

- Соціалізація (Socialization): Діління знань через спілкування та спостереження;
- Екстерналізація (Externalization): Перетворення імпліцитних знань в експліцитні форми, такі як документація або інструкції;
- Комбінація (Combination): Зібрання та об'єднання знань для створення нових ідей та інновацій;
- Інтерналізація (Internalization): Освоєння та внутрішнє усвідомлення знань, щоб вони стали частиною індивідуального досвіду.

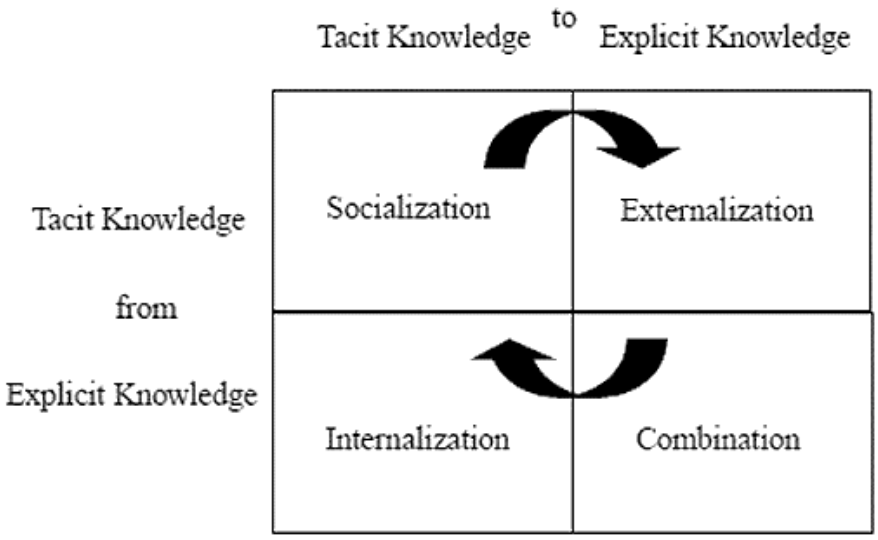


Рисунок 2.1 – Модель перетворення знань SECI

Модель SECI може бути застосована для обміну знаннями в IT-командах шляхом створення спільних просторів для обговорення, документування процесів та створення практик спільної роботи.

Модель також розроблена Ічиро Нонакою та Хіроюке Такеучі і базується на концепції вироблення знань у виробничій сфері.

Складові моделі Nonaka-Takeuchi:

- Знання процесів (Такеучі): Співпраця та обмін досвідом між працівниками;

- Знання продукту (Нонака): Створення нових знань через інновації та дослідження.

Приклади застосування: Модель Nonaka-Takeuchi може бути корисною для IT-проектів у формуванні командної культури, сприяючи співпраці та інноваціям у процесі розробки.

Модель KM-CMM – зрілості управління знаннями (Knowledge Management Capability Maturity Model) – розвивалася для оцінки та вдосконалення здатності організації управляти знаннями.

Ця модель передбачає рівні зрілості: початковий рівень, повторення процесів, визначений процес, управляючий процес, оптимізований процес.

Модель KM-CMM може бути використана для оцінки та покращення процесів управління знаннями в IT-проектах, допомагаючи організаціям досягати високого рівня ефективності та продуктивності.

Ще однією з відомих моделей є OMGEssence [16]. Ця модель використовує основи попередніх, але, крім того, може бути представлена як модель трьох рівнів:

Бізнес-модель;

Програмна (програмно-апаратна) модель;

Модель користувача.

Для створення програмного продукту важливим етапом є визначення вимог, їх узгодження, аналіз можливостей реалізації визначених вимог. Знання щодо вимог формуються на кожному етапі життєвого циклу створення програмного продукту. Крім того, відповідно до методології, фіксуються у вигляді спрінтів, завдань, звітів, міні-блогів тощо.

На рис. 2.2 представлена модель управління знаннями на основі вимог.

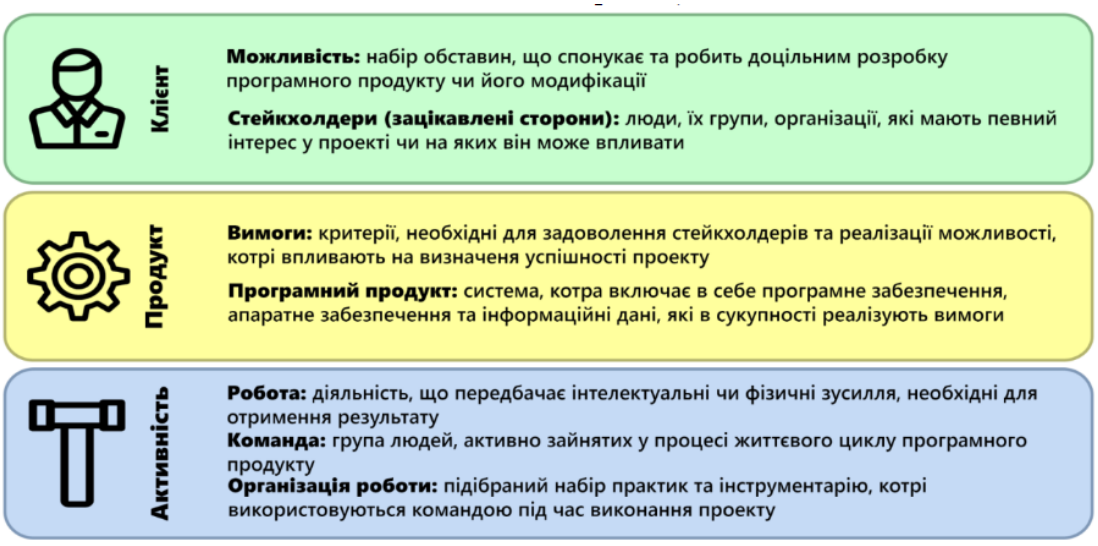


Рисунок 2.2 – Модель управління знаннями на основі вимог

Особливості представленої моделі полягають в тому, що відносно вимог, визначених замовником та розробником формують знання для реалізації

програмного продукту. Структура управління знаннями представлена у вигляді моделі на основі процесів управління вимогами (Рис. 2.3).

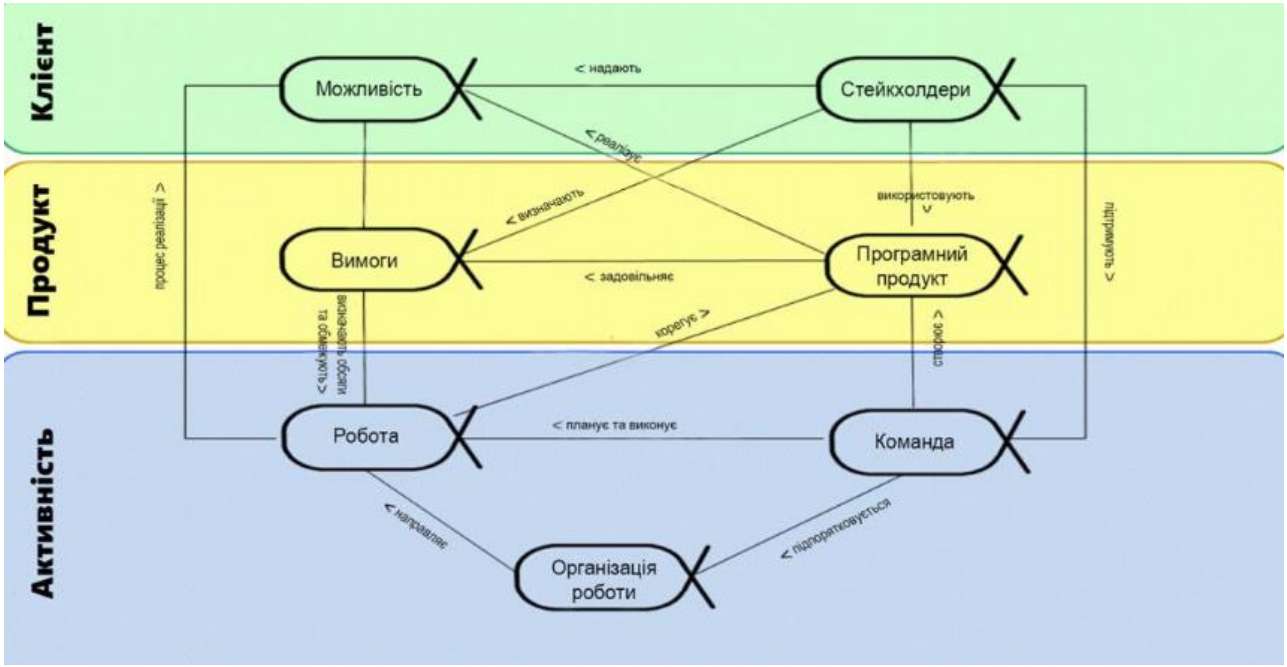


Рисунок 2.3 – Модель управління знаннями на основі вимог в трьох вимірах моделі Активність-Продукт-Клієнт

З метою підтримання знань проєкту визначемо метод управління знаннями на основі системних тріад. Системна тріада може бути представлена у вигляді трьох складових – елементність, зв'язність, цілісність.

В таблиці 2.1 представлені складові системи, яка дозволяє управляти знаннями носіїв знань – учасників команди, а також загальними знаннями, які зафіксовані у вигляді нотаток, документів, візуалізації, дошок тощо [17].

Таблиця 2.1 – Таблична модель управління знаннями за тріадою ЕЗЦ

Тріада	Виявлення знань щодо вимог	Збір знань щодо вимог	Підтримка знань	Використання

Елементність	Окремі знання потенційних користувачів Окремі знання розробників для реалізації Знання (досвід) попередніх проєктів		Генерація, зберігання, передача знань	
Зв'язність	Менеджери та фахівці проєкту			Всі учасники команди
Цілісність	Моніторинг під час всього життєвого циклу	Контроль та можливість внесення змін на кожному етапі	Бази знань, системи рекомендацій, мініблоги	В усіх процесах проєкту та на кожному етапі

Ці моделі пропонують різні підходи до управління знаннями в ІТ-проєктах, спираючись на різні теорії та концепції вироблення та передачі знань. Вибір конкретної моделі залежить від конкретних потреб та контексту проєкту.

2.3 Розробка моделі управління знаннями ІТ-проєкту

Представимо загальну модель управління знаннями в ІТ-проєкті. Для цього виконаємо уточнення поняття знань ІТ-проєкту.

Управління знаннями про проєкт – це процес використання наявних знань і створення нових для досягнення цілей проекту та сприяння навчанню організації (PMBOK 6th 2018). Знаннями ми будемо вважати знання всіх учасників команди проєкту, знання рекомендаційних систем (при наявності), знання, що зберігаються в базах знань організації або команди проєкту, знання, що генеруються при вхаємодії учасників команди з документацією проєкту та інформаційними модулями системи управління проєктом. Знання поділяються на «явні» та «неявні», а управління знаннями пов'язане з керуванням всіх видів знань. Найкращі інструменти та методи управління знаннями не працюють, якщо люди не мотивовані ділитися знаннями.

Ключовим аспектом ефективного управління знаннями є ефективний метод обміну інформацією про проєкти з іншими всередині організації, щоб дати можливість колегам виконувати такі завдання, як аналогічна оцінка витрат,

ідентифікувати спільних зацікавлених сторін або застосовувати отримані уроки в наступних проєктах. Важливо також ідентифікувати існуючі бібліотеки знань та даних, документацію. За відсутності таких ресурсів управління знаннями команди управління проектами повинні прагнути запровадити сукупні ресурси, щоб використати найкращі практики та ефективно пом'якшити очікувані/відомі ризики та проблеми, які стосуватимуться інших проектів. Критично важливі ресурси управління знаннями мають підтримку інших керівників проєктів в організації, щоб сприяти послідовному використанню доступних інструментів.

На рис. 2.4 представлена модель управління знаннями як системна модель зв'язку менеджменту знань та проєктного менеджменту.

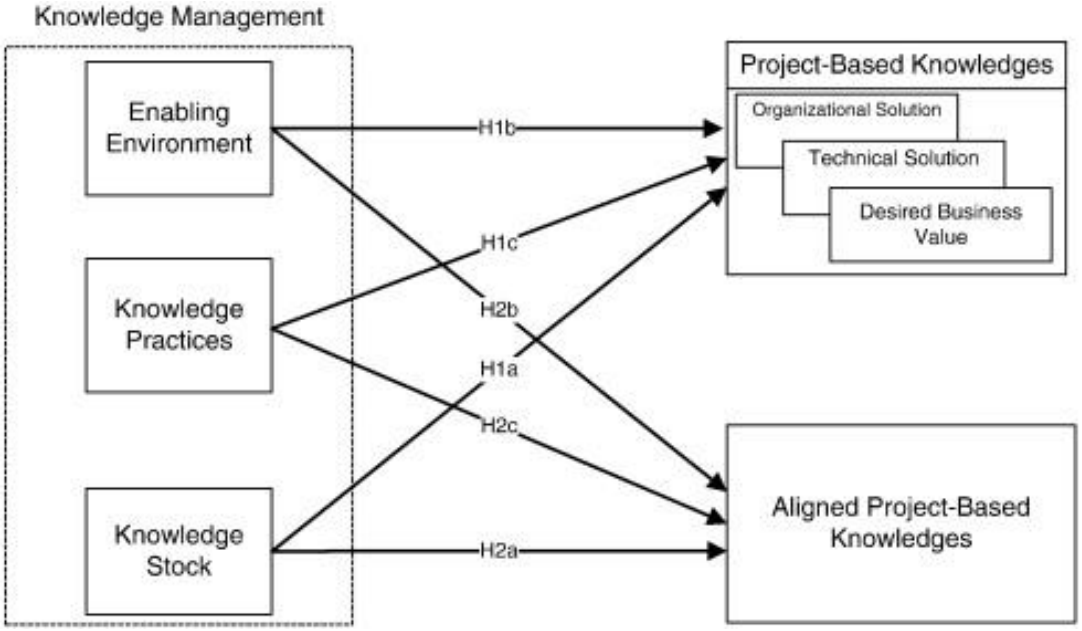


Рисунок 2.4 – Комплексна модель менеджменту знань та проєктного управління [18]

ІТ-проєкти працюють із знаннями як основним вхідним матеріалом. Крім того, оскільки команда проекту є тимчасовою організацією, члени команди можуть мати мало спільного досвіду, баз знань або процедур. Керівник проекту повинен розробити проєктне середовище, в якому знання створюються, обмінюються та використовуються для отримання результатів, бажаних

організацією-клієнтом. З цих причин ефективне управління знаннями в контексті проєкту повинно сприяти досягненню цінності проєктів.

Управління знаннями в контексті проєкту можна визначити як управлінську діяльність, необхідну для отримання необхідних знань, створення сприятливого середовища та управління практиками знань, щоб отримати узгоджений набір знань для поточного проєкту та подальшої проєктної діяльності.

Концепція наявних знань представляє загальну когнітивну здатність, доступну для проєкту на індивідуальному, груповому та організаційному рівнях проєкту.

Поняття наявних знань складається з двох частин. Перший – це запас знань, якими володіють учасники проєкту. А також те, що притаманне процесам проєкту та методам проєктування, включаючи явні знання, представлені в документах, моделях, розробках тощо, а також мета-знання, такі як карти знань. Друга частина – це потенціал збільшення (генерації) знань, включаючи здатність проєкту засвоювати різноманітні знання та використовувати їх, а також його доступ до джерел знань поза межами – досвід інших проєктів та організацій, штучний інтелект тощо. База знань у бізнес-проєктах із підтримкою IT – це знання відповідної області IT-команди, бізнес-команди та команди управління».

Сприятливе середовище в рамках IT-бізнес-проєкту – це поєднання технологічних і соціальних аспектів проєкту, які сприяють застосуванню знань .

Технологічні умови, які підтримують практики знань, складаються з фізичних ресурсів, як правило, IT-інфраструктури, включаючи комунікаційну інфраструктуру, веб-сайти проєктів, спільні сховища та інші елементи технологічної системи управління знаннями.

Соціальні умови стосуються організаційних ресурсів, як правило, організаційних структур і процесів проєкту та клімату проєкту. Організаційні структури та процеси можна розглядати як визначення формальних каналів знань, які підтримують передачу та створення знань.

Практика знань – це діяльність, яка генерує придатні для використання знання в явній або неявній формі. Література з управління знаннями пропонує високорівневі моделі практик знань Nonaka and Takeuchi, однак ці концепції важко реалізувати, і вони не використовувалися в дослідженнях управління проєктами. Практики знань можна визначити як дії, які виконуються для відображення та обміну знаннями всередині та між командами IT, бізнесу та управління в IT-проєкті. Враховуючи запропонований метод управління знаннями в проєкті доцільно визначити особливості практик знань відповідно до методології створення програмного продукту.

Управління знаннями створює певні масиви знань у межах проєкту; знання, необхідні для успішного досягнення цілей проекту. Частина цих знань залишатиметься неявною, але багато з них потрібно зробити явними, щоб їх можна було перевірити, зберегти, поділитися ними та зробити повними.

Знання про бажану бізнес-цінність можна представити як динамічне спільне розуміння бізнес-цілей, які, очікуються за результатами проєкту. Ці знання мають поширюватися серед достатньо широких груп, і вони мають бути чіткими та відповідним чином конкретними.

База знань, наявна в команді проєкту, матиме значний позитивний вплив на узгодження задокументованих знань на основі проєкту. Рівень досвіду та знань людей впливає на їхню здатність бачити зв’язок між різними артефактами проєкту, і такі знання повинні бути узгоджені між всіма учасниками проєкту.

Для удосконалення моделі будуть використані різні моделі створення програмного продукту та їх зв'язок з міжнародними стандартами управління IT-проєктами. Така модель представлена на рис. 2.5.

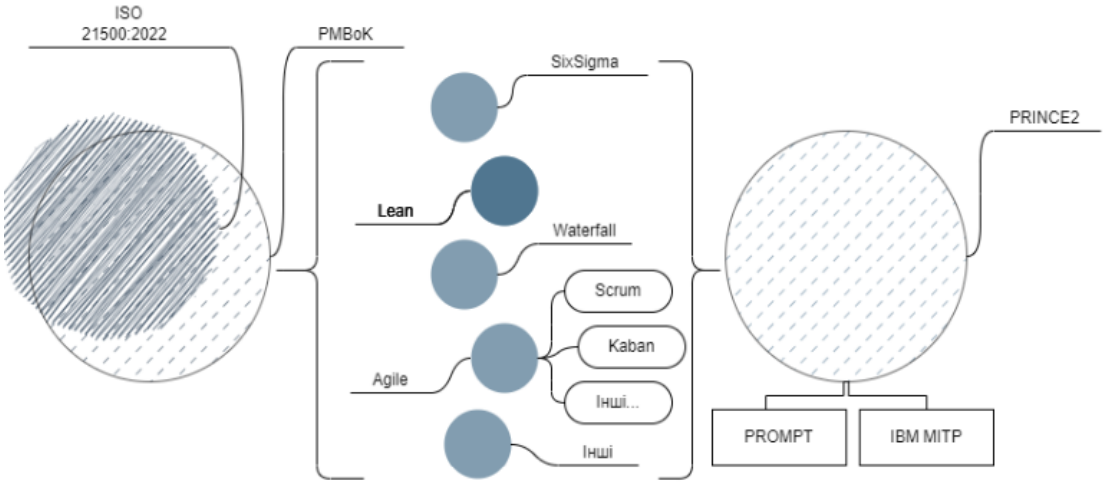


Рисунок 2.5 – Модель зв'язку методологій створення програмних продуктів з міжнародними стандартами [19]

На рисунку 2.6 представлено модель управління знаннями IT-проєкту на основі запропонованого методу.

Така модель включає в себе явні та неявні знання, знання учасників команди, задокументовані знання на етапах відповідно до методології (в даному випадку – беклог, спрінти, статут, технічне завдання), знання поточні – міні блог, записи зборів, знання виконаного проєкту (досвід проєкту у вигляді сховища знань). Запропонована модель містить модулі – поточні нотатки спрінтів, дошка завдань, база знань проєкту. Визначені модулі будуть реалізовані в межах магістерської роботи.

Математична модель може бути представлена як множинна у вигляді основних кортежів знань.

Kn_{pr}= <Kn_{kl}; Kn_t; Kn_s> ; <Kn_v; Kn_{sp}; Kn_d >; <Kn_c; Kn_{rep}; $Kn_{b;}$ Kn_{st}; Kn_l >

де Kn_{pr} – знання проєкту;

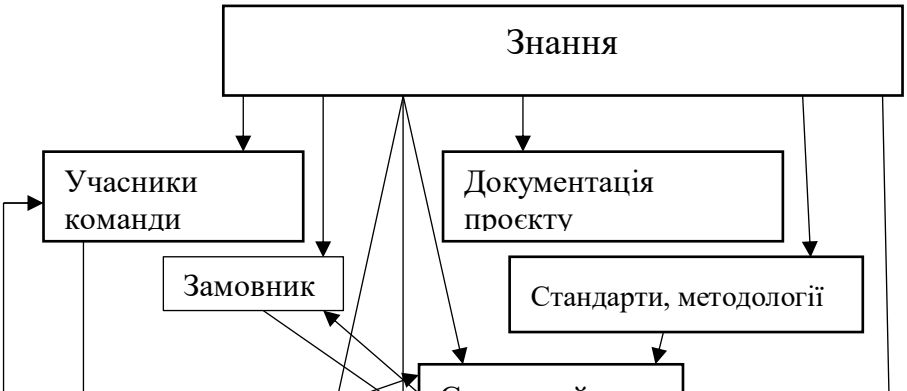


Рисунок 2.6 – Удосконалена модель управління знаннями для ІТ-проєкту

Kn_k – знання замовника;

Kn_t – знання команди;

Kn_s – знання скрам-майстра;

Kn_{spr} – знання за спрінтами;

K_v – знання за вимогами;

Kn_c – знання, згенеровані в нотатках;

Kn_{rep} – знання, згенеровані в звітах;

Kn_b – знання, згенеровані в базах знань

Kn_{st} – стандарти;

Kn_l – знання в системі навчання.

Представлена математична абстракція враховує особливості методології Scrum. Інші методології можуть бути представлені аналогічно.

Така модель дозволяє реалізувати принципи екосистеми:

1. Одноразове введення даних з подальшою обробкою, генерацією знань та доступом для всіх визначених учасників проєкту з багаторазовим використанням.

2. Забезпечення відповідності стандартам, визначеним методологіям технічних рішень створення програмного продукту.
3. Збереження проєктного та організаційного досвіду як бази знань для подальших проєктів.

Системне зберігання, обробка інформації та знань.

2.4 Висновки

В другому розділі магістерської роботи представлено метод управління знаннями ІТ-проєкту.

Запропоновано метод автоматизованого управління знаннями, який базується на принципах інформаційних екосистем, враховує методологію створення програмного продукту та реалізується на основі тріади елементність, зв'язність та цілісність, що дозволяє комплексно залучати всіх учасників ІТ-проєктів до процесу управління знаннями та покращувати якість створюваних програмних продуктів.

Проаналізовані моделі пропонують різні підходи до управління знаннями в ІТ-проєктах, спираючись на різні теорії та концепції вироблення та передачі знань. Вибір конкретної моделі залежить від конкретних потреб та контексту проєкту. В межах магістерської роботи запропоновано модель, яка, акцентує увагу на вимогах до програмного продукту та враховує модель методології створення програмного продукту, що дозволяє формалізувати знання проєкту, механізми їх генерування, зберігання та обміну для підвищення рівня ефективності реалізації поставлених цілей проєкту.

3 ПРОГРАМНІ ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

3.1 Обґрунтування вибору технологій та середовища розробки

Аналіз існуючих аналогів і потреб систем управління знаннями в IT-проєктах показав, що для задоволення потреб у функціональності та динаміці інформаційного середовища необхідно використовувати сучасні веб-технології, які підтримують динамічний контент та інтеграцію з базами даних, що містять інформацію про проєкт та його учасників, завдання та складові частини. Це дає можливість створювати гнучку і масштабовану систему, яка може ефективно обробляти та зберігати знання, а також забезпечувати їх доступність для користувачів.

Саме тому інформаційна система управління проєктами буде містити не тільки модуль управління проєктами та завданнями, а й функціональні можливості формування інформації про проєкт, набір команди та модуль нотаток для ефективного обміну та передачі інформації.

Важливою в управлінні знаннями є наскрізна автоматизація всіх процесів управління проєктами та передачі інформації, що пов’язана із плануванням, розробкою, дизайном, командою проєкту та виконанням завдань. Саме тому, повинні бути сформовані профілі кожного окремого фахівця з можливістю аналітики його діяльності та комунікації для отримання необхідної інформації.

Отже, веб-портал управління знаннями в IT-проєктах буде частиною корпоративної інформаційної системи, який реалізує можливість збереження та передачі інформації, знань та ідей пов’язаних із IT-проєктом.

Загальні алгоритми та програмні компоненти системи управління знаннями в IT-проєктах можна розділити технологічно на дві частини за моделлю клієнт-серверної архітектури – фронтенд та бекенд частини. Бекенд частина відповідає за статичну інформацію, логіку роботи системи, збереження та резервування інформації. Фронтенд частина відповідає за відображення, оновлення та представлення динамічних даних на внутрішніх та зовнішніх сайтах.

Серед основних процедур розробки необхідно виділити такі:

1. Формування основних вкладок порталу управління знаннями.
2. Визначення ролей користувачів порталу.

3. Визначення ключового функціоналу сторінок порталу управління проєктом та необхідними для його виконання знаннями.
4. Виявлення вимог до функціоналу та логіки роботи системи.
5. Визначення архітектури, моделей та полів бази даних.
6. Створення профілю користувача.
7. Формування середовища взаємодії для роботи над проєктами та при виконанні поставлених завдань.
8. Реалізація програмних компонентів згідно із методами та моделями автоматизації процесів управління знаннями.
9. Вдосконалення модулів електронного порталу управління знаннями проєкту.

Враховуючи специфіку розроблюваного програмного продукту, було прийнято рішення розподілити процес розробки на дві незалежні частини. Спочатку зробимо акцент на розробці back-end частини, де зосереджена головна логіка управління даними і запитами, а потім зосередимося на розробці front-end частини, що відповідатиме за відображення інтерфейсу користувача у вигляді веб сторінки.

Такий підхід дозволяє більш ефективно організувати роботу над проєктом, забезпечуючи високу гнучкість та масштабованість системи. Це сприятиме покращенню якості управління IT-проєктами та необхідними знаннями завдяки інтеграції сучасних технологій та інноваційних методів.

Для розробки сучасних застосунків існує безліч середовищ розробки (IDE та редакторів коду). Всі вони мають свої особливості, переваги та недоліки. Деякі з найпоширеніших середовищ розробки, які можуть бути корисними для проєкту, наведено в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика найпоширеніших середовищ розробки

Середовище	WebStorm	IntelliJ IDEA	Eclipse	Visual Studio Code
Особливості	Комерційний IDE від	Потужний IDE від JetBrains,	Відкритий IDE, популярний	Безкоштовний, легкий та

	JetBrains, спеціально розроблений для JavaScript та веб-розробки	підтримує багато мов програмування. Підтримка веб-розробки забезпечується за допомогою плагінів	серед Java-розробників, підтримує розробку для багатьох мов за допомогою плагінів	розширюваний редактор коду від Microsoft. Підтримує велику кількість розширень, які додають функціональність для різних мов програмування і технологій
Переваги	Високий рівень інтеграції з популярними фреймворками, потужні інструменти для рефакторингу та налагодження коду	Інтелектуальні інструменти для аналізу коду, рефакторингу, підтримка багатьох мов програмування та фреймворків	Висока модульність, багато плагінів для різних мов та фреймворків	Швидкість роботи, інтеграція з Git, вбудований термінал, інтелектуальні підказки коду (IntelliSense)
Недоліки	Платний, може бути важким для новачків через велику кількість функцій	Платний, споживає значні ресурси системи	Може бути важким для налаштування, споживає багато ресурсів	Може бути недостатньо функціональним без належних розширень. Але це легко вирішується їх встановленням

Кожне з цих середовищ розробки має свої особливості і може бути корисним для конкретних завдань у межах проєкту. Вибір конкретного середовища розробки залежить від індивідуальних уподобань команди розробників та специфічних вимог проєкту.

Ми зупинили свій вибір на середовищі Visual Studio Code, оскільки цей редактор коду є безкоштовним, легким та високоефективним інструментом для розробки. Він підтримує широкий спектр розширень, що дозволяє легко інтегруватися з нашими обраними технологіями, такими як Node.js, Angular та TypeScript. Крім того, VS Code має вбудовані інструменти для налагодження, інтеграцію з Git і термінал, що значно підвищує продуктивність команди розробників [20].

3.2 Розробка Back-end частини модулів управління знаннями в IT-

проєкті

На сьогоднішній день Back-end частина web-додатків є критичним компонентом, який відповідає за обробку запитів, управління бізнес-логікою та збереження даних. Вона відіграє важливу роль у забезпеченні надійності, продуктивності та безпеки додатку. Back-end забезпечує взаємодію між користувачами та сервером, обробляючи всі запити від фронтенду, який відповідає за відображення інтерфейсу користувача. Це взаємодія відбувається через API (Application Programming Interface), який надає необхідні кінцеві точки для виконання різних операцій, таких як створення, читання, оновлення та видалення (CRUD) даних.

Однією з основних задач back-end частини є забезпечення ефективного та безпечного доступу до бази даних, а також підтримка безперервності бізнес-логіки, яка керує усіма процесами всередині системи. Використання сучасних технологій, таких як Node.js, Express, MongoDB та Mongoose, дозволяє створити масштабовану та продуктивну серверну архітектуру, що відповідає вимогам сучасних IT-проєктів [21].

У даному підпункті ми розглянемо та оберемо відповідні технології, що також входить в процес розробки back-end частини, на рівні проектування бази даних та реалізації API. В подальшому особлива увага буде приділена інтеграції з фронтенд частиною, яка розроблена з використанням Angular, TypeScript та інших сучасних вебтехнологій, забезпечуючи зручний та інтерактивний інтерфейс для користувачів.

Node.js — це середовище виконання JavaScript, яке дозволяє розробникам створювати високопродуктивні та масштабовані серверні додатки. Завдяки своїй асинхронній, подіє-орієнтованій архітектурі, Node.js ефективно обробляє велику кількість одночасних запитів, що є важливим для систем управління знаннями, які потребують швидкого доступу до даних та високу надійність [22].

Express — це мінімалістичний та гнучкий фреймворк для Node.js, що спрощує розробку вебдодатків та API. Він забезпечує потужні засоби для

створення маршрутизації, обробки запитів та відповідей, а також інтеграції з іншими middleware-компонентами. Використання Express дозволяє швидко та ефективно створювати надійні серверні рішення.

Node.js та Express є потужними інструментами для розробки бекенду, які мають кілька значних переваг у порівнянні з іншими мовами та платформами для створення серверних додатків.

1. Асинхронність та подіє-орієнтована архітектура:

Node.js: Асинхронна природа Node.js дозволяє обробляти велику кількість одночасних запитів з мінімальними затримками, що робить його ідеальним для застосунків, які потребують високої продуктивності та швидкості обробки запитів. У той час як традиційні сервери, такі як Apache з PHP, працюють з багатопоточністю, Node.js використовує однопоточний підхід з асинхронним введенням/виведенням, що знижує накладні витрати та покращує масштабованість. Java та .NET також підтримують асинхронні операції, але їх реалізація зазвичай більш складна і вимагає більше коду та налаштувань [23].

2. Єдина мова для клієнтської та серверної частини:

Node.js: Використання JavaScript як на клієнтській, так і на серверній частинах дозволяє зменшити складність розробки та полегшує обмін знаннями між фронтенд та бекенд розробниками. Це сприяє швидшому та ефективнішому розвитку проєкту. Використання різних мов програмування для фронтенду (JavaScript) та бекенду (наприклад, PHP, Java, Ruby або Python) може ускладнити процес розробки та вимагає більшого часу на узгодження між командами.

3. Висока продуктивність та ефективність:

Завдяки V8, JavaScript-двигуну від Google, Node.js забезпечує високу швидкість виконання коду. Це особливо важливо для вебзастосунків, які потребують швидкої обробки даних та високої продуктивності. Хоча такі мови як Java та C# також відомі своєю продуктивністю, їх використання може бути більш ресурсоємним та вимагати більше часу на налаштування середовища розробки.

4. Масштабованість та модульність:

Модульна структура Node.js дозволяє легко розширювати функціонал додатку за рахунок численних пакетів з npm (Node Package Manager). Це спрощує додавання нових можливостей та підтримку коду. У таких мовах як Java та .NET також є свої системи пакетів (Maven, NuGet), але їх екосистеми можуть бути більш складними для початкової конфігурації та інтеграції нових модулів.

5. Підтримка великої спільноти та наявність багатьох бібліотек:

Завдяки великій та активній спільноті розробників, Node.js має велику кількість готових бібліотек та модулів, які можуть значно прискорити розробку та знизити вартість проекту. Інші мови та платформи також мають великі спільноти, але Node.js виділяється швидкістю розвитку та кількістю доступних відкритих проектів [23].

Таким чином, вибір Node.js та Express для розробки бекенду забезпечує високий рівень продуктивності, масштабованості та гнучкості, що є критично важливим для ефективного функціонування. Вони дозволяють створити швидкий та надійний серверний додаток, який легко підтримувати та розширювати відповідно до потреб користувачів.

MongoDB — це документно-орієнтована NoSQL база даних, яка забезпечує високу гнучкість та масштабованість при роботі з великими обсягами даних. Вона дозволяє зберігати інформацію у вигляді JSON-подібних документів, що ідеально підходить для динамічних та нерегламентованих даних, які характерні для систем управління знаннями [25].

Mongoose — це ORM-бібліотека для MongoDB та Node.js, яка надає елегантний спосіб роботи з базою даних, включаючи визначення схем, валідацію та обробку даних. Mongoose дозволяє створювати потужні моделі даних та забезпечує інтуїтивно зрозумілий API для взаємодії з MongoDB.

Вибір MongoDB та Mongoose для розробки бекенд частини вебдодатку, особливо у поєднанні з Node.js та Express, має значні переваги у порівнянні з іншими реляційними та нереляційними базами даних [23]. Нижче наведено кілька ключових аспектів, що пояснюють цей вибір.

1. Відмінна інтеграція з Node.js та Express:

MongoDB використовує документи у форматі BSON (бінарний JSON), що дуже схоже на JSON, який природно підтримується в JavaScript. Це спрощує передачу даних між клієнтом і сервером, зменшуючи потребу в додаткових перетвореннях.

Node.js відомий своєю асинхронною архітектурою. MongoDB, разом з Mongoose, підтримує асинхронні операції, що дозволяє ефективно керувати запитами до бази даних без блокування основного потоку.

Хоча є бібліотеки для роботи з реляційними базами даних (наприклад, MySQL, PostgreSQL) у Node.js, такі як Sequelize або TypeORM, робота з реляційними даними може бути складнішою через необхідність маппінгу об’єктно-реляційних відношень. Інші нереляційні бази даних (наприклад, Cassandra, CouchDB): Можуть бути менш інтегрованими з Node.js та вимагати додаткових налаштувань для ефективної роботи [24].

2. Гнучкість і масштабованість:

MongoDB зберігає дані у вигляді документо-орієнтованої структури, що дозволяє зберігати різнотипні дані в одній колекції. Це особливо корисно для динамічних додатків, де структура даних може змінюватися. MongoDB підтримує горизонтальне масштабування за допомогою шардингу, що дозволяє ефективно керувати великими обсягами даних.

Реляційні бази даних вимагають визначення суворої схеми даних, що ускладнює роботу з динамічними даними. Масштабування може бути складнішим через необхідність синхронізації даних між кількома серверами. Інші нереляційні бази даних можуть також підтримувати горизонтальне масштабування, але часто не мають такої гнучкості у роботі з різнорідними даними, як MongoDB.

3. Простота розробки та швидкість:

Завдяки простій структурі даних і гнучкості MongoDB та Mongoose, розробники можуть швидко розпочати роботу без необхідності складного налаштування схеми. Mongoose, як ORM для MongoDB, дозволяє легко створювати і маніпулювати моделями даних, забезпечуючи валідацію та

цілісність даних на рівні додатка.

Реляційні бази даних потребують більше часу на створення та підтримку складних схем і міграцій бази даних. Інші нереляційні бази даних можуть не мати таких потужних інструментів, як Mongoose, для зручної роботи з даними, що може збільшити час розробки.

4. Висока продуктивність:

MongoDB забезпечує високу швидкість читання і запису, особливо для додатків з великими обсягами даних і складними запитами. Підтримка різних типів індексів допомагає оптимізувати запити і покращувати продуктивність додатка.

Реляційні бази даних можуть забезпечувати високу продуктивність, але зазвичай вимагають більш складної оптимізації запитів і налаштування індексів. Продуктивність інших нереляційних баз даних може варіюватися в залежності від конфігурації та конкретного випадку використання, але MongoDB зазвичай забезпечує більш передбачувані результати.

5. Підтримка та спільнота:

MongoDB та Mongoose мають велику і активну спільноту розробників, що забезпечує доступ до численних ресурсів, документації та прикладів коду. Велика кількість готових рішень та бібліотек полегшує розробку та прискорює вирішення поширених проблем.

Реляційні бази даних також мають великі спільноти, але через складнішу архітектуру можуть вимагати більше часу на освоєння. Інші нереляційні бази даних мають менші спільноти, що може ускладнити пошук ресурсів та прикладів для розробки.

Для розробки застосунку використано MongoDB Cloud. Хмарну платформу, яка включає кілька сервісів, таких як Atlas (хмарна база даних), Realm (платформа для мобільних додатків) та Data Lake (зберігання і аналіз даних). Основним сервісом для зберігання та управління базами даних є MongoDB Atlas.

Наведемо переваги Використання MongoDB Cloud.

1. Масштабованість та гнучкість: MongoDB Cloud дозволяє автоматично масштабувати ресурси в залежності від навантаження, забезпечуючи оптимальну продуктивність. Можливість вибору конфігурацій для різних вимог додатків.

2. Безпека та надійність: Платформа пропонує вбудовані засоби шифрування даних, аутентифікацію та контроль доступу. Автоматизовані резервні копії та можливість швидкого відновлення даних забезпечують високу надійність.

3. Зручність адміністрування: MongoDB Cloud забезпечує автоматизоване управління оновленнями та налаштуваннями баз даних, знижуючи навантаження на адміністраторів. Вбудовані інструменти для моніторингу продуктивності та аналітики допомагають ефективно управляти ресурсами.

4. Інтеграція з іншими сервісами: MongoDB Cloud легко інтегрується з іншими хмарними сервісами, такими як AWS, Google Cloud Platform та Microsoft Azure. Інтеграція з CI/CD пайплайнами та підтримка різних мов програмування сприяють швидкому та ефективному циклу розробки.

Використання MongoDB Cloud дозволяє компаніям зосередитися на розробці додатків, забезпечуючи при цьому надійне, безпечне та масштабоване управління даними в хмарі.

Таким чином, вибір MongoDB та Mongoose у поєднанні з Node.js та Express є виваженим та обґрунтованим з багатьох причин. MongoDB забезпечує високу гнучкість завдяки своїй документо-орієнтованій структурі, що дозволяє легко адаптуватися до змін у вимогах проекту. Висока продуктивність MongoDB, зокрема швидкість операцій читання та запису, є критично важливою для додатків з великими обсягами даних. Mongoose значно спрощує розробку завдяки потужним інструментам для моделювання даних та забезпечення їх цілісності. Інтеграція з Node.js та Express дозволяє ефективно використовувати асинхронні можливості платформи, що забезпечує високу продуктивність і масштабованість додатків.

Окрім того, використання JavaScript як на серверній, так і на клієнтській стороні сприяє єдності коду та спрощує процес розробки, зменшуючи кількість

помилок та підвищуючи ефективність розробника. Враховуючи всі ці фактори, MongoDB та Mongoose у поєднанні з Node.js та Express є оптимальним вибором для створення сучасного, високопродуктивного та гнучкого вебдодатку.

3.3 Розробка архітектури back-end частини інформаційної системи управління знаннями в IT-проєктах

Оскільки розроблювана інформаційна система планується досить комплексною, необхідно обрати правильну та надійну архітектуру. Архітектура бекенд-частини програмного забезпечення визначає, як компоненти сервера взаємодіють між собою та з іншими частинами системи. Існує кілька основних видів архітектури бекенд-частини:

1. Монолітна архітектура – всі компоненти сервера (автентифікація, бізнес логіка, обробка запитів, робота з базою даних) інтегровані в єдиний додаток. Монолітний застосунок – це додаток, що доставляється через єдине розгортання. Це застосунок доставлений у вигляді однієї WAR або застосунок Node із однією точкою входу. Додаток легко розробляти, та розгортати, але складно масштабувати. Схематично монолітну архітектуру зображено на рисунку 3.1.

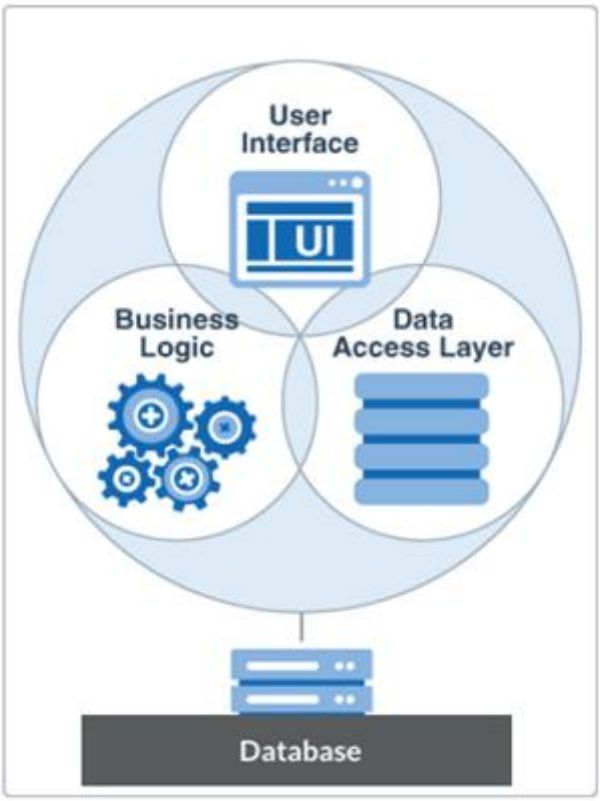


Рисунок 3.1 – Схема монолітної архітектури

2. Мікросервісна архітектура передбачає, що додаток розділений на невеликі незалежні сервіси, кожен з яких виконує одну функцію і може розроблятися, розгортатися та масштабуватися незалежно. Але ним складно управляти через необхідність у складній інфраструктурі для взаємодії сервісів (наприклад, API Gateway, сервіс Mesh). Схема мікросервісної архітектури зображена на рисунку 3.2.

3. Архітектура з використанням безсерверних технологій (Serverless) виконання функцій на вимогу в безсерверному середовищі, де інфраструктура автоматично управляється постачальником хмарних послуг (наприклад, AWS Lambda, Azure Functions). Перевагою є відсутність необхідності в управлінні серверами, але можливі затримки при холодному старті та існує обмеження на тривалість виконання функцій, складність у налагодженні та моніторингу [26]. Схема безсерверної архітектури зображена на рисунку 3.3.

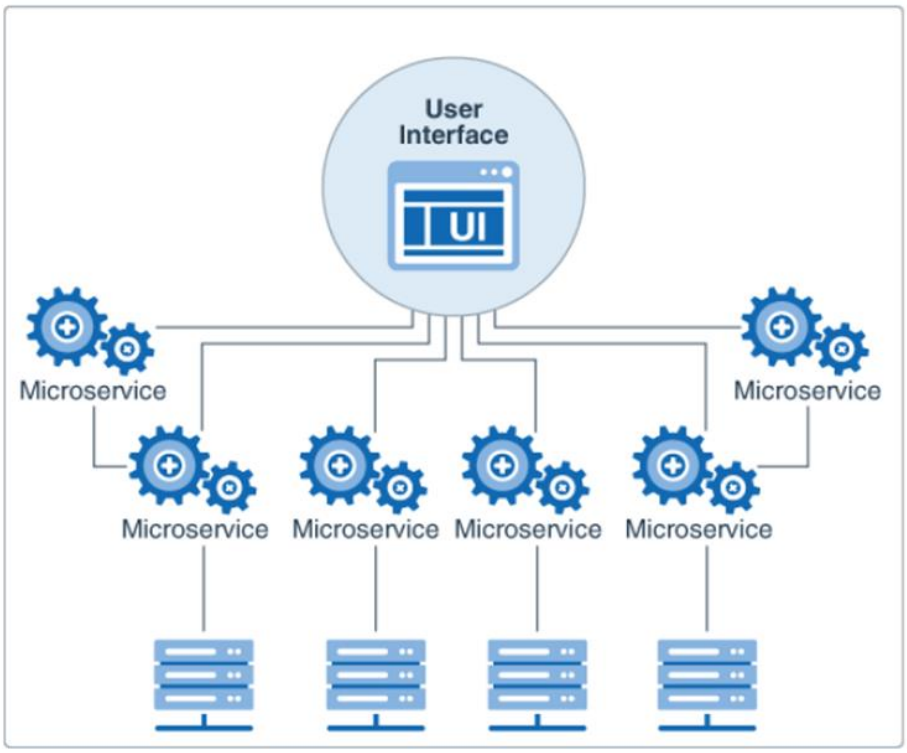


Рисунок 3.2 – Схема мікросервісної архітектури

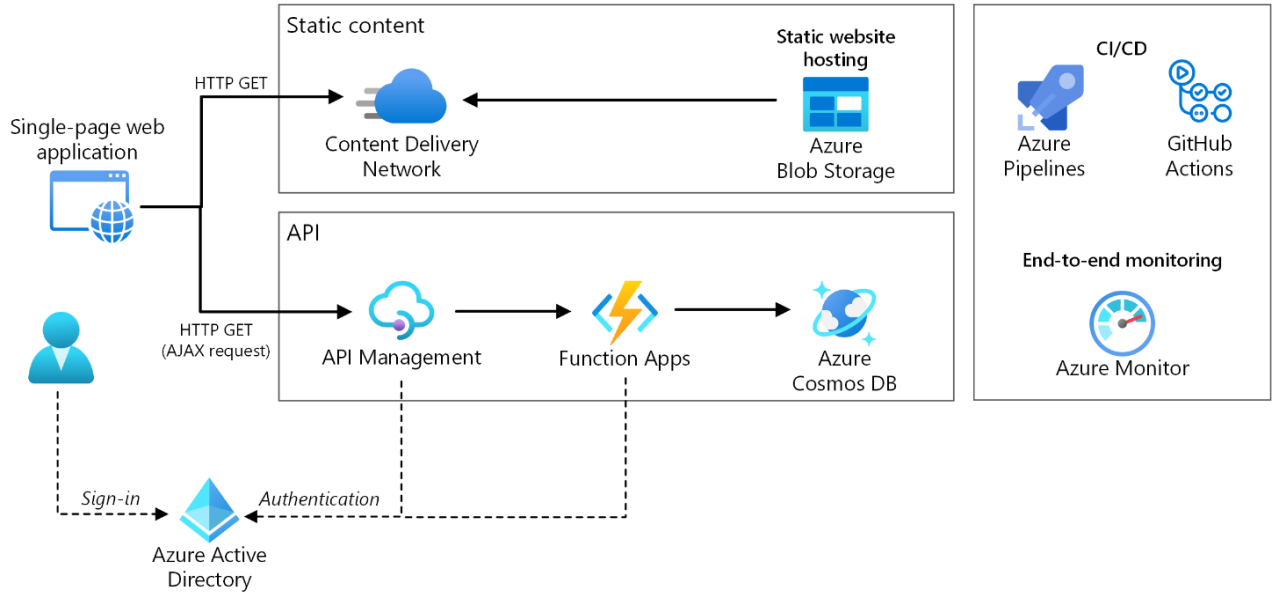


Рисунок 3.3 – Схема безсерверної архітектури

4. Трирівнева (three-tier) архітектура включає: рівень презентації (Presentation Tier), що відповідає за взаємодію з користувачем, відображення інтерфейсу та обробку введених даних; логічний рівень (Logic Tier), що обробляє бізнес-логіку додатку, обробку запитів, маршрутизацію та взаємодію з базою даних. Рівень даних (Data Tier) відповідає за зберігання та управління даними

[27]. Схема трирівневої архітектури наведено на рисунку 3.4.

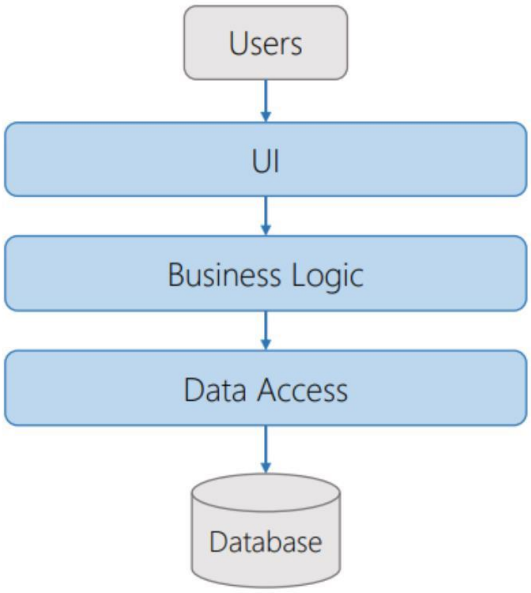


Рисунок 3.4 – Схема трирівневої архітектури

Переваги трирівневої архітектури:

- Кожен рівень відповідає за конкретний аспект додатку, що спрощує розробку, тестування та підтримку.

- Можливість незалежного масштабування кожного рівня відповідно до навантаження.

- Можливість заміни або оновлення одного з рівнів без значного впливу на інші забезпечує гнучкість.

Кожен вид архітектури має свої переваги та недоліки, і вибір залежить від конкретних вимог проекту, масштабу, необхідного рівня масштабованості, швидкості розробки та інших факторів.

Трирівнева архітектура з використанням Angular на фронтенді, Node.js з Express на бекенді та MongoDB з Mongoose і MongoDB Cloud на рівні даних є потужним рішенням для розробки сучасних веб-додатків. Angular дозволяє створювати динамічні та інтерактивні веб-додатки з використанням компонентного підходу, як рівень презентації. Node.js та Express на логічному рівні забезпечують ефективну та швидку роботу серверної частини завдяки своїй

асинхронній природі. Рівень даних виконує MongoDB, як нереляційна база даних, що забезпечує гнучкість у роботі з великими обсягами даних та зберіганні документів у форматі JSON. Mongoose використовується для моделювання даних та забезпечення валідації. MongoDB Cloud надає можливості для хмарного зберігання та масштабування бази даних. Ця комбінація технологій забезпечує високу продуктивність, масштабованість та гнучкість, що робить її оптимальною для різних типів проектів, включаючи розроблюваний додаток.

3.4 Розробка бази даних системи управління проєктами та знаннями

Ефективне управління проєктами та знаннями вимагає надійної та гнучкої бази даних, яка може забезпечити зберігання, обробку та аналіз великих обсягів інформації. Для розроблюваної системи управління проєктами та знаннями було обрано нереляційну базу даних MongoDB у поєднанні з Mongoose та MongoDB Cloud. Тобто дані будуть зберігатися у документах, зовнішніх файлах, а не у вигляді таблиць, що пов’язані між собою, як у реляційних базах даних. Це рішення дозволяє створити структуру бази даних, яка відповідає сучасним вимогам до зберігання документів, забезпечує високу продуктивність і надійність, а також підтримує масштабованість та безпеку даних у хмарному середовищі. Розглянемо процес проектування структури бази даних, основні колекції та їх поля, оскільки необхідно детально обміркувати, яка інформація повинна зберігатися [28].

Враховуючи запланований функціонал, в базі даних потрібно забезпечити зберігання інформації про зареєстрованих користувачів, існуючі проєкти, нотатки, що належать певному користувачу та відносяться до певного проєкту, коментарі, що відносяться до певної нотатки. Також необхідно зберігати категорії нотаток, статуси завдань конкретного проєкту.

Основна взаємодія буде відбуватися із працівниками, проєктами та нотатками. Однією із головними сутностей буде саме інформація про користувача (User), що включатиме в себе основні дані про працівника: ім’я,

прізвище, номер телефону, дату народження, електронну адресу, місто та країну проживання. У діючих працівників є інформація стосовно їхнього поточного проєкту, до якого вони відносяться, їхньої ролі в проєкті та статус адміна в додатку. Після створення сутності користувача, можна створити інші сутності для детального комплексного процесу роботи із інформацією.

Нотатка (Note) включатиме в себе інформацію про заголовок, контент, зображення (необов'язково), категорію, до якої відноситься нотатка, ідентифікатор проєкту, до якого відноситься, та ідентифікатор автора нотатки, дату створення та дату оновлення.

Документ коментаря до нотатки (Comment) містить в собі поля з назвою завдання, його текстом, зображенням до нього (за потребою), ідентифікаторами автора та нотатки, до якої він прив'язаний, дату створення та його оновлення.

Інформація про проєкт (Project) містить в собі поле імені проєкту, дату початку та кінця, склад команди та завдання, що відносяться до цього проєкту.

Завдання (Task) містить інформацію з назвою, текстом завдання, ідентифікаторами відповідального, виконавця та проєкту, до якого відноситься, початком та кінцем виконання завдання, його статусом.

Для уникнення повторень в присвоєнні категорії або статусу виконання, сутності нотаток та завдань мають службові сутності, на які вони посилаються – категорії (Categories) та статус завдання (TaskStatuses).

Вся ця інформація буде зберігатися у окремих документах формату BSON (бінарна версія JSON). Схему бази даних наведено на рисунку 3.5.

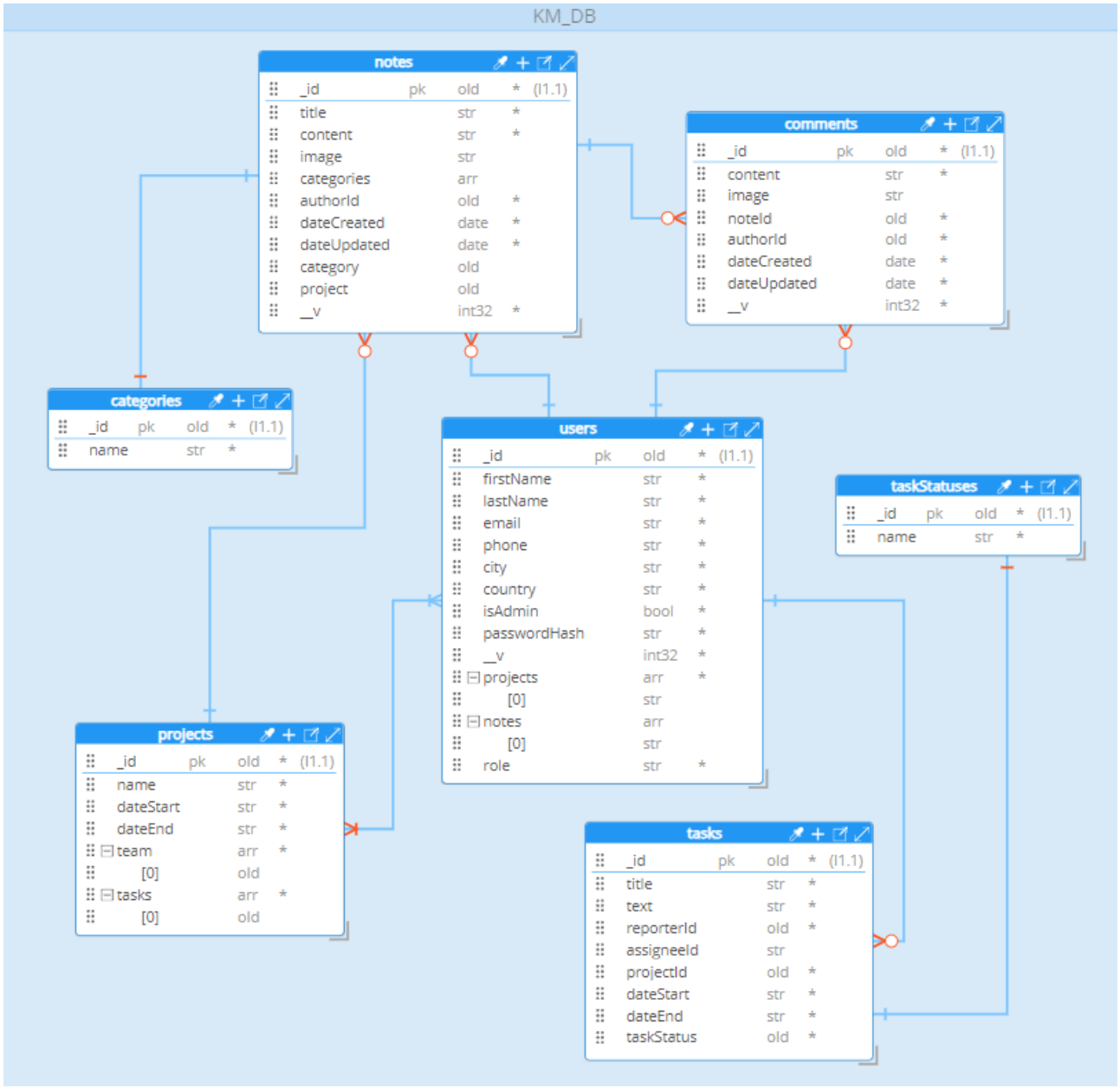


Рисунок 3.5 – Схема бази даних системи управління знаннями

Впровадження бази даних системи управління знаннями на основі MongoDB забезпечує ефективну та масштабовану платформу для зберігання та обробки різноманітної інформації необхідної для управління проєктами, включаючи дані про користувачів, проєкти, нотатки, коментарі та завдання. Взаємодія між сутностями, така як посилання на користувачів у проєктах та нотатках, а також категорії та статуси завдань, забезпечує комплексний підхід до організації даних, сприяючи ефективному управлінню та обміну знаннями в IT-

проєктах. Обрані технології дозволяють створити гнучку і високопродуктивну систему, яка відповідає сучасним вимогам до управління даними. Використання MongoDB Cloud додатково гарантує надійність та безпеку даних, а також спрощує процеси резервного копіювання і масштабування. Завдяки цьому, система буде здатна ефективно підтримувати зберігання та обробку великих обсягів інформації про користувачів, проєкти, нотатки та коментарі, забезпечуючи при цьому високий рівень продуктивності та надійності для користувачів і розробників.

3.5 Розробка програмної компоненти back-end частини системи управління знаннями

Під час розробки будь-якої програмної системи важливо дотримуватися певних принципів щодо вибраної архітектури для забезпечення її правильної та послідовної реалізації. Оскільки було вирішено дотримуватися трирівневої архітектури, розпочнемо процес розробки зі створення основної логіки програми, яка забезпечує обробку запитів, управління даними та взаємодію з базою даних. Розробка програмної компоненти back-end частини системи управління знаннями включає налаштування сервера, створення моделей даних, реалізацію контролерів та маршрутів, а також забезпечення безпеки системи.

На початковому етапі створення та налаштування сервера на основі Node.js та Express.js встановлюється базова структура проекту, яка включає такі основні директорії: models, routes, та helpers. У файлі app.js налаштовується основний сервер, який буде обробляти запити від клієнтів. Фрагмент коду із налаштуванням сервера зображено на рисунку 3.7.

Оскільки ми застосували у проекті MongoDB Cloud, то необхідно створити підключення бази даних до проекту і роботи із нею у файлі .env (рисунок 3.6).

```
CONNECTION_STRING = 'mongodb+srv://kirlyubov:d[redacted]u@cluster0.joi5uck.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0'
```

Рисунок 3.6 – Підключення бази даних

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const morgan = require("morgan");
const mongoose = require("mongoose");
const cors = require("cors");
require("dotenv/config");
const authJwt = require("./helpers/jwt");
const errorHandler = require("./helpers/error-handler");

app.use(cors());
app.options("*", cors());

//middleware
app.use(express.json());
app.use(morgan("tiny"));
app.use(authJwt());
app.use("/public/uploads", express.static(__dirname + "/public/uploads"));
app.use(errorHandler);

//Database
mongoose
  .connect(process.env.CONNECTION_STRING, {
    dbName: "KM_DB",
  })
  .then(() => {
    console.log("Database Connection is ready...");
  })
  .catch((err) => {
    console.log(err);
  });

//Server
app.listen(3000, () => {
  console.log("server is running http://localhost:3000");
});
```

Рисунок 3.7 – Фрагмент коду із налаштуванням сервера

На наступному етапі розробляються моделі даних за допомогою Mongoose. Ці моделі визначають структуру документів у базі даних MongoDB. Для цього було створено конфігураційні файли для кожної сутності, в яких описуються назви полів і їхні типи. Програмний код моделей серверної частини наведено в додатку Г. Приклад коду для генерації сутності User зображено на рисунку 3.8.

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    email: { type: String, required: true },
    phone: { type: String, required: true },
    city: { type: String, default: '' },
    country: { type: String, default: '' },
    isAdmin: { type: Boolean, default: false },
    passwordHash: { type: String, required: true },
    role: { type: String, default: '' },
    projects: [
        { type: mongoose.Schema.Types.ObjectId, ref: 'Project' }
    ]
});

userSchema.virtual('id').get(function () {
    return this._id.toHexString();
});

userSchema.set('toJSON', {
    virtuals: true,
});

exports.User = mongoose.model('User', userSchema);
exports.userSchema = userSchema;
```

Рисунок 3.8 – Фрагмент коду для генерації сутності User

Наступним етапом розробляється рівень бізнес логіки. Контролери відповідають за обробку логіки запитів, тоді як маршрути визначають шляхи для цих запитів. У контроллерах прописано можливі http відповіді від сервера. Наприклад, контролер для користувачів може включати функції для створення, оновлення та видалення користувачів.

Приклад одного із методів контроллера, що відповідає за роботу із нотатками наведено на рисунку 3.9.

Маршрути відповідають за обробку запитів від користувачів та спрямування їх до відповідних контролерів. Це дозволяє забезпечити чітку структуру та легкість у підтримці коду, а також спрощує додавання нових функціональних можливостей до системи. Кожен маршрут включає метод HTTP (GET, POST, PUT, DELETE) та шлях, який відповідає певній операції або ресурсу, що підлягає обробці. Наведемо маршрути, що визначають шляхи, за якими ці функції доступні (рисунок 3.10).

```
const express = require('express');
const router = express.Router();
const multer = require('multer');
const { Note } = require('../models/note');
const { Comment } = require('../models/comment');

router.post('/', uploadOptions.single('image'), async (req, res) => {
    const file = req.file;
    if (!file) return res.status(400).send('No image in the request');

    const fileName = file.filename;
    const basePath = `${req.protocol}://${req.get('host')}/public/uploads/`;

    try {
      let note = new Note({
        title: req.body.title,
        content: req.body.content,
        image: `${basePath}${fileName}`,
        category: req.body.category,
        authorId: req.body.authorId,
        project: req.body.project
      });
      note = await note.save();
      if (!note) {
        return res.status(400).send('The note cannot be created!');
      }
      res.status(201).send(note);
    } catch (error) {
      res.status(500).json({ message: 'An error occurred while creating the note.', error: error.message });
    }
});
```

Рисунок 3.9 – Фрагмент контроллера із методом збереження нотатки

```
const categoriesRoutes = require("./routes/categories");
const projectsRoutes = require("./routes/projects");
const notesRoutes = require("./routes/notes");
const usersRoutes = require("./routes/users");

const api = process.env.API_URL;

app.use(`${api}/users`, usersRoutes);
app.use(`${api}/categories`, categoriesRoutes);
app.use(`${api}/notes`, notesRoutes);
app.use(`${api}/projects`, projectsRoutes);
```

Рисунок 3.10 – Фрагмент коду із маршрутами

Важливим аспектом є забезпечення безпеки системи, включаючи автентифікацію та авторизацію користувачів. В контролерах додаються функції для реєстрації та входу користувачів з генеруванням токенів. Для цього використані такі бібліотеки, як jsonwebtoken для роботи з JWT-токенами, та

bcrypt для хешування паролів (рисунок 3.11).

```
const {User} = require('../models/user');
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { Project } = require('../models/project');

router.post('/', async (req,res)=>{
    let user = new User({
        firstName: req.body.name,
        lastName: req.body.name,
        email: req.body.email,
        passwordHash: bcrypt.hashSync(req.body.password, 10),
        phone: req.body.phone,
        isAdmin: req.body.isAdmin,
        city: req.body.city,
        country: req.body.country,
    })
    user = await user.save();

    if(!user)
    return res.status(400).send('the user cannot be created!')

    res.send(user);
})
```

Рисунок 3.11 – Приклад використання бібліотек jsonwebtoken та bcrypt

Розробка програмної компоненти back-end частини системи управління знаннями успішно завершена, забезпечуючи основу для подальшої інтеграції з фронтенд частиною. Створені модулі та взаємодія через REST API гарантують надійне зберігання, обробку та передачу даних, необхідних для ефективного управління проєктами та знаннями. Це дозволяє нам зосередитися на подальших етапах розробки, включаючи інтеграцію користувацького інтерфейсу та реалізацію повного функціоналу системи.

3.6 Розробка фронтенд частини інформаційної системи управління знаннями

Платформа та фреймворк Angular розроблений і підтримуваний Google. Angular дозволяє створювати динамічні, односторінкові вебдодатки (SPA) з використанням TypeScript. Він надає розробникам засоби для розробки як складних, так і простих додатків, забезпечуючи високу продуктивність і зручність у використанні [29].

Однією з ключових характеристик даного фреймворку є здатність забезпечувати взаємодію між моделлю та відображенням, де будь-яка зміна в моделі автоматично відображається на інтерфейсі, і навпаки. Це критично важливо для підтримки безперервного зв'язку між діями користувача та станом системи. Наприклад, при введенні даних користувачем у текстове поле, модель оновлюється в реальному часі, після чого відправляється запит на сервер. Отримана відповідь коригує модель, що, в свою чергу, автоматично оновлює відображення на інтерфейсі. Цей механізм відомий як «двонаправлене зв'язування даних» (two-way data binding) [30].

Однією з визначальних особливостей Angular є механізм впровадження залежностей (dependency injection), що значно полегшує управління залежностями у додатку. Цей механізм дозволяє централізовано створювати та використовувати services, забезпечуючи високу гнучкість та можливість повторного використання коду. Коли компонент або сервіс потребує певної залежності, Angular автоматично надає її через спеціальний механізм впровадження, що дозволяє уникнути жорсткого зв'язування між компонентами та залежностями. Це покращує тестованість коду та сприяє більшій модульності та розширюваності додатку. Таким чином, впровадження залежностей є ключовим елементом архітектури Angular, що забезпечує ефективне управління складними структурами та взаємодією компонентів додатку.

Angular використовує сувору структуру проекту, яка включає в себе модулі, компоненти, сервіси та інші елементи. Це полегшує підтримку великих проектів і сприяє кращій організації коду. Фреймворк забезпечує двостороннє зв'язування даних «з коробки», що спрощує синхронізацію між моделлю і представленням. Angular має потужні вбудовані засоби для роботи з формами та

їх валідації. Платформа активно підтримується Google і має велику спільноту розробників, що забезпечує стабільні оновлення, виправлення помилок і зручну та багату документацію.

Angular пропонує широкий набір додаткових інструментів, що робить його потужним інструментом для розробки сучасних веб-додатків. Серед них:

- Модуль анімацій (Animations), який дозволяє створювати ефектні анімаційні переходи та ефекти.

- Тест-раннер Karma, що використовується для виконання unit-тестів, забезпечуючи надійне тестування компонентів.

- Фреймворк для unit-тестів Jasmine, який надає потужні можливості для написання та виконання модульних тестів.

- Фреймворк Protractor для проведення end-to-end (e2e) тестування, забезпечуючи комплексне тестування функціоналу додатку.

- Підтримка локалізації та інтернаціоналізації, що дозволяє адаптувати додаток до різних мов та регіонів, забезпечуючи глобальну доступність.

- HTTP-клієнт, який включає модулі або бібліотеки для клієнт-серверної взаємодії, аналогічно до fetch або XHR, що спрощує роботу з API.

- Бібліотека Router для керування браузерною історією та виконання навігації між різними адресами всередині додатку.

Завдяки цим перевагам, Angular став оптимальним вибором для розробки фронтенд частини інформаційної системи управління знаннями, забезпечуючи високу продуктивність, зручність у використанні та легкість в підтримці.

Фрагмент організації папок проєкту представлений на рисунку 3.12.

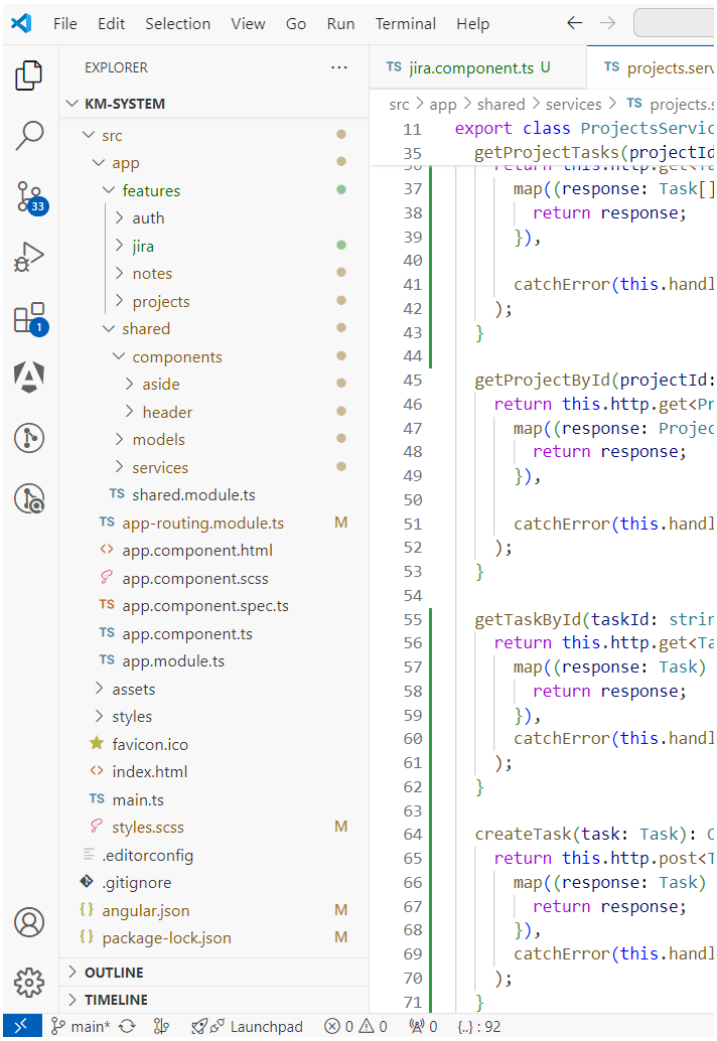


Рисунок 3.12 – Фрагмент організації структури папок проєкту

Фронтенд частина презентує основний функціонал:

- сторінки реєстрації та автентифікації;
- сторінку із доступними проєктами;
- сторінки із нотатками користувача;
- сторінку із модальною формою для створення нотатки;
- можливість додавання коментарів до нотаток;
- сторінку із завданнями, що відносяться до проєкту;
- сторінку окремого завдання;
- модальну форму для створення завдання.

Першим кроком створення інтерфейсу користувача є імплементація

сторінок реєстрації та авторизації в системі. В реєстраційній формі необхідно заповнити поля з іменем, прізвищем, електронною поштою, телефоном, країною та містом проживання, вказати, чи буде користувач адміном та ввести пароль і поле підтвердження паролю. В формі авторизації необхідно заповнити поля із електронною поштою та паролем. Обробляє введену в компоненти інформацію AuthService, що зв'язується із бекендом, де перевіряється введена інформація, і повертається відповідний респонс. Він зберігає інформацію про успішно залогіненого користувача у currentUserSubject, що дає доступ до інформації про користувача із будь-якого місця застосунку. На рисунку 3.13 подано зображення форм реєстрації та авторизації.

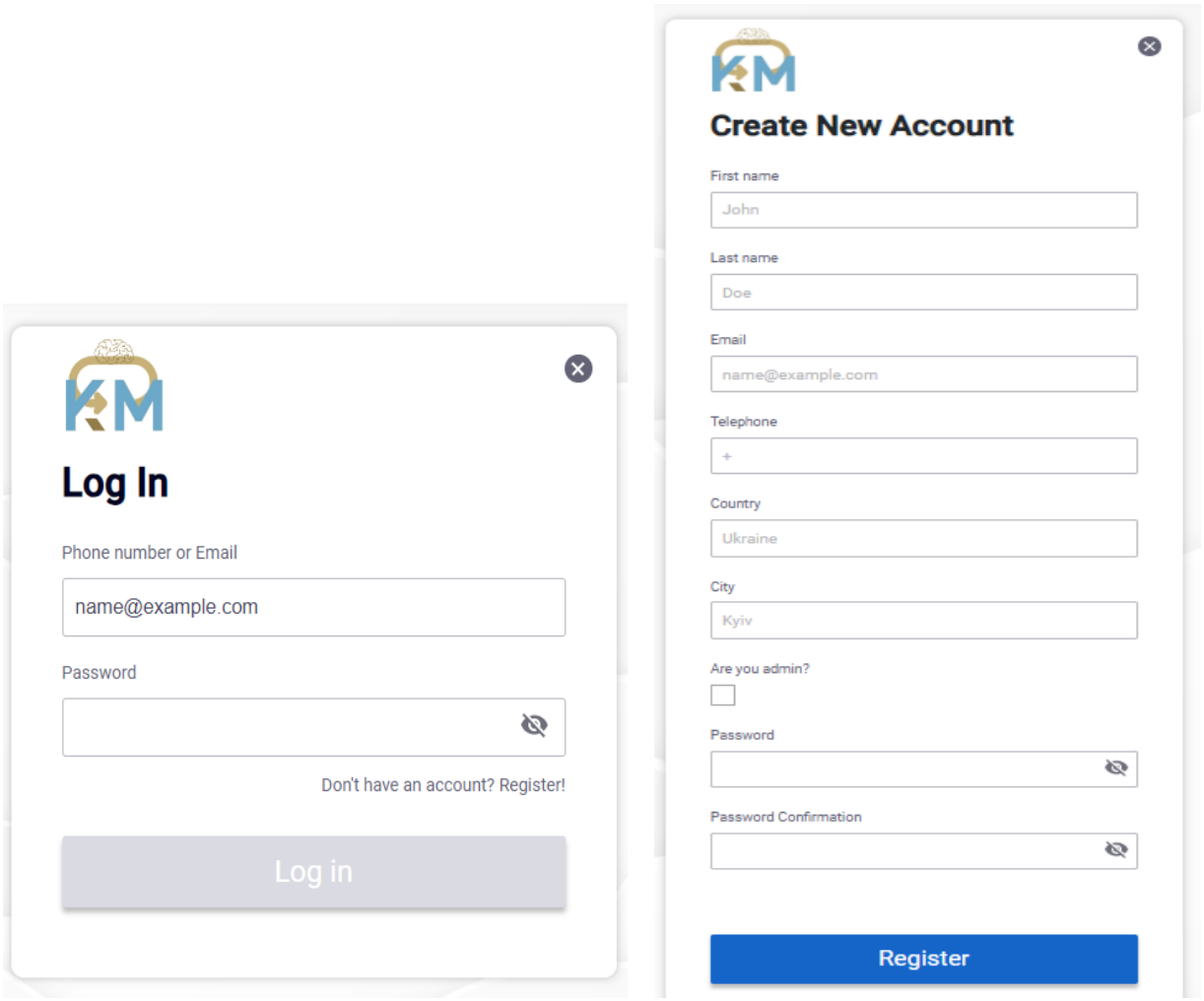


Рисунок 3.15 – Форми реєстрації та авторизації користувача.

Після авторизації користувач потрапляє на сторінку з доступними для нього проєктами, як було спроєктовано. В компоненті окремого проєкту

відображено назву, мету проєкту, його учасників, доступ до нотаток та завдань проєкту. Дані отримуються за допомогою сервісу ProjectsService з використанням методів, які в ньому реалізовано для зв'язку із бекендом. Зображення компоненту із проєктом наведено на рисунку 3.16.

Розроблений програмний продукт дозволяє створювати нотатки, що відносяться до конкретного проєкту та належать певному користувачу. Можливість передачі знання реалізовано за допомогою колективної роботи, а не пересилання нотатки чи зауваження. Імплементовано можливість додавання коментарів до певної нотатки для можливості обговорення конкретного питання або обміну інформацією чи досвідом. Як до нотатки, так і до коментаря реалізовано можливість додавання зображення, що розширює можливості збереження та передавання ідей, знань, завдань та інформації. Компонент відображення нотатки наведено на рисунку 3.17.

Рисунок 3.16 – Вигляд компоненту Проєкту.

Практики управління знаннями вплетені в методи ведення проєктів в IT-галузі. Знання про рух проєкту передається через різноманітні точки дотику та комунікації. Робота зі знаннями вплетена в IT-розробку, є своє управління проєктами і командами – Agile-методи. Саме тому в додатку було розроблено та введено аналог Jira – дошку із завданнями. Зображення із імплементованою

дошкою завдань наведено на рисунку 3.18.

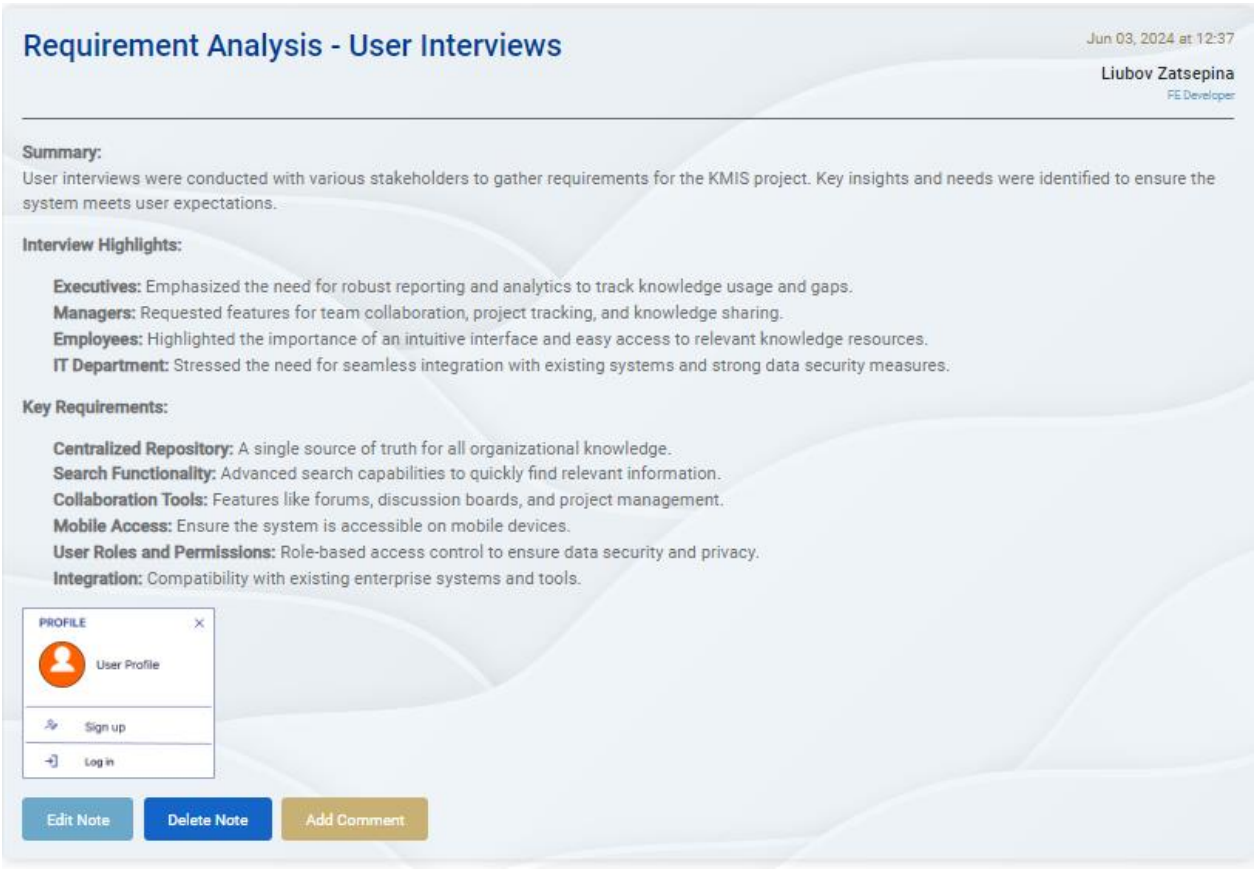


Рисунок 3.17 – Зображення компоненту Нотатки.

Необхідно зазначити, що структура кожного компонента складається з кількох файлів. Перший файл є скриптом компонента, де реалізовані всі необхідні операції з даними, що повинен виконувати даний компонент, а також підключені інші файли компонента або необхідні сервіси [31]. Цей файл має розширення .ts, оскільки TypeScript є основною мовою для розробки скриптів у фреймворку Angular.

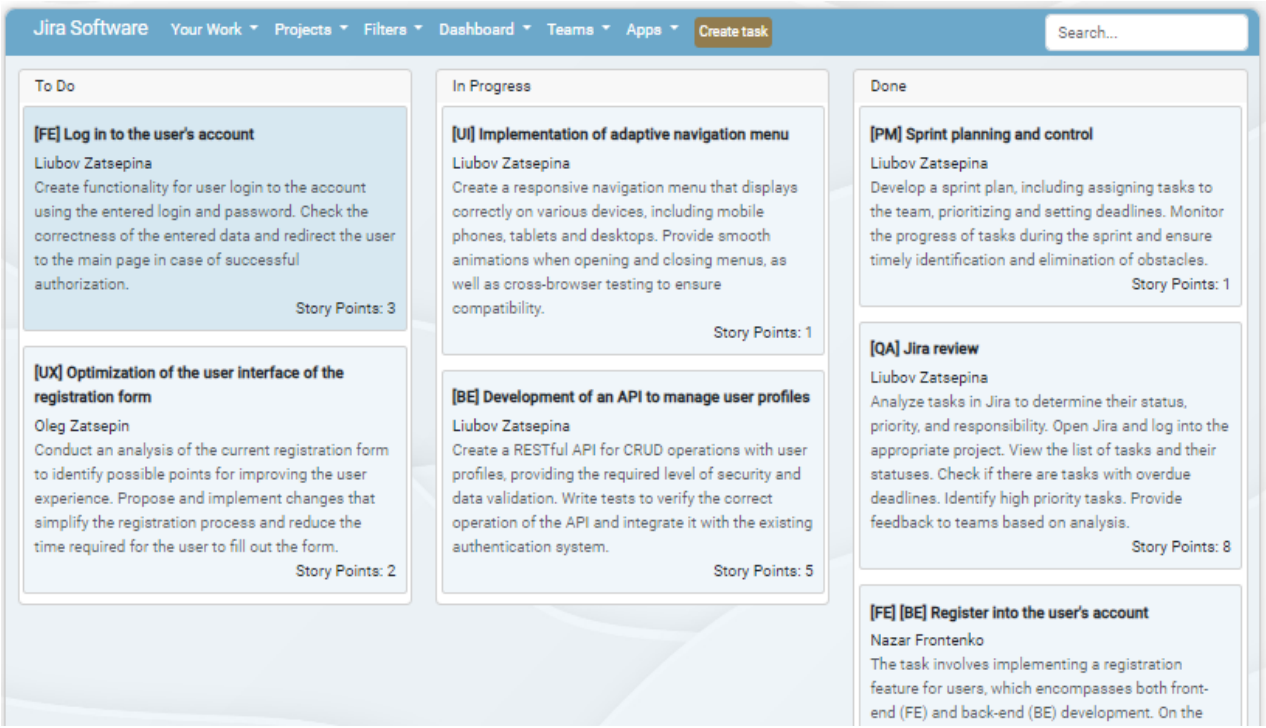


Рисунок 3.18 – Вигляд реалізованої дошки із завданнями.

Другий файл – це шаблон компонента, який містить структуру та розмітку елементів інтерфейсу компонента. Він зазвичай має розширення .html та використовує гіпертекстову розмітку [32]. Третім обов'язковим файлом є файл каскадних стилів, який містить інформацію про кольори елементів та їхні стилі відповідно до єдиного стилю інтерфейсу системи. Цей файл, як правило, має розширення .scss, оскільки для розробки стилів інтерфейсу системи був обраний препроцесор SCSS. Цей препроцесор є стандартним при розробці на фреймворку Angular і має низку переваг порівняно зі звичайним CSS [33]. В додатку Г наведено лістинг коду фронтенд частини корпоративного порталу.

Проведення тестування розробленого програмного продукту дасть змогу оцінити його відповідність встановленим правилам функціонування інформаційної екосистеми, а також запропонованим методам удосконалення управління знаннями. Це дозволить виявити потенційні недоліки та забезпечити високий рівень якості та ефективності впроваджених рішень.

3.7 Висновки

У даному розділі виконано розробку програмних модулів для системи управління знаннями в IT-проєкті. Розробка здійснена за допомогою сучасних технологій, зокрема Angular для фронтенд частини, Node.js і Express для бекенд частини, а також MongoDB у поєднанні з Mongoose для управління базою даних. Використано модель клієнт-серверної архітектури, що забезпечує ефективну взаємодію між користувачем і сервером. База даних зберігається та обробляється у хмарному середовищі MongoDB Cloud, що підвищує масштабованість і надійність системи. Розроблені програмні модулі забезпечують надійну та ефективну підтримку процесів управління знаннями в IT-проєкті та відповідають вимогам сучасної інформаційної екосистеми і вимогам технічного завдання.

4 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ УПРАВЛІННЯ ЗНАННЯМИ

4.1 Опис методик, які використовуються для тестування системи

Тестування програмного забезпечення є критично важливим етапом у процесі розробки, оскільки воно забезпечує виявлення та усунення помилок і недоліків до впровадження системи в експлуатацію. Це дозволяє гарантувати високу якість продукту, забезпечуючи його надійність, безпеку та відповідність вимогам користувачів. Тестування сприяє зниженню ризиків пов’язаних з експлуатацією програмного забезпечення, зменшуючи ймовірність виникнення збоїв та критичних помилок, які можуть негативно вплинути на бізнес-процеси. Крім того, якісне тестування допомагає покращити загальну продуктивність та зручність використання системи, що є ключовими факторами для задоволення потреб кінцевих користувачів.

Функціональне тестування спрямоване на перевірку функціональності системи відповідно до вимог специфікації. Воно включає в себе тестування окремих функцій системи, таких як створення, редагування та видалення нотаток та коментарів, завдань проєкту. Функціональне тестування також охоплює перевірку користувацьких інтерфейсів, інтеграції між різними модулями та правильність обробки даних.

Інтеграційне тестування перевіряє взаємодію між різними модулями та компонентами системи. У випадку розроблюваної системи управління знаннями це може включати перевірку взаємодії між фронтендом на Angular та бекендом на Node.js та Express, а також взаємодію з базою даних MongoDB. Мета інтеграційного тестування – виявити помилки, що можуть виникнути при спільній роботі різних компонентів.

Модульне тестування зосереджується на перевірці окремих компонентів або модулів системи в ізоляції. Для цього використовуються різні фреймворки для тестування, такі як Jasmine для Angular або Mocha і Chai для Node.js.

Модульне тестування допомагає виявити помилки на ранніх етапах розробки та забезпечити якість коду.

Регресійне тестування проводиться для перевірки, що зміни в коді або нові функції не вплинули негативно на вже існуючу функціональність системи. Це особливо важливо для великих систем, де будь-яка зміна може спричинити неочікувані помилки. Регресійне тестування допомагає забезпечити стабільність системи після внесення змін.

Навантажувальне тестування проводиться для оцінки продуктивності системи під різними навантаженнями. Воно допомагає визначити, як система поводиться при високому рівні запитів та великих обсягах даних. Для цього можуть використовуватися інструменти типу JMeter або Gatling. Мета навантажувального тестування – виявити потенційні вузькі місця та забезпечити стабільну роботу системи під навантаженням.

Безпекове тестування спрямоване на виявлення вразливостей у системі, які можуть бути використані зловмисниками для несанкціонованого доступу або пошкодження даних. Це включає перевірку автентифікації та авторизації користувачів, захисту даних та інші аспекти безпеки. Безпекове тестування допомагає забезпечити захищеність системи від зовнішніх та внутрішніх загроз.

Експериментальне тестування – це методика, яка передбачає перевірку системи в умовах, що наближені до реальних сценаріїв використання. Воно включає виконання різноманітних тестових сценаріїв, які можуть бути непередбачуваними або неформалізованими, для виявлення можливих помилок або вразливостей, які не були враховані під час традиційного тестування. Експериментальне тестування дозволяє оцінити, як система поводиться в непередбачуваних ситуаціях, забезпечуючи тим самим її більш надійну та стабільну роботу у реальному середовищі [34].

4.2 Опис сценаріїв експериментальних досліджень програмних модулів системи управління знаннями в ІТ-проєкті

Експериментальні дослідження виконуються з використанням різноманітних сценаріїв функціонування в системі управління знаннями. Серед них – взаємодія різних ролей, таких як працівники за посадами та адміністратори системи. Експериментальне тестування проводиться на різних платформах, включаючи різні браузери, ґаджети та операційні системи [34].

Ціль експериментальних досліджень полягає в перевірці функціональності та ефективності роботи системи управління знаннями. Для цього виконується перевірка роботи кожного модуля та сценарію взаємодії з користувачем. Наприклад, перевіряється коректність реєстрації та авторизації користувачів, додавання та редагування нотаток, коментарів до них, а також створення завдань.

Результати експериментальних досліджень дозволять зробити висновки щодо ефективності та надійності роботи системи управління знаннями. Будуть сформовані рекомендації щодо подальшого вдосконалення та розвитку програмних модулів з урахуванням виявлених недоліків та потреб користувачів.

Для виконання експериментальних досліджень в межах магістерської роботи було використано правила перевірки функціональності. Функціональне тестування порталу спрямовано на виявлення невідповідної роботи посилань, формування профілю тощо. Виконаємо перевірку роботи кожної форми та компоненту, введення даних та формування нотаток, коментарів до них та створення завдань до виконання в проєкті. Перевіримо роботу вікон при незаповненні обов’язкових полів.

Експериментальні дослідження були проведені за таким чек-лістом:

Перевірка правильності роботи головних функцій системи;

Виявлення невірних посилань;

Коректність виконання внутрішніх посилань;

Коректність відображення модальних вікон.

Перевірка полів і сторінок за функціоіналом:

- Реєстрація;
- Авторизація;

- Створення проєкту;
- Відкриття проєкту;
- Додавання нотатки до проєкту;
- Додавання коментаря до нотатки;
- Додавання завдання проєкту.

За результатами експериментального тестування були сформовані рекомендації щодо вдосконалення системи управління знаннями.

4.3 Тестування модулів системи управління знаннями

Тестування модулів системи управління знаннями виконаємо за кейсом перевірки функціоналу.

На рисунку 4.1 бачимо сторінку авторизації а на рисунку 4.2 сторінку із формою реєстрації в системі управління знаннями.

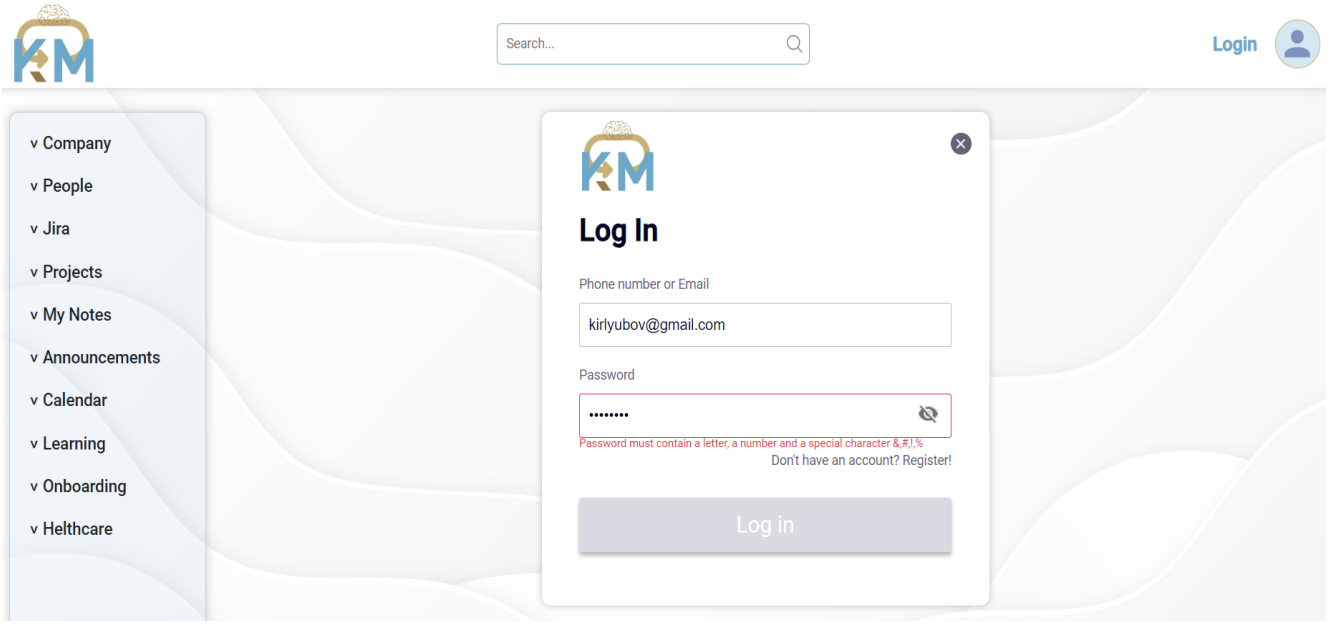


Рисунок 4.1 – Сторінка із формою авторизації

Незареєстрований та неавторизований користувач не має доступу до функціоналу. За допомогою механізму Guard CanActivate від Angular, із будь-якого шляху система нас перенаправляє на сторінку із формою авторизації.

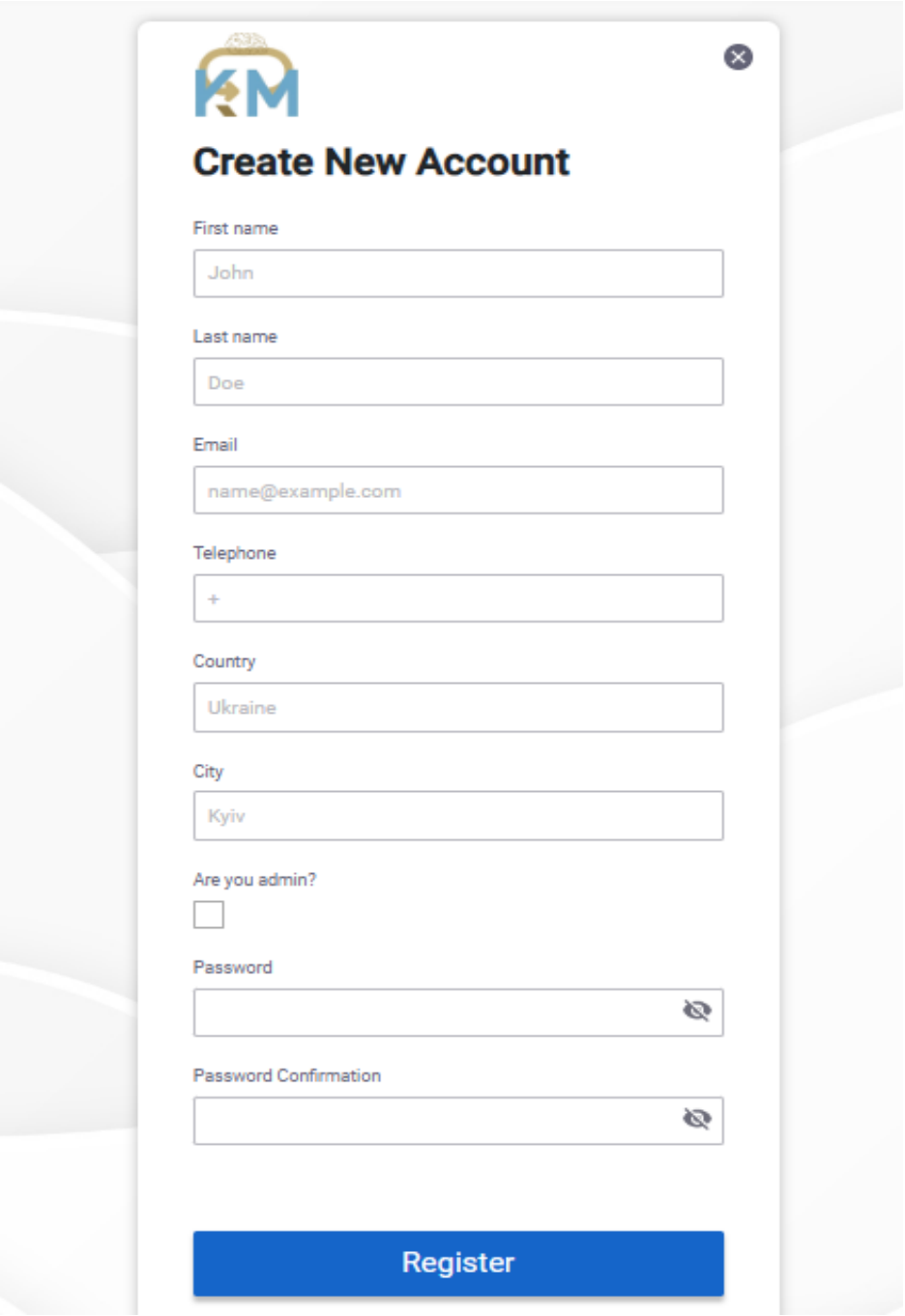


Рисунок 4.2 – Форма реєстрації в системі

У процесі розробки форм реєстрації та входу в систему, реалізованих за допомогою Angular, було забезпечено комплексну валідацію даних. Вона включає перевірку правильності введення користувачем обов’язкових полів, таких як ім’я та прізвище, електронна пошта, номер телефону, пароль, підтвердження пароля та інші необхідні дані.

Для форми реєстрації було встановлено наступні правила валідації:

1. Валідація обов’язкових полів імені та прізвища, щоб забезпечити їх заповнення;

2. Валідація електронної пошти: Перевірка формату введеної адреси електронної пошти, щоб забезпечити її відповідність стандартам та запобігти некоректному введенню;

3. Пароль: Вимога до мінімальної довжини пароля та перевірка на відповідність складності, включаючи використання великих та малих літер, цифр та спеціального символу;

4. Підтвердження пароля: Перевірка на відповідність введеного пароля та його підтвердження для запобігання помилок при введенні.

Для форми входу в систему було впроваджено наступні перевірки:

1. Валідація правильності формату електронної пошти;

2. Перевірка на відповідність введеного паролю вимогам безпеки.

Усі перевірки реалізовані за допомогою реактивних форм Angular, що дозволяє забезпечити високу швидкість реакції на некоректно введені дані та надавати користувачам миттєвий зворотний зв’язок за допомогою валідаційних повідомлень. Це значно підвищує зручність та безпеку використання системи, зменшуючи ймовірність помилок та забезпечуючи цілісність даних.

Після успішного залогінення користувач перенаправляється на сторінку із доступними йому проєктами. Кожен компонент із проєктом є посиланням на сторінку із детальною інформацією про його команду, завдання та нотатки пов’язані із ним. Сторінку із переліком проєктів зображено на рисунку 4.3, а сторінку із конкретним проєктом, відповідно, на рисунку 4.4.

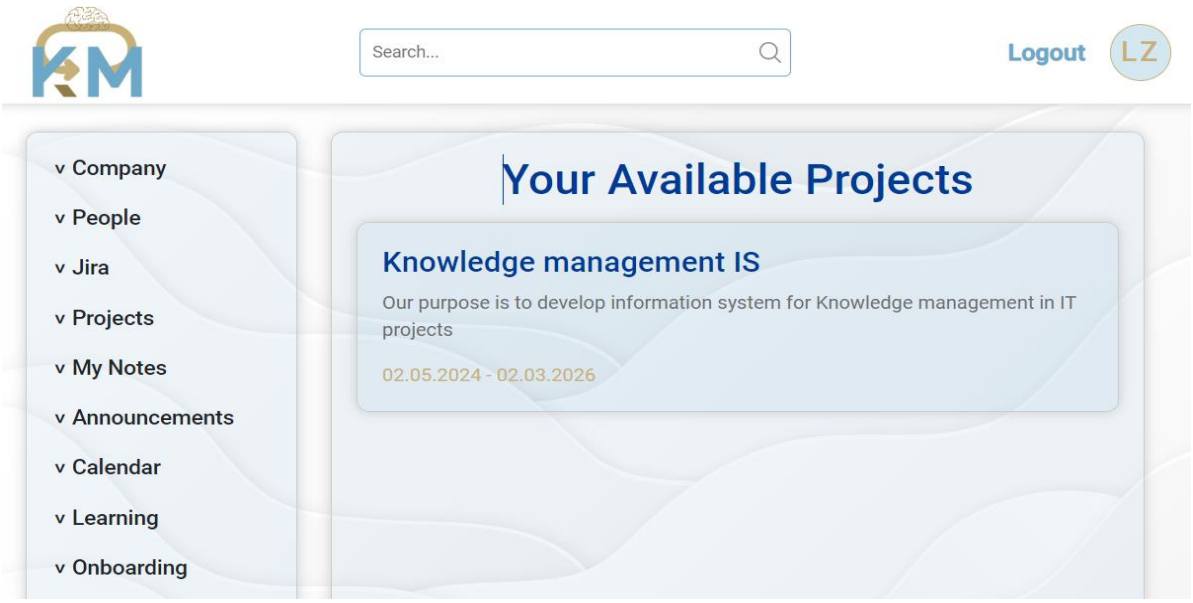


Рисунок 4.3 – Сторінка із переліком доступних проєктів

Із сторінки окремого проєкту ми переходимо на компонент зі списком членів команди. Там ми бачимо необхідну інформацію по кожній особі – ім'я, прізвище, роль на проєкті, посилання на електронну пошту із можливістю відправити співробітнику листа (рисунок 4.5).

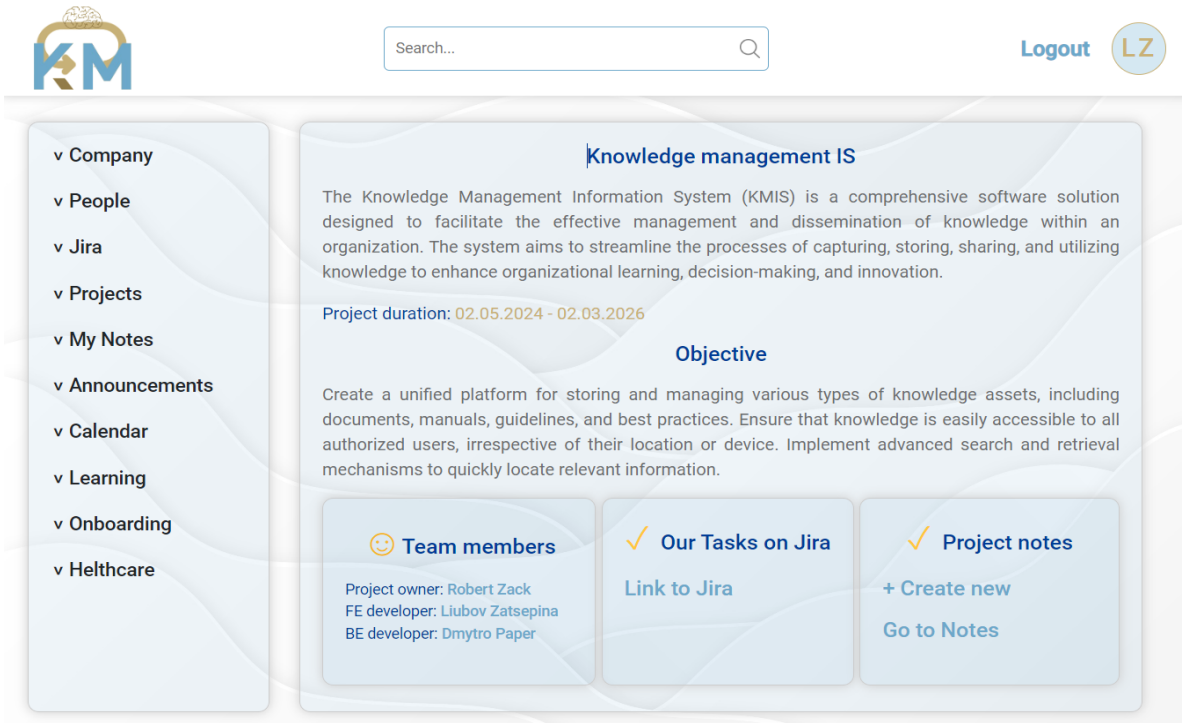


Рисунок 4.4 – Сторінка із інформацією окремого проєкту

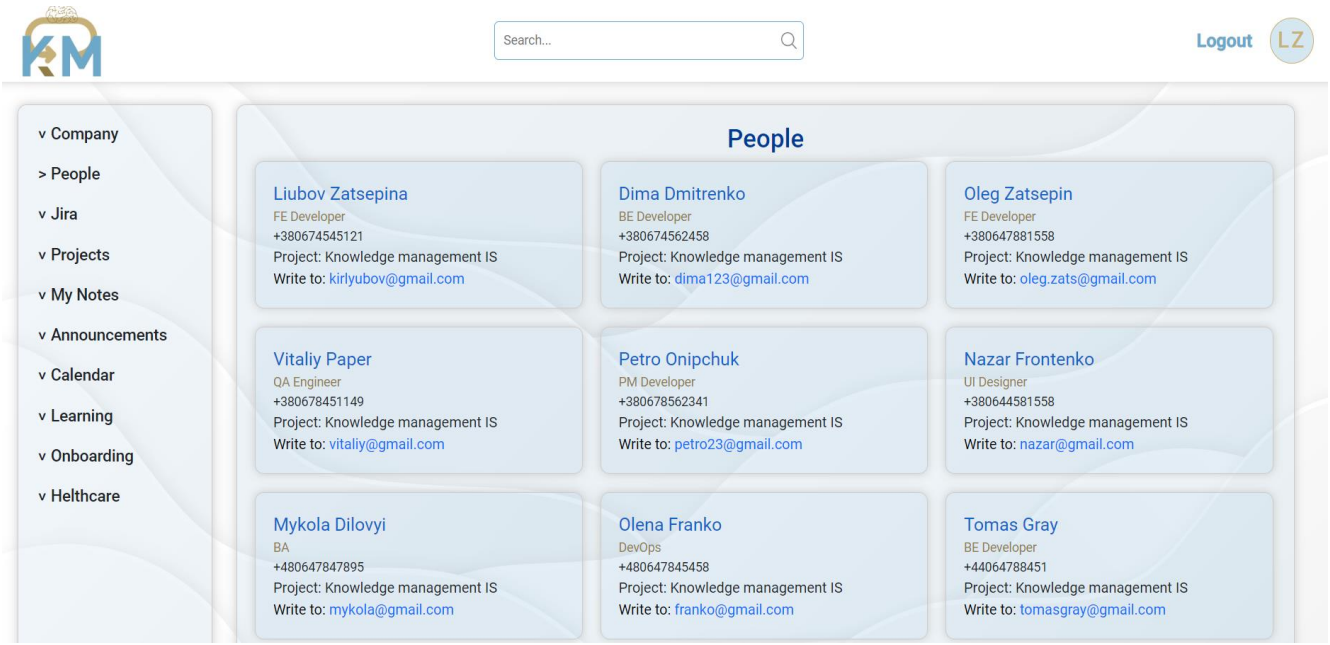


Рисунок 4.5 – Сторінка із інформацією про членів команди проєкту

На рисунку 4.6 подано список створених завдань проекту в імплементованому у розроблюваному продукті аналозі Jira. Це, в свою чергу, дозволяє створювати завдання, поширювати інформацію про рух проєкту без плати за додатковий продукт.

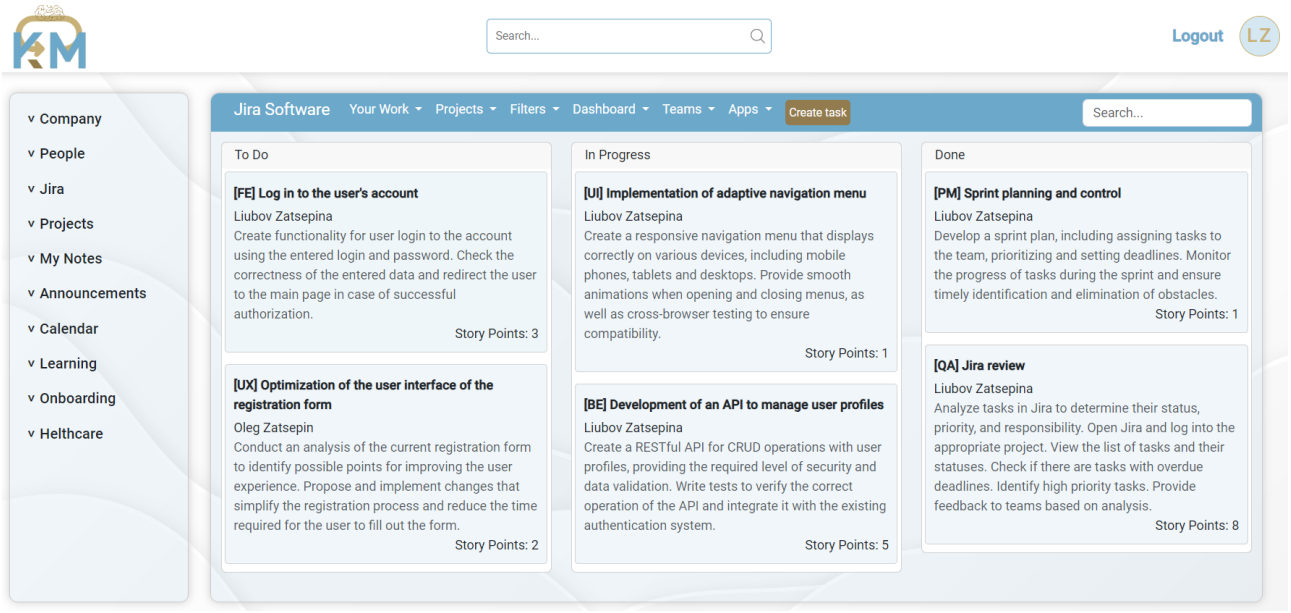


Рисунок 4.6 – Зображення дошки із завданнями в Jira

Важливим компонентом є сторінка створення завдання (рис. 4.7). Заповнюємо поля заголовку завдання, власне контент завдання, виконавця та

відповідального за успішне виконання, оцінку в сторіпоінтах, приналежність до проєкту та іншу інформацію.

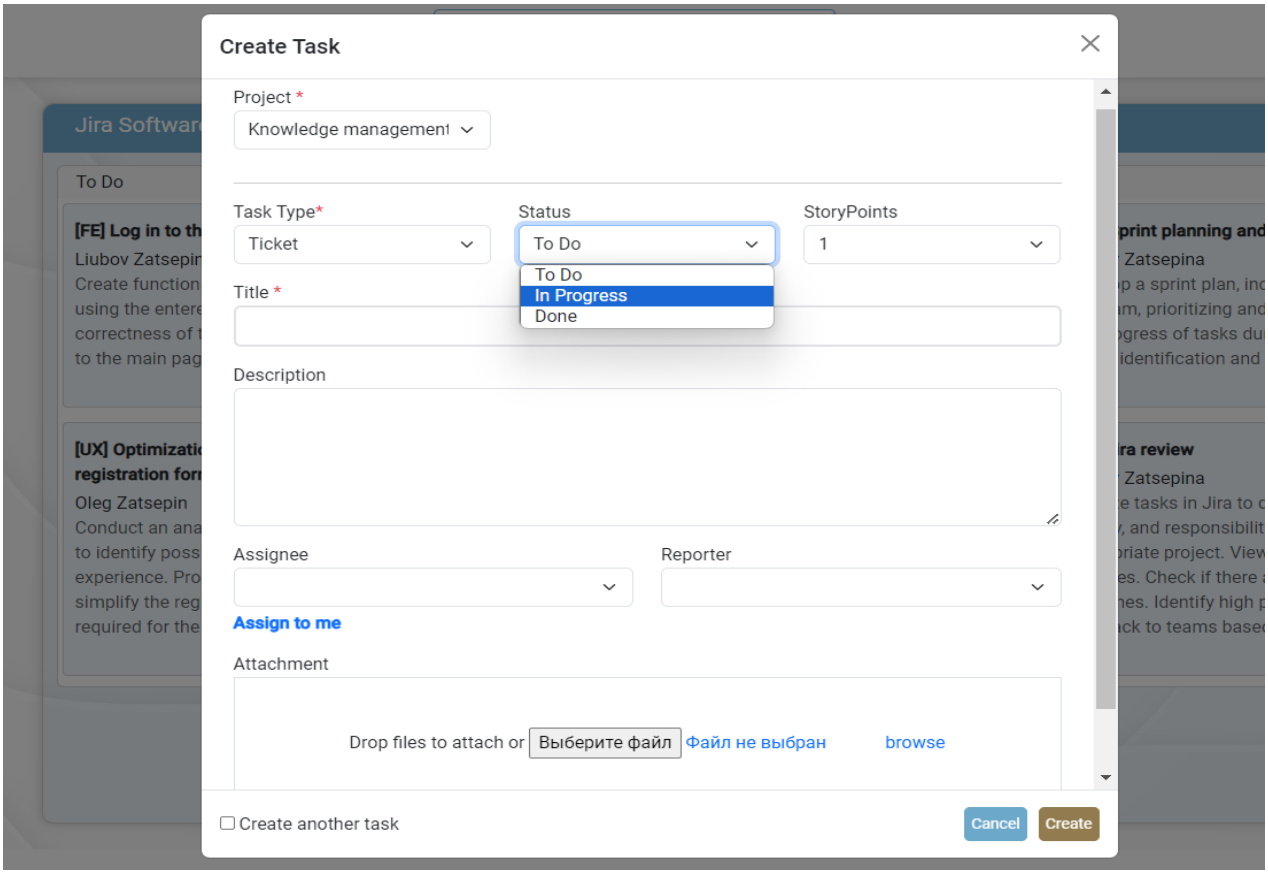


Рисунок 4.7 – Сторінка створення завдання

На рисунку 4.8 наведено сторінку відображення окремого завдання із його детальною інформацією.

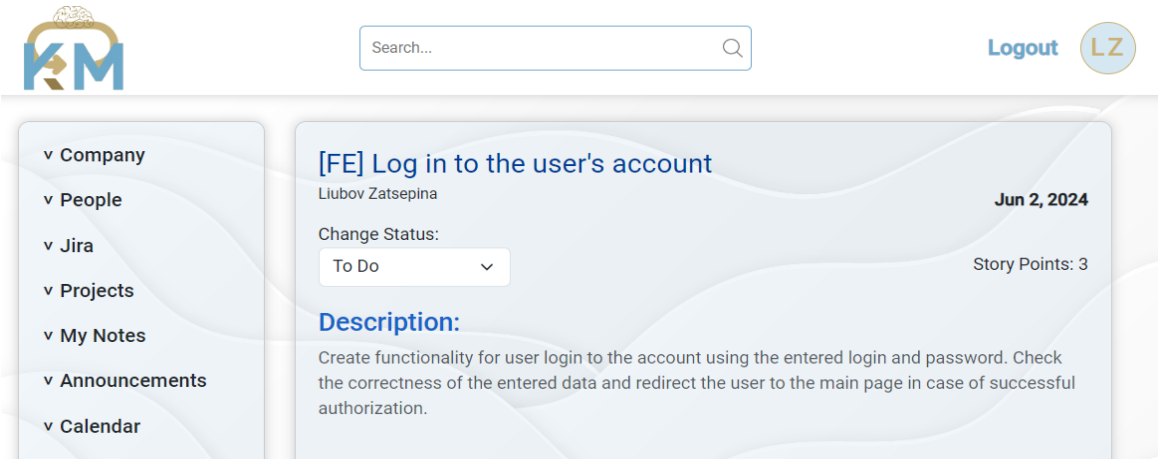


Рисунок 4.8 – Зображення сторінки окремого завдання

Одним із ключових моментів в обміні знаннями в IT-проєкті є створення та обмін нотатками. Список нотаток підтягується в залежності від проєкту, до

якого нотатки було створено (рис. 4.9).

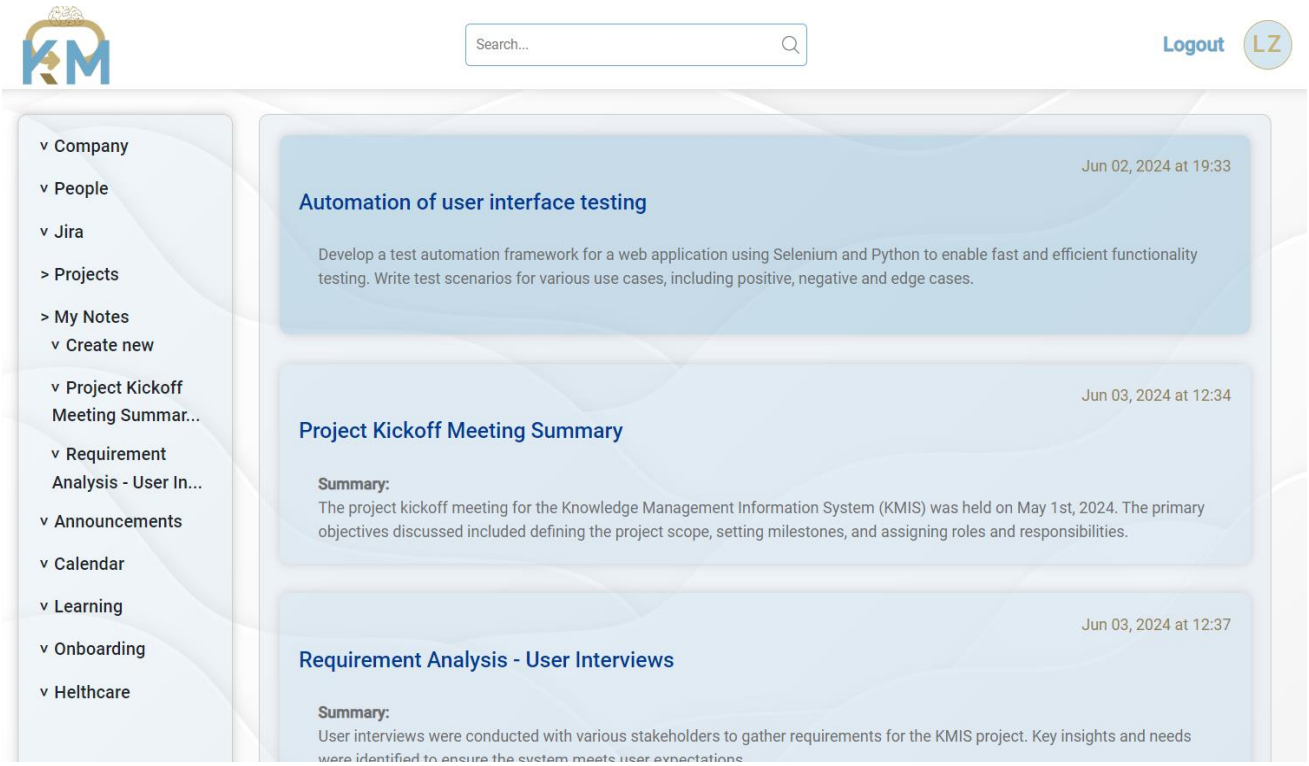


Рисунок 4.9 – Зображення списку нотаток

Створення нотатки реалізує можливість вказування проєкту та категорії, до яких вона належить, завантаження зображення, яке часто полегшує передачу інформації та ідей. При створенні нотатки, автоматично додається інформація про те, хто є її автором. Сторінку створення нотатки наведено на рисунку 4.10. Фрагмент збереженої нотатки наведено на рисунку 4.11. Функціонал реалізовано таким чином, що лише автор нотатки має змогу її змінити чи видалити.

Додавання коментарів до нотаток у системі управління знаннями є важливим аспектом, який забезпечує кілька критично важливих функцій для ефективного використання та розвитку системи.

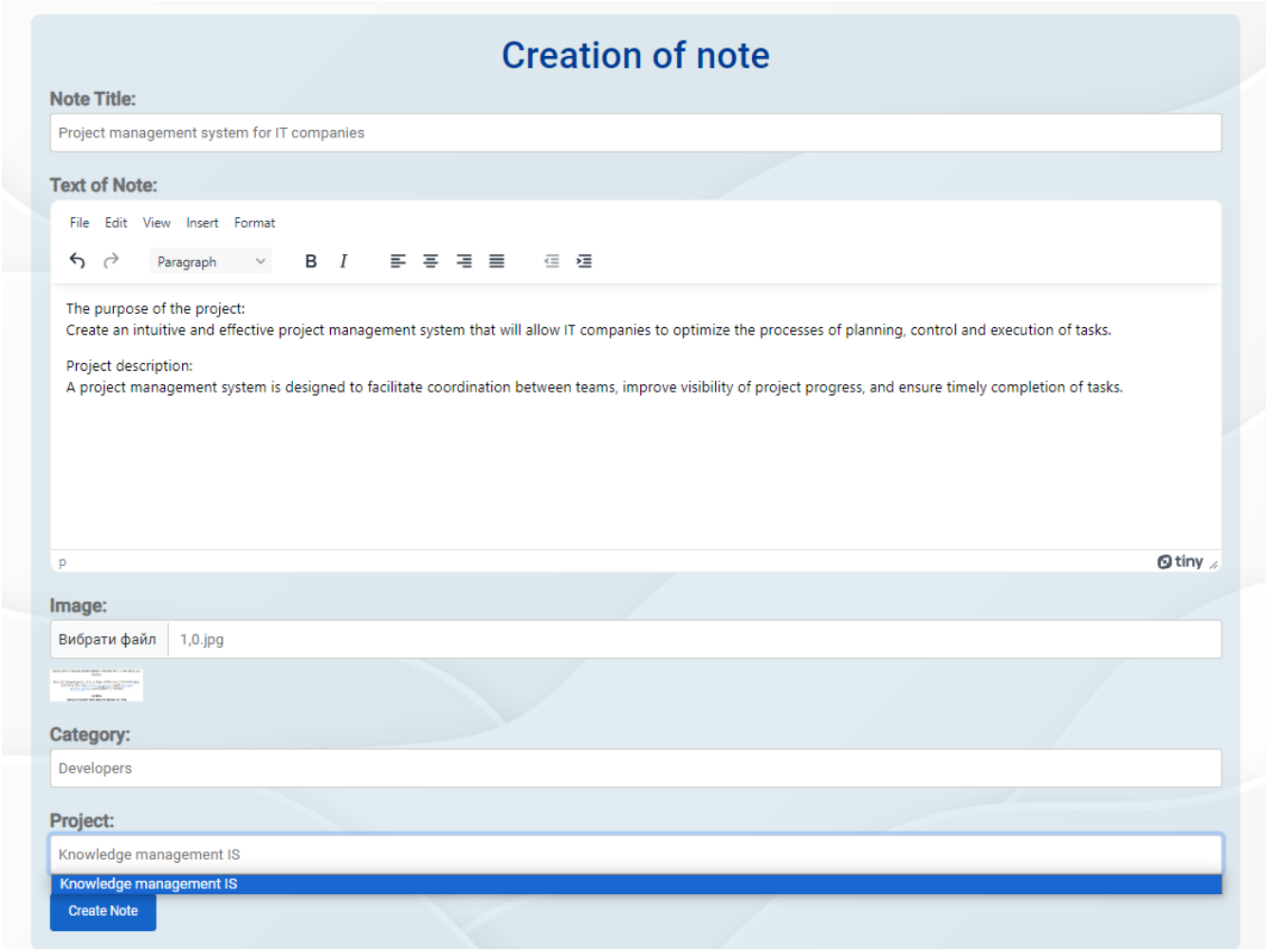


Рисунок 4.10 – Зображення із формою створення нотатки

По-перше, коментарі дозволяють забезпечити прозорість та відкритість обговорення. Вони дають можливість користувачам залишати відгуки, пропозиції та питання стосовно конкретних нотаток, що сприяє кращому розумінню контенту та його подальшому вдосконаленню.

По-друге, коментарі є ефективним інструментом для колективної роботи та співпраці. Вони дозволяють різним членам команди обговорювати нотатки в режимі реального часу, обмінюватися ідеями та вирішувати проблеми без необхідності організовувати додаткові зустрічі або обмінюватися великою кількістю електронних листів.

По-третє, коментарі забезпечують можливість відстежування історії обговорень і рішень. Це особливо важливо для складних проєктів, де рішення можуть змінюватися з часом, і необхідно мати доступ до історії дискусій, щоб

зрозуміти причини прийняття певних рішень.

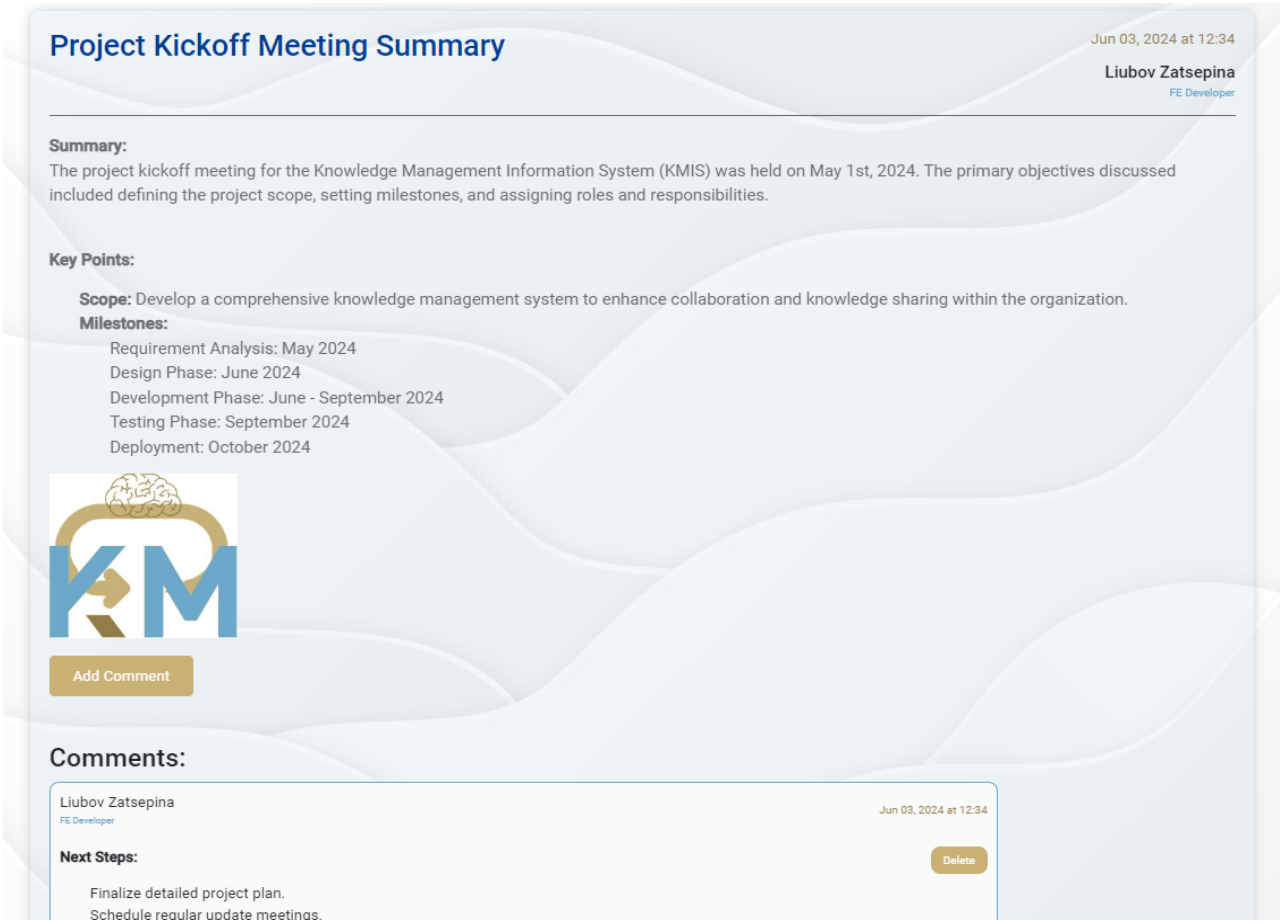


Рисунок 4.11 – Фрагмент створеної нотатки із доданим коментарем

По-четверте, коментарі можуть бути використані для навчання та передачі знань. Нові члени команди можуть швидко ознайомитися з попередніми обговореннями та рішеннями, що знижує криву навчання та підвищує загальну ефективність команди.

Зрештою, коментарі сприяють підвищенню якості контенту. Відгуки від різних користувачів можуть вказати на помилки, неточності або можливості для покращення, що дозволяє авторам швидко вносити необхідні корективи та покращувати якість інформації, яка зберігається у системі.

Таким чином, додавання коментарів до нотаток було імплементовано як необхідний елемент, що забезпечує відкритість, співпрацю, відстежуваність, навчання та підвищення якості в системі управління знаннями. На рисунку 4.12

наведено форму створення коментаря до конкретної нотатки. Відображення збереженого коментаря наведено на рисунку 4.13.

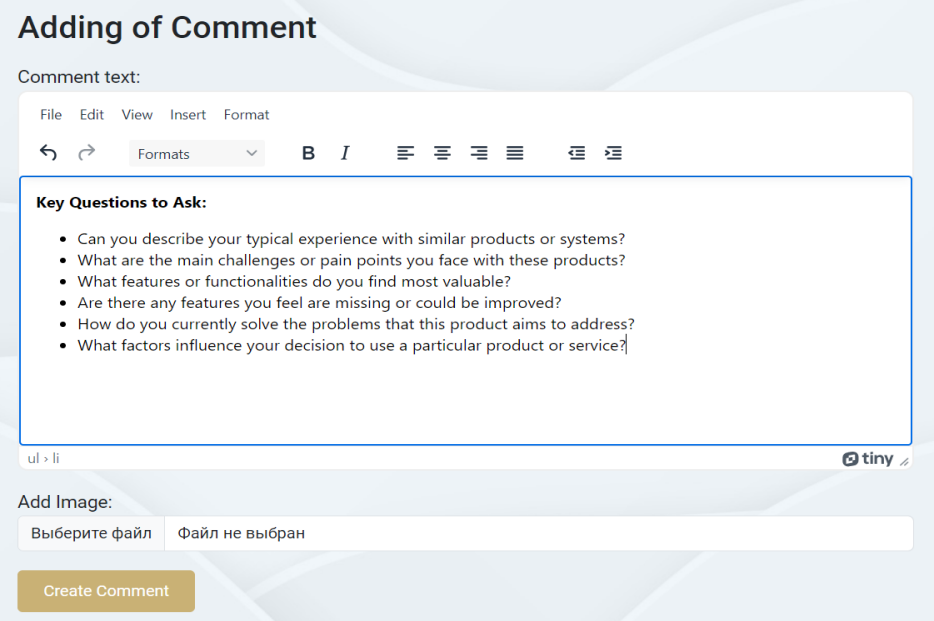


Рисунок 4.12 – Форма створення коментаря до нотатки

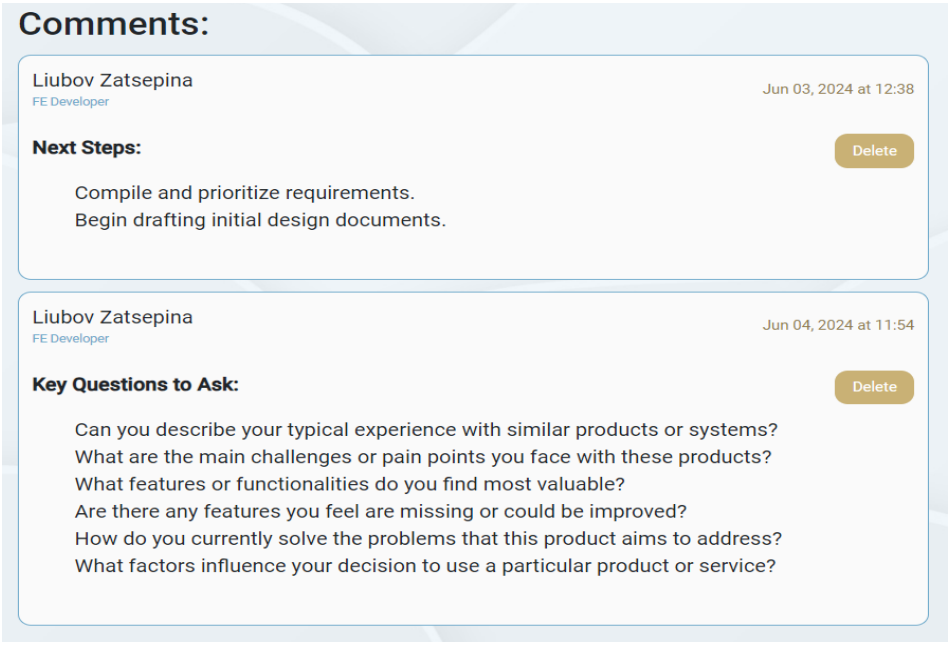


Рисунок 4.13 – Зображення доданих коментарів до нотатки

В ході проведеного тестування за кейсом перевірки функціоналу було перевірено основні функції системи, включаючи реєстрацію, авторизацію, створення проєктів, додавання нотаток, коментарів та завдань. Усі перевірки

продемонстрували коректну роботу системи відповідно до заявлених вимог та сценаріїв використання. Виявлені незначні недоліки були оперативно виправлені, що дозволило підтвердити відповідність програмного продукту стандартам якості та функціональності.

4.4 Проведення тестування додатку утилітами Angular

Тестування в Angular включає в себе використання кількох утиліт і інструментів, які допомагають розробникам писати, виконувати і аналізувати тести. Основні утиліти для тестування в Angular включають:

Jasmine – це бібліотека для написання тестів, яка поставляється з Angular за замовчуванням. Вона надає синтаксис для написання тестів і різні методи для перевірки очікувань.

Karma – це тестовий раннер, який виконує тести в реальних браузерах. Він дозволяє налаштовувати різні конфігурації для запуску тестів і генерувати звіти про покриття коду.

Під час unit-тестування створюються невеликі тести, які перевіряють специфічні аспекти функціональності коду. Це дозволяє виявляти помилки та дефекти на ранніх етапах розробки і забезпечувати стабільність окремих частин програми. Unit-тести автоматизовані, що дає можливість запускати їх повторно при кожній зміні коду для перевірки його стабільності та правильності роботи. Цей вид тестування є важливим для підтримки якості кодової бази, забезпечення рефакторингу та зменшення ризику виникнення непередбачуваних проблем у великих програмних проектах [35].

Під час інтеграційного тестування увага приділяється перевірці точок взаємодії між компонентами та їхньої спільної роботи. Це включає тестування API, баз даних, інтерфейсів користувача та інших елементів системи. Інтеграційне тестування допомагає виявити проблеми, що можуть виникнути під час взаємодії різних частин програми, наприклад, некоректні дані, проблеми синхронізації або неправильні взаємодії. Важливо переконатися, що всі

компоненти програми можуть ефективно інтегруватися та співпрацювати між собою, щоб забезпечити надійну та стабільну роботу системи в цілому.

Cypress – це сучасний інструмент для E2E тестування, який набирає популярності завдяки простоті використання і швидкості. Cypress дозволяє писати тести, які імітують дії користувача, і надає детальну інформацію про помилки.

Під час E2E тестування застосовуються автоматизовані тести, що моделюють і перевіряють користувацьку взаємодію з програмою від початку до кінця. Це включає всі можливі сценарії взаємодії та варіанти введення даних. Головна мета E2E тестування полягає в тому, щоб забезпечити правильну роботу всіх компонентів системи у сукупності та виявити потенційні проблеми на рівні інтеграції та взаємодії між різними частинами. Такий вид тестування є критично важливим для виявлення дефектів, які можуть з’явитися під час реального використання програми користувачами [36].

Під час тестування компонентів та сервісів було проведено ретельний аналіз різних аспектів функціональності програмного забезпечення. Тести охоплювали різні сценарії використання, включаючи позитивні, негативні та крайові випадки. Перевірялися різні аспекти, такі як ініціалізація форм, валідація введених даних, коректність виконання дій при успішному та невдалому вході користувача, автоматичний вихід користувача після закінчення токену тощо.

На рисунку 4.14 показано фрагмент файлу тестів компоненту Нотаток, який демонструє проведення тестування різних аспектів функціональності цього компоненту. У цьому фрагменті можна побачити тести, які перевіряють ініціалізацію форми, коректну валідацію введених даних, а також інші ключові аспекти, які впливають на коректну роботу компоненту.

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { NotesComponent } from './notes.component';
import { NotesService } from '@shared/services/notes.service';
import { ActivatedRoute } from '@angular/router';
import { Note } from '@shared/models/note.model';

describe('NotesComponent', () => {
  let component: NotesComponent;
  let fixture: ComponentFixture<NotesComponent>;
  let notesServiceStub: Partial<NotesService>;
  let activatedRouteStub: Partial<ActivatedRoute>;

  const dummyNotes: Note[] = [
    { id: '1', title: 'Note 1', dateCreated: '2024-01-01T10:00:00Z' },
    { id: '2', title: 'Note 2', dateCreated: '2024-01-02T11:00:00Z' },
  ];

  beforeEach(async () => {
    notesServiceStub = { getProjectNotes: jasmine.createSpy('getProjectNotes').and.returnValue(of(dummyNotes)) };
    activatedRouteStub = { queryParams: of({ projectId: '123' }) };

    await TestBed.configureTestingModule({
      declarations: [NotesComponent],
      imports: [RouterTestingModule],
      providers: [
        { provide: NotesService, useValue: notesServiceStub },
        { provide: ActivatedRoute, useValue: activatedRouteStub },
      ],
    }).compileComponents();

    fixture = TestBed.createComponent(NotesComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should initialize project ID and fetch project notes on init', () => {
    expect(component.prId).toBe('123');
    expect(notesServiceStub.getProjectNotes).toHaveBeenCalledWith('123');
    component.notes$.subscribe(notes => {
      expect(notes).toEqual(dummyNotes);
    });
  });

  it('should display notes', () => {
    const noteElements = fixture.debugElement.queryAll(By.css('.note'));
    expect(noteElements.length).toBe(2);
    expect(noteElements[0].nativeElement.textContent).toContain('Note 1');
```

Рисунок 4.14 – Фрагмент файлу тестів компоненту Нотаток

На рисунку 4.15 представлено приклад роботи тестового раннеру Karma та бібліотеки Jasmine. Під час запуску тестування було досягнуто успішного виконання усіх тестів, що свідчить про коректну роботу розробленого додатку та тестувального середовища. Спочатку були зафіксовані деякі тестові провали, які вказували на проблеми або помилки у коді програми, що потребували подальшої

перевірки та виправлення.

Це сприяло підвищенню якості програмного продукту та забезпечило надійність його роботи.

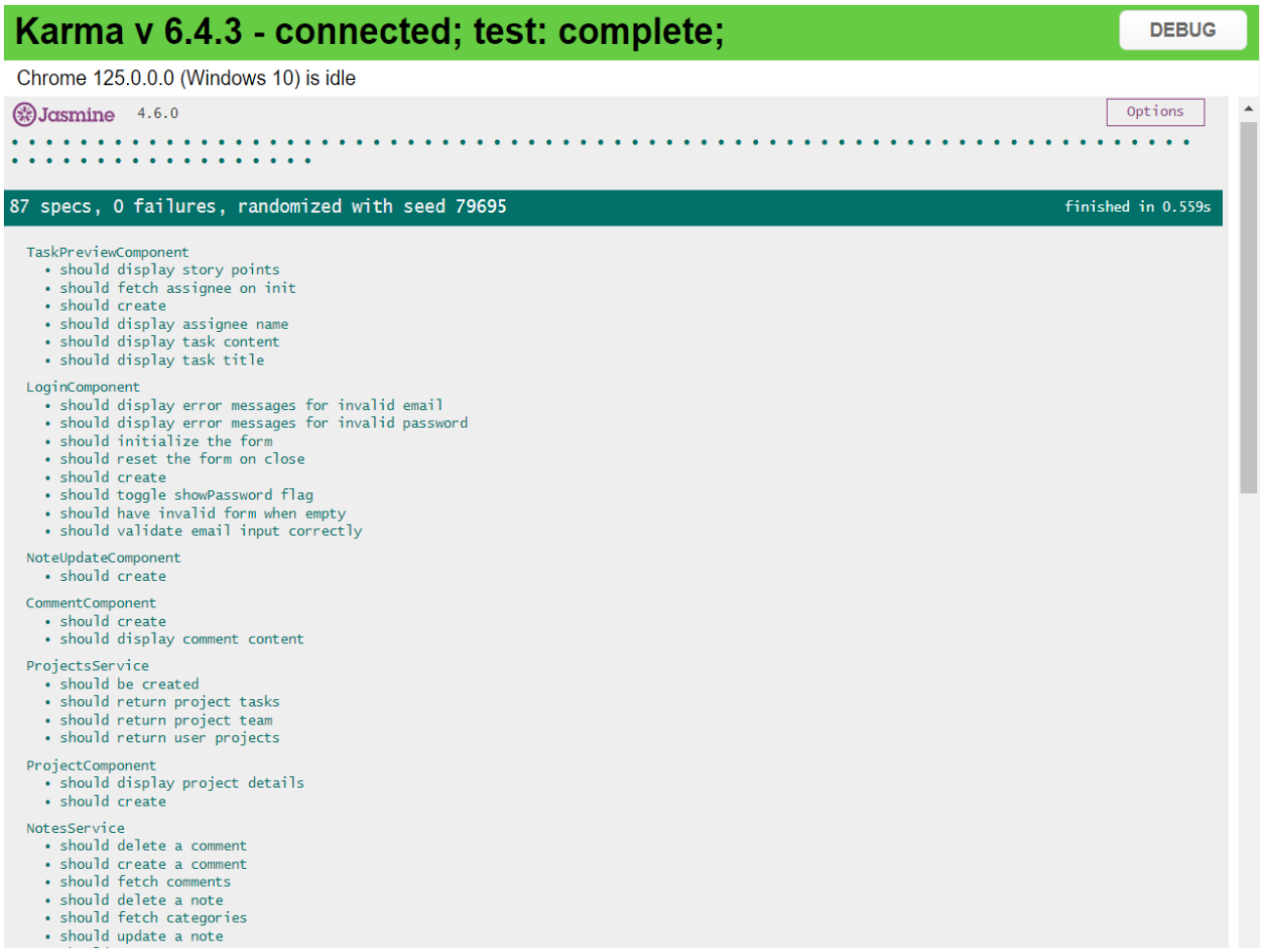
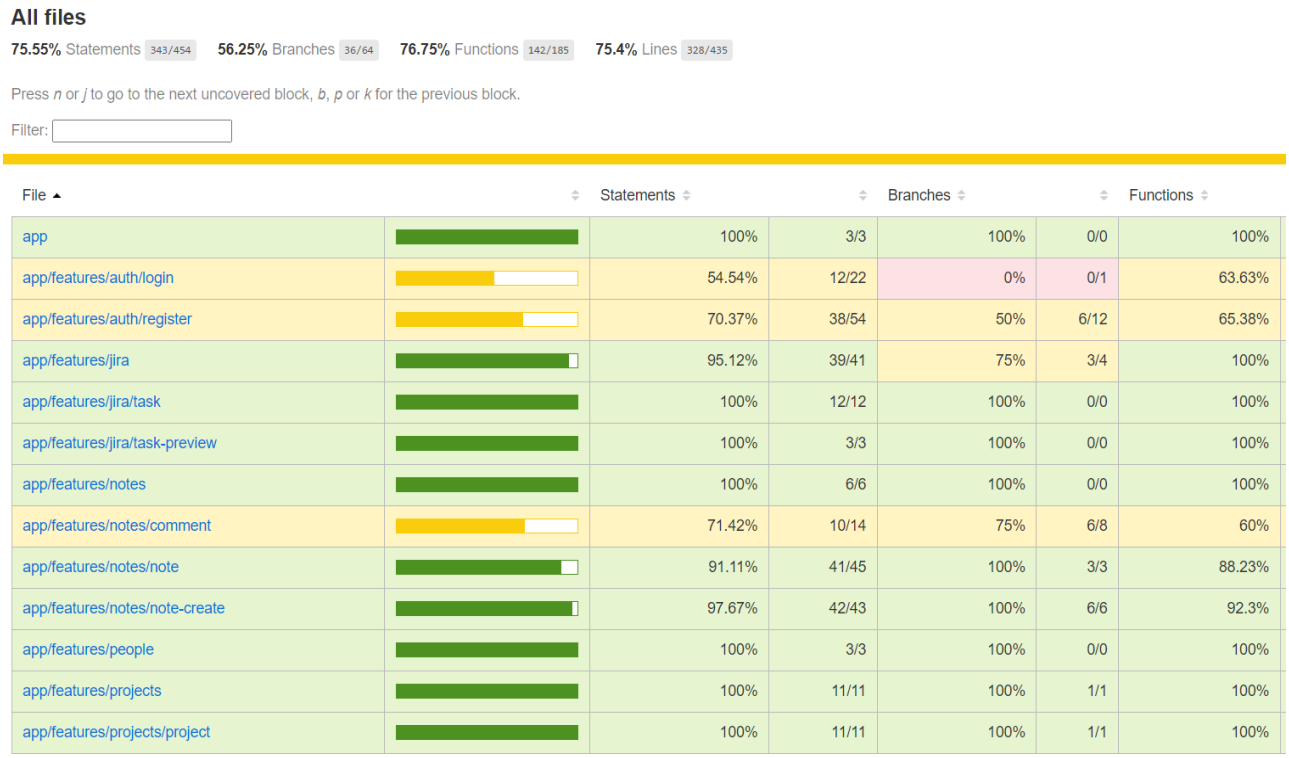


Рисунок 4.15 – Приклад роботи тестового раннеру Karma та бібліотеки Jasmine

На рисунку 4.16 наведено приклад покриття тестами розробленого додатку. Результати вказують на високий рівень покриття тестами, охоплюючи значну частину програмного коду. Це свідчить про відповідність тестових сценаріїв функціональним вимогам програми, а також про високу якість тестування.



All files

75.55% Statements 343/454 **56.25%** Branches 36/64 **76.75%** Functions 142/185 **75.4%** Lines 328/435

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions	
app	100%	3/3	100%	0/0	100%	
app/features/auth/login	54.54%	12/22	0%	0/1	63.63%	
app/features/auth/register	70.37%	38/54	50%	6/12	65.38%	
app/features/jira	95.12%	39/41	75%	3/4	100%	
app/features/jira/task	100%	12/12	100%	0/0	100%	
app/features/jira/task-preview	100%	3/3	100%	0/0	100%	
app/features/notes	100%	6/6	100%	0/0	100%	
app/features/notes/comment	71.42%	10/14	75%	6/8	60%	
app/features/notes/note	91.11%	41/45	100%	3/3	88.23%	
app/features/notes/note-create	97.67%	42/43	100%	6/6	92.3%	
app/features/people	100%	3/3	100%	0/0	100%	
app/features/projects	100%	11/11	100%	1/1	100%	
app/features/projects/project	100%	11/11	100%	1/1	100%	

Рисунок 4.16 – Приклад покриття тестами розробленого додатку

Високий відсоток покриття роботи програми тестами є важливим показником якості програмного продукту, оскільки воно забезпечує впевненість у його надійності та стабільності. За допомогою широкого спектру тестів вдається виявити потенційні проблеми та помилки ще на ранніх етапах розробки, що дозволяє уникнути їх у подальшому експлуатації. Такий підхід сприяє підвищенню задоволення користувачів від продукту та сприяє його успішному впровадженню на ринку. Таким чином, можна стверджувати, що система управління знаннями готова до впровадження та експлуатації в реальних умовах.

4.5 Висновки

У четвертому розділі даної роботи було детально описано методики, які використовуються для тестування програмного продукту. Висвітлено сценарії експериментальних досліджень програмних модулів системи управління знаннями в IT-проєкті, спрямованих на визначення їх ефективності та

відповідності функціональним вимогам. Зокрема, розглянуто використання бібліотеки Jasmine та тестового раннера Karma для забезпечення надійності та стабільності програмного коду.

Проведення тестування додатку утилітами вбудованими в Angular дозволило забезпечити високий рівень покриття тестами та виявити потенційні проблеми ще на ранніх етапах розробки та сприяло підвищенню надійності та якості програмного продукту, а також забезпечує впевненість у його стабільності під час експлуатації.

Проведення експериментальних досліджень дозволило оцінити ефективність роботи програмних модулів системи управління знаннями в реальних умовах. Результати цих досліджень підтвердили ефективність та надійність розробленого програмного продукту, що забезпечує користувачам впевненість у його функціонуванні та відповідність їх потребам.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Нові технічні розробки можуть бути прийнятними та успішно впроваджуватися, лише якщо вони відповідають сучасним вимогам технічного прогресу та економічному контексту. Тому важливо провести оцінку економічної ефективності результатів науково-дослідної роботи.

Магістерська кваліфікаційна робота на тему «Удосконалення методів і засобів управління знаннями в ІТ-проєкті» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок, тобто коли відбувається так звана комерціалізація науково-технічної розробки, оскільки результатом кваліфікаційної роботи є програмний продукт у вигляді веб додатку для управління знаннями в ІТ-проєкті. Для реалізації продукту необхідно знайти потенційного інвестора, переконати його в економічній доцільності, щоб він взявся за втілення цього проекту.

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності. Для проведення технологічного аудиту було залучено трьох незалежних експертів: Черноволик Галина Олександрівна (к.т.н. доц. кафедри ПЗ ВНТУ), Ракитянська Ганна Борисівна (к.т.н. доц. кафедри ПЗ ВНТУ), Майданюк Володимир Павлович (к.т.н. доц. кафедри ПЗ ВНТУ).

Для цього нам необхідно виконати такі кроки:

1) здійснити комерційний аналіз науково-технічної розробки, щоб визначити її науковий рівень і комерційний потенціал; 2) розрахувати витрати на виконання науково-технічної розробки; 3) оцінити економічну вигідність розробки при її впровадженні та комерціалізації і зробити обґрунтований висновок про цілеспрямованість комерціалізації потенційним інвестором.

Оцінювання комерційного потенціалу буде здійснене із застосуванням 5-ти бальної системи оцінювання за критеріями, що наведені в таблиці 5.1 [37].

Таблиця 5.1 – Критерії оцінювання комерційного потенціалу розробки та бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги:					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат Аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Кри-тер.	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому Комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання експертами комерційного потенціалу розробки зведено в таблицю 5.2.

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами:		
1 1. Технічна здійсненність концепції	4	3	4
Ринкові переваги (недоліки):			
2. Ринкові переваги (наявність аналогів)	2	2	3
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	4	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	4	3
Ринкові перспективи			
6. Ринкові перспективи (розмір ринку)	2	3	3
7. Ринкові перспективи (конкуренція)	4	2	3
Практична здійсненність			
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	4	4
Сума балів	$СБ_1$=44	$СБ_2$=43	$СБ_3$=44
Середньоарифметична сума балів $\overline{СБ}$	44		

За даними результатів розрахунків, наведених в таблиці 5.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Порівняємо результат з рівнями комерційного потенціалу розробки, що наведено в таблиці 5.3.

Таблиця 5.3 – Рівні комерційного потенціалу розробки

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

Рівень комерційного потенціалу розробки, становить 44 бали, що відповідає рівню «високий».

Такий рівень комерційного потенціалу було досягнуто завдяки кільком ключовим факторам. По-перше, система була спроектована з урахуванням потреб користувачів та вимог ринку, що дозволило максимально задовольняти їхні потреби. В першу чергу розробка дозволяє ефективно обмінюватися інформацією та економити людські ресурси, необхідні для організації і проведення онбоардингів та передачі інформації, досвіду та знання необхідного для успішного виконання завдань ІТ-проєкту. По-друге, впровадження сучасних технологій розробки програмного забезпечення сприяло підвищенню продуктивності і ефективності роботи системи. Крім того, активна підтримка та постійне оновлення системи дозволить підтримувати її на високому рівні конкурентоспроможності на ринку. Завдяки правильному маркетинговому стратегічному плану, можна посприяти широкому розповсюдженню та прийняттю системи користувачами.

Нова розробка може використовуватися як на комерційних підприємствах, так і у державних закладах для успішного виконання ІТ-проєктів. Безпосереднім споживачем є команди, що проводять розробку ІТ-проєктів.

Технічна готовність складає 95%. Була розроблена архітектура програмного продукту, розроблено прототип, який частково покриває необхідний функціонал. На даний час декілька зацікавлених сторін розглядає можливість комерціалізації розробки.

5.2 Розрахунок витрат на здійснення науково-технічної розробки

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Удосконалення методів та засобів управління знаннями в ІТ-проєкті», під час планування, обліку і калькулювання собівартості науково-дослідної роботи згрупуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці» [37].

5.2.1.1 Розрахунок основних витрат на оплату праці

Проведемо розрахунок витрат на здійснення науково-технічної розробки. Обчислимо основну заробітну $З_о$ розробника, за формулою (5.1):

$$З_о = \frac{M}{T_р} \cdot t[грн], \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника, грн;

$Т_р$ – число робочих днів в місяці, $Т_р$ = 22дні;

t – число днів роботи розробника.

Обчислимо заробітну плату розробника з місячним окладом 18000 грн і кількістю робочих днів у місяці – 22. Число днів роботи розробника 68.

$$З_о = \frac{18000}{22} \cdot 68 = 55636{,}36(\text{грн}).$$

Результати розрахунків зведено до таблиці 5.5.

Таблиця 5.5 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на

				заробітну плату, грн.
Керівник проєкту	20000,00	909,09	68	61818,18
Розробник ПЗ	18000,00	818,18	68	55636,36
Тестувальник ПЗ	15000,00	681,81	68	46363,63
Всього				163818,17

5.2.1.2 Розрахунок додаткової заробітної плати робітників

Обчислимо додаткову заробітну плату робітників $З_д$. Розрахуємо її як 10% від основної заробітної плати за формулою (5.2):

$$З_д = З_о \cdot 0,10 \text{ [грн]}. \quad (5.2)$$

$$З_д = 163818,17 \cdot 0,10 = 16381,82 \text{ (грн)}.$$

5.2.2 Розрахунок відрахування на соціальні заходи

Згідно діючого законодавства до статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов'язкове державне соціальне страхування та для здійснення заходів щодо соціального захисту населення (ЄСВ – єдиний соціальний внесок). У 2024 році нарахування на заробітну плату складають 22% від суми основної та додаткової заробітної плати, як подано формулою (5.3):

$$Н_з = (З_{о\ р} + З_д) \cdot 0,22 \text{ [грн]}. \quad (5.3)$$

$$Н_з = (163818,17+16381,82) \cdot 0,22 = 39644,00 \text{ (грн)}.$$

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні

та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації [37].

Дана стаття витрат розраховується за формулою 5.4:

$$К = \sum_{i=1}^{n} H_i * Ц_i * К_i \ , \quad (5.4)$$

де Н*i* – кількість комплектуючих *i*-го виду, шт.;

Ц*i* – покупна ціна комплектуючих *i*-го найменування, грн.;

К*i* – коефіцієнт транспортних витрат (1,1…1,15).

Інформацію про витрати на матеріали, що використані при розробці програмного продукту, наведено у таблиці 5.6.

Таблиця 5.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено, шт.	Вартість витраченого матеріалу, грн
Папір офісний А4 Zoom	220,00	1	220,00
Ручка	25,00	2	50,00
Тонер для принтера HP LaserJet 1018	150,00	1	150,00
Всього			420,00

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі (Кв), які б використовувалися при проведенні НДР на тему «Удосконалення методів та засобів управління знаннями в ІТ-

проєкті відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення [37]. Витрати на «Спецустаткування для наукових (експериментальних) робіт» в роботі на тему «Удосконалення методів та засобів управління знаннями в ІТ-проєкті» відсутні.

5.2.6 Витрати на програмне забезпечення

До статті витрат «Програмне забезпечення для наукових (експериментальних) робіт» входять витрати на спеціальні програмні засоби та спеціальне програмне забезпечення (програми, алгоритми, бази даних), необхідне для проектування та розробки програмного продукту.

Балансову вартість програмного забезпечення розраховують за формулою 5.5:

$$В_{прг} = \sum_{i=1}^{к} Ц_{i\ прг} * С_{і\ прг} * К_{і}\ , \qquad (5.5)$$

де Ц*iпрг* – ціна придбання/використання одиниці програмного засобу цього виду, грн;

С*iпрг* – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

К*i* – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо (Кі = 1,10…1,12).

k – кількість найменувань програмних засобів.

Отримані результати наведено в таблиці 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів

Найменування програмного засобу	Час використання, місяців	Ціна за місяць, грн	Вартість, грн
Підписка Visual Paradigm Enterprise	1	3660,00	3660,00
Visual Studio Code	3		Безкоштовно
MongoDB Cloud	3		Безкоштовно
Всього			3660,00

5.2.7 Амортизація обладнання, програмних засобів та приміщень

До статті «Амортизація обладнання, програмних засобів та приміщень» відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпечення для проведення науково-дослідної роботи, за його наявності в дослідній організації або на підприємстві [37].

В спрощеному вигляді амортизаційні відрахування розраховуються за формулою:

$$A = \frac{Ц \cdot T}{12 \cdot T_B} \; [грн], \tag{5.6}$$

де $Ц$ – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, грн;

Т – фактична тривалість використання, міс;

$Т_В$ – термін використання обладнання, приміщень тощо, роки.

Розрахуємо амортизаційні витрати на ноутбук ASUS Vivobook 15. Балансова вартість блоку становить 35000 грн, а термін його використання – 3 роки, а фактична тривалість використання 3 місяці.

$$A = \frac{35000 \cdot 3}{12 \cdot 3} = 2920(грн).$$

Величина амортизаційних відрахувань наведена в таблиці 5.8.

Таблиця 5.8 – Величина амортизаційних відрахувань

№	Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, міс.	Величина амортизаційних відрахувань, грн
1	Ноутбук ASUS Vivobook 15	35000	3	3	2920,00
2	Принтер HP LaserJet 1018	10000	1	3	833,00
3	ОС Windows 11	12992	1	3	3248,00
4	Microsoft Office 2019	6160	2	3	770,00
5	Приміщення	400000	20	3	5000,00
Всього:					12771,00

5.2.8 Паливо та енергія для науково-виробничих цілей

До статті «Паливо та енергія для науково-виробничих цілей» належать витрати на придбання у сторонніх підприємств, установ і організацій будь-якого палива, що витрачається з технологічною метою на проведення досліджень [37].

Обчислимо витрати на силову електроенергію за формулою (5.7):

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} * t_i * Ц_e * K_{вni}}{\eta_i}, \quad (5.7)$$

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

t_i – тривалість роботи обладнання на етапі дослідження, год; Це – вартість 1 кВт-години електроенергії, грн;

$К_{вni}$ – коефіцієнт, що враховує використання потужності;

η_i – коефіцієнт корисної дії обладнання.

Вартість електроенергії для непобутових споживачів на даний час у 2024 році становить 7,6 грн за кВт.

$$B_e = 0,5 \cdot 544 \cdot 7,6 \cdot 0,95 / 0,97 = 2025,00 \text{ (грн)}.$$

Сумарні витрати на електроенергію занесемо в таблицю 5.9:

Таблиця 5.9 – Витрати на електроенергію

№	Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
1	Ноутбук ASUS Vivobook 15, 2шт.	0,5	544	2025,00
2	Принтер HP LaserJet 1018	0,45	20	67,00
3	Робоче місце	0,15	544	600,00
Всього				2692,00

5.2.9 Службові відрядження

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень [37].

Витрати за даною статтею відсутні.

5.2.10 Інші витрати

Інші витрати ($B_{ін}$) охоплюють: витрати на Інтернет, управління організацією, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, тощо. Інші витрати приймаємо як 50% від суми основної заробітної плати розробників [37].

Величину інших витрат розрахуємо за формулою (5.8):

$$B_{ін} = З_о \cdot 50\% \text{ [грн]}. \quad (5.8)$$

$$B_{ін} = 184000 \cdot 0,5 = 92000,00 \text{ (грн)}.$$

Обчислимо витрати на виконання даної науково-дослідної роботи, що є сумою всіх попередніх витрат:

$$B = З_о + З_д + Н_з + К + B_{прг} + А + B_е + B_{ін} \text{ [грн]}.$$

В=163818,17+16381,82+39644+4200+3660+12771+2692+92000=335166,99 (грн).

Виконаємо прогнозування загальних витрат *(ЗВ)* на виконання та впровадження результатів виконаної науково-технічної роботи за формулою:

$$ЗВ = B_{заг} / \beta \text{ [грн]}, \quad (5.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Якщо розробка знаходиться на стадії впровадження, то β ≈ 0,9.

ЗВ = 335166,99/0,9=372407,77 (грн).

Витрати на виконання наукової роботи та впровадження її результатів становитиме 372407,77 грн.

5.3 Розрахунок економічної ефективності та обґрунтування економічної доцільності комерціалізації науково-технічної розробки.

У контексті ринкової діяльності, значним позитивним результатом для потенційного інвестора від можливого впровадження науково-технічної розробки може бути зростання чистого прибутку, який він зможе отримати.

Підвищення чистого прибутку створює можливість інвесторам отримувати додаткові фінансові ресурси та покращувати фінансові показники їхньої діяльності.

Важливо, щоб маркетинговий відділ через ефективне впровадження своїх заходів забезпечив достатній інтерес аудиторії до продукту. Ми передбачаємо, що кількість користувачів буде щорічно поступово збільшуватись на 15%. Крім того, вартість послуг компанії, може поетапно збільшуватись на 10% разом із покращенням якості надаваної послуги.

Перші позитивні фінансові результати від впровадження розробки очікуються вже в перші місяці після впровадження.

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

Збільшення чистого прибутку у потенційного інвестора протягом декількох років розраховується за формулою (5.10):

$$\Delta П_i = (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}) \quad (5.10)$$

де $\Delta Ц_o$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

N – основний кількісний показник, який визначає величину попиту на аналогічні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. ρ = 0,3;
ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році ϑ = 18%.

В результаті впровадження наукової розробки покращується якість продукту, що дозволяє підвищити ціну його реалізації на 600 грн. Кількість користувачів збільшиться протягом першого року на 700, другого – 1100, третього – 900. Ціна продукції на 1 користувача до впровадження результатів наукової розробки складала 4400 грн. Кількість користувачів – 36.

Розрахуємо показник прибутку впродовж трьох років відносно базового.

Збільшення чистого прибутку $\Delta П_1$ протягом першого року складе:

$\Delta П_1 = (4400 \cdot 36 + (4400 + 600) \cdot 700) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1 - 18/100) = 749942{,}60$ (грн).

Обчислимо збільшення чистого прибутку $\Delta П_2$ протягом другого року:

$\Delta П_2 = (4400 \cdot 36 + (4400 + 600) \cdot 1100) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1 - 18/100) = 1159925{,}60$(грн).

Збільшення чистого прибутку $\Delta П_3$ протягом третього року становитиме:

$\Delta П_3 = (4400 \cdot 36 + (4400+600) \cdot 900) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1 - 18/100) = 954933{,}80$ (грн).

Отже, розрахунки показують, що комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку.

Далі розраховуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot \text{ЗВ} \qquad (5.11)$$

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

$$PV = 2 \cdot 402501{,}11 = 744815{,}54 \text{ (грн)}.$$

Чистий приведений дохід для інвестора розраховуємо за формулою:

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t} \text{ [грн]}, \quad (5.12)$$

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

т – період часу, протягом якого виявляються результати впровадженої роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; В Україні рівень інфляції за підсумком 2023 року склав 10,6 %, прогнозований рівень на 2024 рік – 8,6 %, на 2025 рік – 5,8%.

t – період часу (в роках) від моменту початку впровадження розробки до отримання інвестором чистого прибутку.

$$ПП = \frac{749942{,}60}{(1+0{,}106)^1} + \frac{1159925{,}60}{(1+0{,}086)^2} + \frac{954933{,}80}{(1+0{,}058)^3} = 2\ 467\ 895{,}05 \text{ (грн)}.$$

Розрахуємо абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$Е_{абс} = (ПП - PV), \text{ [грн]}, \quad (5.13)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

PV – теперішня вартість інвестицій, грн.

$$E_{абс} = 2467895{,}05 - 744815{,}54 = 1723079{,}51 \text{ (грн)}.$$

Оскільки $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток, але це також ще не свідчить про те, щопотенційний інвестор буде зацікавлений у фінансуванні даного проекту.

Розрахуємо ефективність вкладених у наукову розробку інвестицій $E_в$. Для цього використаємо формулу 5.14:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 \quad (5.14)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки. Від початку розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[3]{1 + \frac{1723079,51}{744815,54}} - 1 = 0,49, \text{ або } 49\%.$$

Розраховану величину $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладати не вигідно. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування визначається за формулою (5.15):

$$\tau = d + f, \quad (5.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках. В 2023 році в Україні d становила 0,14…0,2. Приймаємо d = 0,14;

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau = 0,14 + 0,05 = 0,19$$

Оскільки $E_в$ = 49%, що є більшим за τ мін, то потенційний інвестор може бути зацікавлений у фінансуванні науково-технічної розробки.

Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ можна розрахувати за формулою 5.16:

$$T_{ок} = \frac{1}{E_в}, \tag{5.16}$$

$$T_{ок} = \frac{1}{0{,}49} = 2{,}04\ (роки),$$

Оскільки термін окупності становить 2 роки ($T_{ок}$ < 3…5 років), то це свідчить про комерційну привабливість науково-технічної розробки і демонструє потенційному інвестору, що фінансування даної наукової розробки є доцільним.

5.4 Висновки

У даному розділі було проведено оцінювання комерційного потенціалу розробки програмного забезпечення управління знаннями. Проведено комерційний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки є високим.

Аналіз комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками не поступається аналогічним програмним продуктам, що підтверджує її перспективність. Програмний продукт має кращі функціональні показники, а тому є конкурентоспроможним товаром, а існуючі переваги нової розробки дозволять швидко поширити її на ринку.

Згідно із розрахунками всіх статей витрат на виконання науково-технічної роботи загальні витрати на розробку складають 402501,11 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 1723079,51 грн свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 49%, що значно вище за мінімальну бар'єрну ставку дисконтування, яка складає 19%. Це свідчить про потенційну зацікавленість інвесторів у комерціалізації нового програмного продукту.

Розрахунок економічної ефективності вкладених інвестицій та періоду їх окупності показав, що фінансування даної розробки є доцільним, адже термін окупності інвестиції складає 2 роки, а прогнозований прибуток за 3 роки складає 2 467 895,05 гривень.

З усього вищезазначеного робимо висновок про доцільність проведення науково-дослідної роботи за темою «Удосконалення методів і засобів управління знаннями в ІТ-проєкті».

ВИСНОВКИ

Результати дослідження, аналіз теоретичних та практичних розробок у сфері методів та засобів управління знаннями в ІТ-проєктах свідчать про високу актуальність цієї теми. У магістерській кваліфікаційній роботі проведено комплексне дослідження сучасних методів та засобів управління знаннями в ІТ-проєктах, а також розроблено нові підходи до їх удосконалення. На основі аналізу предметної області було визначено, що постійна еволюція технологій та зростання обсягу даних, що використовуються в ІТ-проєктах, створюють необхідність впровадження більш ефективних систем управління знаннями.

В процесі дослідження було здійснено аналіз існуючих інформаційних систем управління знаннями, виявлено їхні переваги та недоліки, а також запропоновано нові методи та засоби для підвищення ефективності цих систем. Зокрема, розроблено метод автоматизованого управління знаннями, який базується на принципах інформаційних екосистем. Цей метод враховує методологію створення програмного продукту та реалізується на основі тріади елементність, зв’язність та цілісність. Такий підхід дозволяє залучати всіх учасників ІТ-проєктів до процесу управління знаннями та покращувати якість створюваних програмних продуктів. Проаналізовані моделі управління знаннями пропонують різні підходи, що ґрунтуються на теоріях та концепціях вироблення та передачі знань, що дозволяє адаптувати їх до конкретних потреб проєкту.

У магістерській кваліфікаційній роботі досягнуто поставлену мету, яка полягала в розробці нових методів та засобів управління знаннями для ІТ-проєктів, що сприяють підвищенню ефективності та якості управлінських рішень, а також забезпечують збереження та ефективне використання знань на всіх етапах життєвого циклу проєкту. Основними досягненнями роботи є:

- Досліджено сучасні інформаційні системи управління знаннями, виявлено їхні переваги та недоліки;
- Проведено аналіз існуючих методів і засобів автоматизації та

цифровізації управління знаннями в ІТ-проєктах;

- Запропоновано нові методи удосконалення процесів управління знаннями, що базуються на сучасних технологіях та підходах;
- Розроблено моделі програмних модулів для управління знаннями, які можуть бути інтегровані в корпоративні портали;
- Створено програмні компоненти для управління знаннями, що дозволяють оптимізувати інформаційні потоки та забезпечити ефективне зберігання та використання знань.

Практична цінність отриманих результатів полягає у створенні програмних компонентів, які можуть бути використані для побудови корпоративних порталів та систем управління знаннями. Розроблені моделі та програмні модулі можуть бути впроваджені в різні ІТ-проєкти, що дозволить оптимізувати процеси управління знаннями, підвищити продуктивність праці та покращити процес ухвалення рішень. Запропоновані підходи сприяють зниженню витрат на розробку та впровадження систем управління знаннями, а також забезпечують їх високу конкурентоспроможність на ринку.

Таким чином, проведене дослідження підтвердило критичну важливість управління знаннями в ІТ-проєктах, продемонструвало ефективність запропонованих методик та розроблених програмних модулів, а також економічну доцільність їх впровадження. Результати роботи можуть бути корисними для подальших досліджень та практичного застосування в сфері управління знаннями ІТ-проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коваленко О.О., Зацепіна Л.В. Управління знаннями в проєктах. Матеріали Інтернетконференції Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 89), червень 2024 р. http://www.konferenciaonline.org.ua/ua/articles/year-4/rozdil-35/pidrozdil-106/pidrozdil2-0/

2. Davenport, Thomas H. Saving IT's Soul: Human Centered Information Management//Harvard Business Review. 1994. March-April. T. 72, № 2. C. 119-131.

3. Duhon, Bryant. It's All in our Heads // Inform. 1998. September. T. 12, № 8.

4. Bolisani E.; Bratianu C. The Emergence of Knowledge Management. In Emergent Knowledge Strategies: Strategic Thinking in Knowledge Management; Bolisani E., Bratianu C., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 23–48. URL: https://www.researchgate.net/publication/318234141_The_Emergence_of_Knowledge_Management

5. Добровінська Я.В., Ситник Н.І. Розроблення системи управління знаннями на підприємстві: зб. тез І Міжнар. наук.-практичної конференції «Бізнес, інновації, менеджмент: проблеми та перспективи» (м. Київ, 23 квітня, 2020 р.). С. 166-167. URL: http://confmanagement.kpi.ua/proc/issue/view/%D0%91%D0%86%D0%9C

6. Document360: Knowledge base Widget. URL: https://docs.document360.com/docs/knowledge-base-widget

7. Confluence: Features for knowledge management. URL: https://www.atlassian.com/software/confluence/features

8. SharePoint. URL: https://uk.wikipedia.org/wiki/SharePoint

9. Notion for engineering. URL: https://www.notion.so/product/notion-for-engineering

10. Trello для технічних відділів. URL: https://trello.com/teams/engineering

11. Ou-Yang L. Cyclic Model for knowledge management capability. A review study Arab Journal of Business Management Performance. 2014. № 4(4). P. 132.

12. Kaur V. Knowledge-Based Dynamic Capabilities, Springer International Publishing, Springer Cham: Springer Nature Switzerland, 2019. P. 33.

13. Ahmed H.M.Z., & Mohamed M.S. The effect of knowledge management critical success factors on knowledge management effectiveness and performance: An empirical research in Egyptian banking sector, The Business and Management Review Journal. 2017. № 9(2). P. 205.

14. Nonaka I. and Takeuchi H. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, New York. 1995. P. 67.

15. Bhardwaj R., Srivastava S., Mishra H. G., & Sangwan S. Exploring micro-foundations of knowledge-based dynamic capabilities in social purpose organizations. Journal of Knowledge Management. 2022. P. 53.

16. OMG Essence 1.2. URL: https://www.omg.org/spec/Essence/1.2/PDF.

17. Осташевський Б.В. Управління знаннями як інструмент формалізації процесів проєкту в умовах невизначеності. URL: https://snku.krok.edu.ua/index .php/vcheni-zapiski-universitetu-krok/article/view/280/307

18. Reich B.H., Gemino A., Sauer C. Knowledge management and project-based knowledge in IT projects: A model and preliminary empirical results // International Journal of Project Management. 2012. Vol. 30, Issue 6.

19. Development of a knowledge base on the integration of project management methodologies / Vitalii Kharuta, Oleh Karun // Bulletin of the National Technical University "KhPI". Series: Strategic management, portfolio, program and project management. 2023. No. 1(7). URL: http://pm.khpi.edu.ua/article/view/290054

20. Visual Studio Code архітектура 2021. URL: https://franzajit.medium.com /understanding-visual-studio-code-architecture-5fc411fca07

21. Keshav S. Practical Node.js: Building Real-World Scalable Web Apps. 2nd Edition. Apress, 2019. 356 p.

22. About Node.js URL: https://nodejs.org/en/about

23. Holmes A. Node.js, MongoDB, and Angular Web Development: The definitive guide to using the MEAN stack to build web applications. 2nd Edition. Addison-Wesley Professional, 2020. 480 p.

24. Kudryashov M. Pro Node.js for Developers: Everything you need to know about using Node.js in enterprise applications. Apress, 2017. 350 p.

25. Han S., Holub M. MongoDB and Mongoose: A Comprehensive Guide for Beginners. Packt Publishing, 2020. 350 с.

26. Web Application Architecture: Definition, Models, Types, and More. URL: https://hack.io/blog/web-application-architecture-definition-models-types-and-more

27. Web Application Architecture: Best Practices and Guides. URL: https:// litslink.com/blog/web-application-architecture

28. Intuitive Web Design: Hot to make your website intuitive to use. URL: https://cxl.com/blog/intuitive-web-design-how-to-make-your-website-intuitive-to-use

29. What is Angular. URL: https://angular.io/guide/what-is-angular/

30. Компоненти Angular. URL: https://web-dev.guru/angular-book/angular components

31. Create User Interface with an Angular component. URL: https://www. infragistics.com/products/ignite-ui-angular/angular/components/general/wpf-to-angular-guide/create-ui-with-components

32. Динамічні компоненти Angular 2023. URL: https://blog.bitsrc.io/ dynamic-components-in-angular-9ddc346e2742

33. Best Practices & Guidelines for web applications using Angular 2021. URL: https://blogs.halodoc.io/angular-best-practices/

34. Огляд видів тестування 2023. URL: https://training.qatestlab.com/blog /technical-articles/review-the-types-of-testing/

35. Angular 5: Unit тести | MarkupUA. URL: https://markupua.com/angular-5-unit-testi/

36. Види тестування та підходи до їх застосування 2023. URL: https:// qalearning.com.ua/

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

38. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк. Вінниця: ВНТУ, 2021. 60 с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., професор Романюк О.Н.

_____

"12" березня 2024 р.

Технічне завдання

на магістерську кваліфікаційну роботу

«Удосконалення методів і засобів управління знаннями в ІТ-проєкті»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. О.О. Коваленко

"_12_" ____березня_____2024 р.

Виконав:

_____ студент гр.ПІ-22мз Л.В. Зацепіна

"_12_" _____березня_____2024 р.

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Удосконалення методів і засобів управління знаннями в ІТ-проєкті».

Галузь застосування – автоматизовані системи управління знаннями в ІТ-проєктах.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 81 від 11 березня 2024 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності управління знаннями за рахунок запровадження нових методів та засобів управління знаннями, створення і використання програмних модулів управління знаннями для інформаційного порталу підприємства, що дозволить оптимізувати процеси передачі знань та інформації в ІТ-проєктах.

Призначення роботи – використання моделей автоматизованого управління проєктами, програмних модулів для корпоративних порталів підприємств та в системах управління знаннями.

4. Вихідні дані для проведення НДР

Середовище розробки Visual Studio Code; мови розробки – JavaScript, TypeScript; фронтенд фреймворк Angular, бекенд – Node.js і Express; база даних – MongoDB; операційна система – Windows 10, методи управління знаннями, засоби управління знаннями в ІТ-проєкті.

5. Технічні вимоги

Для функціонування систем управління знаннями достатньо задовольнити мінімальні вимоги для браузерів Інтернет: Google Chrome, Firefox Mozilla. Операційні системи Windows 8-10, Linux.

6. Конструктивні вимоги

Архітектура програмного продукту повинна відповідати естетичним та ергономічним вимогам функціонування у веб-середовищі.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Обгрунтування доцільності удосконалення методів та засобів управління знаннями	12.03.2024 –26.03.2024
2	Аналіз відомих інформаційних систем та застосунків управління знаннями	27.03.2024 – 05.04.2024
3	Моделювання та вдосконалення методів та засобів управління знаннями в ІТ-проєкті	06.04.2024 –19.04.2024
4	Реалізація та тестування програмних модулів	20.04.2024 – 31.05.2024
5	Економічна частина	26.04.2024 – 06.06.2024

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Удосконалення методів і засобів управління знаннями в ІТ-проєкті.**

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ПІ – 22мз

Науковий керівник: Коваленко О.О.

Unicheck	
Оригінальність	**91,5%**
Схожість	8,5%

Аналіз звіту подібності

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

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку Черноволик Г.О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Зацепіна Л.В.

Керівник роботи Коваленко О.О.

Додаток В – Лістинги програми

Лістинг коду сереверної частини:

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const morgan = require("morgan");
const mongoose = require("mongoose");
const cors = require("cors");
require("dotenv/config");
const authJwt = require("./helpers/jwt");
const errorHandler = require("./helpers/error-handler");

app.use(cors());
app.options("*", cors());

app.use(express.json());
app.use(morgan("tiny"));
app.use(authJwt());
app.use("/public/uploads", express.static(__dirname + "/public/uploads"));
app.use(errorHandler);

const categoriesRoutes = require("./routes/categories");
const projectsRoutes = require("./routes/projects");
const tasksRoutes = require("./routes/tasks");
const notesRoutes = require("./routes/notes");
const usersRoutes = require("./routes/users");

const api = process.env.API_URL;

app.use(`${api}/users`, usersRoutes);
app.use(`${api}/categories`, categoriesRoutes);
app.use(`${api}/notes`, notesRoutes);
app.use(`${api}/projects`, projectsRoutes);
app.use(`${api}/tasks`, tasksRoutes);

mongoose
  .connect(process.env.CONNECTION_STRING, {
    dbName: "KM_DB",
  })
  .then(() => {
    console.log("Database Connection is ready...");
  })
  .catch((err) => {
    console.log(err);
  });

app.listen(3000, () => {
  console.log("server is running http://localhost:3000");
});

const {User} = require('../models/user');
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { Project } = require('../models/project');

router.get(`/`, async (req, res) =>{
    const userList = await User.find().select('-passwordHash');

    if(!userList) {
        res.status(500).json({success: false})
```

```
    }
    res.send(userList);
})

router.get('/:id', async(req,res)=>{
    const user = await User.findById(req.params.id).select('-passwordHash');
    if(!user) {
        res.status(500).json({message: 'The user with the given ID was not found.'})
    }
    res.status(200).send(user);
})

router.post('/', async (req,res)=>{
    let user = new User({
        firstName: req.body.name,
        lastName: req.body.name,
        email: req.body.email,
        passwordHash: bcrypt.hashSync(req.body.password, 10),
        phone: req.body.phone,
        isAdmin: req.body.isAdmin,
        city: req.body.city,
        country: req.body.country,
    })
    user = await user.save();

    if(!user) return res.status(400).send('the user cannot be created!')
    res.send(user);
})

router.put('/:id',async (req, res)=> {
    const userExist = await User.findById(req.params.id);
    let newPassword
    if(req.body.password) {
        newPassword = bcrypt.hashSync(req.body.password, 10)
    } else {
        newPassword = userExist.passwordHash;
    }

    const user = await User.findByIdAndUpdate(
        req.params.id,
        {
            firstName: req.body.name,
            lastName: req.body.name,
            email: req.body.email,
            passwordHash: newPassword,
            phone: req.body.phone,
            isAdmin: req.body.isAdmin,
            city: req.body.city,
            country: req.body.country,
            projects: req.body.projects,
        },
        { new: true}
    )
    if(!user)
    return res.status(400).send('the user cannot be created!')

    res.send(user);
})

router.post('/login', async (req,res) => {
    const user = await User.findOne({email: req.body.email})

    const secret = process.env.secret;
    if(!user) {
        return res.status(400).send('The user not found');
```

```
    }
    if(user && bcrypt.compareSync(req.body.password, user.passwordHash)) {
        const token = jwt.sign(
            {
                userId: user.id,
                isAdmin: user.isAdmin
            },
            secret,
            {expiresIn : '1d'}
        )
        res.status(200).send({user: user, token: token})
    } else {
       res.status(400).send('password is wrong!');
    }
})


router.post('/register', async (req,res)=>{
    let user = new User({
        firstName: req.body.firstName,
        lastName: req.body.lastName,
        email: req.body.email,
        passwordHash: bcrypt.hashSync(req.body.password, 10),
        phone: req.body.phone,
        isAdmin: req.body.isAdmin,
        city: req.body.city,
        country: req.body.country,
    })
    user = await user.save();

    if(!user)
    return res.status(400).send('the user cannot be created!')

    res.send(user);
})


router.delete('/:id', (req, res)=>{
    User.findByIdAndRemove(req.params.id).then(user =>{
        if(user) {
            return res.status(200).json({success: true, message: 'the user is deleted!'})
        } else {
            return res.status(404).json({success: false , message: "user not found!"})
        }
    }).catch(err=>{
       return res.status(500).json({success: false, error: err})
    })
})

router.get('/projects/:id', async(req,res)=>{
    const user = await User.findById(req.params.id).select('projects');
    const projects = await Project.find({ _id: { $in: user.projects } });

    if(!user) {
        res.status(500).json({message: 'The user with the given ID was not found.'})
    }
    if(!projects) {
        res.status(500).json({message: 'The projects in user with the given ID were not found.'})
    }
    res.status(200).send(projects);
})

router.get(`/get/count`, async (req, res) =>{
    const userCount = await User.countDocuments((count) => count)
```

```
    if(!userCount) {
        res.status(500).json({success: false})
    }
    res.send({
        userCount: userCount
    });
})
module.exports = router;

const express = require('express');
const router = express.Router();
const multer = require('multer');
const { Note } = require('../models/note');
const { Comment } = require('../models/comment');

const FILE_TYPE_MAP = {
    'image/png': 'png',
    'image/jpeg': 'jpeg',
    'image/jpg': 'jpg'
};

const storage = multer.diskStorage({
    destination: function (req, file, cb) {
        const isValid = FILE_TYPE_MAP[file.mimetype];
        let uploadError = new Error('invalid image type');

        if (isValid) {
            uploadError = null;
        }
        cb(uploadError, 'public/uploads');
    },
    filename: function (req, file, cb) {
        const fileName = file.originalname.split(' ').join('-');
        const extension = FILE_TYPE_MAP[file.mimetype];
        cb(null, `${fileName}-${Date.now()}.${extension}`);
    }
});

const uploadOptions = multer({ storage: storage });

router.get(`/`, async (req, res) =>{
    const noteList = await Note.find();

    if(!noteList) {
        res.status(500).json({success: false})
    }
    res.status(200).send(noteList);
});

router.get('/:id', async(req,res)=>{
    const note = await Note.findById(req.params.id);

    if(!note) {
        res.status(500).json({message: 'The note with the given ID was not found.'})
    }
    res.status(200).send(note);
});

router.post('/', uploadOptions.single('image'), async (req, res) => {
    console.log('Creation of note ', req.body);

    const file = req.file;
    let imagePath;

    if (file) {
```

```
        const fileName = file.filename;
        const basePath = `${req.protocol}://${req.get('host')}/public/uploads/`;
        imagePath = `${basePath}${fileName}`;
    } else {
        imagePath = '';
    }

    try {
      let note = new Note({
        title: req.body.title,
        content: req.body.content,
        image: `${basePath}${fileName}`,
        category: req.body.category,
        authorId: req.body.authorId,
        project: req.body.project
      });
      note = await note.save();
      if (!note) {
        return res.status(400).send('The note cannot be created!');
      }
      res.status(201).send(note);
    } catch (error) {
      res.status(500).json({ message: 'An error occurred while creating the note.', error: error.message });
    }
});


router.put('/:noteId', uploadOptions.single('image'), async (req, res) => {
    const note = await Note.findById(req.params.noteId);
    if (!note) {
        return res.status(400).send('The note with the given ID was not found.');
    }

    const file = req.file;
    let imagePath;

    if (file) {
        const fileName = file.filename;
        const basePath = `${req.protocol}://${req.get('host')}/public/uploads/`;
        imagePath = `${basePath}${fileName}`;
    } else {
        imagePath = note.image;
    }

    const updatedData = {
        title: req.body.title || note.title,
        content: req.body.content || note.content,
        image: imagePath,
        category: req.body.category || note.category,
        authorId: note.authorId,
        project: note.project
    };

    const updatedNote = await Note.findByIdAndUpdate(
        req.params.noteId,
        updatedData,
        { new: true }
    );

    console.log(updatedNote);

    if (!updatedNote) {
        return res.status(400).send('The note cannot be updated!');
    }
```

```
    res.send(updatedNote);
});

router.delete('/:noteId', async (req, res)=>{
    try {
        const note = await Note.findByIdAndDelete(req.params.noteId);
        if (note) {
            await Comment.deleteMany({ noteId: req.params.noteId });
            return res.status(200).json({ success: true, message: 'The note and its comments are deleted!'
});
        } else {
            return res.status(404).json({ success: false, message: 'Note not found!' });
        }
    } catch (err) {
        return res.status(500).json({ success: false, error: err });
    }
});

router.get('/notesByUser/:userId', async(req,res)=>{
    const notes = await Note.find({ authorId: req.params.userId });
    if(!notes) {
        res.status(500).json({message: 'The notes in user with the given ID were not found.'})
    }
    res.status(200).send(notes);
});

router.get('/notesByProject/:projectId', async(req,res)=>{
    const notes = await Note.find({ project: req.params.projectId });
    if(!notes) {
        res.status(500).json({message: 'The notes in project with the given ID were not found.'})
    }
    res.status(200).send(notes);
});

router.get('/comments/:noteId', async(req,res)=>{
    const comments = await Comment.find({ noteId: req.params.noteId });
    if(!comments) {
        res.status(500).json({message: 'The comments in note with the given ID were not found.'})
    }
    res.status(200).send(comments);
});

router.post('/commentCreate', async (req,res)=>{
    let comment = new Comment({
        content: req.body.content,
        image: req.body.image,
        noteId: req.body.noteId,
        authorId: req.body.authorId
    })
    comment = await comment.save();

    if(!comment)
    return res.status(400).send('the comment can not be created!')

    res.send(comment);
});

router.delete('/comments/:commentId', (req, res)=>{
    Comment.findByIdAndDelete(req.params.commentId).then(comment =>{
        if(comment) {
            return res.status(200).json({success: true, message: 'the comment is deleted!'})
        } else {
            return res.status(404).json({success: false , message: "comment not found!"})
        }
    }).catch(err=>{
```

```
        return res.status(500).json({success: false, error: err})
    })
});
module.exports = router;

const { Project } = require('../models/project');
const { User } = require('../models/user');
const { Task } = require('../models/task');
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');

router.get(`/`, async (req, res) => {
    try {
        const projectList = await Project.find();
        if (!projectList || projectList.length === 0) {
            return res.status(404).json({ success: false, message: 'No projects found' });
        }
        res.status(200).send(projectList);
    } catch {
        res.status(500).json({ success: false, message: 'An error occurred while retrieving projects' });
    }
});

router.get('/:projectId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.projectId)) {
        return res.status(400).json({ success: false, message: 'Invalid Project ID' });
    }
    try {
        const project = await Project.findById(req.params.projectId);
        if (!project) {
            return res.status(404).json({ message: 'The project with the given ID was not found.' });
        }
        res.status(200).send(project);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving the project' });
    }
});

router.get('/userProjectsByUserId/:userId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.userId)) {
        return res.status(400).json({ success: false, message: 'Invalid User ID' });
    }
    try {
        const user = await User.findById(req.params.userId).select('projects');
        if (!user) return res.status(404).json({ message: 'User not found.' });

        const userProjectsIds = user.projects;
        const projects = await Project.find({ _id: { $in: userProjectsIds } });

        if (!projects.length) {
            return res.status(404).json({ message: 'The user with the given ID doesn\'t have active
projects.' });
        }

        res.status(200).send(projects);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving projects.' });
    }
});

router.get('/teamByprojectId/:projectId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.projectId)) {
        return res.status(400).json({ success: false, message: 'Invalid project ID' });
    }
```

```
    try {
        const projectTeam = await Project.findById(req.params.projectId).select('team');
        if (!projectTeam) return res.status(404).json({ message: 'Project not found.' });
        const projectTeamIds = projectTeam.team;
        const team = await User.find({ _id: { $in: projectTeamIds } });

        if (!team.length) {
            return res.status(404).json({ message: 'The project with the given ID doesn\'t have active
members.' });
        }
        res.status(200).send(team);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving team.' });
    }
});
router.get('/tasksByProjectId/:projectId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.projectId)) {
        return res.status(400).json({ success: false, message: 'Invalid project ID' });
    }
    try {
        const tasks = await Task.find({ projectId: { $in: req.params.projectId } });

        if (!tasks.length) {
            return res.status(404).json({ message: 'The project with the given ID doesn\'t have tasks.' });
        }
        res.status(200).send(tasks);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving tasks.' });
    }
});

router.get('/taskById/:taskId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.taskId)) {
        return res.status(400).json({ success: false, message: 'Invalid task ID' });
    }
    try {
        const task = await Task.findById(req.params.taskId);

        if (!task) {
            return res.status(404).json({ message: 'The task with the given ID doesn\'t exist.' });
        }
        res.status(200).send(task);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving task.' });
    }
});

router.post('/userProjectsByIds', async (req, res) => {
    const ids = req.body;
    if (!Array.isArray(ids) || !ids.every(id => mongoose.Types.ObjectId.isValid(id))) {
        return res.status(400).json({ message: 'Invalid IDs' });
    }
    try {
        const objects = await Project.find({ _id: { $in: ids } });
        if (!objects || objects.length === 0) {
            return res.status(404).json({ message: 'The projects with the given IDs were not found.' });
        }
        res.status(200).send(objects);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving projects.' });
    }
});

router.post('/', async (req, res) => {
    try {
```

```
        let project = new Project({
            name: req.body.name,
            dateStart: req.body.dateStart,
            dateEnd: req.body.dateEnd,
            team: req.body.team,
            tasks: req.body.tasks
        });
        project = await project.save();
        if (!project) {
            return res.status(400).send('The project cannot be created!');
        }
        res.send(project);
    } catch {
        res.status(500).json({ message: 'An error occurred while creating the project.' });
    }
});

router.put('/:id', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
        return res.status(400).json({ success: false, message: 'Invalid Project ID' });
    }
    try {
        const project = await Project.findByIdAndUpdate(
            req.params.id,
            {
                name: req.body.title,
                dateStart: req.body.dateStart,
                dateEnd: req.body.dateEnd,
                team: req.body.team,
                tasks: req.body.tasks
            },
            { new: true }
        );
        if (!project) {
            return res.status(400).send('The project cannot be updated!');
        }
        res.send(project);
    } catch {
        res.status(500).json({ message: 'An error occurred while updating the project.' });
    }
});

router.delete('/:id', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
        return res.status(400).json({ success: false, message: 'Invalid Project ID' });
    }
    try {
        const project = await Project.findByIdAndRemove(req.params.id);
        if (project) {
            return res.status(200).json({ success: true, message: 'The project is deleted!' });
        } else {
            return res.status(404).json({ success: false, message: 'Project not found!' });
        }
    } catch {
        return res.status(500).json({ success: false, message: 'An error occurred while deleting the
project.' });
    }
});
module.exports = router;

const { Project } = require('../models/project');
const { User } = require('../models/user');
const { Task } = require('../models/task');
const express = require('express');
const router = express.Router();
```

```
const mongoose = require('mongoose');

router.get('/:taskId', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.taskId)) {
        return res.status(400).json({ success: false, message: 'Invalid task ID' });
    }
    try {
        const task = await Task.findById(req.params.taskId);

        if (!task) {
            return res.status(404).json({ message: 'The task with the given ID doesn\'t exist.' });
        }
        res.status(200).send(task);
    } catch {
        res.status(500).json({ message: 'An error occurred while retrieving task.' });
    }
});

router.post('/', async (req, res) => {
    try {
        let task = new Task({
            title: req.body.title,
            content: req.body.content,
            image: req.body.image,
            storyPoint: req.body.storyPoint,
            dateStart: req.body.dateStart,
            dateEnd: req.body.dateEnd,
            status: req.body.status,
            projectId: req.body.projectId,
            assigneeId: req.body.assigneeId,
            reporterId: req.body.reporterId,
            taskStatus: req.body.taskStatus
        });
        console.log(task);

        task = await task.save();
        console.log(task);

        if (!task) {
            return res.status(400).send('The task cannot be created!');
        }
        res.send(task);
    } catch {
        res.status(500).json({ message: 'An error occurred while creating the task.' });
    }
});

router.put('/:id', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
        return res.status(400).json({ success: false, message: 'Invalid task ID' });
    }
    try {
        const task = await Task.findByIdAndUpdate(
            req.params.id,
            {
                title: req.body.title,
                content: req.body.content,
                image: req.body.image,
                storyPoint: req.body.storyPoint,
                dateStart: req.body.dateStart,
                dateEnd: req.body.dateEnd,
                status: req.body.status,
                projectId: req.body.projectId,
                assaineeId: req.body.assaineeId,
                reporterId: req.body.reporterId,
```

```
                taskStatus: req.body.taskStatus
            },
            { new: true }
        );
        if (!task) {
            return res.status(400).send('The task cannot be updated!');
        }
        res.send(task);
    } catch {
        res.status(500).json({ message: 'An error occurred while updating the task.' });
    }
});

router.delete('/:id', async (req, res) => {
    if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
        return res.status(400).json({ success: false, message: 'Invalid task ID' });
    }
    try {
        const task = await Task.findByIdAndRemove(req.params.id);
        if (task) {
            return res.status(200).json({ success: true, message: 'The task is deleted!' });
        } else {
            return res.status(404).json({ success: false, message: 'Task not found!' });
        }
    } catch {
        return res.status(500).json({ success: false, message: 'An error occurred while deleting the task.'
});
    }
});
module.exports = router;

const {Category} = require('../models/category');
const express = require('express');
const router = express.Router();

router.get(`/`, async (req, res) =>{
    const categoryList = await Category.find();

    if(!categoryList) {
        res.status(500).json({success: false})
    }
    res.status(200).send(categoryList);
})

router.get('/:id', async(req,res)=>{
    const category = await Category.findById(req.params.id);

    if(!category) {
        res.status(500).json({message: 'The category with the given ID was not found.'})
    }
    res.status(200).send(category);
})

router.post('/', async (req,res)=>{
    let category = new Category({
        name: req.body.name
    })
    category = await category.save();

    if(!category)
    return res.status(400).send('the category cannot be created!')

    res.send(category);
})
```

```
router.put('/:id',async (req, res)=> {
    const category = await Category.findByIdAndUpdate(
        req.params.id,
        {
            name: req.body.name
        },
        { new: true}
    )

    if(!category)
    return res.status(400).send('the category cannot be updated!')

    res.send(category);
})

router.delete('/:id', (req, res)=>{
    Category.findByIdAndRemove(req.params.id).then(category =>{
        if(category) {
            return res.status(200).json({success: true, message: 'the category is deleted!'})
        } else {
            return res.status(404).json({success: false , message: "category not found!"})
        }
    }).catch(err=>{
       return res.status(500).json({success: false, error: err})
    })
})
module.exports = router;

function errorHandler(err, req, res, next) {
    if (err.name === 'UnauthorizedError') {
        return res.status(401).json({message: "The user is not authorized"})
    }

    if (err.name === 'ValidationError') {
        return res.status(401).json({message: err})
    }
    return res.status(500).json(err);
}
module.exports = errorHandler;

const expressJwt = require('express-jwt');
function authJwt() {
    const secret = process.env.secret;
    const api = process.env.API_URL;
    return expressJwt({
        secret,
        algorithms: ['HS256'],
        isRevoked: isRevoked
    }).unless({
        path: [
            { url: /\/public\/uploads(.*)/, methods: ['GET', 'OPTIONS'] },
            { url: /\/api\/v1\/categories(.*)/, methods: ['GET', 'OPTIONS'] },
            { url: /\/api\/v1\/notes(.*)/, methods: ['GET', 'OPTIONS', 'POST','PUT', 'DELETE'] },
            { url: /\/api\/v1\/projects(.*)/, methods: ['GET', 'OPTIONS', 'POST'] },
            { url: /\/api\/v1\/tasks(.*)/, methods: ['GET', 'OPTIONS', 'POST'] },
            { url: /\/api\/v1\/users(.*)/, methods: ['GET', 'OPTIONS', 'POST'] },
            `${api}/users/login`,
            `${api}/users/register`,
        ]
    });
}
async function isRevoked(req, payload, done) {
    if (!payload.isAdmin) {
        done(null, true);
    }
```

```
    done();
}
module.exports = authJwt;

const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    email: { type: String, required: true },
    phone: { type: String, required: true },
    city: { type: String, default: '' },
    country: { type: String, default: '' },
    isAdmin: { type: Boolean, default: false },
    passwordHash: { type: String, required: true },
    role: { type: String, default: '' },
    projects: [
        { type: mongoose.Schema.Types.ObjectId, ref: 'Project' }
    ]
});
userSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
userSchema.set('toJSON', {
    virtuals: true,
});
exports.User = mongoose.model('User', userSchema);
exports.userSchema = userSchema;

const mongoose = require('mongoose');
const noteSchema = mongoose.Schema({
    title: { type: String, required: true },
    content: { type: String },
    image: { type: String },
    category: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Category'
    },
    project: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Project'
    },
    dateCreated: { type: Date, default: Date.now },
    dateUpdated: { type: Date, default: Date.now },
    authorId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
});
noteSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
noteSchema.set('toJSON', {
    virtuals: true
});
exports.Note = mongoose.model('Note', noteSchema);

const mongoose = require('mongoose');
const commentSchema = mongoose.Schema({
    content: { type: String },
    image: { type: String },
    noteId: { type: mongoose.Schema.Types.ObjectId, ref: 'Note' },
    authorId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    dateCreated: { type: Date, default: Date.now },
    dateUpdated: { type: Date, default: Date.now },
});
```

```
commentSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
commentSchema.set('toJSON', {
    virtuals: true
});
exports.Comment = mongoose.model('Comment', commentSchema);

const mongoose = require('mongoose');
const taskSchema = mongoose.Schema({
  title: { type: String, required: true },
  content: { type: String },
  image: { type: String },
  storyPoint: { type: Number },
  dateStart: { type: Date, default: Date.now },
  dateEnd: { type: Date, default: Date.now },
  status: { type: String },
  projectId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project'
  },
  assigneeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  reporterId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  taskStatus: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'TaskStatus'
  },
});
taskSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
taskSchema.set('toJSON', {
    virtuals: true
});
exports.Task = mongoose.model('Task', taskSchema);

const mongoose = require('mongoose');
const taskStatusSchema = mongoose.Schema({
    name: {
        type: String,
        required: true,
    }
})
taskStatusSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
taskStatusSchema.set('toJSON', {
    virtuals: true,
});
exports. TaskStatus = mongoose.model('TaskStatus', taskStatusSchema);

const mongoose = require('mongoose');
const projectSchema = mongoose.Schema({
    name: { type: String, required: true },
    dateStart: { type: Date, required: true },
    dateEnd: { type: Date, required: true },
    team: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
    tasks: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Task' }],
});
```

```
projectSchema.virtual('id').get(function () {
    return this._id.toHexString();
});
projectSchema.set('toJSON', {
    virtuals: true
});
exports.Project = mongoose.model('Project', projectSchema);
```

Лістинг коду клієнтської частини:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { authGuard } from '@shared/services/auth.guard';

const routes: Routes = [
  { path: 'auth', loadChildren: () => import('./features/auth/auth.module').then(m => m.AuthModule) },
  { path: 'notes', loadChildren: () => import('./features/notes/notes.module').then(m => m.NotesModule),
canActivate: [authGuard] },
  { path: 'projects', loadChildren: () => import('./features/projects/projects.module').then(m =>
m.ProjectsModule), canActivate: [authGuard] },
  { path: 'jira', loadChildren: () => import('./features/jira/jira.module').then(m => m.JiraModule),
canActivate: [authGuard] },
  { path: 'people', loadChildren: () => import('./features/people/people.module').then(m => m.PeopleModule),
canActivate: [authGuard]},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

import { inject } from '@angular/core';
import { CanActivateFn, Router } from '@angular/router';
import { of } from 'rxjs';
import { catchError, switchMap } from 'rxjs/operators';
import { AuthService } from './auth.service';

export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);

  return authService.getCurrentUser().pipe(
    switchMap((user) => {
      if (user) {
        return of(true);
      } else {
        router.navigate(['/auth/login']);
        return of(false);
      }
    }),
    catchError(() => {
      router.navigate(['/auth/login']);
      return of(false);
    })
  );
};

import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { BehaviorSubject, Observable, throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';

import { User } from '@shared/models/user.model';
```

```
@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private currentUserSubject: BehaviorSubject<User | null> = new BehaviorSubject<User | null>(null);
  clearLogoutTimeout: any;
  api: string = `http://localhost:3000/api/v1`;

  constructor(private http: HttpClient) {
    if (this.isValidToken()) {
      this.currentUserSubject.next(JSON.parse(localStorage.getItem('user')!));
    } else {
      this.logout();
    }
  }

  setCurrentUser(user: User): void {
    this.currentUserSubject.next(user);
  }

  getCurrentUser(): Observable<User | null> {
    return this.currentUserSubject.asObservable();
  }

  registerUser(user: User): Observable<User> {
    return this.http.post<User>(`${this.api}/users/register`, user).pipe(
      map((response: User) => {
        if (response.token) {
          localStorage.setItem('token', response.token);
        }
        return response;
      }),

      catchError(this.handleError),
    );
  }

  login(email: string, password: string): Observable<User> {
    let userData = { email, password };
    return this.http.post<{ user: User, token: string }>(`${this.api}/users/login`, userData).pipe(
      map((response) => {
        if (response.token) {
          localStorage.setItem('user', JSON.stringify(response.user));
          localStorage.setItem('token', response.token);

          this.currentUserSubject.next(response.user);
          this.autoLogout(response.token);
        }
        return response.user;
      }),

      catchError(this.handleError),
    );
  }

  getUserInfo(userId: string): Observable<User> {
    return this.http.get<User>(`${this.api}/users/${userId}`).pipe(
      catchError(this.handleError),
    );
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(`${this.api}/users/${user.id}`, user).pipe(
      map((response: User) => {
```

```
      if (response.token) {
        localStorage.setItem('token', response.token);
        localStorage.setItem('user', JSON.stringify(response));
        this.autoLogout(response.token);
      }
      this.setCurrentUser(response);
      return response;
    }),
    catchError(this.handleError),
  );
}

logout(): void {
  localStorage.removeItem('token');
  localStorage.removeItem('user');
  this.currentUserSubject.next(null);
  if (this.clearLogoutTimeout) {
    clearTimeout(this.clearLogoutTimeout);
  }
}

autoLogout(token: string): void {
  let expirationDate = this.getExpirationDate(token);
  let timeUntilExpiration = expirationDate - Date.now();

  if (timeUntilExpiration <= 0) {
    this.logout();
    return;
  }

  this.clearLogoutTimeout = setTimeout(() => {
    this.logout();
  }, timeUntilExpiration);
}

getAllUsers(): Observable<User[]> {
  return this.http.get<User[]>(`${this.api}/users`).pipe(
    catchError(this.handleError),
  );
}

private isValidToken (): boolean {
  let token = localStorage.getItem('token')!;
  if (token) {
    let expirationDate = this.getExpirationDate(token);
    let timeUntilExpiration = expirationDate - Date.now();
    return timeUntilExpiration > 0;
  }
  return false;
}

private getExpirationDate(token: string): number {
  const payload = this.decodeJwtToken(token);
  if (payload && payload.exp) {
    return payload.exp * 1000;
  }
  return 0;
}

private decodeJwtToken(token: string) {
  const basePayload = token.split('.')[1];
  if (basePayload) {
    const payload = basePayload.replace(/-/g, '+').replace(/_/g, '/');
    const paddedPayload = payload.padEnd(payload.length + ((4 - (payload.length % 4)) % 4), '=');
```

```
      try {
        const decodedPayload = atob(paddedPayload);
        return JSON.parse(decodedPayload);
      } catch (error) {
        console.error('Failed to decode JWT:', error);
        return '';
      }
    }
  }

  private handleError(error: HttpErrorResponse) {
    let errorMessage: string = 'An unknown error occurred!';

    if (error.error instanceof ErrorEvent) {
      errorMessage = `Client Error: ${error.error.message}`;
    } else {
      errorMessage = `Server Error: ${error}`;
      console.error(error);
    }
    console.error(errorMessage);
    return throwError(() => new Error(errorMessage));
  }
}

import { ChangeDetectorRef, Component, OnInit } from '@angular/core';
import { AuthService } from '@shared/services/auth.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrl: './header.component.scss'
})
export class HeaderComponent implements OnInit {

  constructor(private auth: AuthService, private cdr: ChangeDetectorRef) {}
  isLoggedIn: boolean = false;
  initials: string = '';

  ngOnInit() {
    if(this.auth.getCurrentUser()){
      this.auth.getCurrentUser().subscribe((user)=>{
        if (user) {
          this.isLoggedIn = true;
        } else {
          this.isLoggedIn = false;
        }
        this.cdr.markForCheck();
      });
    };

    if(this.auth.getCurrentUser()){
      this.auth.getCurrentUser().subscribe((user)=>{
        if (user) {
          let words: string[] = [user.firstName!, user.lastName!]
          this.initials =  words.map(s => s[0]).join('').toUpperCase();
        } else {
          this.initials = '';
        }
        this.cdr.markForCheck();
      });
    };
  }

  onLogout() {
    this.auth.logout();
```

```
  }
}

import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';

import { AuthService } from '@shared/services/auth.service'

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
})
export class LoginComponent implements OnInit {
  loginForm!: FormGroup;
  showPassword: boolean = false;
  errorMessage: string = '';

  constructor(
    private fb: FormBuilder,
    private authService: AuthService,
    private router: Router,
  ) {}

  ngOnInit(): void {
    this.createForm();
  }

  createForm(): void {
    this.loginForm = this.fb.group({
      email: ['',
        [
          Validators.required,
          Validators.pattern(/^([^\s@]+@[^\s@]+\.[^\s@]+)$/),
        ],
      ],
      password: ['',
        [
          Validators.required,
          Validators.minLength(8),
          Validators.pattern(/^(?=.*[A-Za-z])(?=.*\d)(?=.*[&#%!]).+$/),
        ],
      ],
    });
  }

  get email(): FormControl {
    return this.loginForm.get('email') as FormControl;
  }

  get password(): FormControl {
    return this.loginForm.get('password') as FormControl;
  }

  onLogin(): void {
    if (this.loginForm.valid) {

      this.authService.login(this.email?.value, this.password?.value).subscribe({
        next: () => {
          this.onClose();
          alert('You logined successfully.');
          this.router.navigateByUrl('projects');
        },
        error: (error) => {
```

```
          this.errorMessage = error.message;
          setTimeout(() => {
            this.onClose();
          }, 2000);
          alert('You couldn\'t login.');
          this.router.navigateByUrl('auth/register');
        },
      });
    }
  }

  onClose() {
    this.loginForm.reset();
  }

  toggleShowPassword(): void {
    this.showPassword = !this.showPassword;
  }
}
```

```
<div class="login">
  <div class="login__logo">
      <img src="./assets/KM_LOGO_3.png" alt="">
  </div>

  <div class="login__content">
    <h1 class="login__title">{{ "Log In" | titlecase }}</h1>
    <form class="login__form" [formGroup]="loginForm" (ngSubmit)="onLogin()">

      <div class="login__block loginBlock">
        <label for="email" class="loginBlock__label">Phone number or Email</label>
        <input
          type="text"
          name="email"
          id="email"
          class="loginBlock__input"
          formControlName="email"
          [ngClass]="{
            invalidInput:
              loginForm.get('email')?.invalid &&
              (loginForm.get('email')?.dirty || loginForm.get('email')?.touched)
          }"
          placeholder="name@example.com"
        />
        <div *ngIf="email?.invalid && (email?.touched || email?.dirty)" class="error-message">
          <div *ngIf="email?.errors?.['required']">Phone number or Email is required</div>
          <div *ngIf="email?.errors?.['minlength']">Phone number or Email must be at least 5 characters
long</div>
          <div *ngIf="email?.errors?.['pattern']
            && !email?.errors?.['minlength']
            && !email?.errors?.['maxlength']">Input must contain only email or numbers and +</div>
          <div *ngIf="email?.errors?.['maxlength']">Phone must be no more than 20 characters</div>
        </div>
      </div>
```

```
      <div class="login__block loginBlock">
        <label for="password" class="loginBlock__label">Password</label>
        <input
          type="{{ showPassword ? 'text' : 'password' }}"
          name="password"
          id="password"
          class="loginBlock__input"
          formControlName="password"
          [ngClass]="{
            invalidInput:
              loginForm.get('password')?.invalid &&
              (loginForm.get('password')?.dirty || loginForm.get('password')?.touched)
          }" />
        <img
          [src]="showPassword ? '../../../../assets/password-icons/show-
icon.png':'../../../../assets/password-icons/hide-icon.png'"
          (click)="toggleShowPassword()"
          class="eye" />
        <div *ngIf="password?.invalid && (password?.touched || password?.dirty)" class="error-message">
          <div *ngIf="password?.errors?.['required']">Password is required</div>
          <div *ngIf="password?.errors?.['minlength']">Password must be at least 8 characters long</div>
          <div *ngIf="password?.errors?.['pattern']
            && !password?.errors?.['minlength']
            && !password?.errors?.['maxlength']">Password must contain a letter, a number and a special
character &,#,!,%</div>
          <div *ngIf="password?.errors?.['maxlength']">Password must be no more than 255 characters</div>
        </div>
      </div>
      <div class="login__forgot">
        <a routerLink="/auth/register">Don't have an account? Register!</a>
      </div>
      <button class="login__close" (click)="onClose()">
        <div class="btn__inner">
          <div class="line"></div>
          <div class="line"></div>
        </div>
      </button>
      <div class="errorMessage">
        <div *ngIf="errorMessage" class="error-message">{{ errorMessage }}</div>
      </div>
      <button type="submit" class="login__btn" [disabled]="!loginForm.valid">
        Log in
      </button>
    </form>
  </div>
</div>

import { Component, OnDestroy, OnInit } from '@angular/core';
import { FormBuilder, FormControl, FormGroup, ValidationErrors, Validators } from '@angular/forms';
```

```
import { User } from '@shared/models/user.model';
import { AuthService } from '@shared/services/auth.service';
import { ActivatedRoute, Router } from '@angular/router';
import { Observable, Subject, switchMap, takeUntil } from 'rxjs';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.scss'],
})
export class RegisterComponent implements OnInit, OnDestroy {
  form!: FormGroup;
  showPassword: boolean = false;
  showConfirmPassword: boolean = false;
  errorMessage: string = '';
  isUpdate: boolean = false;
  private destroy$ = new Subject<void>();

  constructor(
    private fb: FormBuilder,
    private authService: AuthService,
    private router: Router,
    private activatedRoute: ActivatedRoute
  ) {}

  ngOnInit(): void {
    this.createForm();
    this.checkFormMode();
  }

  createForm(): void {
    this.form = this.fb.group(
      {
        firstName: ['',
          [
            Validators.required,
            Validators.maxLength(255),
            Validators.minLength(1),
            Validators.pattern(/^[A-Za-z]+$/),
          ],
        ],
        lastName: ['',
          [
            Validators.required,
            Validators.maxLength(255),
            Validators.minLength(1),
            Validators.pattern(/^[A-Za-z]+$/),
          ],
        ],
        email: ['',
          [
            Validators.required,
            Validators.email,
            Validators.maxLength(255),
            Validators.pattern(/^[^\s@]+@[^\s@]+\.[^\s@]+$/),
          ],
        ],
        phone: ['',
          [
            Validators.required,
            Validators.minLength(6),
            Validators.maxLength(20),
            Validators.pattern(/^\+\d+(-?\d+)*$/),
          ],
        ],
```

```
      country: ['',
        [
          Validators.required,
          Validators.maxLength(255),
          Validators.minLength(1),
          Validators.pattern(/^[A-Za-z]+$/),
        ],
      ],
      city: ['',
        [
          Validators.required,
          Validators.maxLength(255),
          Validators.minLength(1),
          Validators.pattern(/^[A-Za-z]+$/),
        ],
      ],
      isAdmin: false,
      password: ['',
        [
          Validators.required,
          Validators.maxLength(255),
          Validators.minLength(8),
          Validators.pattern(/^(?=.*[A-Za-z])(?=.*\d)(?=.*[&#%!]).+$/),
        ],
      ],
      confirmPassword: ['', [Validators.required]],
    },
    { validators: this.passwordMatchValidator }
  );
}

get firstName(): FormControl {
  return this.form.get('firstName') as FormControl;
}

get lastName(): FormControl {
  return this.form.get('lastName') as FormControl;
}

get email(): FormControl {
  return this.form.get('email') as FormControl;
}

get phone(): FormControl {
  return this.form.get('phone') as FormControl;
}

get country(): FormControl {
  return this.form.get('country') as FormControl;
}

get city(): FormControl {
  return this.form.get('city') as FormControl;
}

get isAdmin(): FormControl {
  return this.form.get('isAdmin') as FormControl;
}

get password(): FormControl {
  return this.form.get('password') as FormControl;
}

get confirmPassword(): FormControl {
  return this.form.get('confirmPassword') as FormControl;
```

```
}

getTitle(): string {
  return this.isUpdate ? 'Update profile information' : 'Create new account';
}

getButtonText(): string {
  return this.isUpdate ? 'Save' : 'Register';
}

passwordMatchValidator(formGroup: FormGroup): ValidationErrors | null {
  const password = formGroup.get('password')?.value;
  const confirmPassword = formGroup.get('confirmPassword')?.value;
  if (password !== confirmPassword) {
    formGroup.get('confirmPassword')?.setErrors({ passwordMismatch: true });
    return { passwordMismatch: true };
  }
  formGroup.get('confirmPassword')?.setErrors(null);
  return null;
}

onSubmit(): void {
  if (this.form.valid) {
    const userData: User = {
      firstName: this.firstName?.value,
      lastName: this.lastName?.value,
      email: this.email?.value,
      phone: this.phone?.value,
      city: this.city?.value,
      country: this.country?.value,
      isAdmin: this.isAdmin.value,
      password: this.password?.value,
    };

    const request: Observable<User> = this.isUpdate
      ? this.authService.updateUser(userData)
      : this.authService.registerUser(userData);

    request.pipe().subscribe({
        next: () => this.handleSuccess(),
        error: (error) => this.handleError(error),
      });
  }
}

toggleShowPassword(): void {
  this.showPassword = !this.showPassword;
}

toggleShowConfirmPassword(): void {
  this.showConfirmPassword = !this.showConfirmPassword;
}

private checkFormMode(): void {
  const token = localStorage.getItem('token');
  if (token) {
    this.isUpdate = true;
  }
  this.activatedRoute.data
    .pipe(takeUntil(this.destroy$))
    .subscribe((data) => {
      if (data['user']) {
        this.isUpdate = true;
        this.form.patchValue(data['user']);
      }
```

```
      });
  }

  private handleSuccess(): void {
    this.router.navigateByUrl('auth/login');
  }

  private handleError(error: any): void {
    this.errorMessage = error.message;
    const redirectUrl = this.isUpdate ? '/users/update' : '/users/register';
    this.router.navigateByUrl(redirectUrl);
  }

  onClose() {
    this.form.reset();
    this.router.navigateByUrl('');
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }
}

<div class="container">
  <div class="form__wrapper">
    <form class="form" [formGroup]="form" (ngSubmit)="onSubmit()" appTabDirective>
      <div class="form__header">
        <div class="header__logo">
          <img src="./assets/KM_LOGO_3.png" alt="">
        </div>
      </div>
      <div class="form__body">
        <h2 class="form__title">{{ getTitle() | titlecase }}</h2>
        <div class="form__group">
          <label for="firstName" class="form__label">First name</label>
          <input type="text" id="firstName" class="form__input" formControlName="firstName"
placeholder="John"
                 name="firstName"
                 [ngClass]="{ 'error': firstName.invalid && (firstName.touched || firstName.dirty)}">

          <div *ngIf="firstName?.invalid && (firstName?.touched || firstName?.dirty)" class="error-message">
            <div *ngIf="firstName?.errors?.['required']">First name is required</div>
            <div *ngIf="firstName?.errors?.['pattern'] && !firstName?.errors?.['maxlength']">First name must
contain only letters</div>
            <div *ngIf="firstName?.errors?.['maxlength']">First name must be no more than 255
characters</div>
          </div>
        </div>
        <div class="form__group">
          <label for="lastName" class="form__label">Last name</label>
          <input type="text" id="lastName" class="form__input" formControlName="lastName" placeholder="Doe"
                 name="lastName"
                 [ngClass]="{ 'error': lastName.invalid && (lastName.touched || lastName.dirty)}">
          <div *ngIf="lastName?.invalid && (lastName?.touched || lastName?.dirty)" class="error-message">
```

```
            <div *ngIf="lastName?.errors?.['required']">Last name is required</div>
            <div *ngIf="lastName?.errors?.['pattern'] && !lastName?.errors?.['maxlength']">Last name must
contain only letters</div>
            <div *ngIf="lastName?.errors?.['maxlength']">Last name must be no more than 255 characters</div>
          </div>
        </div>
        <div class="form__group">
          <label for="email" class="form__label">Email</label>
          <input type="email" id="email" class="form__input" formControlName="email"
placeholder="name@example.com" name="email"
        [ngClass]="{ 'error': email.invalid && (email.touched || email.dirty)}">
          <div *ngIf="email?.invalid && (email?.touched || email?.dirty)" class="error-message">
            <div *ngIf="email?.errors?.['required']">Email is required</div>
            <div *ngIf="email?.errors?.['pattern'] || email?.errors?.['email']">Email is invalid</div>
          </div>
        </div>
        <div class="form__group">
          <label for="phone" class="form__label">Telephone</label>
          <input type="text" id="phone" class="form__input" formControlName="phone" placeholder="+"
name="phone" [ngClass]="{ 'error': phone.invalid && (phone.touched || phone.dirty)}">
          <div *ngIf="phone?.invalid && (phone?.touched || phone?.dirty)" class="error-message">
            <div *ngIf="phone?.errors?.['required']">Phone is required</div>
            <div *ngIf="phone?.errors?.['minlength']">Phone must be at least 6 characters long</div>
            <div *ngIf="phone?.errors?.['pattern']
              && !phone?.errors?.['minlength']
              && !phone?.errors?.['maxlength']">Phone must contain only numbers and +</div>
            <div *ngIf="phone?.errors?.['maxlength']">Phone must be no more than 20 characters</div>
          </div>
        </div>
        <div class="form__group">
          <label for="country" class="form__label">Country</label>
          <input type="text" id="country" class="form__input" formControlName="country" placeholder="Ukraine"
                 name="country"
                 [ngClass]="{ 'error': country.invalid && (country.touched || country.dirty)}">

          <div *ngIf="country?.invalid && (country?.touched || country?.dirty)" class="error-message">
            <div *ngIf="country?.errors?.['required']">Country is required</div>
            <div *ngIf="country?.errors?.['pattern'] && !country?.errors?.['maxlength']">Country must contain
only letters</div>
            <div *ngIf="country?.errors?.['maxlength']">Country must be no more than 255 characters</div>
          </div>
        </div>
        <div class="form__group">
          <label for="city" class="form__label">City</label>
          <input type="text" id="city" class="form__input" formControlName="city" placeholder="Kyiv"
                 name="city"
                 [ngClass]="{ 'error': city.invalid && (city.touched || city.dirty)}">

          <div *ngIf="city?.invalid && (city?.touched || city?.dirty)" class="error-message">
```

```
            <div *ngIf="city?.errors?.['required']">City name is required</div>
            <div *ngIf="city?.errors?.['pattern'] && !city?.errors?.['maxlength']">City must contain only
letters</div>
            <div *ngIf="city?.errors?.['maxlength']">City must be no more than 255 characters</div>
          </div>
        </div>

        <div class="form__group">
          <label for="isAdmin" class="form__label">Are you admin?</label>
          <input type="checkbox" id="isAdmin" class="form__checkbox" formControlName="isAdmin"
name="isAdmin">
        </div>
        <div class="form__group">
          <label for="password" class="form__label">Password</label>
          <div class="input-wrapper input-wrapper-password">
            <input id="password" class="form__input" formControlName="password" name="password"
                   [type]="showPassword ? 'text' : 'password'"
                   [ngClass]="{ 'error': password.invalid && (password.touched || password.dirty)}"
            >
            <div class="toggle-password" (click)="toggleShowPassword()">
              <img [src]="showPassword ? '../../../../assets/password-icons/show-icon.png' :
'../../../../assets/password-icons/hide-icon.png'"
                   alt="show/hide password"/>
            </div>
          </div>
          <div *ngIf="password?.invalid && (password?.touched || password?.dirty)" class="error-message">
            <div *ngIf="password?.errors?.['required']">Password is required</div>
            <div *ngIf="password?.errors?.['minlength']">Password must be at least 8 characters long</div>
            <div *ngIf="password?.errors?.['pattern']
              && !password?.errors?.['minlength']
              && !password?.errors?.['maxlength']">Password must contain a letter, a number and a special
character &,#,!,%</div>
            <div *ngIf="password?.errors?.['maxlength']">Password must be no more than 255 characters</div>
          </div>
        </div>

        <div class="form__group">
          <label for="confirmPassword" class="form__label">Password Confirmation</label>
          <div class="input-wrapper input-wrapper-password">
            <input id="confirmPassword" class="form__input" formControlName="confirmPassword"
                   name="confirmPassword"
                   [type]="showConfirmPassword ? 'text' : 'password'"
                   [ngClass]="{ 'error': confirmPassword.invalid && (confirmPassword.touched ||
confirmPassword.dirty)}" >
            <div class="toggle-password" (click)="toggleShowConfirmPassword()">
              <img
                [src]="showConfirmPassword ? 'assets/password-icons/show-icon.png' : 'assets/password-
icons/hide-icon.png'"
                alt="show/hide password"/>
```

```
            </div>
          </div>
          <div *ngIf="confirmPassword?.invalid && (confirmPassword?.touched || confirmPassword?.dirty)"
               class="error-message">
            <div *ngIf="confirmPassword?.errors?.['required']">Password confirmation is required</div>
            <div *ngIf="confirmPassword?.errors?.['passwordMismatch']">Passwords do not match</div>
          </div>
        </div>
        <div class="form__group">
          <div class="form__error-message">
            <div *ngIf="errorMessage" class="error-message">{{ errorMessage }}</div>
          </div>
        </div>
        <div class="form__group">
          <button type="submit" class="form__button" [disabled]="form.invalid || form.pristine">{{
getButtonText() }}</button>
        </div>
      </div>
      <button class="form__close" (click)="onClose()">
        <div class="btn__inner">
          <div class="line"></div>
          <div class="line"></div>
        </div>
      </button>
    </form>
  </div>
</div>

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { NotesComponent } from './notes.component';
import { NoteComponent } from './note/note.component';
import { NoteCreateComponent } from './note-create/note-create.component';
import { NoteUpdateComponent } from './note-update/note-update.component';

const routes: Routes = [
  { path: '', pathMatch: 'full', component: NotesComponent },
  { path: ':id', component: NoteComponent },
  { path: 'create', component: NoteCreateComponent },
  { path: 'update/:id', component: NoteUpdateComponent }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class NotesRoutingModule { }

import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { NotesService } from '@shared/services/notes.service';
import { Note } from '@shared/models/note.model';
import { ActivatedRoute } from '@angular/router';
@Component({
  selector: 'app-notes',
  templateUrl: './notes.component.html',
```

```
  styleUrl: './notes.component.scss'
})
export class NotesComponent {
  constructor(private notesService: NotesService, private route: ActivatedRoute) {}
  prId!: string;
  notes$!: Observable<Note[]>;
  ngOnInit() {
    this.route.queryParams
      .subscribe(params => {
        this.prId = params['projectId'];
        this.notes$ = this.notesService.getProjectNotes(this.prId);
      }
    );
  }
}
```

```
<div class="note" *ngIf="note">
  <div class="note__up">
    <h2 class="note__title">{{ note.title }}</h2>
    <div class="note__header">
      <div class="note__date">{{ note.dateCreated | date : "MMM dd, yyyy 'at' HH:mm" }}</div>
      <div class="note__author" *ngIf="author">
        <div class="note__author-name">{{ author.firstName }} {{ author.lastName }}</div>
        <div class="note__author-role">{{ author.role }}</div>
      </div>
    </div>
  </div>
  <div class="note__content" [innerHTML]="note.content"></div>
  <img style="width: 200px;" [src]="note.image" alt="image" *ngIf="note.image">
  <button class="note__edit" [routerLink]="['/notes/update', note.id!]" *ngIf="isAuthor">Edit Note</button>
  <button class="note__delete" (click)="deleteNote(note.id!)" *ngIf="isAuthor">Delete Note</button>
  <button class="note__add" (click)="addComment()">Add Comment</button>
  <div class="create" *ngIf="commentCreation">
    <h2 class="create__title">Adding of Comment</h2>
    <form [formGroup]="commentForm" (ngSubmit)="onCommentCreate()">
      <div class="form-group">
        <label for="content">Comment text:</label>
        <editor formControlName="content" id="content"
apiKey="x2kj148c3unh8b6qvj3uwk2poqwxozayxnc7mw5nif072vlq" [init]="{plugins: 'link'}"></editor>
      </div>
      <div class="form-group">
        <label for="image">Add Image:</label>
        <input type="file" formControlName="image" id="image" class="form-control"/>
      </div>
      <button type="submit" class="btn-primary">Create Comment</button>
      <button class="note__edit" (click)="onClose()">Cancel</button>
    </form>
  </div>
  <div class="note__comments" *ngIf="(comments$ | async)?.length">
    <h3>Comments:</h3>
    <app-comment *ngFor="let comment of (comments$ | async)" [comment]="comment"
(deleteCommentEvent)="onDeleteComment($event)"></app-comment>
  </div>
```

```
</div>

import { Component, OnInit } from '@angular/core';
import { map, Observable, switchMap, tap } from 'rxjs';
import { ActivatedRoute, ParamMap, Router } from '@angular/router';
import { FormBuilder, FormGroup } from '@angular/forms';
import { NotesService } from '@shared/services/notes.service';
import { Note } from '@shared/models/note.model';
import { Comment } from '@shared/models/comment.model';
import { AuthService } from '@shared/services/auth.service';
import { User } from '@shared/models/user.model';
@Component({
  selector: 'app-note',
  templateUrl: './note.component.html',
  styleUrl: './note.component.scss',
})
export class NoteComponent implements OnInit {
  constructor(
    private notesService: NotesService,
    private route: ActivatedRoute,
    private router: Router,
    private formBuilder: FormBuilder,
    private notesServise: NotesService,
    private auth: AuthService
  ) {}

  noteId: string = '';
  note$!: Observable<Note>;
  note!: Note;
  comments$!: Observable<Comment[]>;
  author!: User;
  isAuthor: boolean = false;
  currentUserId!: string;
  commentCreation: boolean = false;
  commentForm!: FormGroup;
  errorMessage: string = '';

  ngOnInit() {
    this.auth.getCurrentUser().pipe(
      tap((user) => {
        if (user?.id) {
          this.currentUserId = user.id;
        }
      }),
      switchMap(() => this.route.paramMap),
      map((params: ParamMap) => params.get('id')!),
      switchMap((id) => {
        this.noteId = id;
        return this.notesService.getNoteById(this.noteId);
      }),
      tap((note) => {
        this.note = note;
        if (this.currentUserId === this.note.authorId) {
          this.isAuthor = true;
        }
      }),
      switchMap((note) => this.auth.getUserInfo(note?.authorId!))
    ).subscribe((author) => {
      this.author = author;
    });
  }

  deleteNote(noteId: string) {
    this.notesService.deleteNote(noteId).subscribe();
    this.router.navigateByUrl(`/projects`);
```

```
  }

  onClose() {
    this.commentForm.reset();
    this.commentCreation = false;
  }

  addComment() {
    this.commentCreation = true;

    this.commentForm = this.formBuilder.group({
      content: [''],
      image: [''],
    });
  }

  onCommentCreate() {
    if (this.commentForm.valid) {
      const authorId = this.currentUserId;
      const noteId = this.noteId;
      const commentValue = { ...this.commentForm.value, authorId, noteId };

      this.notesServise.createComment(commentValue).subscribe({
        next: () => {
          this.onClose();
          this.comments$ = this.notesService.getComments(this.noteId);
          alert('Comment was successfully created.');
        },
        error: (error) => {
          this.errorMessage = error.message;
          setTimeout(() => {
            this.onClose();
          }, 2000);
          alert('Comment was not created.');
        },
      });
    }
  }

  onDeleteComment(commentId: string) {
    this.notesServise.deleteComment(commentId).subscribe();
    alert('Comment was successfully deleted.');
    this.notesService.getComments(this.noteId)
    this.router.navigateByUrl(`/projects`);
  }
}

import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { Router } from '@angular/router';
import { Category } from '@shared/models/category.model';
import { Project } from '@shared/models/project.model';
import { User } from '@shared/models/user.model';
import { AuthService } from '@shared/services/auth.service';
import { NotesService } from '@shared/services/notes.service';
import { ProjectsService } from '@shared/services/projects.service';

@Component({
  selector: 'app-note-create',
  templateUrl: './note-create.component.html',
  styleUrls: ['./note-create.component.scss'],
})
export class NoteCreateComponent implements OnInit {
  noteForm!: FormGroup;
  errorMessage: string = '';
```

```
currentUser!: User;
currentUserId!: string;

categories!: Category[];
projects!: Project[];
imageDisplay!: string | ArrayBuffer | null;

constructor(
  private formBuilder: FormBuilder,
  private notesServise: NotesService,
  private projectsServise: ProjectsService,
  private router: Router,
  private auth: AuthService
) {}

ngOnInit(): void {
  this.noteForm = this.formBuilder.group({
    title: [''],
    content: [''],
    image: [''],
    category: [''],
    project: ['']
  });

  this.auth.getCurrentUser().subscribe((user) => {
    if (user) {
      this.currentUser = user;
      if (user.id) this.currentUserId = user.id;
    }
  });

  this.notesServise.getCategories().subscribe((categories) => {
    if (categories) {
      this.categories = categories;
    }
  });

  this.projectsServise.getUserProjects(this.currentUserId).subscribe((projects) => {
    if (projects) {
      this.projects = projects;
    }
  });
}

onSubmit(): void {
  if (this.noteForm.valid) {
    const noteFormData = new FormData();

    Object.keys(this.noteForm.controls).forEach(key => {
      noteFormData.append(key, this.noteForm.get(key)?.value);
    });

    noteFormData.append('authorId', this.currentUserId);
    this.notesServise.createNote(noteFormData).subscribe({
      next: () => {
        let proj = this.noteForm.get('project')?.value;
        this.onClose();
        this.router.navigateByUrl(`/notes?projectId=${proj}`);
        alert('Note was successfully created.');
      },
      error: (error) => {
        this.errorMessage = error.message;
        setTimeout(() => {
          this.onClose();
        }, 2000);
```

```
          alert('Note was not created.');
          this.router.navigateByUrl('/notes/create');
        },
      });

    }
  }

  onImageUpload(event: any) {
    const file = event.target.files[0];
    if (file) {
      this.noteForm.patchValue({image: file});
      this.noteForm.get('image')?.updateValueAndValidity();
      const fileReader = new FileReader();
      fileReader.onload = () => {
        this.imageDisplay = fileReader.result;
      }
      fileReader.readAsDataURL(file);
    }
  }

  onClose() {
    this.noteForm.reset();
  }
}

import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { ParamMap, Router, ActivatedRoute } from '@angular/router';
import { forkJoin, map, Subscription, switchMap } from 'rxjs';
import { Category } from '@shared/models/category.model';
import { Note } from '@shared/models/note.model';
import { NotesService } from '@shared/services/notes.service';

@Component({
  selector: 'app-note-update',
  templateUrl: './note-update.component.html',
  styleUrls: ['./note-update.component.scss']
})
export class NoteUpdateComponent implements OnInit, OnDestroy {
  private subscription: Subscription = new Subscription();
  note!: Note;
  noteId!: string;
  noteForm!: FormGroup;
  errorMessage: string = '';
  categories!: Category[];
  imageDisplay!: string | ArrayBuffer | null;

  constructor(
    private formBuilder: FormBuilder,
    private notesService: NotesService,
    private router: Router,
    private route: ActivatedRoute,
  ) {}

  ngOnInit(): void {
    this.subscription = this.route.paramMap.pipe(
      map((params: ParamMap) => params.get('id')!),
      switchMap(id => {
        this.noteId = id;
        return forkJoin([
          this.notesService.getNoteById(this.noteId),
          this.notesService.getCategories(),
        ]);
      })
```

```
  ).subscribe(([note, categories]) => {
    this.note = note;
    this.categories = categories;
    this.initializeForm(note);
  });
  this.noteForm = this.formBuilder.group({
    title: [''],
    content: [''],
    image: [''],
    category: ['']
  });
}
ngOnDestroy() {
  if (this.subscription) {
    this.subscription.unsubscribe();
  }
}

initializeForm(note: Note): void {
  this.noteForm.patchValue({
    title: note.title,
    content: note.content,
    image: note.image,
    category: note.category,
  });

  if (note.image) {
    this.imageDisplay = note.image;
  }
}

onSubmit(): void {
  if (this.noteForm.valid) {
    const updatedFields = new FormData();

    Object.keys(this.noteForm.controls).forEach(key => {
      if (this.noteForm.get(key)?.dirty) {
        updatedFields.append(key, this.noteForm.get(key)?.value);
      }
    });

    this.notesService.updateNote(this.noteId, updatedFields).subscribe({
      next: () => {
        this.onClose();
        this.router.navigateByUrl(`/notes?projectId=${this.note.project}`);
        alert('Note was successfully updated.');
      },
      error: (error) => {
        this.errorMessage = error.message;
        setTimeout(() => {
          this.onClose();
        }, 2000);
        alert('Note was not updated.');
        this.router.navigateByUrl('/projects');
      },
    });
  }
}

onImageUpload(event: any) {
  const file = event.target.files[0];
  if (file) {
    this.noteForm.patchValue({image: file});
    this.noteForm.get('image')?.markAsDirty();
    const fileReader = new FileReader();
```

```
      fileReader.onload = () => {
        this.imageDisplay = fileReader.result;
      }
      fileReader.readAsDataURL(file);
    }
  }

  onClose() {
    this.noteForm.reset();
  }
}
```

```
<div class="comment" *ngIf="comment">
  <div class="comment__author" *ngIf="author">
    <div class="coment__author_first">
      <div class="comment__author-name">{{ author.firstName }} {{ author.lastName }}</div>
      <div class="comment__author-role">{{ author.role }}</div>
    </div>
    <div class="comment__dateCreated">{{ comment.dateCreated | date : "MMM dd, yyyy 'at' HH:mm" }}</div>
  </div>
  <div class="comment__contents">
    <div class="comment__content" [innerHTML]="comment.content"></div>
    <button class="comment__delete" *ngIf="isAuthor" (click)="onDeleteComment(comment.id!)">Delete</button>
  </div>
  <div class="comment__image"><img src="{{ comment.image }}" /></div>
</div>
```

```
import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
import { Comment } from '@shared/models/comment.model';
import { User } from '@shared/models/user.model';
import { AuthService } from '@shared/services/auth.service';
import { NotesService } from '@shared/services/notes.service';

@Component({
  selector: 'app-comment',
  templateUrl: './comment.component.html',
  styleUrl: './comment.component.scss'
})
export class CommentComponent implements OnInit {
  constructor(private authService: AuthService, private notesServise: NotesService ) {}
  @Input() comment!: Comment;
  @Output() deleteCommentEvent = new EventEmitter<string>();

  author!: User;
  isAuthor: boolean = false;
  currentUserId!: string;

  ngOnInit() {
    this.authService.getCurrentUser().subscribe((user) => {
      if (user?.id) this.currentUserId = user.id;
    });
    if (this.comment && this.comment.authorId && this.currentUserId === this.comment.authorId) {
      this.isAuthor = true;
    }
    if (this.comment && this.comment.authorId) {
      this.authService.getUserInfo(this.comment.authorId).subscribe(author => {
        this.author = author;
      });
    }
  }
  onDeleteComment(idComment: string) {
```

```
    this.deleteCommentEvent.emit(idComment);
  }
}

import { Component, OnInit, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';
import { switchMap } from 'rxjs/operators';
import { Project } from '@shared/models/project.model';
import { AuthService } from '@shared/services/auth.service';
import { ProjectsService } from '@shared/services/projects.service';

@Component({
  selector: 'app-projects',
  templateUrl: './projects.component.html',
  styleUrls: ['./projects.component.scss'],
})
export class ProjectsComponent implements OnInit, OnDestroy {
  projects: Project[] = [];
  userId!: string;
  private subscription: Subscription = new Subscription();
  constructor(private auth: AuthService, private projectsService: ProjectsService) {}
  ngOnInit() {
    this.subscription = this.auth.getCurrentUser().pipe(
      switchMap(user => {
        this.userId = user?.id!;
        return this.projectsService.getUserProjects(this.userId);
      })
    ).subscribe(projects => {
      this.projects = projects;
    });
  }
  ngOnDestroy() {
    if (this.subscription) {
      this.subscription.unsubscribe();
    }
  }
}

import { Component, OnInit, OnDestroy, ChangeDetectionStrategy } from '@angular/core';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { Subscription } from 'rxjs';
import { map, switchMap } from 'rxjs/operators';
import { Project } from '@shared/models/project.model';
import { ProjectsService } from '@shared/services/projects.service';

@Component({
  selector: 'app-project',
  templateUrl: './project.component.html',
  styleUrls: ['./project.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProjectComponent implements OnInit, OnDestroy {
  projectId!: string;
  project!: Project;
  private subscription: Subscription = new Subscription();

  constructor(private route: ActivatedRoute, private projectsService: ProjectsService) {}

  ngOnInit() {
    this.subscription = this.route.paramMap.pipe(
      map((params: ParamMap) => params.get('id')!),
      switchMap(id => {
        this.projectId = id;
        return this.projectsService.getProjectById(this.projectId);
      })
```

```
    ).subscribe(project => {
      this.project = project;
    });
  }

  ngOnDestroy() {
    if (this.subscription) {
      this.subscription.unsubscribe();
    }
  }
}

import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { MasterService } from '@shared/services/master.service';
import { AuthService } from '@shared/services/auth.service';
import { ProjectsService } from '@shared/services/projects.service';
import { Project } from '@shared/models/project.model';
import { User } from '@shared/models/user.model';
import { Task } from '@shared/models/task.model';

@Component({
  selector: 'app-jira',
  templateUrl: './jira.component.html',
  styleUrls: ['./jira.component.scss']
})
export class JiraComponent implements OnInit {
  currentUserId: string = '';
  currentProject!: Project;
  projectList: Project[] = [];
  taskList: Task[] = [];
  userList: User[] = [];
  issueTypes: string[]= ['Ticket','Defect','RnD Work'];
  status: string[]= ['To Do','In Progress','Done'];
  storyPoint: number[]= [1, 2, 3, 5, 8, 13, 21, 34, 55, 89];

  taskObj: any = {
    'title': '',
    'content': '',
    'status': 'To Do',
    'type': 'Ticket',
    'taskStatus': '6638ab9fd702ab3e83af0da8',
    'storyPoint': 1,
    'image': "",
    'assigneeId': '',
    'reporterId': '',
    'projectId': '',
    'dateStart': new Date().toISOString(),
    'dateEnd': '2025-09-12T05:58:41.065Z'
  }

  constructor(private http: HttpClient, private master: MasterService, private auth: AuthService, private
projectsServise: ProjectsService) {
    this.auth.getCurrentUser().subscribe((user) => {
      if (user?.id) {
        this.currentUserId = user.id;
        this.taskObj.createdBy = user.id;
      }
    });
  }

  ngOnInit(): void {
    this.getAllProjects();
  }
```

```
  setProject(obj: any) {
    this.master.onProjectChange.next(obj);
    this.currentProject = obj;
    this.getAllUsers();
    this.getAllProjectTasks();
  }

  getAllProjects() {
    this.projectsServise.getUserProjects(this.currentUserId).subscribe((projects) => {
      if (projects) {
        this.projectList = projects;
        this.currentProject = this.projectList[0];
        this.master.onProjectChange.next(this.projectList[0]);
        this.getAllUsers();
        this.getAllProjectTasks();
      }
    });
  }

  getAllUsers() {
    this.projectsServise.getProjectTeam(this.currentProject.id!).subscribe((users: any)=>{
      this.userList = users;
    })
  }

  getAllProjectTasks() {
    this.projectsServise.getProjectTasks(this.currentProject.id!).subscribe((tasks: any)=>{
      this.taskList = tasks;
    })
  }

  filterTasks(status: string) {
    return this.taskList.filter(m => m.status == status)
  }

  onTaskCreate() {
    this.projectsServise.createTask(this.taskObj).subscribe((res: Task)=>{
      if(res) {
       alert('Successfully created task.');
       this.getAllProjectTasks();
       this.master.onTicketCreate.next(true);
      } else {
       alert('An error occurred while creating the task.');
      }
   })
  }
}

import { Component, Input, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { Task } from '@shared/models/task.model';
import { User } from '@shared/models/user.model';
import { AuthService } from '@shared/services/auth.service';

@Component({
  selector: 'app-task-preview',
  templateUrl: './task-preview.component.html',
  styleUrl: './task-preview.component.scss'
})
export class TaskPreviewComponent implements OnInit {
  constructor(private auth: AuthService) {}

  @Input() task!: Task;
  assignee$!: Observable<User>;
```

```
  ngOnInit() {
    this.assignee$ = this.auth.getUserInfo(this.task.assigneeId!);
  }
}

import { Component } from '@angular/core';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { map, Observable } from 'rxjs';
import { Task } from '@shared/models/task.model';
import { User } from '@shared/models/user.model';
import { AuthService } from '@shared/services/auth.service';
import { ProjectsService } from '@shared/services/projects.service';
@Component({
  selector: 'app-task',
  templateUrl: './task.component.html',
  styleUrl: './task.component.scss',
})
export class TaskComponent {
  constructor(private route: ActivatedRoute, private projectService: ProjectsService, private auth:
AuthService) {}
  taskId: string = '';
  task!: Task;
  assignee$!: Observable<User>;
  status: string[]= ['To Do','In Progress','Done'];

  ngOnInit() {
    this.route.paramMap
      .pipe(map((params: ParamMap) => params.get('id')!)).subscribe((id) => {
        this.taskId = id;
        this.projectService.getTaskById(this.taskId).subscribe((task) => {
          this.task = task;
          this.assignee$ = this.auth.getUserInfo(this.task.assigneeId!);
        });
      });
  }
}

import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { catchError, map, Observable, throwError } from 'rxjs';
import { Project } from '@shared/models/project.model';
import { User } from '@shared/models/user.model';
import { Task } from '@shared/models/task.model';

@Injectable({
  providedIn: 'root'
})
export class ProjectsService {

  constructor(private http: HttpClient) {}
  api: string = `http://localhost:3000/api/v1`;

  getUserProjects(userId: string): Observable<Project[]> {
    return this.http.get<Project[]>(`${this.api}/projects/userProjectsByUserId/${userId}`).pipe(
      map((response: Project[]) => {
        return response;
      }),

      catchError(this.handleError),
    );
  }

  getProjectTeam(projectId: string): Observable<User[]> {
    return this.http.get<User[]>(`${this.api}/projects/teamByprojectId/${projectId}`).pipe(
      map((response: User[]) => {
```

```
        return response;
      }),

      catchError(this.handleError),
    );
  }

  getProjectTasks(projectId: string): Observable<Task[]> {
    return this.http.get<Task[]>(`${this.api}/projects/tasksByProjectId/${projectId}`).pipe(
      map((response: Task[]) => {
        return response;
      }),

      catchError(this.handleError),
    );
  }

  getProjectById(projectId: string): Observable<Project> {
    return this.http.get<Project>(`${this.api}/projects/${projectId}`).pipe(
      map((response: Project) => {
        return response;
      }),

      catchError(this.handleError),
    );
  }

  getTaskById(taskId: string): Observable<Task> {
    return this.http.get<Task>(`${this.api}/tasks/${taskId}`).pipe(
      map((response: Task) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  createTask(task: Task): Observable<Task> {
    return this.http.post<Task>(`${this.api}/tasks`, task).pipe(
      map((response: Task) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  private handleError(error: HttpErrorResponse) {
    let errorMessage: string = 'An unknown error occurred!';

    if (error.error instanceof ErrorEvent) {
      errorMessage = `Client Error: ${error.error.message}`;
    } else {
      errorMessage = `Server Error: ${error}`;
      console.error(error);
    }
    console.error(errorMessage);
    return throwError(() => new Error(errorMessage));
  }
}

import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class MasterService {
```

```
  public onProjectChange = new Subject();
  public onTicketCreate = new Subject();
  constructor() { }
}

import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { catchError, map, Observable, throwError } from 'rxjs';
import { Note } from '@shared/models/note.model';
import { Category } from '@shared/models/category.model';
import { Comment } from '@shared/models/comment.model';

@Injectable({
  providedIn: 'root',
})
export class NotesService {
  constructor(private http: HttpClient) {}
  api: string = `http://localhost:3000/api/v1`;

  createNote(note: FormData): Observable<Note> {
    return this.http.post<Note>(`${this.api}/notes/`, note).pipe(
      map((response: Note) => {
        console.log(response);
        return response;
      }),
      catchError(this.handleError),
    );
  }

  updateNote(noteId: string, note: FormData): Observable<Note> {
    return this.http.put<Note>(`${this.api}/notes/${noteId}`, note).pipe(
      map((response: Note) => {
        console.log('updateNote-', response);
        return response;
      }),
      catchError(this.handleError),
    );
  }

  getNotes(): Observable<Note[]> {
    return this.http.get<Note[]>(`${this.api}/notes/`).pipe(
      map((response: Note[]) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  getUserNotes(userId: string): Observable<Note[]> {
    return this.http.get<Note[]>(`${this.api}/notes/notesByUser/${userId}`).pipe(
      map((response: Note[]) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  getProjectNotes(projectId: string): Observable<Note[]> {
    return this.http.get<Note[]>(`${this.api}/notes/notesByProject/${projectId}`).pipe(
      map((response: Note[]) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }
```

```
  getNoteById(id: string): Observable<Note> {
    return this.http.get<Note>(`${this.api}/notes/${id}`).pipe(
      catchError(this.handleError),
    );
  }

  deleteNote(notetId: string) {
    return this.http.delete<{success: boolean, message: string}>(`${this.api}/notes/${notetId}`).pipe(
      map((response) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  getCategories(): Observable<Category[]> {
    return this.http.get<Category[]>(`${this.api}/categories/`).pipe(
      map((response: Category[]) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  getComments(noteId: string) {
    return this.http.get<Comment[]>(`${this.api}/notes/comments/${noteId}`).pipe(
      map((response: Comment[]) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  createComment(comment: Comment): Observable<Comment> {
    return this.http.post<Comment>(`${this.api}/notes/commentCreate`, comment).pipe(
      map((response: Comment) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  deleteComment(commentId: string) {
    return this.http.delete<{success: boolean, message:
string}>(`${this.api}/notes/comments/${commentId}`).pipe(
      map((response) => {
        return response;
      }),
      catchError(this.handleError),
    );
  }

  private handleError(error: HttpErrorResponse) {
    let errorMessage: string = 'An unknown error occurred!';
    if (error.error instanceof ErrorEvent) {
      errorMessage = `Client Error: ${error.error.message}`;
    } else {
      errorMessage = `Server Error: ${error}`;
      console.error(error);
    }
    console.error(errorMessage);
    return throwError(() => new Error(errorMessage));
  }
}
```

Додаток Г

ІЛЮСТРАТИВНА ЧАСТИНА

«УДОСКОНАЛЕННЯ МЕТОДІВ І ЗАСОБІВ УПРАВЛІННЯ ЗНАННЯМИ В ІТ-ПРОЄКТІ»

Рисунок Г.1 – Слайд презентації №1

Рисунок Г.2 – Слайд презентації №2

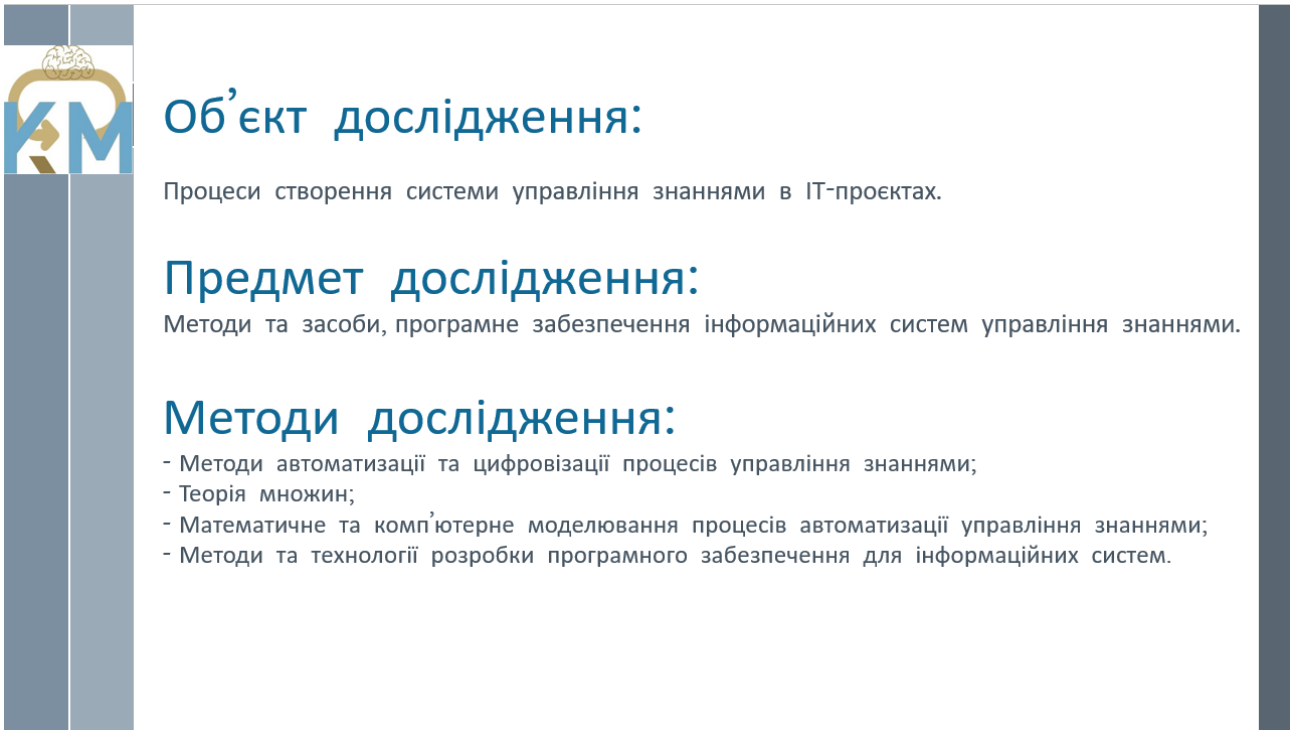


Рисунок Г.3 – Слайд презентації №3

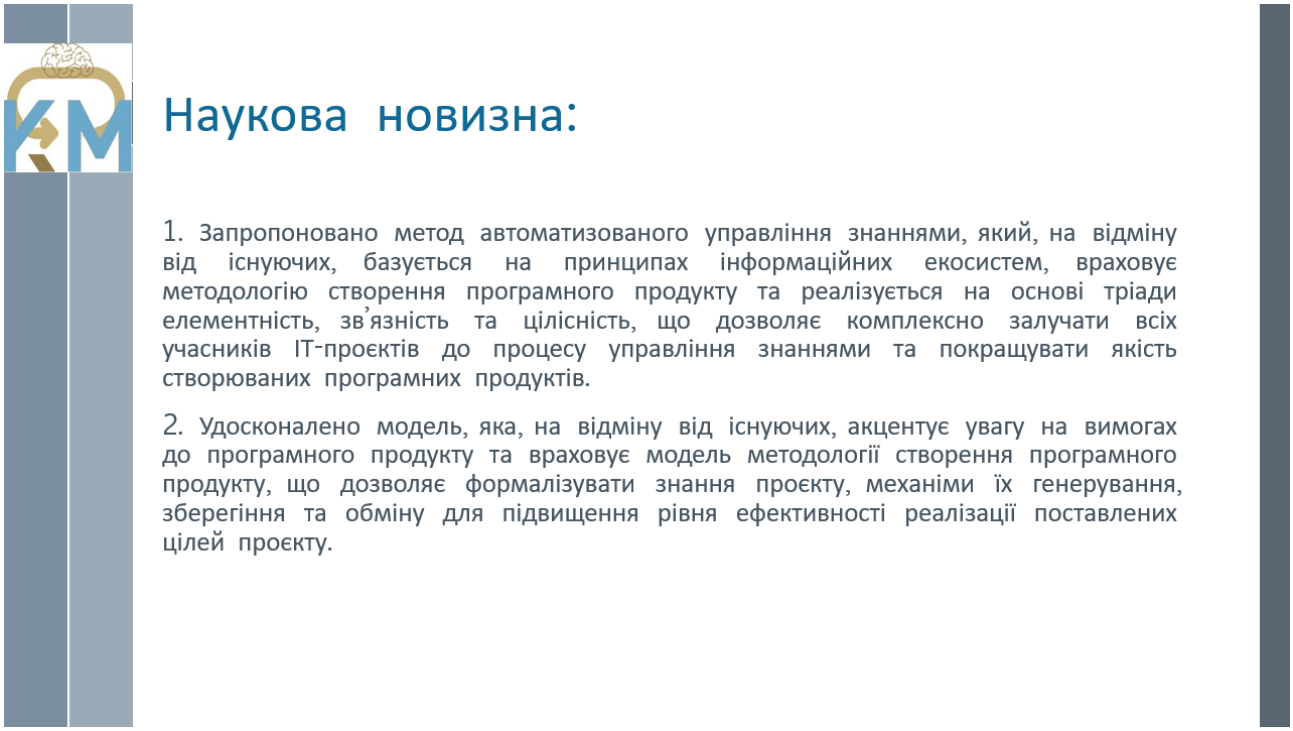


Рисунок Г.4 – Слайд презентації №4

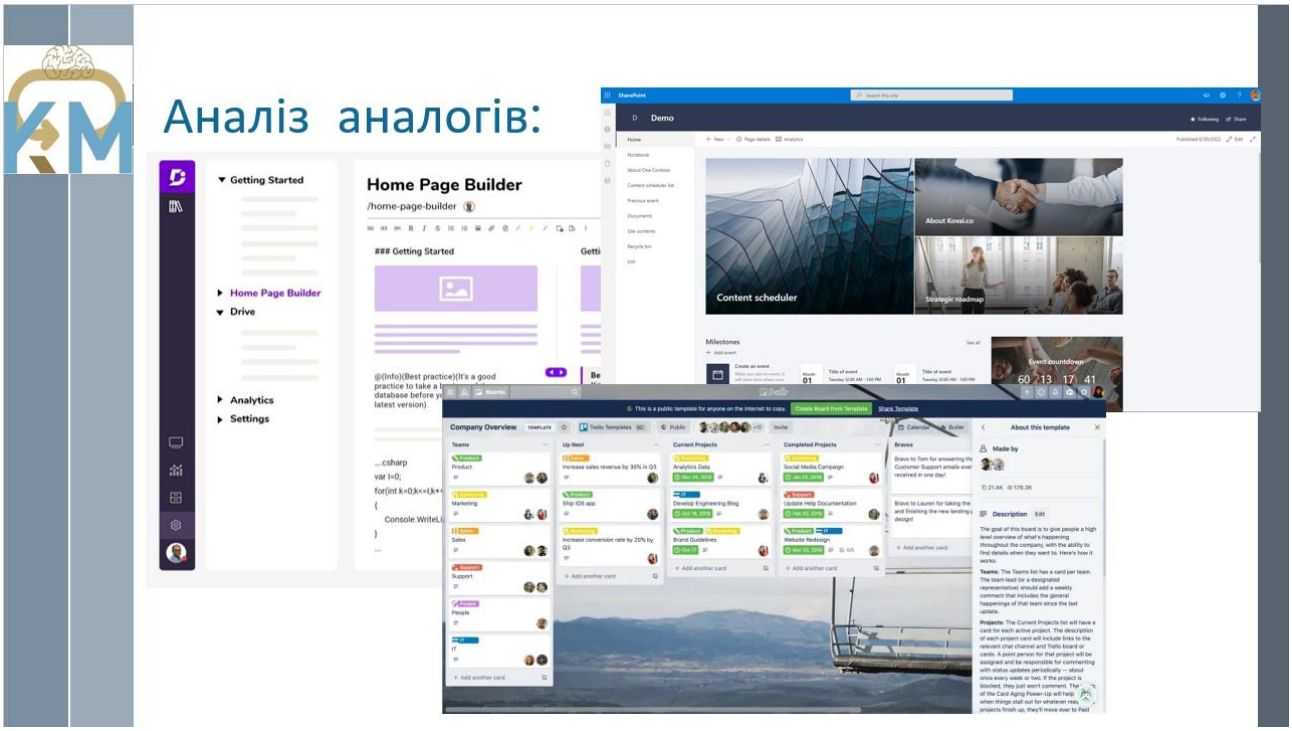


Рисунок Г.5 – Слайд презентації №5

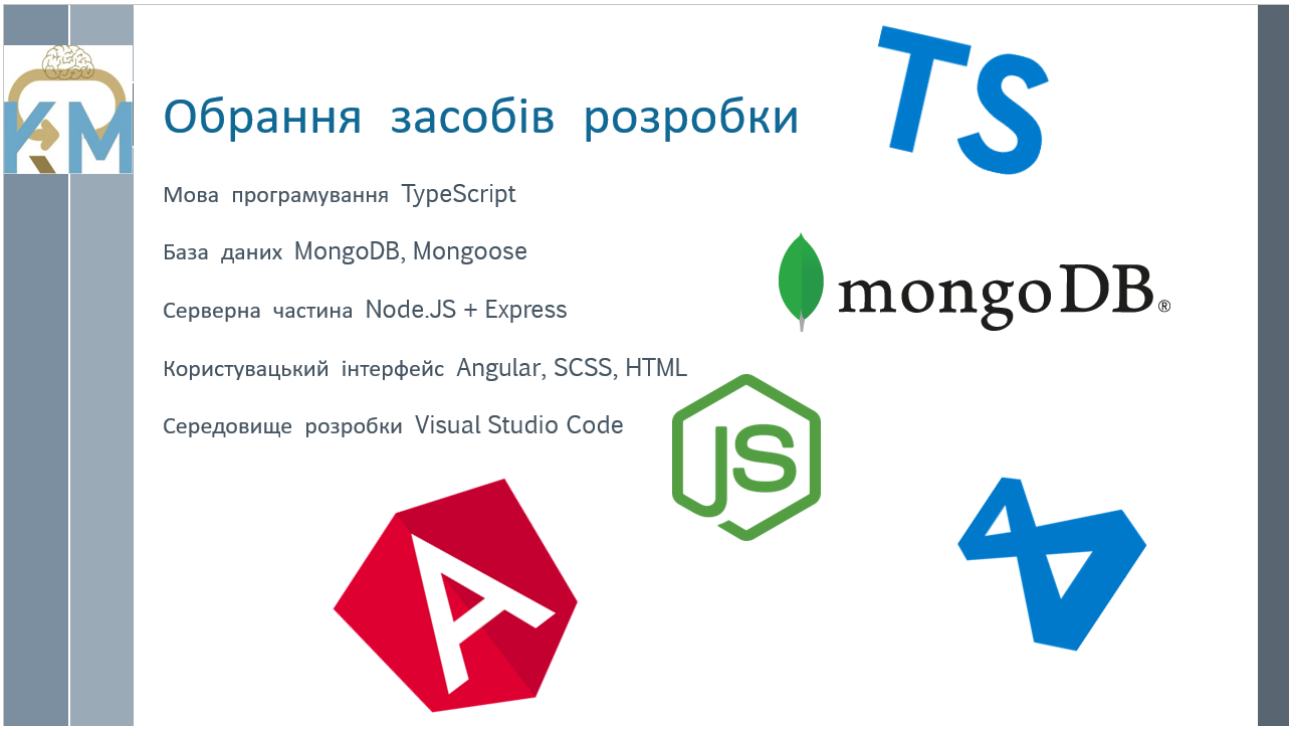


Рисунок Г.6 – Слайд презентації №6

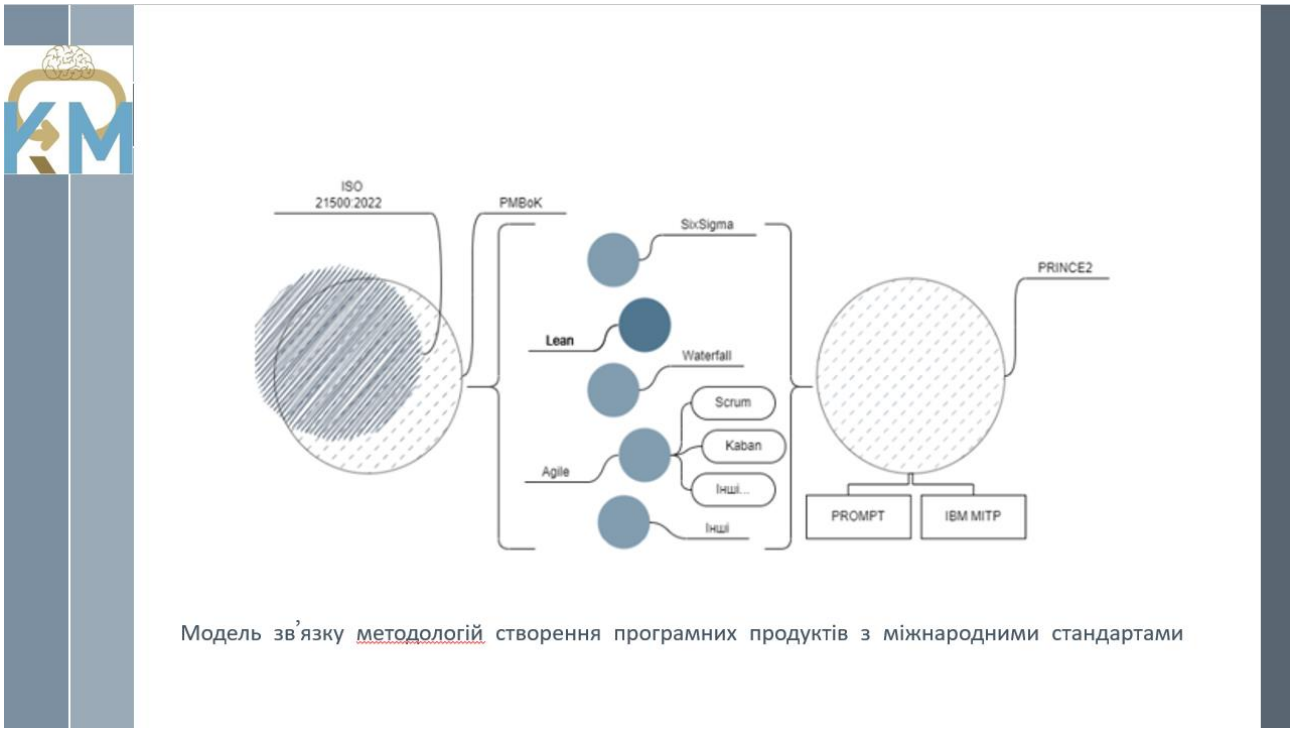


Рисунок Г.7 – Слайд презентації №7

Рисунок Г.8 – Слайд презентації №8

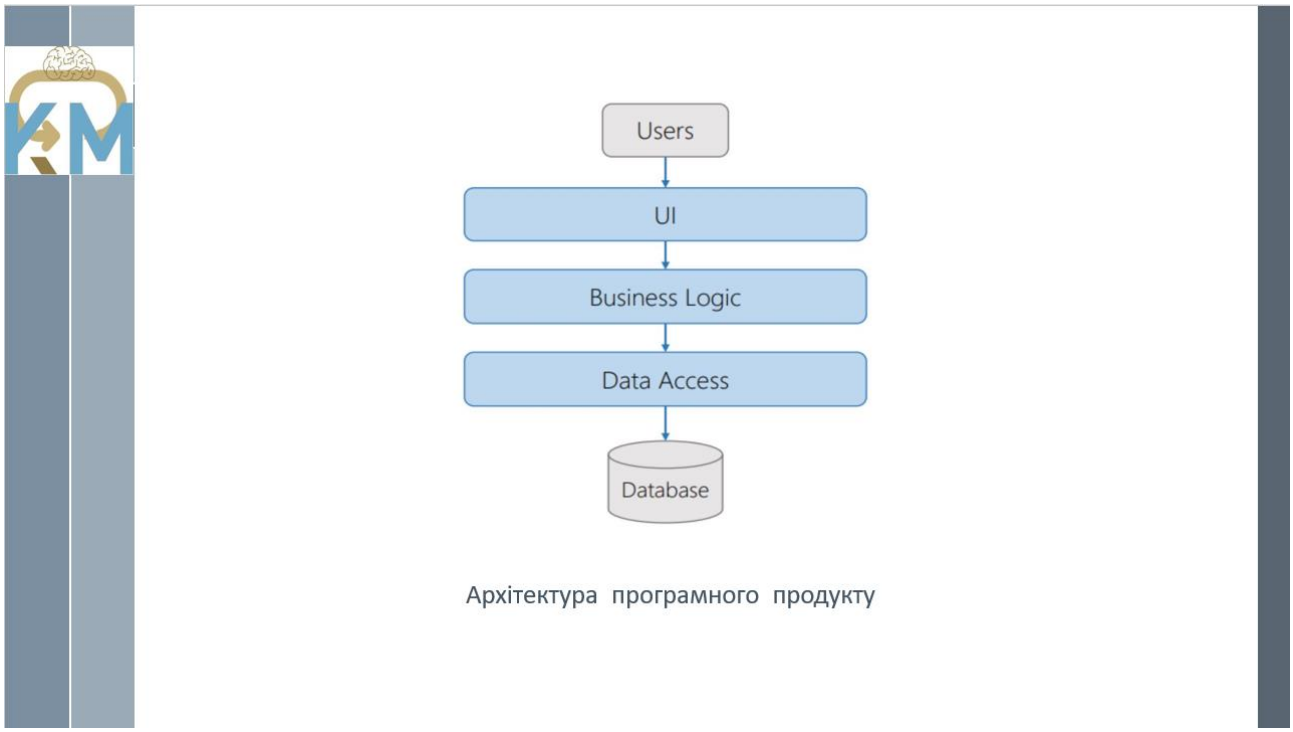


Рисунок Г.9 – Слайд презентації №9

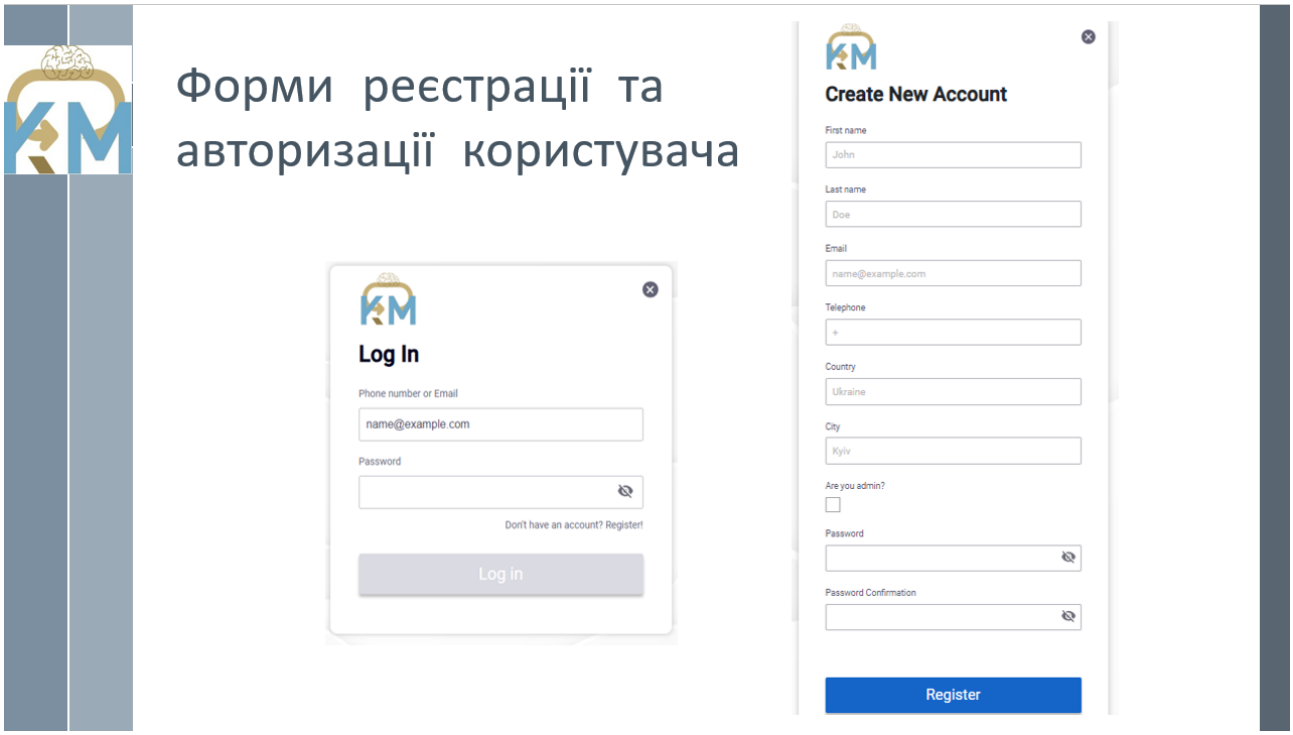


Рисунок Г.10 – Слайд презентації №10

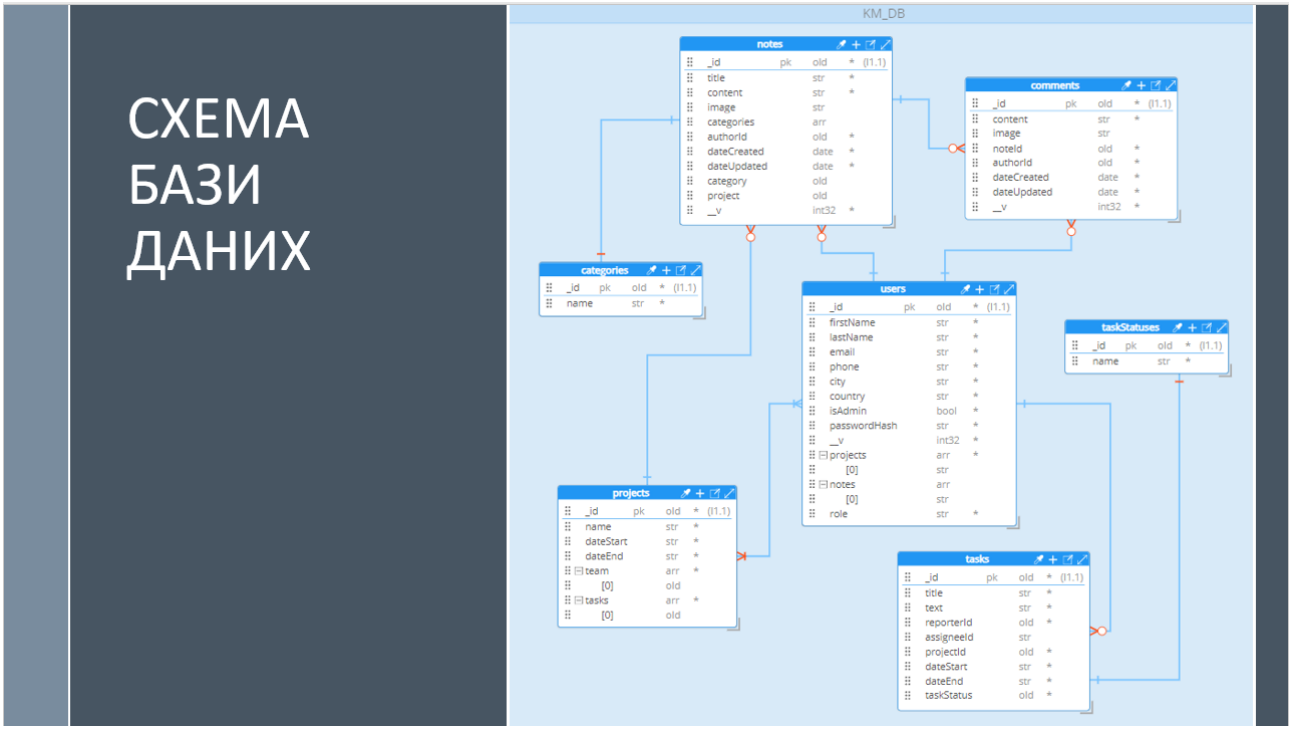


Рисунок Г.11 – Слайд презентації №11

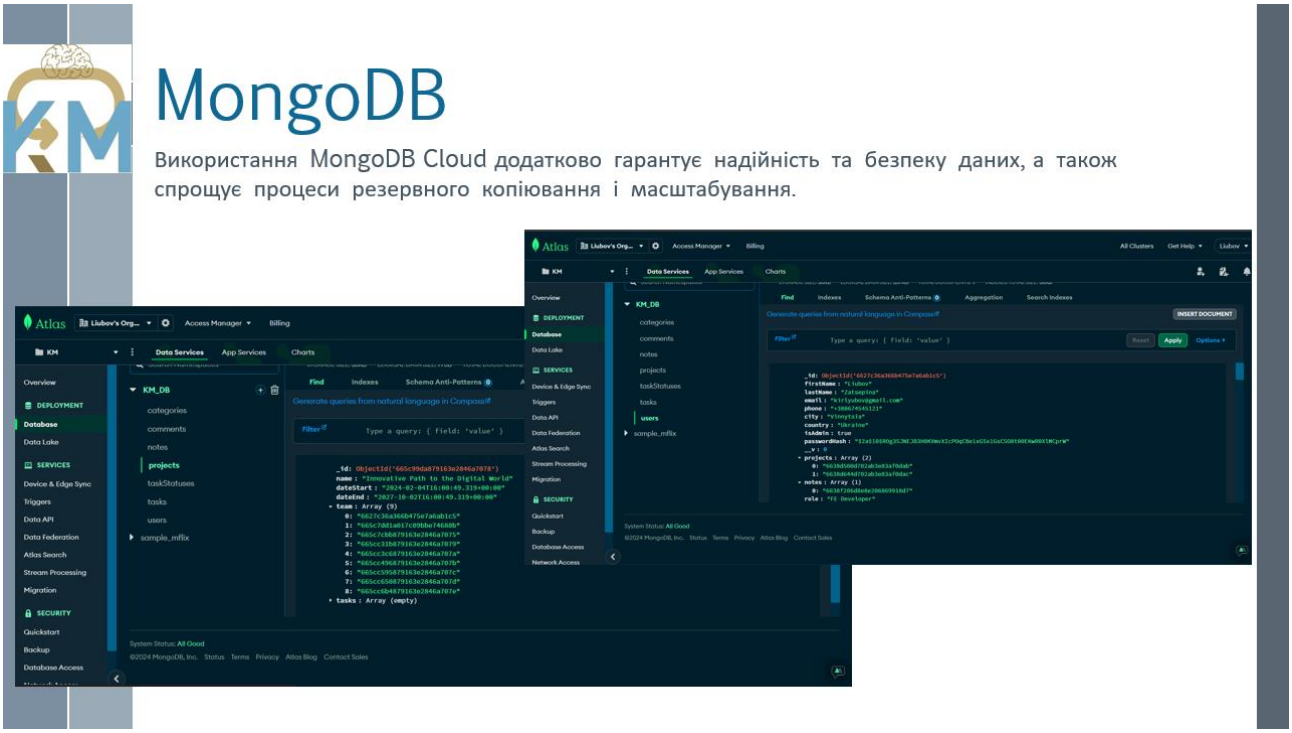


Рисунок Г.12 – Слайд презентації №12

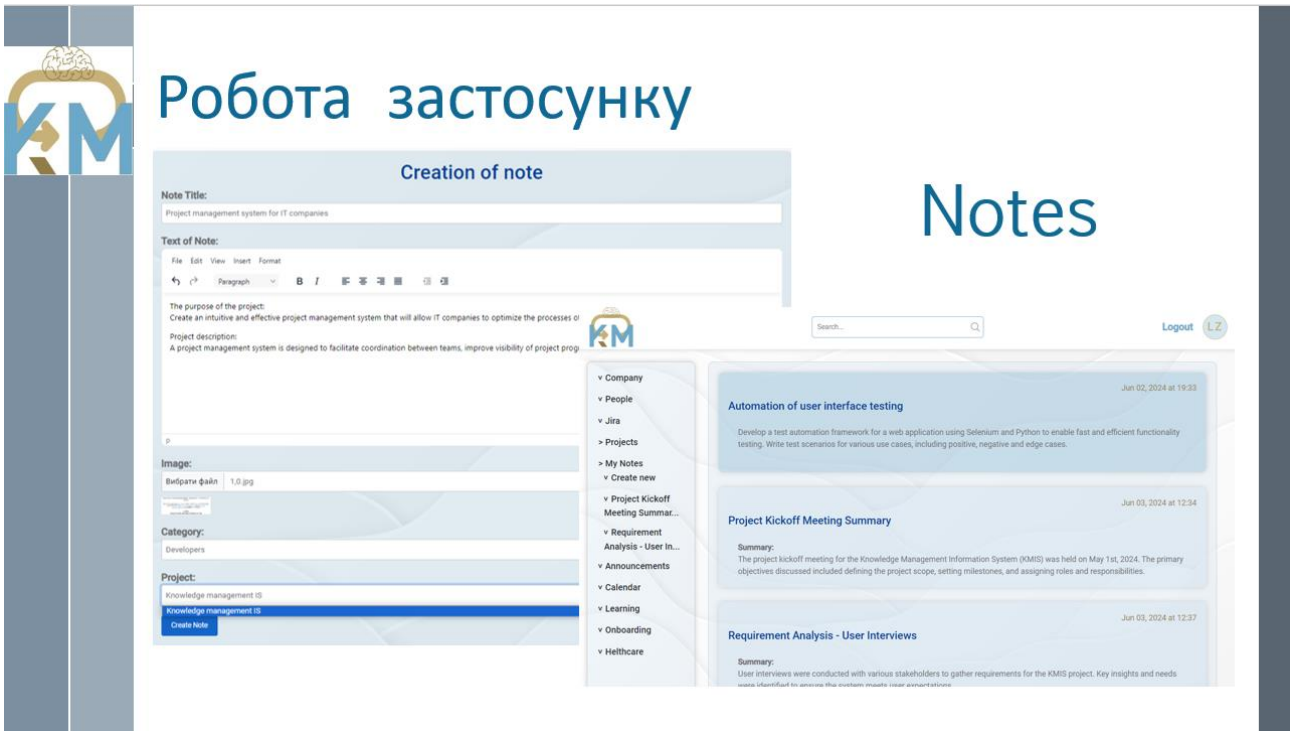


Рисунок Г.13 – Слайд презентації №13

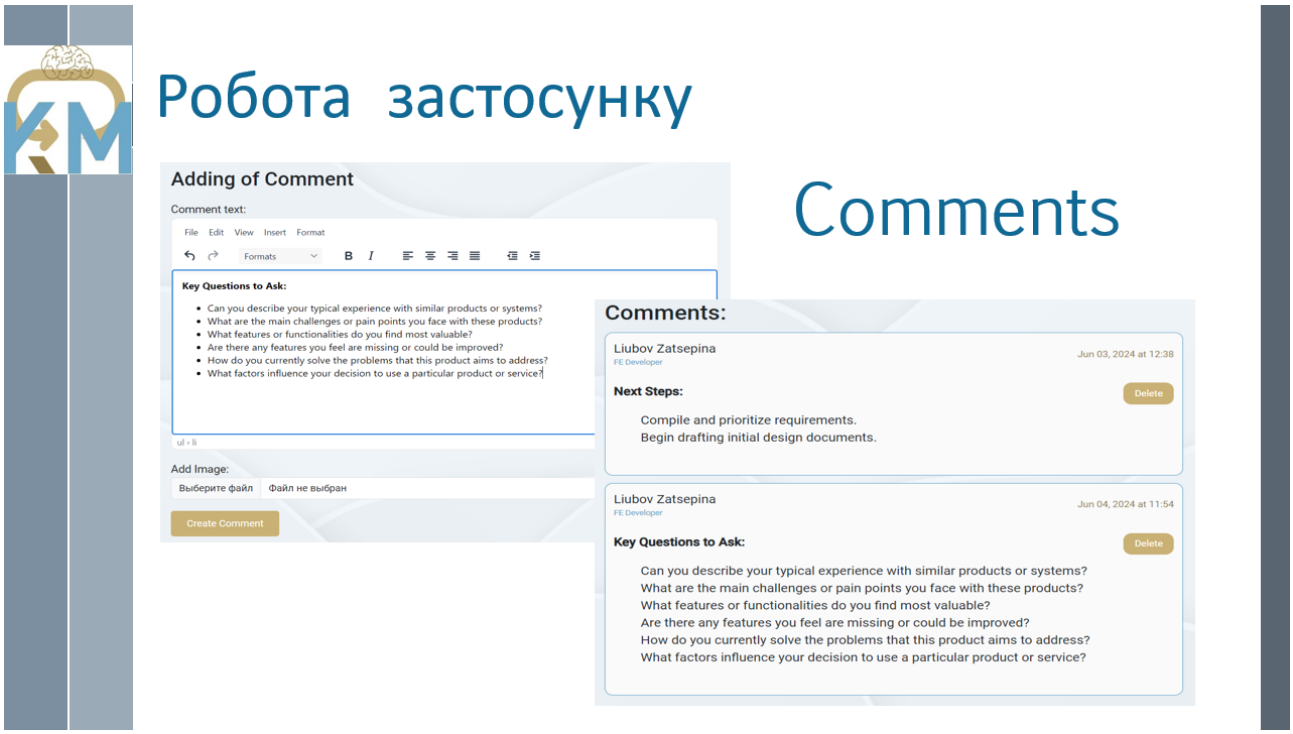


Рисунок Г.14 – Слайд презентації №14

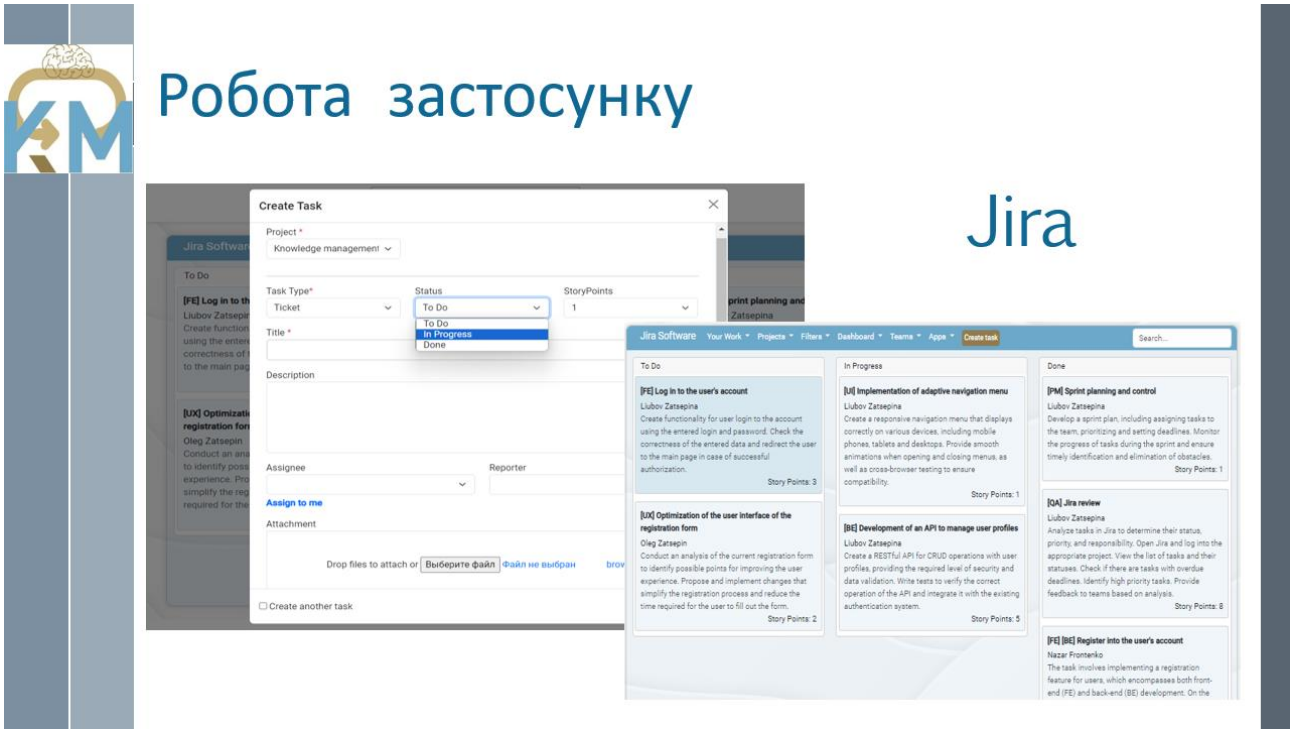


Рисунок Г.15 – Слайд презентації №15

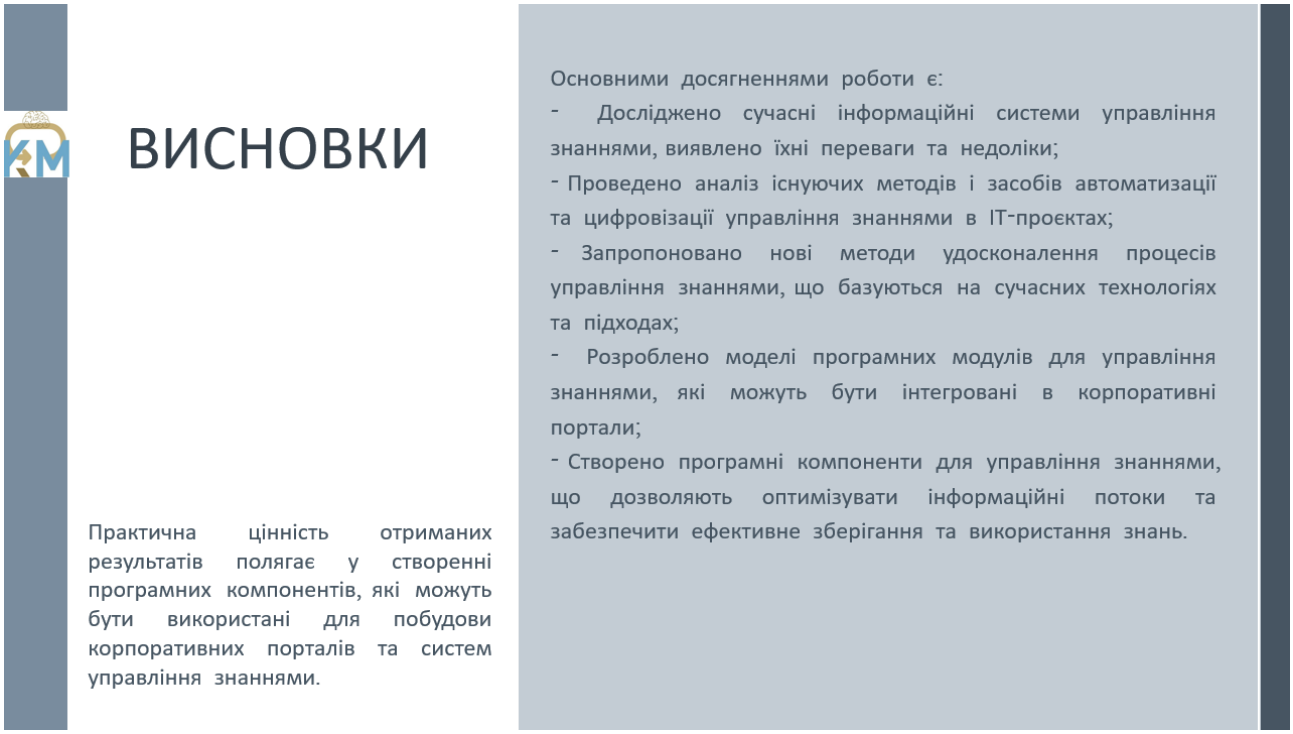


Рисунок Г.16 – Слайд презентації №16

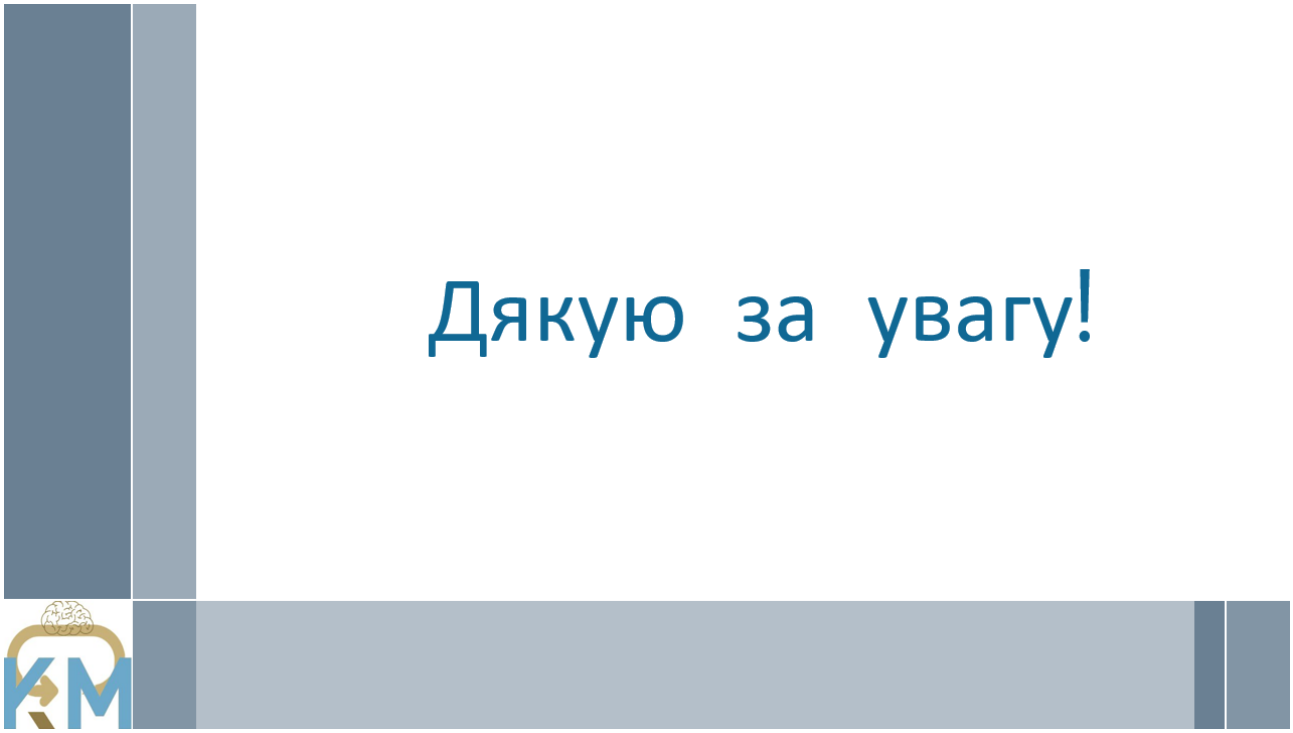


Рисунок Г.17 – Слайд презентації №17