Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія управління проектами з використанням технології Azure»

Виконав: студент 2-го курсу, групи
2КН-22м спеціальності 122
«Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Кривенко І. І.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Богач.І.В.
(прізвище та ініціали)
«04» 12 2023 р.

Опонент: д.т.н., проф. каф. АІІТ
_____ Квєтний Р.Н.
(прізвище та ініціали)
«07» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«08.12.» 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
29.08. **2023 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кривенку Івану Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія управління проектами з використанням технології Azure
керівник роботи_ к.т.н., доцент кафедри КН Богач І.В.
затверджені наказом вищого навчального закладу від "13"09.2023р №142
2. Строк подання студентом роботи 13.09. 2023 року
3. Вихідні дані до роботи:
Angular framework, Mongo DB, Express.js, HTML, CSS, ECMAScript, TypeScript, SCSS, Token, Cryptography, Tests, Integration tests, Jest, JUnit; часові обмеження відповіді системи – 10000 мс, кількість користувачів – до 100, опис завдання – не більше 1000 символів, формат фото користувача – .jpg or .png, інтернет підключення – від 500 кб/с, кількість оперативної пам'яті – від 1гб, кількість ядер процессора – від 2, браузер – Chrome, Foxfire.
4. Зміст текстової частини:
Вступ, аналіз сучасного стану систем управління проектами; розробка інформаційної технології управління проектами з використанням технології Azure; програмна реалізація інформаційної технології управління проектами з використанням технології Azure; економічна частина; висновки; перелік використаних джерел; додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язковихкреслень):
Алгоритм роботи веб-додатку управління проєктами з використанням технології Azure; інтерфейс користувача; UML-діаграми роботи частин веб-додатку; результати тестування веб-додатку управління проектами з використанням технології Azure.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціалита посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Богач І.В., к.т.н., доц. каф. КН	29.08.23	10.11.23
4	Ратушняк О.Г.,к.т.н., доц. каф. ЕПВМ	10.23	08.11.23

7. Дата видачі завдання 29.08. 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	При-мітка
1	Аналіз сучасного стану систем управління проектами	1.09.23-07.09.23	розділ 1
2	Розробка інформаційної технології управління проектами з використанням технології Azure	08.09.23-20.09.23	розділ 2
3	Програмна реалізація та тестування інформаційної технології управління проектами з використанням технології Azure	21.09.23-20.10.23	розділ 3
4	Підготовка економічної частини	21.10.23-29.10.23	розділ 4
5	Апробація та/або впровадження результатів дослідження	30.10.23-05.11.23	теза доповіді
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	06.11.23 – 10.11.23	Подсер. записки графічн. матер. презентації

Студент _____ Кривенко І. І.

Керівник роботи _____ Богач І.В.

АНОТАЦІЯ

УДК 004.89

Кривенко І. І. Інформаційна технологія управління проектами з використанням технології Azure. Магістерська кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Вінниця: ВНТУ, 2023. 119 с.

Укр. мовою. Бібліогр.: 24 назв; рис.: 18; табл. 11.

Дана магістерська кваліфікаційна робота присвячена розробці та реалізації інформаційної технології для управління проектами з використанням такої технології Cloud Computing як Azure. На даний час створюється все більше проектів різних типів, кожен з яких потребує впорядкованого та чіткого керування. Тому додаток для управління проектами є незамінною частиною цього процесу. За допомогою систем управління проектами менеджер або керівник проекту має змогу дізнатись статус певних задач або проекту в цілому, змінити статус певного процесу або назначити працівника на певну задачу. Робота розглядає такі ключові аспекти: Аналіз сучасних технологій управління проектами та їх переваги; Розробка веб-додатку з використанням JavaScript та React.js для зручного управління товарними позиціями та знижками; Оптимізація продуктивності для забезпечення швидкого та ефективного взаємодії користувачів з додатком; Проведення тестів та валідація функціональності; Економічна оцінка витрат та користі від впровадження нової системи.

Ця робота спрямована на розробку інноваційної інформаційної технології, яка дозволить ефективно управляти проектами, з допомогою аналізу завдань та оптимізації ділових процесів для проектів.

Ключові слова: інформаційна технологія, управління проектами, JavaScript, Cloud Computing, оптимізація продуктивності.

ABSTRACT

Kryvenko I. I. Information technology for project management using Azure technology. Master’s qualification paper of a specialty 122 – Computer science, educational program – Artifitial intelligence systems. Vinnytsia: VNTU, 2023. 119 p.

In Ukrainian language. Bibliographer: 24 titles; fig .: 18; table 11.

This Master's thesis is devoted to the development and implementation of information technology for project management using Cloud Computing technology such as Azure. Currently, more and more projects of various types are being created, each of which requires orderly and clear management. Therefore, a project management application is an indispensable part of this process. With the help of project management systems, a manager or project manager can find out the status of certain tasks or the project as a whole, change the status of a certain process or assign an employee to a certain task. The work considers the following key aspects: Analysis of modern project management technologies and their advantages; Development of a web application using JavaScript and React.js for convenient management of product positions and discounts; Performance optimization to ensure fast and efficient user interaction with the application; Conducting tests and validating functionality; Economic assessment of the costs and benefits of implementing a new system.

This work is aimed at the development of innovative information technology that will allow effective project management, with the help of task analysis and optimization of business processes for projects.

Keywords: Information Technology, Project Management, JavaScript, Cloud Computing, Performance Optimization.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Управління проектами є доволі розповсбдженою проблемою в сучасному світі, адже кожен проект, кожна ідея, що повинна пройти від етапу ідеї до реалізації повинна бути контрольованим проектом, адже тільки коли процеси контрольовані, реально досягнути успіху в чомусь. З ростом популярності Cloud Computing систем та збільшенням конкуренції в цьому секторі, підприємства шукають шляхи поліпшення використання цих технологій в їхніх системах.

Аналіз сучасних підходів до управління проектами підкреслює необхідність інноваційних технологічних рішень для оптимізації процесів, пов'язаних з управлінням проектами та правильним розподіленням ресурсів для виконання всіх завдань ефективно. Інформаційні технології можуть значно полегшити цей процес, забезпечуючи зручний доступ до актуальних методів розробки чи виконання завдань, зменшуючи трудомісткість управління та надаючи інструменти для аналізу та взаємодії зі робітниками.

Зростання Cloud Computing у сфері інформаційних систем та послуг робить інформаційну технологію управління проектами ключовим елементом конкурентоспроможності. Інтеграція цих технологій дозволить бізнесу швидше реагувати на зміни на ринку та задовольняти очікування клієнтів у реальному часі.

Таким чином, актуальність цього дослідження полягає в необхідності розробки та впровадження інформаційної технології, яка допоможе підприємствам ефективно управляти проектами на всіх етапах за допомогою веб-додатку, забезпечуючи підвищення конкурентоспроможності та задоволення потреб сучасних споживачів. У майбутньому, подальший прогрес і вдосконалення цих технологій відзначають великий потенціал для поліпшення якості обслуговування клієнтів та оптимізації бізнес-процесів.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових

досліджень кафедри комп'ютерних наук Вінницького національного технічного університету та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання досліджень. Метою дослідження магістерської кваліфікаційної роботи є розширення функціональності та покращення методів управління проектами за рахунок використання хмарної технології Azure.

Для досягнення даної мети, передбачається реалізувати наступні завдання:

1. Аналіз сучасних технологій управління проектами, що використовують технології аналізу даних.

2. Проектування та розробка веб-додатку за допомогою фреймворків Angular та Express.js для управління проектами з використанням хмарної технології Azure.

3. Оптимізація продуктивності додатку для забезпечення швидкого та ефективного використання.

4. Розробка імплементації для аналізу завдань та рекомендації працівників в веб-додатку.

5. Тестування розробленої інформаційної технології та порівняння з аналогами.

6. Економічна оцінка витрат та користі від впровадження нової системи управління проектами з використанням технології Azure.

Дослідження та реалізація цих завдань спрямовані на досягнення мети розробки інформаційної технології, яка полегшить управління проектами, забезпечуючи покращення користувацького досвіду та підтримку бізнес-процесів для підприємств.

Об'єкт дослідження є процес розробки компонентів веб-додатку з функціоналом управління проектами та ресурсами в веб-додатку.

Предметом дослідження є програмні засоби та технології, які використовуються для створення та функціонування компонентів веб-додатку, що дозволяє управляти проектами та аналізом завдань з використанням технології Azure.

Методи дослідження. У роботі використані такі методи наукових досліджень: аналіз сучасних технологій та практик управління проектами з використанням хмарних технологій Azure з метою визначення кращих практик та проблем, які потрібно вирішити; вивчення аналогічних програмних рішень та сервісів, використовуваних для управління проектами, для виявлення корисних властивостей та інноваційних підходів; дослідження алгоритмів та методів управління проектами з метою покращення управління ресурсами та універсального аналізу завдань; аналіз проблем та вимог, пов'язаних з організацією та управлінням проектаим, для забезпечення належної функціональності та зручності використання; проведення експериментів із програмним забезпеченням для валідації та оцінки розроблених рішень та алгоритмів; застосування об'єктно-орієнтованого програмування для розробки програмного забезпечення управління проектами з використанням технології Azure.

Наукова новизна одержаних результатів полягає в наступному:

1. Вперше запропоновано модель аналізу завдань на основі їх початкового опису та надання рекомендацій щодо працівників на основі оцінки завдань із застосуванням сервісу хмарних технологій Azure, що дозволило спростити процес управління проектами.

2. Вдосконалено інформаційну технологію управління ресурсами проекту, яка відрізняється від існуючих застосуванням комбінацій моделей нейронних мереж та методів нечіткої логіки, що дозволяє спростити процес управління проектами на відміну від існуючих аналогів.

Практичне значення отриманих результатів дослідження полягає в наступному:

1. Розширено функціонал процесу управління проектами шляхом запровадження аналізу попереднього опису завдань з допомогою сторонніх API нейронних мереж, а також надання рекомендацій щодо найбільш підходячого працівника для виконання задачі на основі хмарної технології Azure, що підвищує ефективність керування ресурсами на проекті.

2. Розроблено програмний засіб для управління проектами з використанням технології Azure на основі Angular та Express.js, а також використанням MongoDB, як бази даних.

Розроблена інформаційна технологія сприяє покращеню процесів управління проектами шляхом аналізу попереднього опису завдання, а також надання рекомендацій, а також підвищує їх ефективність.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, теоретичні положення в магістерській роботі базуються на чіткому та обґрунтованому визначенні завдань та цілей дослідження. Перевірка та аналіз роботи програмного забезпечення, розробленого в рамках дослідження, дозволяють підтвердити вірність та функціональність теоретичних положень.

Особистий внесок студента. Усі результати, що наведені у магістерській кваліфікаційній роботі, тримані самостійно. У працях, які написано у співавторстві, здобувачу належать: аналіз використання хмарної технології Azure в побудові веб-додатку [1], аналіз документо-орієнтованих баз даних на прикладі MongoDB [2].

Апробація результатів роботи. Результати досліджень було апробовано на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи-2024» (МН-2024) [1], «LI науково-технічній конференції підрозділів Вінницького національного технічного університету НТКП ВНТУ, факультет інтелектуальних інформаційних технологій та автоматизації (2022)» [2].

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано матеріали доповіді науково-практичної конференцій [1-2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ СИСТЕМ УПРАЛІННЯ ПРОЕКТАМИ З ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ АНАЛІЗУ ДАНИХ

1.1 Огляд існуючих систем управління проєктами

«Jira» — це програмне забезпечення для управління проектами, розроблене австралійською компанією Atlassian. Програмне забезпечення базується на гнучкій методології. Це ПЗ використовується в кількох цілях – відстеження помилок, відстеження проблем і керування проектами. Багато підприємств також використовують програмне забезпечення JIRA нестандартним способом як засіб автоматизації складу, документообігу, оптимізації витрат тощо. Інформаційна панель JIRA містить кілька корисних функцій, які дозволяють легко вирішувати проблеми. Одне з найбільш затребуваних гнучких рішень для управління проектами, JIRA нещодавно налаштувала деякі зі своїх продуктів для всіх типів команд і організацій, включаючи IT, маркетинг, операції, фінанси, кадри, юридичні та інші відділи [3].

Основні плюси Jira:

- Спільна робота команди**.** Програмний продукт надає зручні інструменти для спільної роботи команди. Інструменти дозволяють відстежувати прогрес роботи, коментувати завдання та обговорювати спірні моменти в роботі.
- Інтеграція з іншими інструментами**.** Jira легко інтегрується з іншими інструментами розробки програмного забезпечення, такими як Bitbucket, GitHub, Confluence та інші. Це полегшує взаємодію між різними етапами розробки.
- Широкий спектр необов’язкових, але корисних можливостей**,** використовуючи різні плагіни, які доступні в магазині Jira, можна розширити наявний функціонал. Ось найпопулярніші з них: Tempo, Script Runner, EazyBI, Big Picture, Structure.

Хоча Jira є потужним інструментом, деякі користувачі можуть відзначити складність для новачків у навчанні та освоєнні інтерфейсу. Також, при великій кількості задач або проектів, може виникати перевантаження інформацією, що

робить важчим відстеження важливих елементів проекту. Деякі користувачі також вказують на високі витрати часу на налаштування та адміністрування системи..

«Trello» – це веб-додаток для управління завданнями та проєктами, який дозволяє користувачам організовувати свою роботу за допомогою "дошок", "списків" та "карток". Основна інформація про Trello включає наступне:

Основні переваги Trello:

- Простота використання: Trello відзначається інтуїтивним і легким інтерфейсом, що дозволяє користувачам швидко організовувати та відстежувати завдання.

- Гнучкість: Додаток підходить для різних видів проєктів, від особистих завдань до спільної роботи в команді.

- Спільна робота: Trello підтримує колективну роботу, дозволяючи користувачам додавати учасників до дошок та спільно працювати над завданнями.

- Зручність для керівництва проєктами: Використання "дошок" та "списків" дозволяє створювати категорії та відстежувати хід виконання завдань.

- Інтеграція з іншими інструментами: Trello підтримує інтеграцію з різними додатками та сервісами, що полегшує роботу користувачів.

Недоліки Trello:

- Обмежені можливості безкоштовної версії: Деякі корисні функції доступні лише в платних версіях Trello.

- Брак розширених інструментів для великих проєктів: Для складних проєктів, які вимагають розширених функцій управління, Trello може бути недостатнім.

- Не відповідає всім видам бізнесу: Додаток може бути не підходити для деяких сфер бізнесу, де важливі специфічні вимоги.

Trello використовується для створення "дошок", на яких розміщуються "списки" завдань, а кожне завдання представлене "карткою". Користувачі можуть додавати завдання, коментарі, дедлайни, мітки та відстежувати хід

виконання завдань. Додаток побудований на технологіях веб-розробки, таких як HTML, CSS, JavaScript, та використовує сервери для зберігання та синхронізації даних між користувачами.

Програмне забезпечення «Microsoft Project» є інструментом для управління проєктами, який використовується для створення графіків, планів та координації ресурсів та часу в рамках проєктів. Воно включає такі корисні функції, як створення діаграм Ганта, використання дошок Канбан та календарів проєктів, і призначене для використання професіоналами у сфері управління проєктами.

Microsoft Project також відомий іншими назвами, такими як MS Project або Project Professional, яка є офіційною назвою цього програмного забезпечення на сьогодні. Крім того, існують інші продукти Microsoft Project із схожими назвами, такі як Project Online, Project Server та Project для Інтернету. Доступ до цих продуктів можливий через підписку на Microsoft Project Plan, яка має три доступні рівні цін.

Початково "Проєкт" був розроблений як програмне додаток для MS-DOS, написаний Роном Бредехофтом, і виник з його бачення створення рецепту для сніданку з яєць Бенедикта в термінах управління проєктами. Пізніше, програма була розроблена компанією Microsoft під керівництвом Алана М. Бойда та була представлена як внутрішній інструмент для керування різними проєктами всередині компанії. Ця історія пояснює корінь назви "Проєкт".

Програма дозволяє створювати бюджети на основі завдань та ресурсів, призначаючи їх завданням та визначаючи вартість ресурсів на основі їхніх ставок. Кожен ресурс може мати власний календар для визначення доступності, і їхні ресурси можуть використовуватися в різних проєктах. Програма створює розклади критичних шляхів та може бути вирівнювана за ресурсами. Microsoft Project дозволяє визначати різні рівні доступу для різних користувачів та зберігати дані в корпоративній мережі для спільного використання.

1.2 Аналіз потреб та вимог користувачів веб додатків

Аналіз потреб та вимог користувачів веб-додатків є ключовим етапом у процесі розробки та вдосконалення інтерактивних онлайн-ресурсів. Цей розділ детально розглядає важливість систематичного та комплексного підходу до збору та оцінки інформації, яка визначає потреби та очікування кінцевих користувачів.

Важливим елементом в аналізі є визначення цільової аудиторії веб-додатку. Систематичний аналіз демографічних характеристик, поведінкових патернів та потреб споживачів дозволяє точно визначити, які функціональності та сервіси будуть найбільш цікавими та корисними для кінцевого користувача. Важливо враховувати різноманітність та індивідуальні вподобання різних груп користувачів.

Під час аналізу потреб і вимог користувачів, слід враховувати їхні очікування щодо інтерфейсу та взаємодії з додатком. Ретельне вивчення користувачів, їхніх досвідів та фідбеку дозволяє виявити можливості для оптимізації інтерфейсу та забезпечити максимальний рівень зручності використання.

Аналіз потреб також включає в себе вивчення конкурентного середовища та аналіз аналогічних веб-додатків на ринку. Це надає можливість ідентифікувати сильні та слабкі сторони конкурентів, враховувати інновації та уникати повторення помилок інших гравців.

Особливу увагу слід приділити визначенню основних функціональних вимог до веб-додатку. Це можуть бути такі вимоги, як безпека, швидкодія, масштабованість та інші технічні характеристики, які впливають на загальний функціонал і ефективність додатку.

Також, веб-додаток повинен швидко виконувати функції, які він надає, а також володіти хорошою швидкодією, цього можна досягти за допомогою так званого асинхронного підходу до написання коду, який є основою JavaScript, Node.js, а також всіх феріймворків, що працюють на основі JavaScript. Концепція

асинхронного підходу полягає в тому, що весь процес роботи веб-додатку базується на одному потоці, тобто все відбувається крок за кроком, а також рядок за рядком, а тому, якщо візьмемо до прикладу синхронний код, то всі його етапи будуть виконуватись один за одним, а наступний процес буде очікувати на закінчення попереднього процесу, а при умові, що процес може вимагати багато часу на виконання та забирати багато ресурсів, це не підходить для сучасного додатку, а особливо погано впливає на користувацький досвід, так як для того, щоб виконувати якісь функцій, користувач повинен велику кількість часу очікувати на закінчення попереднього процесу. Тоді на допомогу приходить асинхронність в JavaScript. Асинхронний JavaScript є однією з важливих частин мови, оскільки він керує тим, як ми виконуємо довгострокові завдання, наприклад отримання даних із сервера або API.

У спрощених словах, ми можемо розглядати асинхронний код як код, який починає завдання зараз і завершує його пізніше. JavaScript за своєю природою є синхронною мовою. А це означає, що JavaScript може виконувати лише один код за раз — зверху вниз. Нитка схожа на впорядковану послідовність операторів, як показано на рисунку 1.1 [4]:

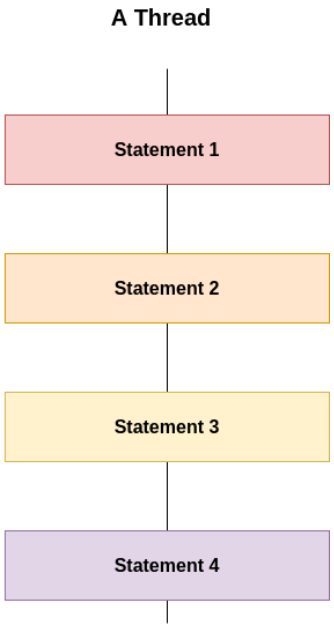

Рисунок 1.1 – Потік в JavaScript

В загальному ідея асинхронності дуже допомагає, коли мова заходить про оптимізацію роботи з веб-додатками, це стосується як клієнтської частини, так і back-end частини. Наприклад, коли користувач викликає якусь дію, яка вимагає багато ресурсів та часу, JavaScript починає цей процес, але делегує його Event Loop механізму, що спеціально створений для роботи з асинхронними задачами. В нього доволі зрозумілий принцип роботи, якщо розбрати на рисунку, що зображений на рисунку 1.2 [5]:

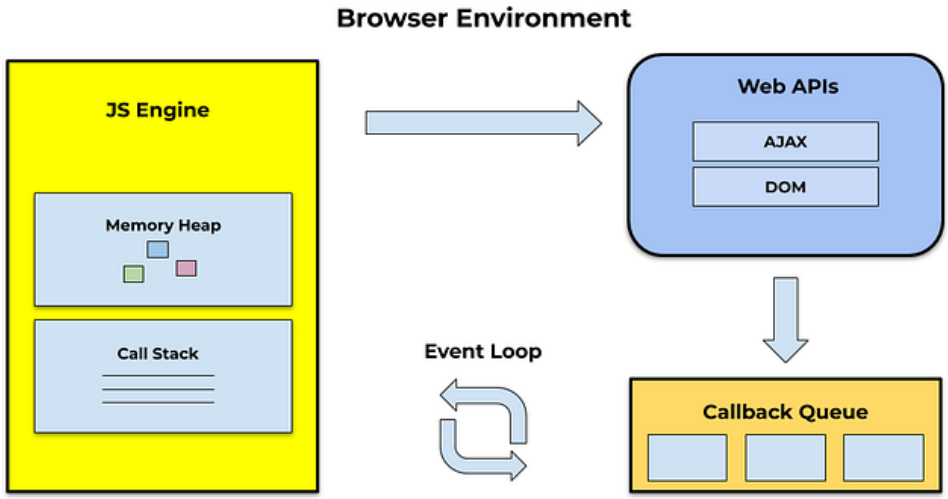

Рисунок 1.2 – Event loop

Він працює за таким принципом: користувач викликає функцію, що вимагає багато ресурсів, при цьому функція приймає callback function, як аргумент, Event loop відправляє цю функцію в Callback queue, і після закінчення процесу, нам браузеру повертається ця функція і викликається, запускаючи певний процес або повертаючи певні дані, але для користувача це майже непомітно, адже він може користуватися всіма іншими наявними функціями додатку. Саме завдяки цій технології, досвід користування додатком набуває набагато вищого рівня.

Підсумовуючи, аналіз потреб та вимог користувачів веб-додатків вимагає глибокого розуміння аудиторії та контексту використання. Цей етап створює фундамент для подальшої розробки, де враховуються усі аспекти, що впливають

на задоволення та успішність користувачів у використанні цього інтерактивного веб-додатку.

1.3 Проєктування та методи розробки компонентів системи

Angular компонент - це ізольована частина функціоналу з власною логікою, HTML-шаблоном та CSS-стилями. Іншими словами, клас стає Angular компонентом, якщо його оголошення передує декоратору @Component() з об'єктом налаштувань. Основна ідея полягає в тому, що кожен Angular-компонент повинен виконувати одне конкретне завдання і не повинен навантажуватися зайвими функціями. Важливо уникати створення надто великих та завантажених компонентів для використання в системі [6].

Зазвичай компоненти варто розробляти як найпростіше можливими, для забезпечення їхньої легкості підтримки та тестування. Однак вони також повинні бути гнучкими, щоб їх можна було перевикористовувати і не потребувати створення клонів компонентів, які мають схожий вигляд та функціонал, але мають деяку різницю. Крім того, слід уникати перевантаження компонентів зайвою кількістю параметрів, які активують або вимикають певну поведінку компонента, особливо якщо ці параметри рідко використовуються.

Основна робота в Angular виконується через компоненти, принаймні один з яких повинен бути кореневим. Наприклад, можна створити два компоненти: app.component.ts (який виступає кореневим) та projects-list.component.ts. Кореневий компонент визначається в файлі app.module.ts, як зазначено у декораторі @NgModule [7].

Хоча можна обійтися одним компонентом і розміщувати весь код у ньому, у середніх та великих проєктах зазвичай рекомендується розподілити функціональність між різними компонентами, кожен з яких відповідає за певний аспект. У нашій системі для досягнення зручності та масштабованості буде використовуватися система компонентів, які будуть групуватися в різні типи структур в залежності від обраної сторінки системи.

Компонент - це базовий клас, який стає функціональним завдяки декоратору @Component(), який містить наступну інформацію:

- Template - вказує шаблон для компонента. Це дозволяє вбудовувати відображення компонента без створення окремого файлу шаблону. Проте для великих шаблонів, особливо тих, що містять багато коду (15-20 рядків і більше), цей підхід може бути не дуже зручним.
- Styles - аналогічно до шаблону, визначає стилі компонента у самому файлі компонента.
- Providers - використовується для ініціалізації сервісів, які будуть використовуватись у цьому компоненті.
- Animations - вказує анімації, які пов'язані з цим компонентом.

Сам вигляд компонента визначається через використання шаблону-компаньйона. Цей шаблон є формою HTML, яка визначає, як компонент буде відображатися.

Зазвичай темплейти ієрархічно організовані, що дозволяє змінювати, показувати або приховувати цілі розділи або сторінки інтерфейсу як єдине ціле. Кожен компонент має свій власний шаблон, який визначає, як він буде представлений. Крім того, компоненти можуть містити ієрархію темплейтів, включаючи вбудовані темплейти, що використовуються іншими компонентами. Структура прикладу архітектури Angular додатку зображена на рисунку 1.3.

Ієрархія концепцій може включати різні уявлення, які походять від компонентів, які належать до одного і того ж NgModule. Однак часто вона також охоплює уявлення, що виникають з компонентів, визначених в різних NgModules.

Шаблон схожий на стандартний HTML, але він містить деякі ознаки шаблону Angular, що безпосередньо впливає на HTML, відповідно до логіки програми та стану програми та даних DOM. Шаблон може використовувати прив'язку даних для координації між даними програми та DOM, а також так звані пайпи (pipe) для перетворення даних перед їх відображенням та директиви для впровадження логіки програми в відображені дані [10].

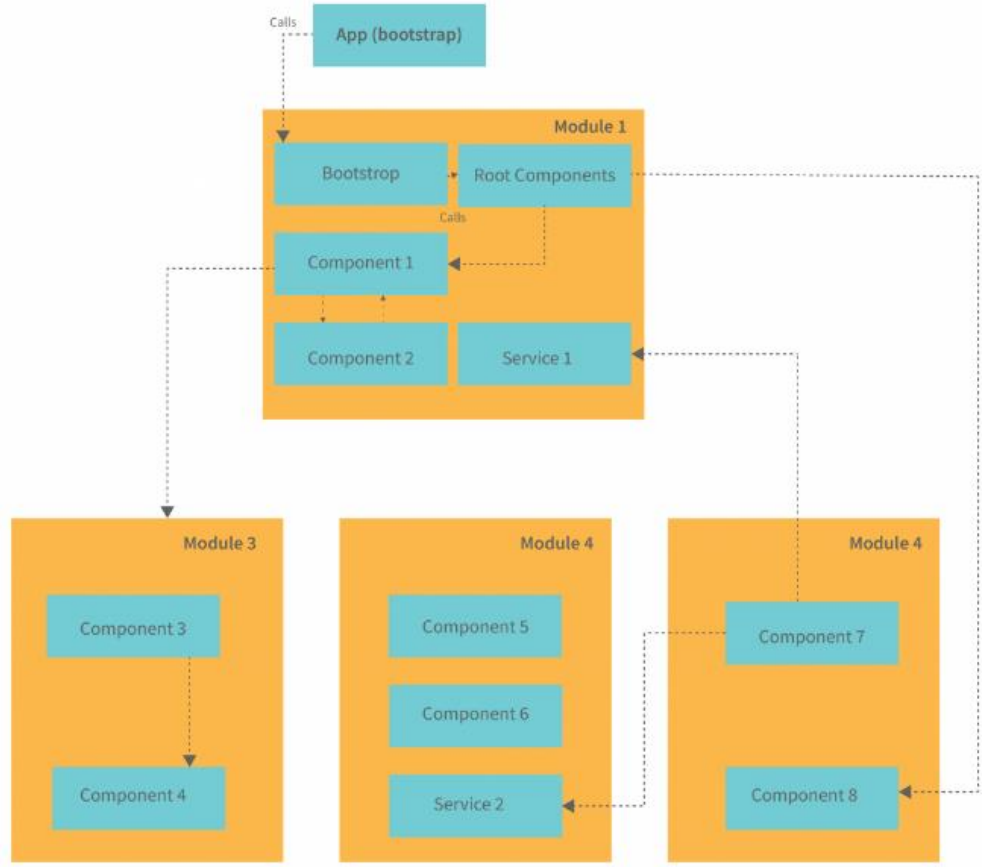

Рисунок 1.3 – Схема архітектури компонента «Angular»

У шаблоні використовуються добре нам знайомі елементи HTML розмітки, такі як , <header>, <p> та інші, а також синтаксис шаблону Angular, такий як *ngFor, {{project.name}}, (click), [project] і <app-project-detail>. Елементи синтаксису шаблону вказують Angular, як реалізувати відображення HTML-коду на екрані, використовуючи програмну логіку та дані.

Фреймворк відповідає за передачу значень даних до HTML-елементів та обробку відповідей користувача, включаючи слідкування за станом значень. Angular підтримує двосторонню прив'язку даних, механізм, який забезпечує взаємодію між частинами шаблону та частинами компонента. Крім того, Angular дозволяє додавати розмітку прив'язки до шаблону HTML для визначення способу зв'язку обидві сторони [11].

Структурні директиви модифікують макет, додаючи, видаляючи або замінюючи елементи в DOM. Наприклад, дві вбудовані структурні директиви визначають спосіб візуалізації даних:

ngFor: Ця ітеративна структурна директива повідомляє Angular відображати окремий компонент для кожного елемента в масиві даних.

ngIf: Ця умовна структурна директива відображає компонент тільки в тому випадку, коли вказаний вираз повертає true [12].

Директиви атрибутів модифікують зовнішній вигляд або поведінку існуючого DOM-елемента. У шаблонах вони виглядають як стандартні атрибути HTML і відтак називаються атрибутами-директивами. Візьмемо для прикладу таку директиву як ngModel, що є директивою атрибута, яка впроваджує двосторонню прив'язку даних. Вона модифікує поведінку наявного елемента, зазвичай <input>, встановлюючи значення його властивості та реагуючи на події зміни..

Метадані для директив зазвичай визначаються через декоратор і пов'язують декорований клас з елементом, який вибирається за допомогою селектора. У шаблонах, директиви зазвичай відображаються як атрибути елемента або використовуються як імена атрибутів, призначення або прив'язки. На рисунку 1.4 показано дві форми розмітки прив'язки даних.

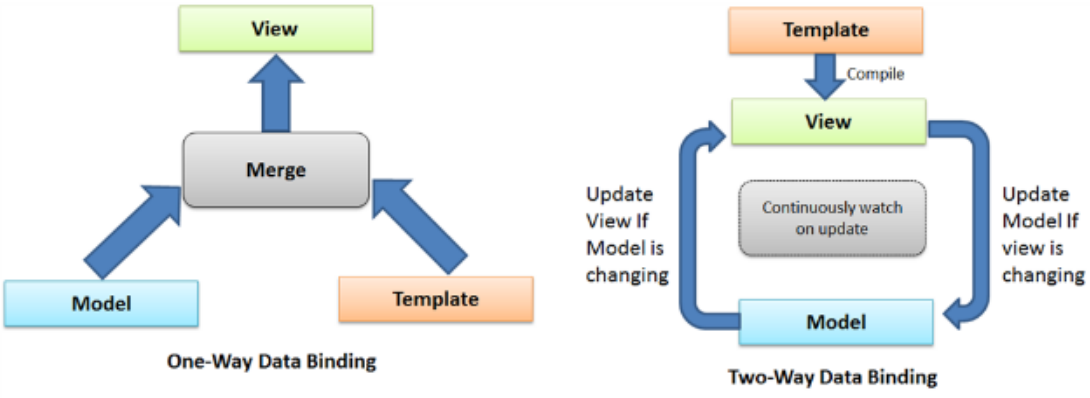

Рисунок 1.4 – Схема прив'язки даних в «Angular»

1.4 Висновок до розділу 1

В першому розділі розглянуто основні аспекти роботи: проведено огляд та аналіз існуючих систем управлінян проєктами, представлено основні поняття по темі, наведено переваги створення додатку з використанням фреймворку Angular.

Розроблена система проектів та завдань для користувачів та їх комунікації для керування цими проєктами, яка повинна мати рядок пошуку, який запитує у внутрішнього API список всіх проєктів і відображає відповіді. І щоразу, коли користувач натискає на проєкт, його буде перенаправлено на сторінку проєкту. Там він зможе слідкувати за прогресом виконання завдання та проєкту в цілому, створювати та видаляти завдання, при створенні, в залежності від опису завдання, користувачеві будуть рекомендуватись працівники та оцінюватись складність завдання. Також було розроблено сервер і базу даних для створення користувачів, проєктів, завдань, збереження їх даних і додавання можливості добавити/видалити завдання чи працівника з проєкту, налаштування, змінити дані користувача.

Актуальним є розробка повної та самодостатньої системи управління проєктами на основі безкоштовного фреймворку Angular та додаткових бібліотек і програмного забезпечення, що на відміну від існуючих дає можливість залучити штучний інтелект для оцінки завданнь та надання рекомендацій, щодо працівників, що найкраще підходять для роботи над завданням. Доступ до системи є на різних платформах, таких як Desktop, IOS та Android.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ УПРАВЛІННЯ ПРОЄКТАМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ AZURE

2.1 Концепція системи управління проєктами з допомогою аналізу даних на Azure та з використанням Angular та Express.js

Концепція системи управління проєктами з допомогою аналізу даних на Azure та Angular повинна включати в себе розгляд таких аспектів, як реєстрація та авторизація користувачів, профілі користувачів, створення та видалення проєктів, створення та видалення завдань, зміна статусу завдання, присвоєння завдань для певного користувача, оцінка складності задачі та надання рекомендацій, щодо працівників, які найкраще підходять для виконання цієї задачі. Ось загальний опис цієї концепції:

Реєстрація та авторизація:

• Користувачі можуть створити обліковий запис, вказавши свою адресу електронної пошти, ім’я, прізвище, пароль, а також по бажанню: компанію та команду в якій працюють, свій рівень занань.

• Важливо забезпечити безпеку і захист персональних даних користувачів.

Профілі користувачів:

• Кожен користувач має особистий профіль, де відображаються його дані, проєкт над яким працює та завдання над якими він працює.

• Користувачі можуть редагувати особисту інформацію.

Створення проектів:

• Всі проєкти, над якими працює користувач будуть відображені на головній сторінці.

• Користувачі можуть видаляти та редагувати данні проєкту, якщо мають права на це.

• Для зручності користувачів система повинна підтримувати фільтрацію проєктів.

Робота з завданнями:

• Користувачі можуть створювати, видаляти та редагувати завдання.

• Користувачі можуть отримати оцінку завдання в залежності від опису завдання, а також отримати рекомендації щодо користувачів, що найкраще підходять для виконання цього завдання.

Відстеження прогресу виконання завдання та проекту в цілому:

• Користувачі мають можливість змінювати статус завдання (не розпочато, в процесі виконання, тестується, готове).

• Користувачі можуть слідкувати за прогресом проєкту, та отримувати такі данні, як кількість виконаних завдання, час створення проєкту, заплановану дату закінчення та прогрес в цілому.

Мобільна версія:

• Розробка мобільного додатка або адаптивної веб-версії для зручного використання на різних пристроях.

Безпека:

• Захист від несанкціонованого доступу, обробка вірусів, захист від DDoS-атак.

• Забезпечення конфіденційності даних користувачів.

Ця концепція є загальною і може бути розширена відповідно до конкретних потреб системи та принципів дизайну, щоб створити зручний та привабливий інтерфейс для користувачів.

2.2 Планування реалізації роботи Back-end в заємодії з технологією cloud computing Azure

Express.Js - це фреймворк, створений над Node.js. В основному використовується для розробки програм на стороні сервера. Більшість функцій у Express.Js дуже схожі на звичаний Node.js додаток з покращеним та полегшеним синтаксисом, що спрощує та пришвидшує розробку бек-енд додатків.

Express.js - це веб-фреймворк для розробки веб-додатків та API на платформі Node.js. Він є одним з найпопулярніших та найвикористовуваніших фреймворків для створення серверної частини веб-додатків, завдяки своєму лаконічному дизайну, гнучкості та великій активній спільноті розробників [8].

Основні риси та концепції Express.js включають:

Маршрутизація: Express дозволяє визначати маршрути, які визначають спосіб обробки запитів. Це дозволяє розробникам легко визначати, які дії виконувати для різних URL-шляхів.

Middleware – це функції, які мають доступ до об'єктів запиту та відповіді, і можуть модифікувати їх або виконувати певні операції перед тим, як запит буде оброблено маршрутом. Це дозволяє створювати ланцюжки обробки запитів та використовувати middleware для різних завдань, таких як аутентифікація, логування тощо.

Шаблонізатори: Express підтримує використання різних шаблонізаторів для генерації HTML-коду на сервері. Це дозволяє легко розробляти сторінки та переробляти дані для виведення на веб-сторінках.

Статичні файли: Express може служити статичні файли, такі як зображення, стилі та сценарії, що дозволяє легко забезпечити доступ до ресурсів на веб-сайті.

Робота з HTTP-запитами та відповідями: Express надає простий та зручний інтерфейс для роботи з HTTP-запитами та відповідями, що спрощує обробку та взаємодію з клієнтами.

Розширені можливості: Express також дозволяє використовувати різноманітні доповнення (middleware) та доповнювачі, які розширюють його базовий функціонал для вирішення конкретних задач або інтеграції з іншими бібліотеками.

Легкість використання та навчання: Express має простий та легкий для зрозуміння синтаксис, що робить його доступним для початківців та зручним для досвідчених розробників. Його документація та активна спільнота розробників роблять процес вивчення та використання досить простим та ефективним.

Ми можемо налаштовувати класи та компоненти так, як хочемо. Express.Js повністю підтримує Typescript, але також підтримує чистий JavaScript. Коли справа доходить до Express.Js, головною його особливістю є впровадження залежностей [8].

Перед тим, як починати розробку на Express.js, необхідно ознайомитись з основними принципами роботи цього фреймфорку та базовими компонентами:

- Controllers
- Providers
- Modules

Контролери в Express.js відповідають за обробку будь-яких вхідних запитів і повернення відповідей клієнтській стороні програми. Наприклад, якщо ви зробите виклик API до певної кінцевої точки, скажімо, /home, контролер отримає цей запит і на основі доступних ресурсів поверне відповідну відповідь.

Express.js було структуровано таким чином, щоб механізм маршрутизації міг контролювати, який контролер відповідатиме за обробку конкретного запиту.

Щоб визначити контролер у Express.js, створіть файл TypeScript і додайте декоратор @Controller(), як показано в наступному фрагменті коду:

```
import { Body, Controller} from 'express;
import { AuthService } from './authorization.service.ts';
import { SignInPayloadDto } from './auth.dto';
import { AuthGuard } from './authorithation.guard';

@Controller('authorization')
export class AuthorizationController {

  @HttpCode(200)
  @Post('login')
  signIn(@Body() signInDto: SignInPayloadDto) {
    return this.authService.signIn(signInDto.email,
signInDto.password);   }       }
```

Усередині цього контролера ми можемо реалізувати такі HTTP-запити, як GET, POST, PUT, DELETE.

У Express.Js розробник може створити провайдер та вставити його в контролер або інший провайдер. Їх ще називають сервісами. Провайдер/сервіси використовуються для обробки будь-якої форми складної логіки, як-от отримання даних із бази даних і виклик стороннього API.

Щоб визначити постачальника послуг, вам слід створити файл TypeScript із декоратором @Injectable() у верхній частині.

```
import { JwtService } from '@nestjs/jwt';
import { UsersService } from 'src/modules/users/users.service';
@Injectable()
export class AuthService {
  constructor(
    private readonly usersService: UsersService,
    private readonly jwtService: JwtService,
  ) {}
  async signIn(email: string, password: string): Promise<any> {
    if (!(await this.usersService.isUserExist(email))) {
      return;
    }
    const userCreds = await this.usersService.getUserCredentials(email);
    if (!userCreds || userCreds.password !== password) {
      throw new UnauthorizedException();
    }
    const user = await this.usersService.findOne(email);
    return {
      access_token: await this.jwtService.signAsync({ payload: user }),
    };
  }
}
```

Модулі дозволяють групувати пов’язані файли. Це файли Typescript, прикрашені декоратором @Module. Цей доданий декоратор надає метадані, які Express використовує для організації структури програми.

Кожен додаток Express.Js повинен мати принаймні один модуль. Під час масштабування програми добре мати кілька модулів, оскільки це зберігає структуру програми.

Окрім 3 вищезазначених типів файлів, є деякі типи файлів, про які ми не можемо забувати, реалізуючи програми за допомогою Express.Js.

- DTO: об’єкт передачі даних – це об’єкт, який визначає спосіб передачі даних через мережу.
- Інтерфейси: інтерфейси TypeScript використовуються для перевірки типів і визначення типів даних, які можна передати контролеру або службі Express.
- Ін’єкція залежностей: ін’єкція залежностей — це шаблон проєктування, який використовується для підвищення ефективності та модульності програм. Він часто використовується найбільшими фреймворками, щоб зберегти код чистим і зручнішим у використанні. Express.js також використовує його для створення пов’язаних компонентів.

```
import { Module } from '@nestjs/common';
import { ProjectsService } from './projects.service';
import { ProjectsController } from './projects.controller';

@Module({
  controllers: [ProjectsController],
  providers: [ProjectsService]
})
export class ProjectsModule {}
```

Схему взаємодії основних компонентів серверної частини можна розглянути на рисунку 2.1.

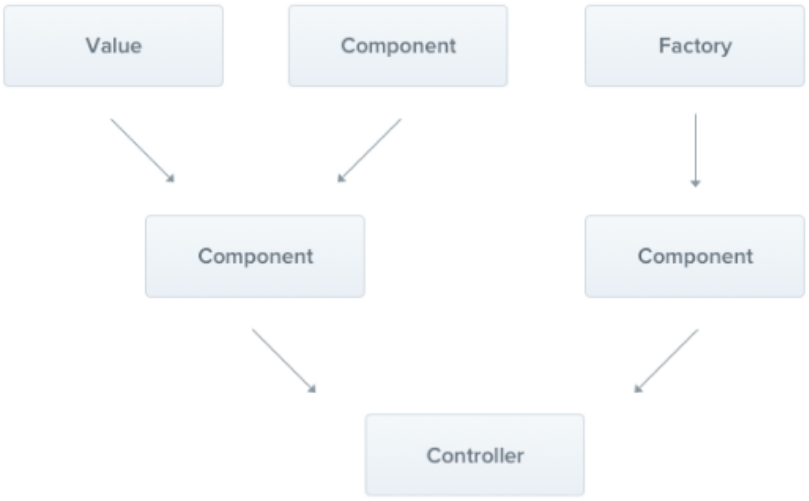

Рисунок 2.1 – Схема взаємодії основних компонентів серверної частини

Так як наш додаток буде також використовувати аналіз даних та cloud computing в аналізі складності завдань і рекомендації працівників, також необхідно описати схему та принципи взаємодії серверної частини та функціональності обробки даних та прийняття рішень.

Взаємодія серверної частини додатку для управління проєктами із функціональністю, що приймає дані у вигляді опису завдання та списку учасників проєкту і проводить оцінку складності завдання, а також рекомендацію працівників для цього завдання, представляє собою складний процес, який включає в себе використання штучного інтелекту та розгортання на платформі Azure.

Ця функціональність може бути ключовою в управлінні проєктами, оскільки допомагає здійснювати розподіл завдань та ресурсів ефективним способом. Давайте детально розглянемо процес взаємодії та функціональність цієї системи.

Приймання даних: Початковий етап включає надходження даних у систему. Оператор або користувач вводить опис завдання та список учасників проєкту через інтерфейс додатку.

Аналіз опису завдання: Система аналізує опис завдання, використовуючи при цьому різноманітні алгоритми обробки природної мови (NLP). Це дозволяє зрозуміти суть завдання, його пріоритетність та інші параметри, що впливають на його розподіл.

Оцінка складності завдання: Штучний інтелект використовує алгоритми, які враховують різні фактори, такі як обсяг роботи, терміни виконання, необхідні навички, ризики тощо. На основі цього аналізу система присвоює завданню оцінку складності.

Рекомендація працівників: На основі оцінки складності завдання та навичок, які необхідні для його виконання, система рекомендує кандидатів серед учасників проєкту. Ці рекомендації можуть включати якість відповідних навичок, досвід роботи та доступність працівників.

Прийняття рішення: Оператор або користувач може прийняти рекомендацію системи або внести свої власні корективи. Після прийняття рішення, завдання розподіляється відповідним працівникам.

Моніторинг та звітність: Система веде моніторинг виконання завдань та надає звіти про стан проєкту. Це допомагає операторам та учасникам проєкту відстежувати прогрес та при необхідності внесення змін.

Розгортання на Azure: Для забезпечення високої доступності та швидкості роботи, ця функціональність може бути розміщена на платформі Azure. Azure надає рішення для обчислень, зберігання даних, масштабування та безпеки.

Така система допомагає ефективно розподіляти завдання в рамках проєкту, оптимізувати ресурси та прискорювати прийняття рішень. Використання штучного інтелекту та хмарних рішень на Azure підвищує продуктивність та забезпечує якісний управлінський процес для проєктів будь-якої складності.

Для більш детального опису та розуміння Azure, для початку розглянемо такий термін, як Cloud Computing. Хмарні обчислення — це просто практика використання Інтернету для доступу до сховища, програмного забезпечення та послуг замість зберігання, встановлення та запуску програм на вашому власному обладнанні.

Отже, для зберігання файлів ви можете використовувати службу хмарних обчислень, як-от Microsoft OneDrive або Google Drive. Замість того, щоб зберігати ці файли на вашому власному комп’ютері, де вони займають місце на вашому жорсткому диску, вони зберігаються на комп’ютерах Microsoft — величезних серверах, до яких можна отримати доступ, коли вони вам знадобляться, через підключення до Інтернету.

Хмарне програмне забезпечення працює так само. Програмне забезпечення буде встановлено та запущено на віддаленому сервері, що належить компанії, яка виробляє програмне забезпечення, і коли ви хочете використовувати його, ви підключаєтесь до веб-сайту та отримуєте доступ до свого облікового запису звідти, використовуючи веб-браузер, а не традиційну програму для настільного комп’ютера. .

Існує майже безмежна кількість способів використання хмарних обчислень, які дають користувачам доступ до величезної кількості обчислювальної потужності, яку вони інакше не змогли б створити самі. Хмарні обчислення мають багато переваг, тому вони так швидко розвиваються.

Усе, що вам потрібно для доступу до хмарних додатків і служб, — це пристрій і підключення до Інтернету, що означає, що тягар купівлі та обслуговування обладнання та серверів зменшується для компаній. Хмарне програмне забезпечення часто дешевше та оплачується за принципом оплати за використання, тому компаніям не потрібно викладати величезні суми грошей, щоб придбати програмне забезпечення наперед.

Хмарні обчислення також роблять програмне забезпечення, файли та інші служби доступними будь-де, у будь-який час, на будь-якому пристрої за умови підключення до Інтернету [13].

Microsoft Azure — це служба хмарних обчислень, яку пропонує Microsoft. Існує понад 600 служб, які підпадають під егіду Azure, але загалом це веб-платформа, на якій можна створювати, тестувати, керувати та розгортати програми та служби.

На Azure розміщено широкий спектр програмного забезпечення як послуги (SaaS), платформи як послуги (PaaS) та інфраструктури як послуги (IaaS). Azure пропонує три основні області функціональності; Віртуальні машини, хмарні служби та служби програм.

Однією з найпопулярніших і найкорисніших послуг, доступних через Azure, є віртуальні машини.

Віртуальна машина — це комп'ютерний файл, іноді званий зображенням, який діє як справжній комп'ютер. Віртуальні машини зазвичай працюють у вікні, як традиційні комп'ютерні програми. Цей комп'ютер всередині комп'ютера відокремлений від решти системи, щоб будь-які зміни або програмне забезпечення, запущене всередині віртуальної машини, не «просочувалися» на головну машину [14].

Однією з найпоширеніших сфер застосування Azure є саме машинне навчання та штучний інтелект, тому саме для цього була обрана саме ця платформа.

Схема роботи додатку, разом з клієнтською частиною, серверною частиною, базою даних та хмарним обчислення буде виглядати наступним чином наведена на рисунку 2.2.

2.3 Розробка веб-інтерфейсу додатку для управління проєктами

Мобільні програми та веб-додатки, незалежно від свого призначення для користувачів чи бізнес-функціональності типу CRM і ERP, мають структуру, що складається з кількох ключових компонентів. Ця структура включає в себе Front-end (клієнтську візуальну частину), Back-end (серверну складову програми) та базу даних. Основні обчислення та логіка реалізуються на серверній частині, тоді як Front-end відповідає за відображення і взаємодію з користувачами через графічний інтерфейс.

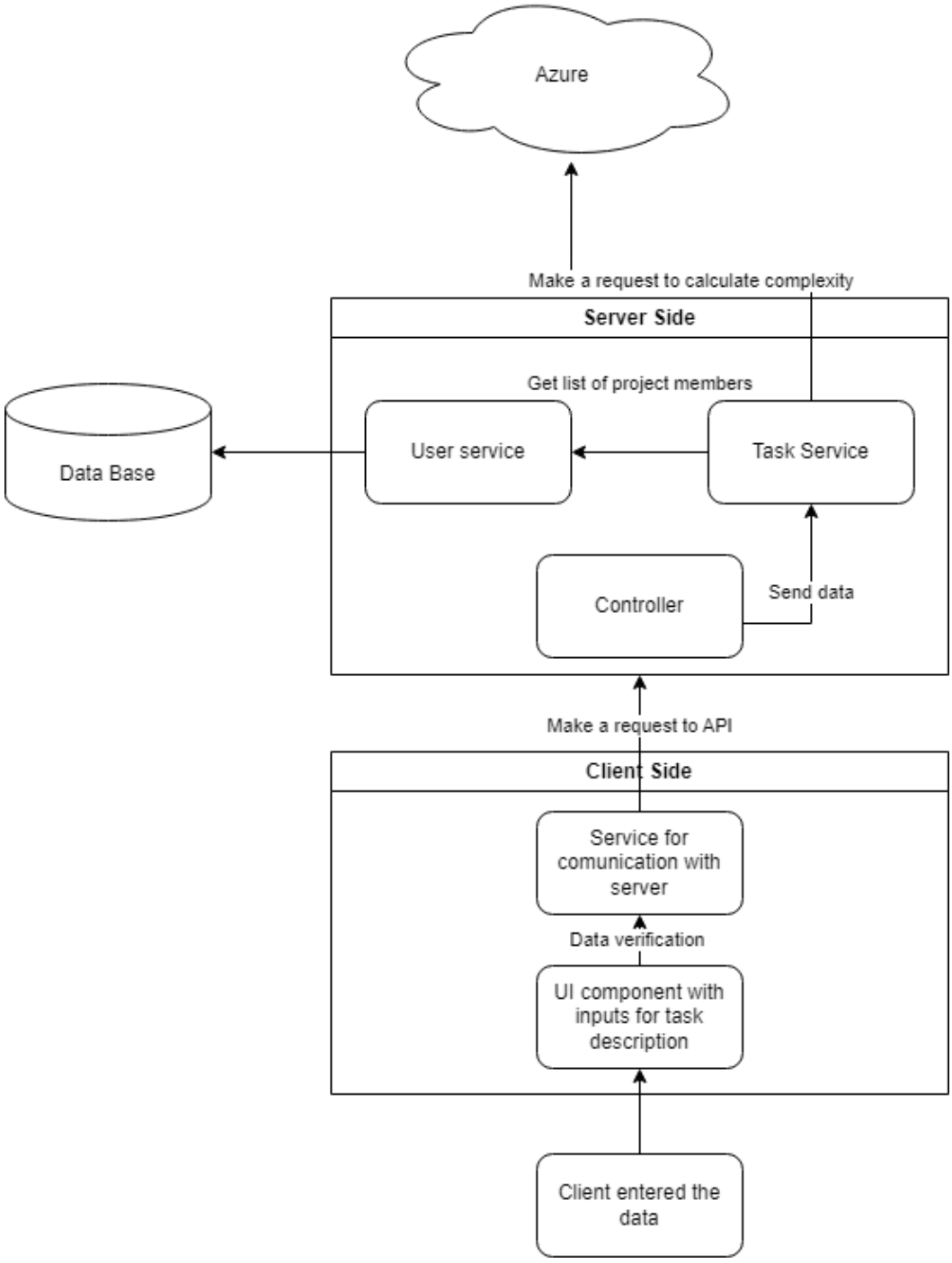

Рисунок 2.2 – Схема роботи додатку для управління проєктами з допомогою штучного інтелекту

Проте важливо зауважити, що розробка дизайну інтерфейсу не є першим етапом в процесі створення цифрового рішення. Для створення високоякісного веб-інтерфейсу необхідно виконати прототипування системи, яке включає створення мокапів, блок-схем та діаграм. Цей прототип, або програмна

архітектура системи, повинен враховувати всю бізнес-логіку та структуру комунікацій в системі.

Одина з основних переваг використання веб-інтерфейсу полягає в тому, що не потрібно встановлювати клієнтські програми на комп'ютери користувачів або слідкувати за версіями DLL-бібліотек. Все відбувається безпосередньо в браузері, і стандарти підтримуються всіма сучасними браузерами. Це спрощує процес оновлення та забезпечує високу доступність.

З погляду безпеки даних, всі дані зберігаються на сервері, де вони централізовано резервуються та захищаються. Використання протоколу HTTPS та обмін даними у форматі JSON забезпечує безпеку даних. При використанні веб-додатків у моделі SaaS (Software as a Service) через мережу, немає потреби в ліцензіях та інсталяції, що спрощує процес розгортання та оновлення.

Розвиток веб-інтерфейсів акцентується на таких ключових аспектах:

- Мінімалізм та простота: Зростає важливість спрощення інтерфейсу та оптимізація взаємодії користувача.
- Інтерактивність та асинхронність: Динаміка інтерфейсу підвищується, і користувачі спостерігають за змінами, не перезавантажуючи сторінки.
- Контекстність: Інформація та контроль відображаються відповідно до контексту та потреб користувача.
- Синхронізація та комет: З'являються програми, які спроможні синхронізувати інтерфейс з серверними подіями.
- Повноекранний лейаут: Заохочується максимальне використання доступного екранного простору та адаптація під різні роздільні здатності.
- Спрощений мобільний інтерфейс: Враховуючи розповсюдження мобільних пристроїв, дизайн пристосовується для малих екранів та сенсорних інтерфейсів.
- Підтримка стандартів: Важливість дотримання стандартів для забезпечення сумісності та зручності користування продуктом.

- Ця еволюція інтерфейсів допомагає створювати ефективні, зручні та інтуїтивно зрозумілі додатки та веб-сервіси, які задовольняють сучасні потреби користувачів. [15]

Мокапи веб-інтерейсу системи управління проєктами розроблялись за допомогою графічного онлайн редактора інтерфейсів Figma. Головна сторінка системи зображена на рисунку 2.3. та сторінка проєкту зображена на рисунку 2.4.

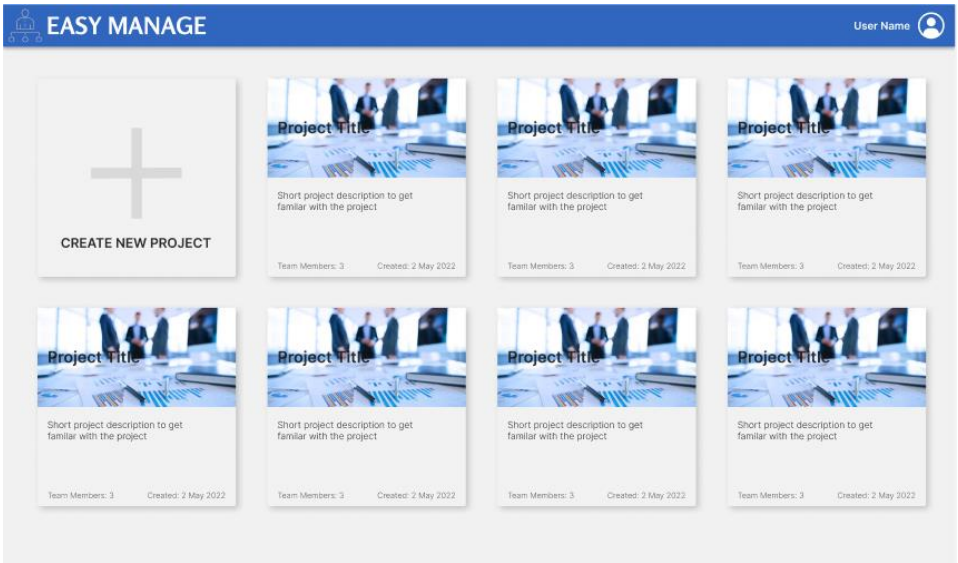

Рисунок 2.3 – Інтерфейс головної сторінки системи розроблений у «Figma»

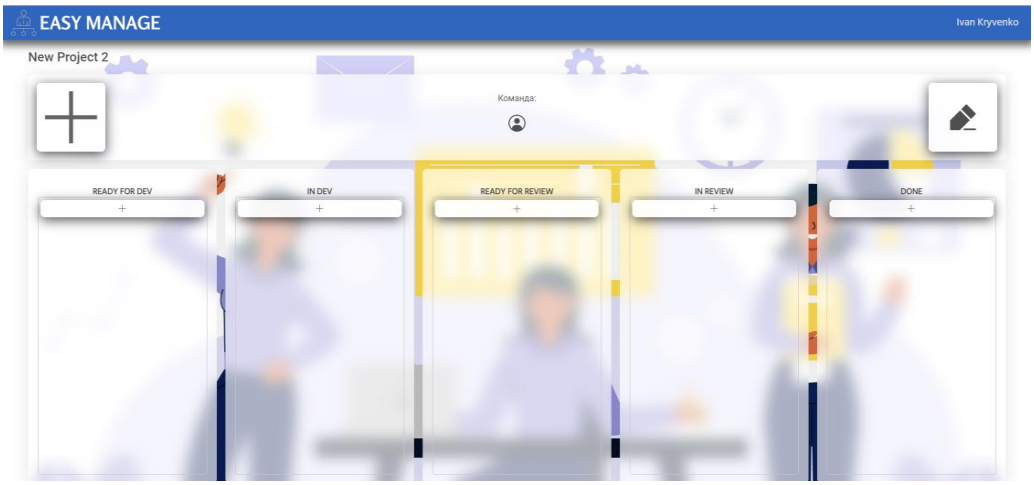

Рисунок 2.4 – Інтерфейс сторінки проєкту розроблений у «Figma»

Кожен елемент, що є доступний у розробленій системі веб-додатку, може бути налаштований індивідуально під потреби проекту. Розробник не має

необхідності витрачати час і ресурси на розробку окремого рішення для підключення соціальних мереж, модуля управління замовленнями, інструментів фінансової аналітики чи блоку авторизації користувача в веб-інтерфейсі. Крім того, UX/UI-компоненти є достатньо добре оптимізовані для використання у веб-програмах.

2.4 Висновок до розділу 2

У цьому розділі представлено основні компоненти, розроблено частину веб-інтерфейсу відповідно до актуальних вимог та стандартів і описано принципи роботи системи.

Формат обміну даними - JSON та шифрування даних за допомогою криптографії для забезпечення надійності системи. За допомогою Azure забезпечимо спрощення обчислення та оцінки складності завдань; за допомогою Angular framework забезпечимо модульність та масштабування проєкту; за допомогою Nest.js забезпечимо просту та ефективну серверну частину додатку та за допомогою Mongo DB забезпечимо збереження контенту та інформації користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ УПРАВЛІННЯ ЗНИЖКАМИ НА ТОВАРНІ ПОЗИЦІЇ

3.1 Обґрунтування вибору мови, фреймворку та середовища програмування для реалізації веб-додатку

Переваги та недоліки використання TypeScript на відміну від JavaScript наведені у таблиці 3.1.

Таблиця 3.1 - Переваги та недоліки TypeScript

Особливості	TypeScript	JavaScript
Можливість використання типізування даних та використання інтерфейсів	+	–
Для виконання написаного коду його необхідно компілювати	+	–
Сторонні бібліотеки є легкими у використанні	+	–
Важкість освоєння	+	–
Маленька спільнота розробників та користувачів	-	–
Можна стверджувати, що код буде безпечним під час написання	+	–

Фреймворки у веб-розробці служать основоположними структурами, які дають змогу розробникам оптимізувати створення програм. Ці потужні інструменти надають набір попередньо встановлених бібліотек і компонентів, пропонуючи структурований підхід до створення веб-рішень.

Angular розроблений і підтримуваний Google, є комплексним і багатофункціональним зовнішнім фреймворком, розробленим для створення динамічних і великомасштабних веб-додатків. Однією з його відмінних особливостей є використання TypeScript [16]. TypeScript - надмножина JavaScript, яка додає статичний тип та інші розширені функції для підвищення якості коду та зручності обслуговування. Angular дотримується архітектурного шаблону Model-View-Controller (MVC), організовуючи код у чітко визначені компоненти для ефективної розробки та обслуговування.

Архітектурний шаблон Model-View-Controller (MVC) — це парадигма проектування, яка широко використовується в розробці програмного забезпечення для організації коду та розділення проблем [17]. У MVC модель представляє дані програми та бізнес-логіку, представлення обробляє презентацію та користувальницький інтерфейс, а контролер діє як посередник, керуючи введенням даних користувача та відповідно оновлюючи як модель, так і представлення.

Angular властива можливість двостороннього прив’язування даних, що полегшує автоматичну синхронізацію між моделлю та представленням, зменшуючи потребу в ручних маніпуляціях DOM. Фреймворк також включає такі потужні функції, як впровадження залежностей, модульність через NgModule і надійний маршрутизатор для керування навігацією додатків. Інтерфейс командного рядка (CLI) Angular прискорює розробку, надаючи інструменти для створення каркасів, тестування та розгортання програм [18].

Що стосується тестування, Angular має вбудований набір тестів, який підтримує модульне тестування, наскрізне тестування та інтеграційне тестування. Jasmine і Protractor є популярними варіантами для тестування

додатків Angular, що дозволяє розробникам переконатися в надійності та правильності свого коду.

Екосистема Angular продовжує розвиватися, регулярно оновлюючи нові функції та вдосконалення, щоб задовольнити потреби сучасної веб-розробки, що постійно розвиваються.

React, розроблений Facebook, є ефективною бібліотекою JavaScript для створення інтерфейсів користувача. Основним елементом дизайну React є концепція віртуальної DOM, спрощеного представлення фактичної DOM, яка дозволяє здійснювати ефективні оновлення та покращувати продуктивність.

Одностороннє прив’язування даних React спрощує керування станом і підвищує передбачуваність програми.

Основним будівельним блоком додатків React є компонент — самодостатній, багаторазово використовуваний фрагмент коду, який інкапсулює як інтерфейс користувача, так і логіку.

Компоненти можна компонувати для створення складних користувальницьких інтерфейсів, сприяючи створенню модульної кодової бази, яку можна підтримувати. Використання React віртуальної DOM дозволяє оновлювати лише ті частини фактичної DOM, які змінилися, мінімізуючи вплив оновлень на продуктивність [19].

Порівняння бібліотеки React та фреймворку Angular наведені в таблиці 3.2.

Таблиця 3.2 – Порівняння React та Angular

Критерії	React	Angular
Вбудовані інструменти тестування	-	+
Модульність	+	+
Масштабованість	+	+
Двостороння прив’язка даних	-	+
Управління станом	-	+
Спільнота та підтримка	+	+

Приклад функционального коду селектора наведений нижче:

```
@Component({
  selector: 'app-select',
  templateUrl: './select.component.html',
  styleUrls: ['./select.component.scss']
})
export class SelectComponent implements ControlValueAccessor {
  selectedOption: string;
  @Input() options: Array<Option>;
  @Input() innerValue: Option;
  disabled = false;
  private onChanged: (id: string) => void;
  constructor(private eRef: ElementRef, private ref:
ChangeDetectorRef) {}
  @Input()
  @HostListener('document:click', ['$event'])
  clickOut(event: Event): void {
    if (!this.eRef.nativeElement.contains(event.target) &&
this.showList) {
      this.showList = false;
    }
  }
}
```

Другий файл представляє собою шаблон компонента, який містить в собі структуру та розмітку елементів інтерфейсу. Цей файл має формат .html, що вказує на гіпертекстову розмітку. Приведено нижче приклад коду структури та розмітки елементів інтерфейсу для компонента селекту.

```
<div class="select noselect">
  <div><button (click)="showOptions($event)" [disabled]="disabled">
      {{innerValue.name}}
  </button>
  </div>
  <div *ngIf="showList">
```

```
        <button (click)="selectOption(option.id)" *ngFor="let option of
options">
          {{option.name}}
        </button>
      </div>
    </div>
```

Третім файлом, що не є обов'язковим, але за допомогою якого можна зробити наш компонент красивішим та привабливішим для користувача, є файл зі стилями, який має розширення .css, .scss або .sass, він може виглядати, як звичайни файл зі стилями, але якщо використоувавати певні пре-процессори для технології CSS, то він може виглядати так, як на прикладі нижче:

```
    .list-of-options {
      width: 6px;
      border-radius: 10px;
    }

    .list-of-options-inner {
      border-radius: 20px;
    }
    p {
      display: flex;
      align-items: center;
      padding: 0.3rem;
      font-size: 1rem;
      &:hover {
        transform: scale(1.1);
      }
    }
  }
```

І на останок четвертий файл, що також є необов'язковим, а саме файл з тестами для розробленого компонента, він містить в собі код для unit тестів, які є хорошим рішенням для тестування всієї системи в окремих дрібних деталях.

3.2 Розробка компонентів для обробки API

Для ефективної функціональності додатку та системи в цылому було впроваджено значну кількість API. Вони відповідають за взаємодію системи з даними проектів, задач та інформацією користувача. Реалізація цих API виконана з використанням мови програмування TypeScript та фреймворку на основі Node.js – Express.js. Для взаємодії з клієнтом необхідно передавати додаткові параметри для використання реалізованих API та очікувати відповіді. Кожен запит також повинен супроводжуватися унікальним ключем, який ідентифікує конкретного користувача, що ініціює запит до API. Це забезпечує захист системи від невірного доступу до даних та гарантує безпеку від зломів та витоків інформації користувача. Нижче наведено список реалізованих в системі API та їх функціонал:

1. app.use("/api/users", cors(cors)) – отримати всіх користувачів системи;
2. app.use("/api/auth", cors(cors)) – реєстарція та вхід в обліковий запис;
3. app.use("/api/user-change-info", cors(corsOptions)) – зміна даних користувача;
4. app.use("/api/addUser", cors(corsOptions)) – додати користувача до проєкту;
5. app.use("/api/deleteUser ", cors(corsOptions)) – видалити користувача з проєкту;
6. app.use("/api/createProgect ", cors(corsOptions)) – створити новий проєкт;
7. app.use("/api/editProject ", cors(corsOptions)) –редагувати дані проєкту;
8. app.use("/api/deleteProject ", cors(corsOptions)) – видалити проєкт;
9. app.use("/api/createTask ", cors(corsOptions)) – створити завдання;
10. app.use("/api/editTask ", cors(corsOptions)) – редагувати завдання;
11. app.use("/api/assignUser ", cors(corsOptions)) – призначити завдання користувачеві;
12. app.use("/api/evaluate ", cors()) – оцінити складність завдання;
13. app.use("/api/deleteTask ", cors(corsOptions)) – видалити завдання;

14. app.use("api/changeStatus", cors()) – змінити статус завдання.

Доступ до наведених вище API мають сервіси, які знаходяться в списку дозволених користувачів, що доволі сильно підвищує безпеку даних, адже за його допомого, доступ до розробленого API мають лише користувачі з цього списку та перешкоджає запитам з інших ресурсів на поточні API. Наразі список адресів виглядає наступним чином:

```
const whitelist = [
  "https://easyManage.herokuapp.com",
  "https://easyManage-backend.herokuapp.com",
];
```

Захищений протокол передачі гіпертексту (HTTPS) — HTTPS встановлює зв’язок між браузером і веб-сервером. Він використовує протокол Secure Socket Layer (SSL) і Transport Layer Security (TLS) для встановлення зв’язку. Нова версія SSL – TLS (Transport Layer Security).

HTTPS використовує звичайний протокол HTTP та додає поверх нього рівень SSL/TLS. Робочий процес HTTP та HTTPS залишається незмінним, браузери та сервери все ще спілкуються один з одним за допомогою протоколу HTTP. Однак це робиться через безпечне з’єднання SSL. З’єднання SSL відповідає за шифрування та дешифрування даних, якими обмінюються, щоб забезпечити безпеку даних.

Рівень захищених сокетів (SSL)

Основна відповідальність SSL полягає в забезпеченні безпеки та надійності передачі даних між системами зв’язку. Це стандартна технологія безпеки, яка використовується для шифрування та дешифрування даних під час передачі запитів.

Як обговорювалося раніше, HTTPS – це, по суті, той самий старий HTTP, але з SSL. Для встановлення захищеного зв’язку між пристроями, що

спілкуються, SSL використовує цифровий сертифікат під назвою SSL-сертифікат. [20].

HTTP передає дані в гіпертекстовому форматі між браузером і веб-сервером, тоді як HTTPS передає дані в зашифрованому форматі. Як наслідок, HTTPS захищає веб-сайти від трансляції їхньої інформації таким чином, щоб будь-хто, хто підслуховує мережу, міг легко побачити. Під час передачі між браузером і веб-сервером HTTPS захищає дані від доступу та зміни хакерами. Навіть якщо передачу буде перехоплено, хакери не зможуть її використати, оскільки повідомлення зашифроване:

```
    getUserInfoById(id: string, token: string): Observable<UserInfo> {
       let httpHeaders = new HttpHeaders({'Content-Type':
'application/text', 'authorization-token': someToken});
       return this.http.get<User>(`${GET_USER_BY_ID}/${user_id}`, {
         headers,
       });
    }
```

Для розробки вище описаного API було використано різні засоби, які призначені для того, щоб підвищити надійність API та покращити роботу із ним.

3.3 Розробка веб-інтерфейсу додатку

Оскільки користувачі потребують більш надійних і чутливих додатків на стороні клієнта, розробники відповідають дивовижними бібліотеками JavaScript. Але в міру того, як програми стають складнішими, код на стороні клієнта починає виглядати дуже засміченим з купою бібліотек, набитими разом у невпорядкованій купі зв’язків подій, викликів jQuery Ajax і функцій аналізу JSON.

Нам потрібно розробляти наші програми на стороні клієнта з тим самим підходом, який ми роками використовували для коду на стороні сервера. Нам потрібен каркас. Завдяки надійній структурі JavaScript ми можемо підтримувати

код упорядкованим, зменшувати дублювання та використовувати стандарт кодування, зрозумілий іншим розробникам.

Потрібно створити програму для підтвердження концепції (PoC), щоб зробити наш інтерфейс керування продуктом набагато оперативнішим. Поточний інтерфейс повільний і вимагає повного оновлення сторінки щоразу, коли користувач додає або змінює інформацію про продукт. Начальник хоче, щоб ми створили односторінковий інтерфейс, і ми повинні зробити все це, не змінюючи наш серверний додаток. Нам потрібен фреймворк, який дозволить нам швидко розробляти програму, зберігаючи все в порядку, щоб ми могли підтримувати його протягом тривалого часу.

У нас є вибір фреймворків JavaScript, які допоможуть вирішити цю проблему. Для цього випадку ми вибираємо Angular, оскільки його архітектура в стилі Model-View-Controller (MVC) проводить паралелі з деякими поширеними веб-фреймворками, такими як Ruby on Rails, ASP.NET MVC і Spring MVC для Java. Шаблон MVC дозволяє нам організувати наш код загальним способом, який запобігає змішуванню таких речей, як логіка відображення, з об’єктами домену. На додаток до шаблону архітектури MVC, підтримка Angular двостороннього зв’язування даних, шаблонів HTML, глибоких посилань і ін’єкцій залежностей робить його чудовим підходом для нашого PoC.

Angular використовує філософію, відмінну від інших фреймворків JavaScript. Інші обходять проблему того, що HTML не розроблено для динамічного перегляду, тоді як Angular вирішує цю проблему напряму, розширюючи HTML.

Такі фреймворки, як jQuery, зосереджені на маніпулюванні DOM і націлюванні елементів за допомогою селекторів CSS. Angular використовує властивості, додані до HTML-елементів, для контролю та маніпулювання переглядом. Давайте розглянемо кілька понять і термінів, щоб замочити ноги:

Передача стану представлення (REST) — це стиль архітектури програмного забезпечення, який зазвичай використовується в Інтернеті як засіб зв’язку між кількома системами.

Resource — це вбудований спосіб зв’язку Angular із веб-сервісами RESTful.

Router — це модуль Angular, який дозволяє нам вказувати маршрути та те, як наш додаток на них реагуватиме.

Види доповнюють послугу маршруту. Вони дозволяють нам створювати фрагменти HTML — часто звані шаблонами — які ми можемо замінити на своїй сторінці.

Директиви — це маркери на елементах HTML, які вказують Angular прикріпити поведінку до цих елементів та їхніх дітей, якщо такі є. Angular поставляється з набором вбудованих директив, таких як ng-view і ng-app. Ми можемо створювати власні директиви, якщо вбудовані не відповідають нашим потребам.

Області — це об’єкти в моделі програми, які імітують структуру DOM програми. Області впорядковано ієрархічно, і кожна програма має одну кореневу область. Кореневу область можна використовувати як механізм для надання шаблону шини повідомлень публікації та підписки.

Веб-інтерфейс системи написаний самостійно з використання таких бібліотек та інструментів за допомогою яких можна підвищити ефективність написання розмітки а також зв’язування даних з самою мовою гіпертекстової розмітки та каскадних стилів. Даний підхід, що використовується для розробки клієнтської та графічної частини системи надає найбільші можливості для масштабування та пере використання компонентів системи.

Структура компоненту складається з декількох файлів. Перший файл це скрипт компоненту, він реалізує всі операції над даними що має виконувати даний компонент та підключає в собі інші файли компонента. Даний файл має характерний формат .ts так як мовою для розробки скриптів на Angular є TypeScript.

Інтерфейс системи підтримує всі можливості що були закладенні на початку в систему і має наступний вигляд. На рисунку 3.1 зображено компонент реєстрації та логіна для користувачів системи.

На рисунку 3.2 зображено компонент з посиланнями на сторінки системи для авторизованого та неавторизованого користувача.

Для забезпечення взаємодії між компонентами інтерфейсу та передачі даних між базою даних і шаблонами сайту було використано додаткові інструменти, які надає фреймворк Angular. Зокрема, застосовувалися двостороння та одностороння прив'язка даних, структурна директива *ngFor для відображення великої кількості однакових елементів, *ngIf для відображення компонента за умовою, ng-template для динамічної заміни компонентів, та ng-routerOutlet для заміни компонентів відповідно до поточного маршруту системи [21].

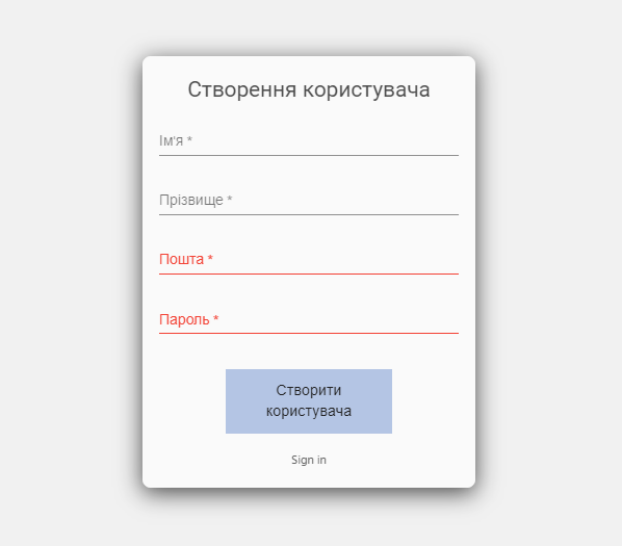

Рисунок 3.1 - Компоненти реєстрації та логіна користувача

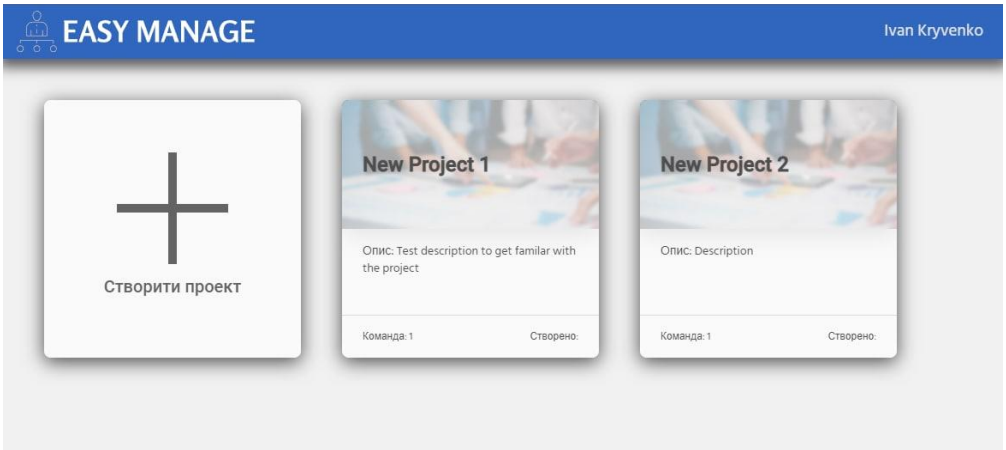

Рисунок 3.2 - Компонент з посиланнями на проекти, які доступні користувачеві системи

На рисунку 3.3 зображено інтерфейс сторінки із списком проєктів користувача, а також, на рисунку 3.4 зображено сторінку проекту з стовпцями завдань.

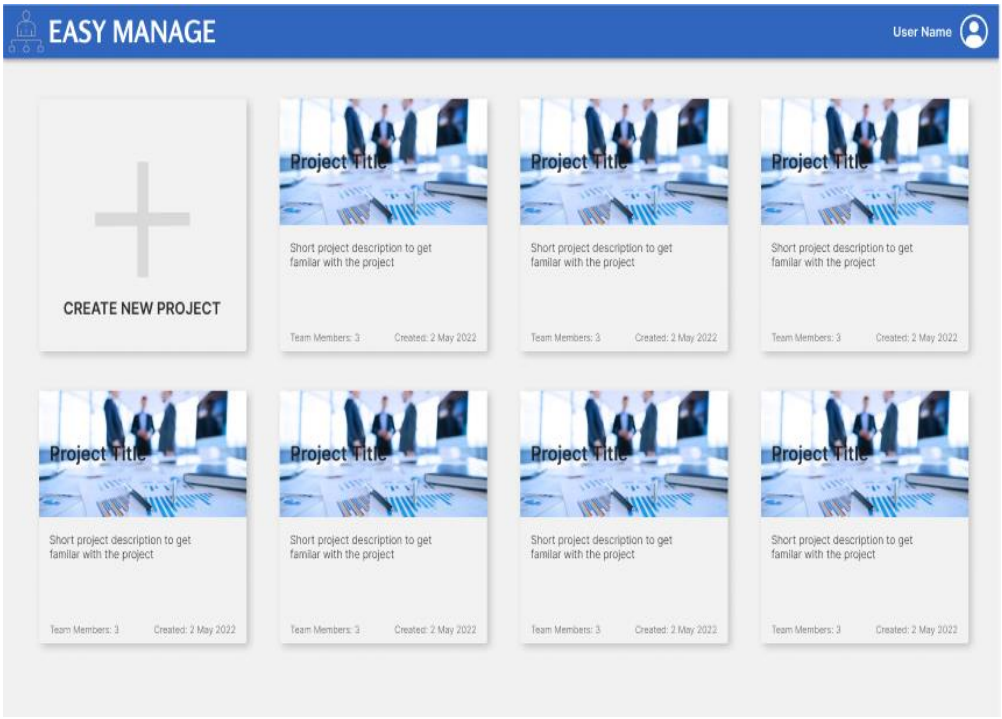

Рисунок 3.3 - Інтерфейс сторінки із списком каналів користувача

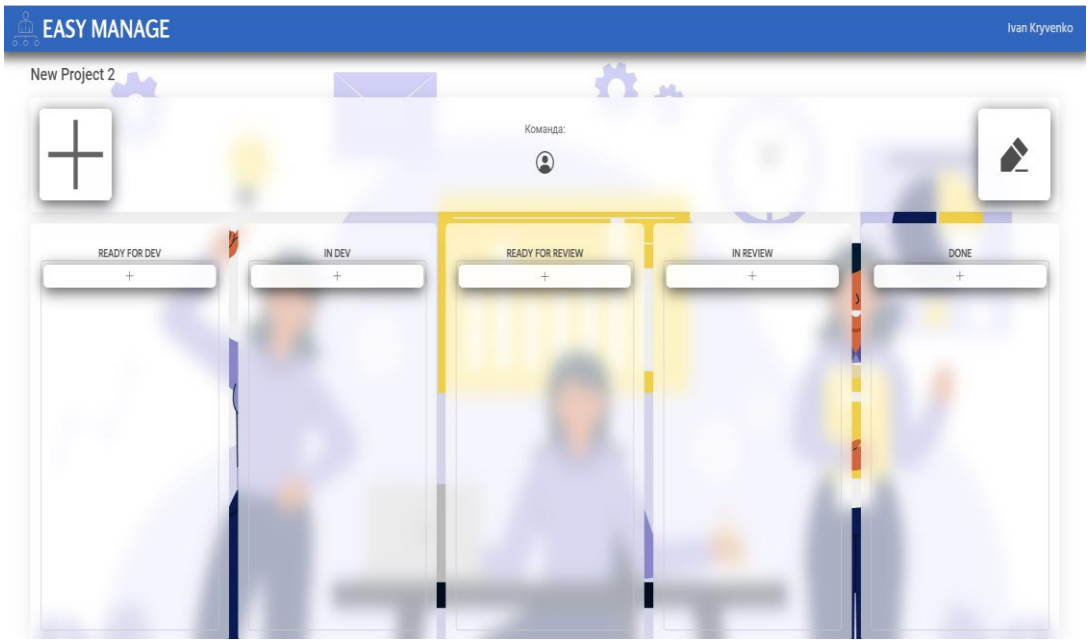

Рисунок 3.4 - Інтерфейс сторінки проекту

На рисунку 3.5 зображено інтерфейс сторінки з налаштуваннями системи користувача.

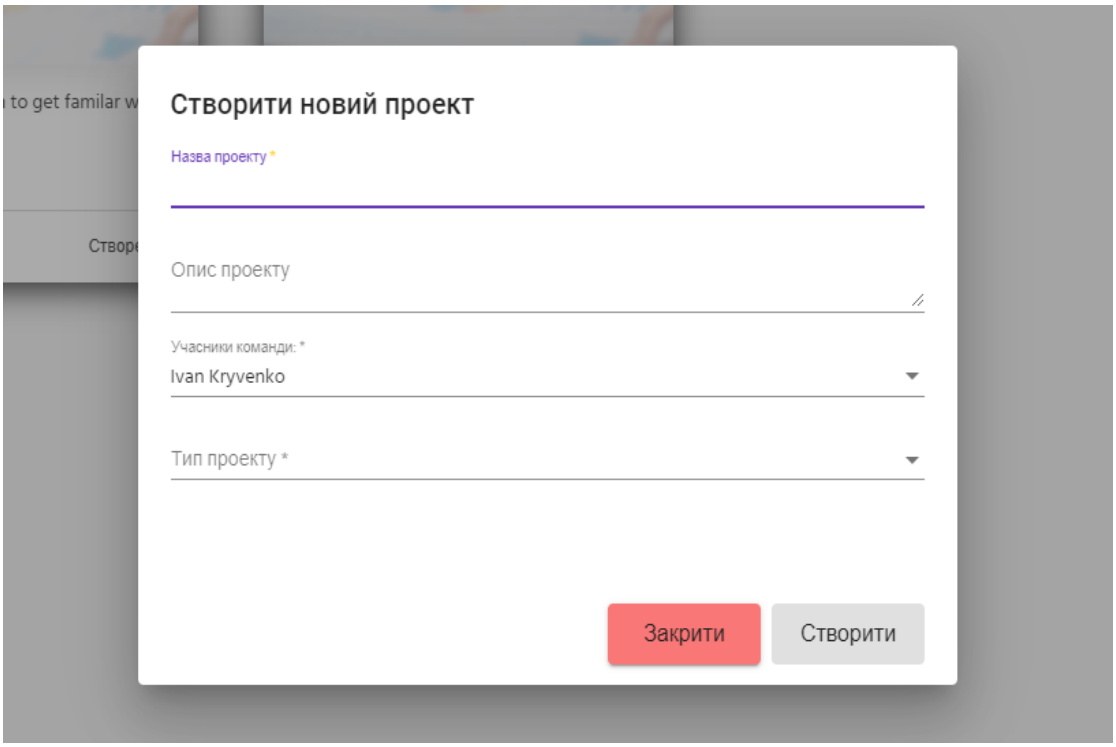

Рисунок 3.5 – Інтерфейс модального вікна створення проєкту

3.4 Реалізація функціоналу аналізу даних на Azure в зв'язці з додатком для управління проектами

Azure Functions — це безсерверний обчислювальний сервіс, наданий Microsoft Azure, який дозволяє розробникам створювати та розгортати програми, керовані подіями, без необхідності керувати інфраструктурою. Завдяки функціям Azure розробники можуть зосередитися виключно на написанні коду для обробки конкретних подій, залишаючи базову інфраструктуру та аспекти масштабування платформі Azure. Це забезпечує швидшу розробку, зниження операційних витрат і покращену масштабованість.

Ключові особливості функцій Azure

Обчислення, керовані подіями: функції Azure запускаються різними подіями, такими як HTTP-запити, таймери, повідомлення в чергах, завантаження

файлів або зміни даних у базах даних. Цей підхід, керований подіями, дозволяє розробникам реагувати на певні події в реальному часі без необхідності підтримувати постійні екземпляри сервера.

Безсерверна архітектура: Azure Functions дотримується парадигми безсерверних обчислень, тобто немає серверів для підготовки або керування. Платформа автоматично обробляє масштабування інфраструктури, забезпечуючи розподіл ресурсів на основі фактичного попиту, що робить її високоефективною.

Підтримка мов і платформ: Azure Functions підтримує кілька мов програмування, зокрема C#, JavaScript, Java, Python і TypeScript. Ця гнучкість дозволяє розробникам працювати з улюбленою мовою та безперешкодно використовувати існуючі кодові бази.

Ціноутворення за принципом оплати: функції Azure пропонують платіжну модель за принципом оплати за використання. Ви платите лише за обчислювальні ресурси, використані під час виконання ваших функцій. Цей економічно ефективний підхід гарантує, що з вас стягується плата лише за фактичне використання без будь-яких попередніх зобов’язань.

Інтеграція зі службами Azure: функції Azure легко інтегруються з різними службами Azure, такими як Azure Storage, Azure Cosmos DB, Azure Service Bus, Azure Event Grid тощо. Ця інтеграція дозволяє розробникам створювати складні програми, поєднуючи потужність різних служб.

Переваги використання функцій Azure

Швидший час виходу на ринок: завдяки можливості зосередитися виключно на бізнес-логіці та обробці подій розробники можуть швидко створювати та розгортати програми, скорочуючи час розробки та прискорюючи час виходу на ринок.

Економія коштів. Оскільки функції Azure автоматично масштабуються залежно від попиту, немає потреби платити за неактивні ресурси сервера, що призводить до економії коштів, особливо для програм із різним навантаженням.

Масштабованість і еластичність: Azure Functions автоматично масштабується для обробки великої кількості подій одночасно. Ця еластичність гарантує, що ваші програми зможуть обробляти великий трафік і раптові стрибки без будь-якого зниження продуктивності.

Безсерверне керування: Microsoft Azure керує всім керуванням сервером, включаючи оновлення, виправлення безпеки та масштабування, звільняючи розробників від операційних витрат і дозволяючи їм зосередитися на розробці коду.

Архітектура, керована подіями: керована подіями природа функцій Azure дає змогу відокремлювати компоненти програми, що полегшує створення масштабованих і стійких архітектур мікросервісів.

Типи функцій Azure:

Тригер HTTP: цей тип функції запускається запитом HTTP. Його можна використовувати для створення веб-інтерфейсів API і обробки взаємодій на основі HTTP.

Тригер таймера: тригер таймера виконує функцію за попередньо визначеним розкладом або через регулярні проміжки часу. Це корисно для виконання таких завдань, як синхронізація даних, періодична обробка або створення звітів.

Тригер Blob: цей тип функції запускається, коли новий або оновлений blob додається до контейнера Azure Storage. Це дає змогу автоматизувати процеси на основі змін у blob-об’єктах зберігання.

Тригер черги: тригер черги запускається, коли повідомлення додається до черги сховища Azure. Він забезпечує спосіб обробки повідомлень в архітектурі на основі черги, дозволяючи створювати програми, керовані подіями.

Тригер сітки подій: тригер сітки подій викликається, коли подія публікується в темі або домені сітки подій Azure. Він забезпечує реактивну обробку подій і може використовуватися для побудови керованих подіями архітектур.

Тригер Cosmos DB: цей тип функції запускається, коли в документі в контейнері Azure Cosmos DB вносяться зміни. Це дозволяє створювати сценарії обробки та синхронізації даних у реальному часі.

Тригер службової шини: тригер службової шини викликається, коли нове повідомлення надходить у чергу або підписку на тему Azure Service Bus. Він підходить для створення роз'єднаних систем на основі обміну повідомленнями.

Тригер концентратора подій: тригер концентратора подій викликається, коли нові події публікуються в концентраторі подій Azure. Він забезпечує сценарії прийому та обробки подій із високою пропускною здатністю.

Давайте зануримося в реальний приклад із використанням функції HTTP Trigger у функціях Azure:

Припустімо, що у нас є дані в озері даних Azure 2-го покоління (ADLS 2-го покоління), прочитайте дані з функції Azure і запишіть назад в обліковий запис зберігання ADLS 2-го покоління.

Щоб виконати завдання читання даних з Azure Data Lake Gen2 (ADLS Gen2) за допомогою функції Azure, а потім записати їх назад до того самого облікового запису зберігання, ми будемо виконувати двоетапний підхід h: По-перше, ми створимо функцію Azure у хмарі Azure, а по-друге, ми отримаємо доступ і розгорнемо код за допомогою Visual Studio Code (VS Code) із вашого локального середовища розробки.

Durable функція є частиною azure функції. Це розширення azure функції з відкритим кодом. Використовуючи durable функції, ми можемо реалізувати функції зі збереженням стану в безсерверному середовищі. Розширення керує станом, контрольними точками та перезапусками.

Є 3 основні компоненти durabale функції:

1. Стартер функція.

2. Функція оркестратор.

3. Функція діяльності.

Під його назвою durable функції ми чітко розуміємо одну з їх цілей, яка полягає в обробці тривалих служб без тайм-ауту. Архітектура Durable функцій зображена на рисунку 3.6 [22]:

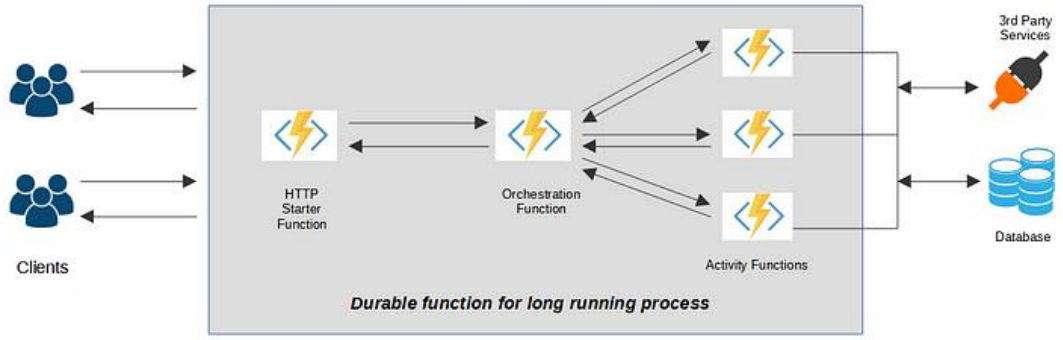

Рисунок 3.6 – Архітектура Durable функцій для тривалого API

Функція HTTP Starter: HTTP запускає durable function для виклику функції оркестратора та зв’язування запиту з нею. Ми можемо вказати динамічний маршрут і HTTP-методи в конфігурації функції.

Функція оркестратор: дозволяє керувати виконанням інших довгострокових функцій у функціональному додатку. Ця функція визначає робочий процес із збереженням стану в коді та викликає функції активності. Функція Orchestrator має бути детермінованою, оскільки код виконуватиметься знову і знову, доки функція активності не завершиться.

Функція активності: функції активності є незалежною функцією без стану для однієї задачі, яка не має інформації про більші робочі процеси. Ця функція викликається функцією оркестратором, виконує завдання та, можливо, повертає результат. Функції активності можна викликати паралельно та/або в ланцюжку.

Наскільки ми знаємо, є деякі запити API, які потребують часу через важкі операції у серверних або сторонніх службах. Тому нам доводиться довго чекати на стороні клієнта, і виникає проблема з тайм-аутом. Схема, що зображена на рисунку 3.7, допоможе нам зрозуміти, як це працює.

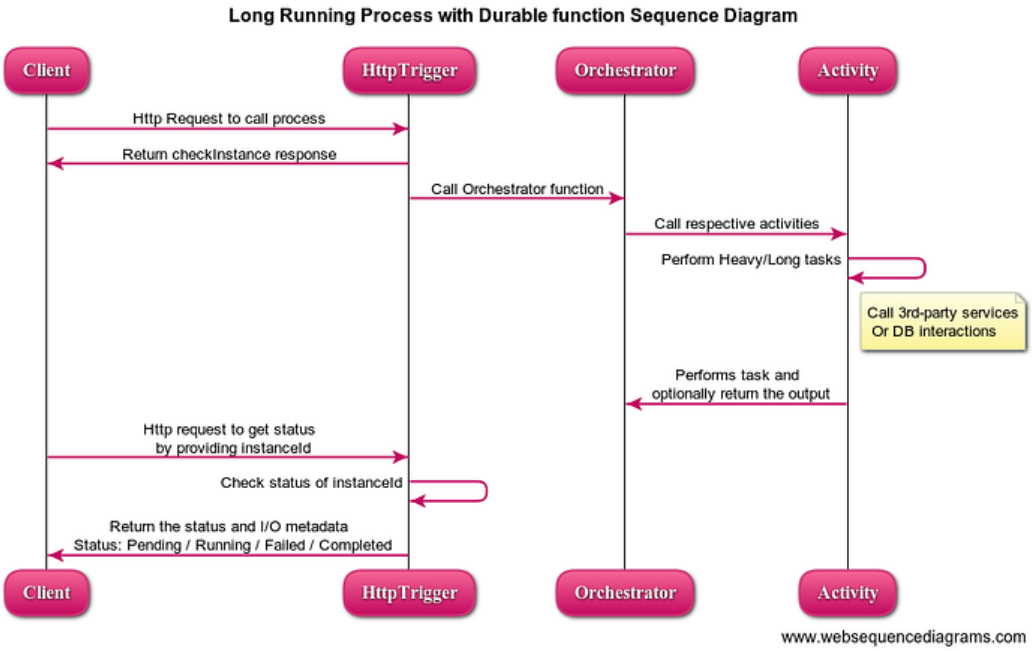

Рисунок 3.7 – Часова діаграма роботи durable функції

Для початку роботи з Azure функціями необхідно мати обліковий запис в Microsoft Azure, встановлену Node.js на локальному комп'ютері для того, аби працювати з нею локально та мати можливість тестувати написаний функціонал. Також дуже зручний інтерфейс для роботи з Azure надає Visual Studio Code з розширенням Azure. Тому після виконання необхідних кроків, можна приступити до створення функції.

Для початку, після того, як ми встановили розширення в VS Code, нам необхідно створити нову Azure функцію, що зображено на рисунку 3.8. Після того, як ми натиснемо створи нову функції, буде необхідно обрати бажану папку де буде розміщувати функція, мову програмування, в нашому випадку JavaScript, модель, в нашому випадку – остання четверта, темплейт для функції – HTTP Trigger, а також ім'я, яке нам найбільше підходить [22].

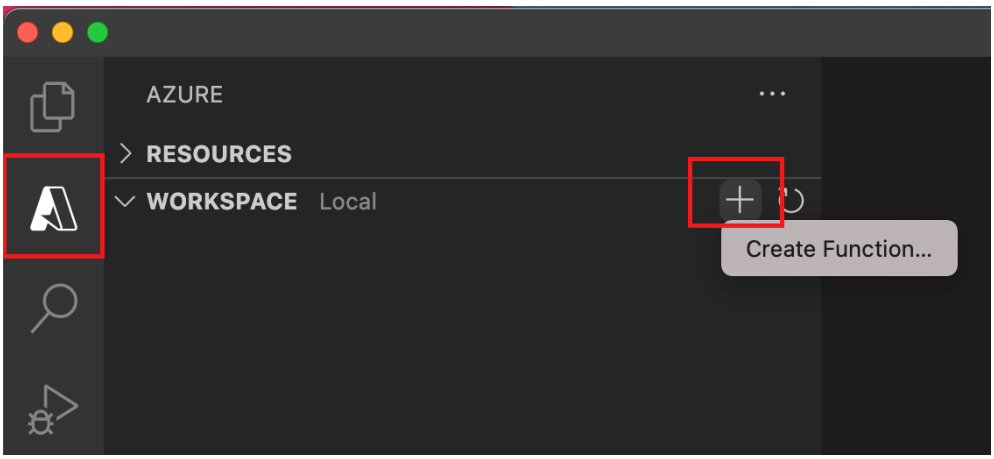

Рисунок 3.8 – Створення нової Azure функції

При створенні нової Azure функції в репозиторії буде створено такі файли, як *host.json* – файл, що містить в собі глобальні конфігурації для функціонального додатку, та буде виглядати приблизно так, в залежності від версії моделі:

```
{
  "version": "2.0",
  "logging": {
    "applicationInsights": {
      "samplingSettings": {
        "isEnabled": true,
        "excludedTypes": "Request"
      }
    }
  }
}
```

analizeTask/function.json – файл, що містить в собі конфігурацію для специфічної функції, в нашому випадку – *analyzeTask.*

```
{
  "bindings": [
    {
      "authLevel": "function",
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": ["get", "post"]
    },
    {
      "type": "http",
      "direction": "out",
```

```
      "name": "res"
    }
  ]
}
```

localSettings.json – файл, що містить в собі налаштування користувача для локальної розробки, а також інструменти для підключення до сервісів Azure:

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "node"
  }
}
```

package.json – файл, що містить метадані про проект:

```
{
  "name": "your-function-app-name",
  "version": "1.0.0",
  "main": "index.js",
  "dependencies": {
    "azure-functions": "^1.0.3"
  }
}
```

Для реалізації функції надання рекомендацій необхідно краще ознайомитись з таким терміном, як нечітка логіка, адже саме завдяки цьому ми будемо надавати рекомендації щодо найбільш підходячого працівника.

Нечітка логіка є узагальненням класичної логіки на випадок, коли істинність розглядається як лінгвістична змінна, що приймає значення типу: "дуже істинно", "більш-менш істинно", "не дуже хибно" і т.п. Зазначені лінгвістичні значення представляються нечіткими множинами. Основна відмінність від класичної логіки полягає в тому, що замість значень “ Істина “ і “ Хибність “ в нечіткій логіці використовується ступінь істинності, що приймає значення з нескінченної множини від 0 (Хибність) до 1 (Істина) включно. Отже логічні операції вже не можуть бути представленими таблицями істинності. У нечіткій логіці вони задаються функціями і лише в крайніх випадків - коли значення змінних виключно 1 або 0 - згадані вище функції дають таблиці істинності операцій класичної логіки.

Важливим етапом визначення показників для вибору працівника є формування експертної системи у вигляді бази нечітких знань, яка містить механізм нечіткого-логічного висновку, яка являє собою систему логічного виведення, яка базується на алгоритмі отримання нечітких висновків на основі нечітких передумов з використанням понять нечіткої логіки. Процес нечіткого виведення поєднує в собі всі основні концепції теорії нечітких множин: функції належності, лінгвістичні змінні, нечіткі логічні операції, методи нечіткої імплікації і нечітку композиції. Системи нечіткого виведення дозволяють вирішувати завдання автоматичного керування, класифікації даних, розпізнавання образів, прийняття рішень, машинного навчання та багато інших.

Експертну інформацію про показники надання рекомендацій про гітари буде представлено у виді нечіткої продукційної моделі подання знань у вигляді «ЯКЩО – ТО», які пов'язані з вхідними змінними $x_1 – x_w$ з одним з типів рішень:

$$\text{ЯКЩО } (x_1 = a_1^{11})\text{I}(x_2 = a_2^{11})\text{I} \dots \text{I}(x_w = a_w^{11})\text{АБО}(x_1 = a_1^{12})\text{I}(x_2 = a_2^{12})\text{I} \dots \text{I}$$
$$(x_w = a_w^{12})\text{АБО} \dots \left(x_1 = a_1^{1\delta_1}\right)\text{I}\left(x_2 = a_2^{1\delta_1}\right)\text{I} \dots \text{I}\left(x_w = a_w^{1\delta_1}\right)$$
$$\text{ТО } y = d_1 \text{ і тд.},$$

де x – вхідна змінна, а з коефіцієнтами – лінгвістична оцінка вхідної змінної,

δ_q – кількість правил, що визначають значення змінної y.

У якості термінів-множин вхідних змінних будуть використовуватися значення, що відповідають показникам «низька», «середня», «висока». Так отримані дані про важливість параметрів будуть переведені у ці показники, після чого буде надаватись рекомендація яка гітара підходить користувачу.

Отже, беручи до уваги недосконаліть існуючих методів визначення показників для рекомендації працівника викликало необхідність створення нечіткої моделі для оцінювання даного показника на основі результатів експертного оцінювання. Теорія нечітких множин дозоляє описувати неточні

поняття, якісні і наші знання про навколишній світ, а також оперувати цими знаннями, щоб отримувати нову інформацію. А наявність математичних засобів для відображення нечіткості вихідної інформації дозволяє побудувати модель яка є адекватною до реальності.

Для побудови функцій належності та визначення бази знань експертної системи вибору спорту розглянемо основні характеристики, на які звертають увагу експерти при оцінці гітар.

Від достовірності залежить те чи будуть здійснюватись рекомендації правильно та точно.

Перехід від базової шкали до терм-множини лінгвістичної змінної відбувається в контексті нечіткої логіки і теорії нечітких множин. Цей перехід дозволяє виразити лінгвістичні концепти або змінні, такі як "високий", "середній", "низький", у вигляді нечітких множин. Базова шкала представляє собою простий числовий діапазон або множину значень, які відповідають лінгвістичній змінній. Наприклад, якщо розглядається лінгвістична змінна "температура" і базова шкала визначена в діапазоні від 0 до 100, це означає, що значення температури можуть бути виражені числами від 0 до 100 [14].

Якщо користувач виконає більше 75% різких змін у своїх твердженнях у комплексних питаннях, то достовірність – критична, якщо користувач виконує від 27% до 75% ірізких змін у твердженннях – нестандартна, і якщо відповіді користувача стабільна, тобто менше 27%, то достовірність – у межах норми.

analizeTask/index.js – сам файл з функцією, який буде виконуватись при виклику функції користувачем:

```
module.exports = async function (context, req) {
    try {
        context.log('JavaScript HTTP trigger function processed a 
request.');
        const { task, users } = req.body;

        // Validate input
```

```
            if (!task || !users || !Array.isArray(users) ||
users.length === 0) {
                context.res = {
                    status: 400,
                    body: "Invalid input. Please provide a valid task
and a non-empty list of users."
                };
                return;
            }
            const taskDifficulty = evaluateTaskDifficulty(task);
            const points = assignPoints(taskDifficulty);
            const recommendedUser = recommendUser(users);

            // Prepare the response
            const response = {
                points,
                recommendedUser
            };

            // Return the response
            context.res = {
                status: 200,
                body: response
            };
        } catch (error) {
            context.log.error(error);
            context.res = {
                status: 500,
                body: "Internal Server Error"
            };
        }
    };
    logic)
    function evaluateTaskDifficulty(task) {
        const { description, technologies } = task;
        const difficulty = description.length + technologies.length;
        return difficulty;
    }

    function assignPoints(difficulty) {

        const pointScale = [0, 1, 2, 3, 5, 8, 13, 21];

        if (difficulty < 10) {
            return pointScale[0];
        } else if (difficulty < 20) {
            return pointScale[1];
        } else if (difficulty < 30) {
            return pointScale[2];
        } else if (difficulty < 40) {
            return pointScale[3];
```

```
    } else if (difficulty < 50) {
        return pointScale[4];
    } else if (difficulty < 60) {
        return pointScale[5];
    } else if (difficulty < 70) {
        return pointScale[6];
    } else {
        return pointScale[7];
    }
}
function recommendUser(users) {
    const availableUsers = users.filter(user => user.taskLoad < 5);
    availableUsers.sort((a, b) => b.experience - a.experience);
    return availableUsers[0];
}
```

3.5 Тестування веб-додатку для управління проєктами з використанням технології Azure

Тестування програмного забезпечення — це процес оцінки функціональності програмного додатку з метою виявити, чи відповідає розроблене програмне забезпечення визначеним вимогам чи ні, і виявити дефекти, щоб переконатися, що продукт без дефектів, щоб виробляти якісний продукт.

В свою чергу баг (проблема) – це неспівпадіння між тим, що саме було очікувано і що отримано. На даний момент часу, є три йбільш популярні та ефективні методи тестування, а саме: Unit тести, інтеграційні тести, E2E тести.

Unit-тестування (модульне тестування) – процес розробки програмного забезпечення, в якому найменші тестовані частини програми, звані одиницями, індивідуально перевіряються на належне функціонування.

Інтеграційне тестування – тип тестування програмного забезпечення, коли компоненти програмного забезпечення поступово інтегруються, а потім тестуються як єдина група.

End-to-end тестування є захоплюючою технікою, оскільки вона часто використовується в гнучких і багатьох інших методологіях тестування, але все

одно її нелегко зрозуміти. Це як модульне та функціональне тестування, але тести виходять за межі окремих блоків [23].

Jest - це популярний фреймворк для тестування JavaScript-коду, спроєктований для простоти використання та швидкості виконання тестів. Він розроблений Facebook і часто використовується для тестування коду Angular, хоча його можна використовувати для будь-якого проєкту, що використовує JavaScript.

Основні особливості Jest:

- Легка налаштуваність – Jest надає стандартні налаштування для багатьох проєктів. Ви можете почати використовувати його вже заздалегідь встановленим, і він буде працювати із коробки. Крім того, він підтримує конфігурацію через файл jest.config.js.

Автоматичне визначення тестів – Jest автоматично знаходить тести у вашому проєкті. Це означає, що вам не потрібно додавати жодних списків тестів чи налаштовувати додаткові параметри для їх знаходження.

Snapshot тести – Snapshot-тести дозволяють записати стан вашого об'єкта або компоненту та перевірити, чи не змінився він з часом. Це особливо корисно для тестування React-компонентів та візуальних елементів.

Mock-функції – Jest дозволяє легко створювати та використовувати mock-функції, що дозволяє замінювати реальні функції у тестах та визначати їх поведінку.

Паралельне виконання тестів – Jest може виконувати тести паралельно, що прискорює час виконання для проєктів з великою базою тестів.

Покриття коду – Вбудований засіб для аналізу покриття коду дозволяє вам визначити, яка частина вашого коду була протестована, а яка ні.

Широкі можливості конфігурації – Jest можна легко налаштувати залежно від потреб вашого проєкту, дозволяючи вам включати або виключати різноманітні функції та розширювати його базовий функціонал.

Jest надає все, що потрібно для тестування JavaScript-коду: від простих unit-тестів до складних тестів інтеграції та енд-тестування.

Допустимо, є асинхронна функція, яка виконує запит до сервера і повертає дані. Наприклад, файли api.js та api.test.js:

```
function fetchData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({ data: 'Hello, Jest!' });
    }, 1000);
  });
}
module.exports = fetchData;


// api.test.js
const fetchData = require('./api');

test('повертає дані асинхронно', async () => {
  const result = await fetchData();
  expect(result.data).toBe('Hello, Jest!');
});

test('повертає дані асинхронно з використанням .resolves', () => {
  return expect(fetchData()).resolves.toEqual({ data: 'Hello, Jest!'
});
});

test('помилка при невалідних даних', async () => {
  await expect(Promise.reject('Помилка')).rejects.toBe('Помилка');
});
```

У цьому прикладі ми використовуємо ключове слово async для оголошення асинхронного тесту. Також Jest надає можливість використовувати асинхронні функції, щоб уникнути callback-базового коду.

Тести використовують функцію fetchData, яка повертає обіцянку (Promise) імітуючи асинхронний запит. Використовуючи ключове слово await, ми чекаємо, коли обіцянка буде виконана, і потім перевіряємо результат за допомогою expect.

Також Jest надає спеціальні функції, такі як resolves та rejects, які полегшують тестування асинхронних операцій.

Якщо потрібно створити mock функцію чи щось інше для тесту це можна легко створити за допомогою можливостей RTL як наведено на рисунку 3.9.

```
// declare which API requests to mock
const server = setupServer(
  // capture "GET /greeting" requests
  rest.get('/greeting', (req, res, ctx) => {
    // respond using a mocked JSON body
    return res(ctx.json({greeting: 'hello there'}))
  }),
)

// establish API mocking before all tests
beforeAll(() => server.listen())
// reset any request handlers that are declared as a part of our tests
// (i.e. for testing one-time error scenarios)
afterEach(() => server.resetHandlers())
// clean up once the tests are done
afterAll(() => server.close())

// ...

test('handles server error', async () => {
  server.use(
    // override the initial "GET /greeting" request handler
    // to return a 500 Server Error
    rest.get('/greeting', (req, res, ctx) => {
      return res(ctx.status(500))
    }),
  )

  // ...
})
```

Рисунок 3.9 - Використання mock функцій для тесту

Для орзроблюваного проекту було створено unit-test, які призначенні для тестування окремих найменших функціональностей додатку окремо, в незалежності одна від одної, в штучно створеному середовищі:

```
import { ComponentFixture } from '@ testing';

describe('AppComponent', (component) => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [AppComponent],
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

  it(`should have as title 'demo-angular-jest'`, () => {
    expect(component.title).toEqual('demo-angular-jest');
  });
});
```

Для охоплення всіх компонентів веб-додатку було створено сотні тестів. Ці тести охоплюють всі необхідні модулі функціонування веб-додатку, взаємодію з сервером, вірне відображення даних і т.д. Результати тестування наведено на рисунку 3.10.

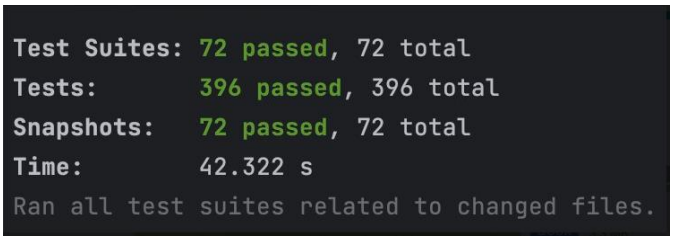

Рисунок 3.10 – Успішне проходження всіх тестів веб-додатку, та перевірки на правильне відображення необхідних даних

3.6 Висновок до розділу 3

У цьому розділі підготовлено середовище розробки системи, розроблено компоненти для обробки API, розроблено адаптивний веб-інтерфейс відповідно до актуальних вимог та стандартів, спроектовано мікро сервісну архітектуру системи. Було визначено важливість вибору мови та середовища програмування, таких як JavaScript і TypeScript, для розробки сучасного веб-додатку. Були розглянуті ключові компоненти реалізації, такі як використання Angular, Express.js та Azure, взаємодія з сервером через RESTful API та використання бази даних. Створено сучасний та зручний користувацький інтерфейс, а також визначені основні компоненти для зручної роботи користувачу з додатком.

Процес тестування системи, що включає верифікацію функціональності та стабільності додатку, був проведений. Всі ці аспекти взяті разом створюють комплексний погляд на розробку та реалізацію інформаційної технології управління проєктами з використанням технології Azure. Це підкреслює важливість правильного вибору технологій, грамотного програмування та тестування для створення ефективного та надійного веб-додатку.

4 ЕКОНОМІЧНА ЧАСТИНА

Ефективне впровадження науково-технічної розробки стає реальністю при відповідності її поточним вимогам науково-технічного прогресу та урахуванні економічних аспектів. Визначення економічної ефективності отриманих результатів науково-дослідної роботи є ключовою частиною цього процесу.

Магістерська робота, що присвячена розробці та дослідженню "Інтелектуальна система управління проектами з використанням технології Azure", відноситься до науково-технічних робіт, спрямованих на введення на ринок. Вирішення питання про комерціалізацію розробки може виникнути протягом виконання самої роботи, що створює можливість її подальшого виведення на ринок. Цей напрямок розглядається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Проте для успішної реалізації цього процесу необхідно пройти кілька етапів:

1. Проведення комерційного аудиту науково-технічної розробки, включаючи визначення її науково-технічного рівня та комерційного потенціалу.
2. Розрахунок витрат на реалізацію науково-технічної розробки.
3. Розрахунок економічної ефективності впровадження та комерціалізації науково-технічної розробки для потенційного інвестора, а також обґрунтування економічної доцільності комерціалізації з точки зору інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою "Інтелектуальна система управління проектами з використанням технології Azure" є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 4.1 [13].

Таблиця 4.1 – Критерії оцінювання комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	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 працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
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
Ринкові перспективи					
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	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 4.1

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
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
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 повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 4.2. Для опитування було залучені експерти: Ковальчук В. О., директор ТОВ «ЕКСАДЕЛ УКРАЇНА»; Татаровська К. О., HR менеджер ТОВ «ЕКСАДЕЛ УКРАЇНА»; Збитківський В. С., Front-End Developer ТОВ «ЕКСАДЕЛ УКРАЇНА».

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Ковальчук В. О.	Татаровська К. О.	Збитківський В. С.
	Бали:		
1. Технічна здійсненність концепції	2	2	2
2. Ринкові переваги (наявність аналогів)	2	3	1
3. Ринкові переваги (ціна продукту)	1	1	2
4. Ринкові переваги (технічні властивості)	2	3	3
5. Ринкові переваги (експлуатаційні витрати)	1	1	0
6. Ринкові перспективи (розмір ринку)	4	3	4
7. Ринкові перспективи (конкуренція)	0	0	0
8. Практична здійсненність (наявність фахівців)	3	3	4
9. Практична здійсненність (наявність фінансів)	2	2	1
10. Практична здійсненність (необхідність нових матеріалів)	3	3	4
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	$СБ_1$=27	$СБ_2$=29	$СБ_3$=29
Середньоарифметична сума балів $СБ_c$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{27+29+29}{3} = 28.3$		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 4.3 [24].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Інтелектуальна система управління проектами з використанням технології Azure" становить 28 балів, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки середній.

Результатом магістерської кваліфікаційної роботи є інтелектуальна система (веб-додаток) для управління проектами з використанням технології Azure, яка має розширений функціонал порівняно з аналогічними, що є присутніми на ринку.

Користувачами системи можуть бути менеджери проектів різних розмірів та масштабів, а також розробники чи ті хто управляє своїми проектами наодинці для відстежування прогресу та результатів роботи.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему "Інтелектуальна система управління проектами з використанням технології Azure", під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [24]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 25000 \cdot 5 / 21 = 5682 \; грн.$$

Проведені розрахунки зведемо до таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Менеджер проекту	25000	1136,4	5	5682
Інженер-програміст	40000	1818,2	25	45455
Розробник	25000	1136,4	25	28409
Всього				79545

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему "Інтелектуальна система управління проектами з використанням технології Azure" розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \quad (4.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (4.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [13];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

Витрати на основну заробітну плату було зведено до таблиці 4.5.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6500{,}00 \cdot 1{,}1 \cdot 1{,}65 / (21 \cdot 8) = 72{,}4$ грн.

$З_{р1} = 72{,}4 \cdot 2 = 144{,}8$ грн.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
1. Підготовчі	2	1	65,8	131,6
2. Дизайн	3	2	72,4	217,2
3. Розробка UI	2	3	88,8	177,7
4. Розробка Back-End	6	5	111,9	671,2
5. Розробка аналітичної частини	5	5	111,9	559,3
6. Тестування	2	1	72,4	144,8
Всього				1901,7

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (79545+ 1901,7) · 11 / 100% = 8959,19 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (79545+ 1901,7+8959,19) · 22 / 100% = 19889,4 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Інтелектуальна система управління проектами з використанням технології Azure".

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (4.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір А 4	180	1	180
Ручка	40	1	40
Диск оптичний CD	20	2	40
Flesh-пам'ять	190	1	190
Всього			450
З врахуванням коефіцієнта транспортування			495

4.2.4 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.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (4.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 35680,00 \cdot 1 \cdot 1,11 = 39604,80$ *грн.*

Отримані результати зведемо до таблиці 4.7:

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Персональний комп’ютер	3	25000	82500
Роутер	1	1500	1650
Всього			92565

4.2.5 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних

для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 10000 \cdot 3 \cdot 1,11 = 33000$ грн.

Отримані результати зведемо до таблиці 4.8:

Таблиця 4.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ліцензія Azure	3	10000	33000
Всього			33000

4.2.6 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \ , \qquad (4.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$А_{обл} = (75000 \cdot 1) / (2 \cdot 12) = 3125$ грн.

Проведені розрахунки зведемо до таблиці 4.9.

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	75000	2	1	3125,00
Приміщення лабораторії	235000	20	1	979,17
Всього				4104,17

4.2.7 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot K_{впі}}{\eta_i}, \qquad (4.10)$$

де $W_{уі}$ – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е = 7,5$ грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i<1$.

$В_е = 0,25 \cdot 275,0 \cdot 7,5 \cdot 0,5 / 0,8 = 322,27$ грн.

4.2.8 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему "Інтелектуальна система управління проектами з використанням технології Azure" 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.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (4.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$В_{св} = (79545+ 1901,7) \cdot 20 / 100\% = 16289,44$ грн.

4.2.9 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$I_в$ = (79545+ 1901,7) · 50 / 100% = 40723,59 грн.

4.2.10 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (79545+ 1901,7) · 100 / 100% = 81447,18 грн.

Витрати на проведення науково-дослідної роботи на тему "Інтелектуальна система управління проектами з використанням технології Azure". розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.14)$$

$В_{заг}$= 79545+ 1901,7+8959,19+19889,4 +495+92565+33000+4104,17+ 322,27 +16289,44 +40723,59 + 81447,18 = 379242,4 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 379242,4 / 0,9 = 421380,45 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою "Інтелектуальна система управління проектами з використанням технології Azure" передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 5000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [24]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1 \cdot 500 + 5000 \cdot 900) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 845676,59$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1 \cdot 500 + 5000 \cdot (900 + 800)) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 1597727,8$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1 \cdot 500 + 5000 \cdot (900 + 800 + 700)) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 2255409,8$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 845676{,}59 /(1+0{,}18)^1 + 1597727{,}8 /(1+0{,}18)^2 + 2255409{,}8 /(1+0{,}18)^3 =$$
$$= 3125548{,}44 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (4.18)$$

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

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 421380,45 грн.

$$PV = k_{інв} \cdot ЗВ = 2 \cdot 421380{,}45 = 842760{,}89 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (4.19)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 3125548,44 грн;

PV – теперішня вартість початкових інвестицій, 842760,89 грн.

$E_{абс} = ПП - PV$ = 3125548,44 - 842760,89 = 2282787,54 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+2282787,54 /842760,89)^{1/3} - 1 = 0,86.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (4.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 0,86 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{4.22}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,86 = 1,2 р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Висновок до розділу 4

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Інтелектуальна система управління проектами з використанням технології Azure" становить 28 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки середній.

Також термін окупності становить 1,2 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може

спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою "Інтелектуальна система управління проектами з використанням технології Azure".

ВИСНОВКИ

Підчас виконання магістерської кваліфікаційної роботи було розв’язано задачу розробки інформаційної технології та програмного забезпечення управління проектами з використанням технології Azure.

В першому розділі роботи проведено аналіз інформаційних технологій та систем, пов'язаних з управлінням проектами. Бувзроблений докладний огляд інструментів та технологій, що найпоширеніше використовуються для управління проектами з використанням cloud computing технологій. Визначено, які технології на сьогоднішній день є найбільш актуальними та популярними. Вивчені потреби цільової аудиторії, яка має буде використовувати додаток для управління проектами з вбудованим аналізом складності завдань. Також було проведено аналіз очікуваннь користувачів системи щодо графічного інтерфейсу, клієнтської частини та функціоналу додатків такого типу. Був проведений аналіз таких характеристик веб-платформ та операційних систем, які мають певний плив на розробку та можливу реалізацію технології для управління проектами.

У другому розділі було розглянуто основні аспекти необхідні для розробки інформаційної технології для управління проектами з використанням технології Azure. В другому розділі був проведений аналіз архітектури інформаційної системи, що розробляється та визначено клієнтську частину, яку розроблятимемо з використанням Angular і TypeScript, а також back-end частини з використанням Express.js та Azure. Також важливо відмітити, що IDE. Яка буде використовуватись для розробки додатку з управління проектами є саме Visual Studio Code.

У третьому розділі докладно представлено інформацію щодо програмної реалізації інформаційної технології для управління проектами, використовуючи технологію Azure у веб-додатку. Проаналізовано вибір мови та середовища програмування, таких як JavaScript і TypeScript, також був проаналізований вибір фреймворка, що буде використовуватись для розробки, та обраний один серед React.js та Angular і визначено їх важливість для розробки сучасного веб-

додатку. Детально описано модель розробки проекту, яка використовується в системі та надає загальний вигляд на процес взаємодії менеджерів та робітників. Розглянуті ключові компоненти реалізації, такі як використання Angular, взаємодія з сервером через RESTful API, використання бази даних, а також важливість вибору інтегрованого середовища розробки Visual Studio Code. Створено зручний та сучасний користувацький інтерфейс, де продемонстровано основні компоненти для ефективної роботи з додатком. Також проведено процес тестування системи, що включає в себе верифікацію функціональності та стабільності додатку. Усі ці аспекти взаємодіють, утворюючи комплексний погляд на розробку та впровадження інформаційної технології управління проектами з використанням технології Cloud Computing, що підкреслює важливість розумного вибору технологій, компетентного програмування та тестування для створення надійного та ефективного веб-додатку.

Розроблене програмне забезпечення забезпечує:

- швидку відповідь від серверу приблизно 7000мс, використану об’єктно-орієнтовану мову програмування JavaScript, обмеження в максимальній кількості працівників у 100 працівників та обмеження максимальної кількості символів в описі завдання в 1000 символів;

- покращення ефективності управління ресурсами, які доступні для роботи на проекті, шляхом залучення API штучного інтелекту для аналізу попереднього опису завдань, а також методів нечіткої логіки.

- спрощення процесу планування та управління проектами та процесами в проектах різних розмірів та складностей, що робить його більш доступним та зручним для користувачів.

У четвертому розділі були проведені розрахунки витрат на розробку та виготовлення нового технічного рішення, сума яких становить 842760 гривень. Прогнозувалась орієнтована величина витрат для кожної з категорій. Також визначався чистий прибуток, який виробник може потенційно отримати від реалізації розробленого технічного рішення, та встановлювався термін окупності витрат виробника, а також економічний ефект для споживача при використанні

даної розробки. Аналіз розрахунків свідчить про те, що розробка та використання виробництвом є більш вигідними в порівнянні з аналогічними рішеннями, що підтверджується періодом окупності, що становить приблизно 1.2 року.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання хмарної технології Azure при розробці сучасних веб-додатків / І. І. Кривенко, І. В. Богач – URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/19543

2. 1. Огляд документо-орієнтованих баз даних на прикладі MongoDB / І. І. Кривенко, І. В. Богач / матер. LI Науко-технічній конференції підрозділів ВНТУ 2021. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15766.

3. Epics in Jira – URL: https://www.atlassian.com/agile/project-management/epics

4. How JavaScript works – URL: https://medium.com/sessionstack-blog/how-javascript-works-a-complete-guide-to-asynchronous-javascript-34ea34f8bd91

5. Asynchronous JavaScript (Event Loop) – URL: https://medium.com/gradeup/asynchronous-javascript-event-loop-1c8de41298dd

6. What is Framework 2022. – URL: https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/

7. What is Angular 2015. URL: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

8. What is Express.js? 2021. URL: https://gouravmukhija.medium.com/what-is-expressjs-64b0b4334f2e

9. What is Vue 2019. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Vue_getting_started

10. Next.js 2022. URL: https://www.geeksforgeeks.org/nextjs/

11. Angular tutorial 2019. URL: https://www.geeksforgeeks.org/angular-tutorial/?ref=gcse

12. What is API? 2021. URL: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

13. What is Cloud Computing? 2022. URL: https://aws.amazon.com/what-is-cloud-computing/

14. Azure durable functions 2024. URL: https://vinayak-kedari-14357.medium.com/azure-durable-functions-for-long-running-apis-in-javascript-d214454123b5

15. Azure function realization 2024. URL: https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-node?pivots=nodejs-model-v4

16. Angular bootstrapping 2022. URL: https://angular.io/guide/bootstrapping

17. Model–view–controller URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

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

19. React.js Fundamentals. URL: https://react.dev/

20. What is HTTPS? https://www.cloudflare.com/learning/ssl/what-is-https/

21. Document Object Model 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

22. Everything you need to know about Azure https://www.nigelfrank.com/insights/everything-you-ever-wanted-to-know-about-microsoft-azure.

23. What is End-to-end testing? 2022. URL: https://katalon.com/resources-center/blog/end-to-end-e2e-testing

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

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інтелектуальна система управління проектами з використанням технології Azure

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 88,6% Схожість 11,4%

Аналіз звіту подібності (відмітити потрібне):

✓ 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 виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 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.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Кривенко І.І.

Керівник роботи Богач І.В.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

app.component.ts

```
import { Component } from '@angular/core';
import { SpinnerService } from './services/spinner.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  loading = this.spinnerService.isSpineActive$;

  constructor(public spinnerService: SpinnerService) {
  }
  title = 'Easy Manage';
}
```

app.module.ts

```
import { SpinnerService } from './services/spinner.service';
import { SpinnerHttpInterceptor } from './services/http.interceptor.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomePageComponent } from './models/home-page/home-page.component';
import { AuthorizationPageComponent } from './models/authorization-page/authorization-page.component';
import { NotFoundPageComponent } from './models/not-found-page/not-found-page.component';
import { ReactiveFormsModule } from '@angular/forms';
import { LoginFormComponent } from './components/login-form/login-form.component';
import { SpineComponent } from './components/spine/spine.component';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { HeaderComponent } from './components/header/header.component';
import { DialogModule } from 'primeng/dialog';
import { MatDialogModule } from '@angular/material/dialog';
import { CreateProjectModalComponent } from './components/create-project-modal/create-project-modal.component';
import { MatChipsModule } from '@angular/material/chips';
import { MatMenuModule } from '@angular/material/menu';
import { MatIconModule } from '@angular/material/icon';
import { MatSelectModule } from '@angular/material/select';
```

```
import { MatInputModule } from '@angular/material/input';
import { MatButtonModule } from '@angular/material/button';
import { ProjectComponent } from './components/project/project.component';
import { ProjectsListComponent } from './components/projects-list/projects-list.component';
import { ProjectCardComponent } from './components/project-card/project-card.component';
import { AvatarModule } from 'primeng/avatar';
import { AvatarGroupModule } from 'primeng/avatargroup';
import { TaskComponent } from './components/task/task.component';
import { StepBoardComponent } from './components/step-board/step-board.component';
import { DragDropModule } from '@angular/cdk/drag-drop';
import { MatTooltipModule } from '@angular/material/tooltip';
@NgModule({
  declarations: [
    AppComponent,
    HomePageComponent,
    AuthorizationPageComponent,
    NotFoundPageComponent,
    LoginFormComponent,
    SpineComponent,
    HeaderComponent,
    CreateProjectModalComponent,
    ProjectComponent,
    ProjectsListComponent,
    ProjectCardComponent,
    TaskComponent,
    StepBoardComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    HttpClientModule,
    BrowserAnimationsModule,
    DialogModule,
    MatDialogModule,
    MatChipsModule,
    MatMenuModule,
    MatIconModule,
    MatSelectModule,
    MatInputModule,
    MatButtonModule,
    DragDropModule,
    AvatarModule,
    AvatarGroupModule,
    MatTooltipModule
  ],
  providers: [{
      provide: HTTP_INTERCEPTORS,
      useClass: SpinnerHttpInterceptor,
```

```
      multi: true
    },
    SpinnerService
  ],
  bootstrap: [AppComponent],
  entryComponents: [CreateProjectModalComponent]
})
export class AppModule { }
```

project-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MatDialog, MatDialogConfig } from "@angular/material/dialog";
import { CreateProjectModalComponent } from '../create-project-modal/create-project-
modal.component';
import { ProjectService } from 'src/app/services/project.service';
import { Router } from '@angular/router';
@Component({
  selector: 'app-projects-list',
  templateUrl: './projects-list.component.html',
  styleUrls: ['./projects-list.component.scss']
})
export class ProjectsListComponent implements OnInit {
  isModalVisible = false;
  projects = [];
  constructor(
    private modal: MatDialog,
    public projectService: ProjectService,
    public router: Router
  ) { }
  ngOnInit(): void {
    this.getProjectsList();
  }
  getProjectsList(): void {
    this.projectService.getProjects().subscribe(data => {
      this.projects = JSON.parse(data.data).projectsList;
    }, (err) => {
      console.error(err);
    });
  }
  openModal() {

    const modalConfig = new MatDialogConfig();
    modalConfig.autoFocus = true;
    modalConfig.hasBackdrop = true;
    modalConfig.width = '600px';
    let modalRef = this.modal.open(CreateProjectModalComponent, modalConfig);
    modalRef.afterClosed().subscribe(id => {
      this.getProjectsList();
      this.router.navigate(['/project', id]);
    });
```

```
  }
  openProject(projectId: string): any {
    this.router.navigate(['/project', projectId]);
  }
}
```

project.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { ProjectService } from 'src/app/services/project.service';
import { Project } from 'src/app/interfaces/project.interface';
import { AvatarModule } from 'primeng/avatar';
import { AvatarGroupModule } from 'primeng/avatargroup';
import { UserService } from 'src/app/services/user.service';
import { FormBuilder, Validators } from '@angular/forms';
import { CdkDragDrop, moveItemInArray, transferArrayItem } from '@angular/cdk/drag-
drop';
@Component({
  selector: 'app-project',
  templateUrl: './project.component.html',
  styleUrls: ['./project.component.scss']
})
export class ProjectComponent implements OnInit {
  projectDetail: Project;
  detailsLoaded: Promise<boolean>;
  usersLoaded: Promise<boolean>;
  workersToDisplay: string[];
  projectUsers: any[];
  isCreatingTask = {};
  stepsFormGroup = {};
  sortedTasks = {};
  constructor(
    public activeRoute: ActivatedRoute,
    public projectService: ProjectService,
    public userService: UserService,
    private formBuilder: FormBuilder,
    private router: Router
  ) {
  }
  ngOnInit(): void {1
    this.getProjectDetails();
  }
  getProjectDetails(): void {
    this.activeRoute.params.subscribe(params => {
      this.projectService.getProjectDetails(params.id).subscribe(data => {
        this.projectDetail = JSON.parse(data.data).projectDetail;
        this.isCreatingTask = {
          ...this.projectDetail.steps
        }
        this.detailsLoaded = Promise.resolve(true);
```

```
        this.workersToDisplay = this.projectDetail.workers.slice(0, 2);
        this.generateFormControls(this.projectDetail.steps);
        this.sortTasks(this.projectDetail.steps);
        this.getUsersOfProject();
      }, err => {
        console.error(err);
      });
    });
  }
  getUsersOfProject(): any {
    let usersList;
    this.userService.getUserList().subscribe(data => {
      usersList = JSON.parse(data.body);
                          this.projectUsers     =     usersList.filter(user     =>
      this.projectDetail.workers.includes(user.id));
      this.usersLoaded = Promise.resolve(true);
    });
  }
  generateFormControls(steps: string[]): void {
    steps.forEach(step => {
      this.stepsFormGroup[step] =  this.formBuilder.group({
        description: ['', [
          Validators.required,
          Validators.maxLength(255)
        ]],
        worker: ['', Validators.required]
      });
    });
  }
  sortTasks(steps: string[]): void {
    steps.forEach(step => {
       this.sortedTasks[step] = this.projectDetail.tasks.filter(task => task.status
=== step);
    });
  }

  createTask(status: string): void {
    if (this.stepsFormGroup[status].value.description === '' &&
      this.stepsFormGroup[status].value.worker === '') {
        return;
    }
    let task = {
      description: this.stepsFormGroup[status].value.description,
      worker: this.stepsFormGroup[status].value.worker,
      status
    }
    this.projectDetail.tasks.push(task);
                              this.projectService.updateTasks(this.projectDetail.id,
this.projectDetail.tasks).subscribe((data) => {
      let updateTasks = JSON.parse(data.data).updatedTasks;
      this.projectDetail.tasks = updateTasks;
```

```
      this.sortTasks(this.projectDetail.steps);
    }, err => {
      console.error(err);
    });
    this.isCreatingTask[status] = false;
    this.stepsFormGroup[status].description = '';
    this.stepsFormGroup[status].worker = '';
  }
  closeCreating(status: string): void {
    this.stepsFormGroup[status].description = '';
    this.stepsFormGroup[status].worker = '';
    this.isCreatingTask[status] = false;
  }
  getWorkerName(userId): string {
    let user = this.projectUsers?.find(user => user.id = userId);
    return `${user.first_name} ${user.last_name}`
  }
  backToProjects(): void {
    this.router.navigate(['/projects']);
  }
  drop(event: CdkDragDrop<any>) {
    let updatedTasks = [];

    if (event.previousContainer.element === event.container.element) {
                    moveItemInArray(event.container.data,      event.previousIndex,
event.currentIndex);
    } else {
      transferArrayItem(
        event.previousContainer.data,
        event.container.data,
        event.previousIndex,
        event.currentIndex,
      );
      event.container.data.forEach(task => task.status = event.container.id);
    }
    for (let tasks in this.sortedTasks) {
      updatedTasks = [...updatedTasks, ...this.sortedTasks[tasks]]
    }
    if (JSON.stringify(this.projectDetail.tasks) !== JSON.stringify(updatedTasks)) {
      this.projectDetail.tasks = updatedTasks;

      this.projectService.updateTasks(this.projectDetail.id,
updatedTasks).subscribe(() => {
        }, err => {
          console.error(err);
      });
    }
  }
  getTeamMembersList(): string {
    let teamMembers = [];
```

```
    this.projectUsers?.forEach(user => {
      teamMembers.push(this.getWorkerName(user));
    });
    return teamMembers.join(', ');
  }
  removeTask(task: any): void {
    console.log(task);
  }
}
```

create-project.component.ts

```
import { Component, Inject, OnInit } from '@angular/core';
import { FormBuilder, FormControl, Validators } from '@angular/forms';
import { MAT_DIALOG_DATA, MatDialogRef } from "@angular/material/dialog";
import { UserService } from 'src/app/services/user.service';
import { UserDetail } from 'src/app/interfaces/userDetailInterface';
import { ProjectService } from 'src/app/services/project.service';
@Component({
  selector: 'app-create-project-modal',
  templateUrl: './create-project-modal.component.html',
  styleUrls: ['./create-project-modal.component.scss']
})
export class CreateProjectModalComponent implements OnInit {
  currentUserData = JSON.parse(localStorage.getItem('userData'));
  userData = {
      firstName: this.currentUserData.first_name,
      lastName: this.currentUserData.last_name,
      id: this.currentUserData.id
  };
  usersList: UserDetail[];
  projectTypesList = [
    {name: 'IT', value: 'IT'},
    {name: 'Business', value: 'BUSINESS'},
    {name: 'Other', value: 'OTHER'}
  ];
  newProjectDetails = this.fb.group({
    projectTitle: ['',[
      Validators.required,
      Validators.minLength(3),
      Validators.maxLength(50)
    ]],
    projectDescription: ['', Validators.maxLength(500)],
    teamMembers: [[this.userData.id], [Validators.required]],
    projectType: ['', [Validators.required]]
  });
  constructor(private fb: FormBuilder,
    private dialogRef: MatDialogRef<CreateProjectModalComponent>,
    @Inject(MAT_DIALOG_DATA) data,
    public userService: UserService,
    public projectService: ProjectService
```

```
  ) {  }
  ngOnInit(): void {
    this.userService.getUserList().subscribe(data => {
      let parsedUserList = JSON.parse(data.body);
      this.usersList = parsedUserList;
      this.usersList = this.getUserList(parsedUserList);
      }, (e) => {
        console.error(e);
      });
  }
  close() {
    this.dialogRef.close();
  }
  getUserList(userData: UserDetail[]): any {
    return userData.map(user => {
      return {
        firstName: user.first_name,
        lastName: user.last_name,
        id: user.id
      }
    });
  }

  createProject(): any {
    let projectData = this.newProjectDetails.value;
    projectData.createdBy = this.userData.id;
    this.projectService.createProject(projectData).subscribe(data => {
      let newProjectId = JSON.parse(data.data).newProject.id;
      this.dialogRef.close(newProjectId);
    }, (err) => {
      console.error(err);
    });
  }
}
```

app.js (сервер)

```
const express = require('express');
var cors = require('cors');
require('dotenv').config();
require("./src/config/db").connect();
const app = express();
app.use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');
    next();
});
app.use(cors());
app.use(express.json({ extended: false }));
app.use(express.json());
app.use(express.urlencoded({
    extended: true
```

```
}));
app.get('/', (req, res) => {
    res.send('Hello World!');
});
require('./src/routes/user.router')(app);
require('./src/routes/project.router')(app);
module.exports = app;
```

project.router.js

```
module.exports = app => {
    const project = require('../controllers/project.controller');
    app.post('/createProject', project.create);
    app.put('/editProject', project.editProject);
    app.post('/getProjectForUser', project.getProjectsForUser);
    app.get('/projectDetail', project.getProject);
    app.delete('/projectDelete', project.delete);
    app.put('/projectUpdateMembers', project.addOrDeleteWorkers);
    app.put('/projectUpdateTasks', project.deleteOrUpdateProjectTasks);
};
```

project.controller.js

```
const Project = require('../services/project.service');
exports.create = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const newProjectData = {
        project_title: req.body.projectTitle,
        project_description: req.body.projectDescription,
        workers: req.body.teamMembers,
        type_of_project: req.body.projectType,
        created_by: req.body.createdBy
    }
    Project.createProject(newProjectData, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Creating success!",
                data: JSON.stringify(data, null, 2),
            });
        }
    })
}
exports.getProject = (req, res) => {
```

```
    if (!req.query) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const projectId = req.query.projectId;
    Project.getProjectDetails(projectId, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Project was found successfully!",
                data: JSON.stringify(data, null, 2),
            });
        }
    });
}
exports.getProjectsForUser = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const userId = req.body.userId;

    Project.getProjectsForUser(userId, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Projects were found successfully!",
                data: JSON.stringify(data, null, 2),
            });
        }
    });
}
exports.editProject = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const projectEditData = {
        id: req.body.projectId,
        project_title: req.body.projectTitle,
        project_description: req.body.projectDescription,
        type_of_project: req.body.projectType,
```

```
        status: req.body.status
    }
    Project.editProject(projectEditData, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Project were updated successfully!",
                data: JSON.stringify(data, null, 2),
            });
        }
    });
}
exports.delete = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const projectId = req.body.projectId;
    Project.deleteProject(projectId, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Project were deleted successfully!",
                data: data,
            });
        }
    });
}
exports.deleteOrUpdateProjectTasks = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const dataForUpdate =  {
        projectId: req.body.projectId,
        tasks: req.body.tasks
    }
    Project.deleteOrUpdateProjectTasks(dataForUpdate, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
```

```
            res.status(data.status).send({
                message: "Project tasks were updated successfully!",
                data: JSON.stringify(data, null, 2),
            });
        }
    });
}

exports.addOrDeleteWorkers = (req, res) => {
    if (!req.body) {
        res.status(400).send({
            message: `Request body shouldn't be empty`
        });
    }
    const dataForUpdate =  {
        projectId: req.body.projectId,
        workers: req.body.teamMembers
    }
    Project.addOrDeleteWorkers(dataForUpdate, (err, data) => {
        if (err) {
            res.status(err.status).send({
                message: err.message,
            });
        } else {
            res.status(data.status).send({
                message: "Project members were updated successfully!",
                data: JSON.stringify(data, null, 2),
            });
        }
    });
}
```

project.service.js

```
const Project = require('../models/project.model');
const uniqId = require('uniqid');
const projectTypesSteps = require('../config/project.config');
Project.createProject = async (newProjectData, result) => {
    try {
            let  {  project_title,  project_description,  workers,  type_of_project,
created_by } = newProjectData;

        if (!workers) {
            workers = [];
        }
        if (!(project_title && project_description && workers && type_of_project &&
created_by)) {
            console.log('not all project fields were entered');
            result({
                status: 400,
```

```
                message: "Not all project fields were entered"
            }, null);
            return;
        }
        const projectId = uniqId('project:');
        const isIdRepeat = await Project.findOne({id: projectId});

        while(!!isIdRepeat) {
            projectId = uniqId('project:');
            isIdRepeat = await Project.findOne({id: projectId});
        }
        let steps;
        switch(type_of_project) {
            case 'IT':
                steps = projectTypesSteps.projectTypesSteps.IT;
                break;
            case 'BUSINESS':
                steps = projectTypesSteps.projectTypesSteps.BUSINNES;
                break;
            case 'OTHER':
                steps = projectTypesSteps.projectTypesSteps.OTHER;
                break;
            default:
                steps = projectTypesSteps.projectTypesSteps.OTHER;
                break;
        }
        const newProject = await Project.create({
            id: projectId,
            project_title,
            project_description,
            workers: workers.split(', '),
            steps,
            type_of_project,
            creation_date: Date.now(),
            status: 'NEW',
            created_by
        });
        result(null, {status: 201, newProject});
    } catch (err) {
        console.log(err);
    }
}
Project.getProjectDetails = async (projectId, result) => {
    try {
        const projectDetail = await Project.findOne({id: projectId});
        if (!projectDetail) {
            result({
                status: 404,
                message: "Project wasn't found"
            });
            return;
```

```
        }
        result(null, {
            status: 202,
            projectDetail
        });
    } catch (err) {
        console.log(err);
    }
}
Project.getProjectsForUser = async (userId, result) => {
    try {
        const allProjects = await Project.find({});
                const projectsForUser = await allProjects.filter(item =>
item.workers.includes(userId));
        projectsList = projectsForUser.map(project => {
            return {
                project_title: project.project_title,
                project_description: project.project_description,
                workers: project.workers,
                creating_date: project.creating_date,
                type_of_project: project.type_of_project,
                id: project.id
            }
        })
        result(null, {
            status: 202,
            projectsList
        });
    } catch (err) {
        console.log(err);
    }
}
Project.editProject = async (editedProjectData, result) => {
    try {
        const projectForUpdateId = editedProjectData.id;

        if (!!(await Project.findOne({id: projectForUpdateId}))) {
            result({
                status: 404,
                message: "Project not found"
            }, null);

            return;
        }
        const updatedProject = await Project.updateOne({id: projectForUpdateId}, {
            project_title: editedProjectData.project_title,
            project_description: editedProjectData.project_description,
            type_of_project: editedProjectData.tipe_of_project,
            status: projectForUpdateId.status,
        });
        result(null, {
```

```
            status: 201,
            updatedProject
        });
    } catch (err) {
        console.log(err);
    }
}
Project.deleteProject = async (projectId, result) => {
    try {
        if (await Project.findOne({id: projectId})) {
            result({
                status: 400,
                message: "Project already deleted ot not exist"
            });

            return;
        }
        await Project.deleteOne({id: projectId});

        result(null, {
            status: 200,
            message: "Project deleted"
        });
    } catch (err) {
        console.log(err);
    }
}
Project.deleteOrUpdateProjectTasks = async (dataForUpdate, result) => {
    try {
        if (!await Project.findOne({id: dataForUpdate.projectId})) {
            result({
                status: 404,
                message: "Project not found"
            });

            return;
        }
        let tasksForUpdate = JSON.parse(dataForUpdate.tasks);
                  const    updatedProjectData    =    await    Project.updateOne({id:
dataForUpdate.projectId}, {tasks: tasksForUpdate});
        const updatedProject = await Project.findOne({id: dataForUpdate.projectId});
        const updatedTasks = await updatedProject.tasks;

        result(null, {
            status: 200,
            updatedTasks
        });
    } catch (err) {
        console.log(err);
    }
}
```

```
Project.addOrDeleteWorkers = async (dataForUpdate, result) => {
    try {
        if (await Project.findOne({id: dataForUpdate.projectId})) {
            result({
                status: 404,
                message: "Project not found"
            });

            return;
        }

                  const    updatedProjectData    =    await    Project.updateOne({id:
dataForUpdate.projectId}, {workers: dataForUpdate.workers});
        result(null, {
            status: 200,
            updatedProjectData
        });
    } catch (err) {
        console.log(err);
    }
}
module.exports = Project;
```

db.js (config)

```
const mongoose = require('mongoose');
const { MONGO_URI } = process.env;
exports.connect = () => {
    mongoose
        .connect(MONGO_URI, {
            useNewUrlParser: true,
        })
        .then(() => {
            console.log('Successfully connected to database');
        })
        .catch((err) => {
            console.log('DB connection failed');
            console.error(err);
            process.exit(1);
        });
}
```

project.model.js

```
const mongoose = require('mongoose');
const projectSchema = new mongoose.Schema({
    id: { type: String },
    project_title: { type: String, default: null },
    project_description: { type: String, default: null },
    tasks: [{
```

```
        description: String,
        worker: String,
        status: String
    }],
    steps: [String],
    workers: [String],
    creation_date: { type: Date, default: Date.now },
    type_of_project: { type: String },
    status: { type: String },
    created_by: { type: String }
});
module.exports = mongoose.model("projects", projectSchema);
```

project-list.component.html

```
<div class="projects-list">
    <div class="projects-list-container">
        <div class="create-project" (click)="openModal()">
            <img class="add-icon" src="../../../assets/images/add.svg">
            <h2>Створити проект</h2>
        </div>
        <app-project-card *ngFor="let project of projects"
            (click)="openProject(project.id)"
            projectTitle="{{ project.project_title }}"
            description="{{ project.project_description }}"
            type="{{ project.type_of_project }}"
            teamMembers="{{ project.workers.length }}"
            creationDate="{{ project.creation_date }}">
        </app-project-card>
    </div>
</div>
```

project.component.html

```
<div class="project-detail-container">
    <app-header></app-header>
    <div class="project-header">
        <div class="back-to-projects-link" (click)="backToProjects()">
            <img src="../../../assets/images/arrow-left.svg">

            <p>Назад до проектів</p>
        </div>
                <h2   *ngIf="detailsLoaded   |   async"   class="project-title">{{
projectDetail?.project_title }}</h2>
    </div>
    <div class="top-panel">
        <div *ngIf="detailsLoaded | async" class="project-controll-panel">
            <p class="text-description">
                {{ projectDetail.project_description }}
            </p>
```

```
            <div *ngIf="usersLoaded | async" class="members" #tooltip="matTooltip"
                matTooltip="{{ getTeamMembersList() }}"
                [matTooltipPosition]="'above'">
                <h3>Команда: </h3>
                <p-avatarGroup styleClass="mb-3">
                    <p-avatar *ngFor="let member of projectDetail.workers"
                        image="../../../assets/images/user-circle.svg"
                        size="large" shape="circle">
                    </p-avatar>
                    <p-avatar
                        *ngIf="projectDetail.workers.length > 2"
                                                label="projectDetail.workers.slice(2,
projectDetail.workers.slice.length - 1).length"
                                shape="circle"  size="large"  [style]="{'background-
color':'#9c27b0', 'color': '#ffffff'}">
                    </p-avatar>
                </p-avatarGroup>
            </div>
            <div class="manage-project" #tooltip="matTooltip"
                matTooltip="Рудагувати"
                [matTooltipPosition]="'above'">
                <img class="add-icon" src="../../../assets/images/edit.svg">
            </div>
        </div>
    </div>
    <div *ngIf="detailsLoaded | async" class="project-board" cdkDropListGroup>
        <div *ngFor="let step of projectDetail.steps" class="step">
            {{ step.replaceAll('_', ' ') }}
            <div class="step-card-container"
                cdkDropList
                [cdkDropListData]="sortedTasks[step]"
                (cdkDropListDropped)="drop($event)"
                id="{{step}}">
                    <app-task *ngFor="let task of sortedTasks[step]"
                    taskDescription="{{ task.description }}"
                    workerName="{{ getWorkerName(task.worker) }}"
                    cdkDrag></app-task>

                <div *ngIf="isCreatingTask[step]" class="create-task">
                    <form class="task-form" [formGroup]="stepsFormGroup[step]">
                       <mat-form-field class="example-full-width" appearance="fill">
                            <mat-label>Опис:</mat-label>
                                                            <textarea   matInput
formControlName='description'></textarea>
                        </mat-form-field>
                        <mat-form-field appearance="fill">
                            <mat-label>Учаник</mat-label>
                            <mat-select formControlName="worker">
                                     <mat-option *ngFor="let user of projectUsers"
[value]="user.id">
                                    {{ user.first_name }} {{ user.last_name }}
```

```
                                </mat-option>
                            </mat-select>
                        </mat-form-field>
                    </form>
                    <div class="button-row">
                              <button mat-raised-button (click)="createTask(step)"
[disabled]="!stepsFormGroup[step].valid">Створити</button>
                                                          <button    mat-raised-button
(click)="closeCreating(step)">Відміна</button>
                    </div>
                </div>
                <div *ngIf="!isCreatingTask[step]"
                    (click)="isCreatingTask[step] = !isCreatingTask[step]"
                    class="add-task-button">
                    <img class="add-icon" src="../../../assets/images/add.svg">
                </div>
            </div>
        </div>
    </div>
</div>
```

project.component.scss

```
@import '../../colors.scss';
.project-detail-container {
    height: 100vh;
    max-height: 100vh;
    background-image: url(../../../assets/images/boardBack.jpg);
    background-repeat: no-repeat;
    background-size: cover;
    .project-header {
        padding: 1rem 3rem;
        display: flex;
        flex-direction: column;
        .project-title {
            height: 3rem;
            font-size: 24px;
            margin-bottom: 0;
        }
        .back-to-projects-link {
            cursor: pointer;
            display: flex;
            font-size: 16px;
            text-decoration: underline;
            p {
                margin-bottom: 0;
            }
        }
    }
    .top-panel {
        width: 100%;
```

```
    padding: 1rem 3rem 0 3rem;
    .project-controll-panel {
        @include glass-background;
        width: 100%;
        height: 6rem;
        display: flex;
        padding: 1rem;
        justify-content: space-between;
        .text-description {
            font-size: 16px;
        }
        .manage-project {
            width: 4rem;
            height: 4rem;
            display: flex;
            align-items: center;
            justify-content: center;
            @include glass-background;
            @include shadow;
            transition: 0.3s;
            cursor: pointer;
            img {
                width: 3rem;
                height: 3rem;
            }
            &:hover {
                transform: scale(1.02);
                ;
            }
        }
        .manage-project {
            img {
                width: 2rem;
                height: 2rem;
            }
        }
        .members {
            width: 4rem;
            height: 4rem;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
        }
    }
}
.project-board {
    display: flex;
    min-height: calc(100% - 22rem);
    margin-bottom: 0;
    justify-content: space-between;
```

```
padding: 1rem 3rem 3rem 3rem;
column-gap: 1rem;
.step {
    width: 100%;
    height: auto;
    @include glass-background;
    display: flex;
    flex-direction: column;
    align-items: center;
    font-size: 1rem;
    font-weight: 600;
    padding-top: 2rem;
    padding-bottom: 1rem;
    .step-card-container {
        height: 100%;
        width: 90%;
        border: 1px solid rgb(201, 201, 201);
        border-radius: 5px;
        display: flex;
        flex-direction: column;
        row-gap: 0.5rem;
        padding: 0.3rem;
        .add-task-button {
            @include glass-background;
            @include shadow;
            display: flex;
            align-items: center;
            justify-content: center;
            height: 2rem;
            transition: 0.3s;

            &:hover {
                transform: scale(1.02);
            }
        }
        .create-task {
            display: flex;
            flex-direction: column;
            padding: 0.4rem;
            @include glass-background;

            .mat-form-field {
                width: 100%;
            }
            .button-row {
                display: flex;
                flex-direction: row;
                justify-content: space-between;
            }
        }
    }
```

```
        }
    }
    .cdk-drag-preview {
        box-sizing: border-box;
        border-radius: 4px;
        box-shadow: 0 5px 5px -3px rgba(0, 0, 0, 0.2),
                    0 8px 10px 1px rgba(0, 0, 0, 0.14),
                    0 3px 14px 2px rgba(0, 0, 0, 0.12);
      }
      .cdk-drag-placeholder {
        opacity: 0;
      }
      .cdk-drag-animating {
        transition: transform 250ms cubic-bezier(0, 0, 0.2, 1);
      }
      .example-box:last-child {
        border: none;
      }
      .example-list.cdk-drop-list-dragging .example-box:not(.cdk-drag-placeholder) {
        transition: transform 250ms cubic-bezier(0, 0, 0.2, 1);
      }
}
```

project-list.component.scss

```
@import '../../colors.scss';
.projects-list {
    height: 100%;
    padding: 3rem;
    .projects-list-container {
        display: flex;
        flex-wrap: wrap;
        gap: 3rem;
       .create-project {
            height: 300px;
            width: 300px;
            @include glass-background;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            color: $grey;
            @include shadow;
            transition: 0.3s;
            cursor: pointer;
            .add-icon {
                height: 150px;
            }
            &:hover {
                transform: translateY(-5px);
            }
```

```
        }
    }
}
```

analizeTask/index.js

```
module.exports = async function (context, req) {
    try {
        context.log('JavaScript HTTP trigger function processed a request.');

        // Parse the request body
        const { task, users } = req.body;

        // Validate input
        if (!task || !users || !Array.isArray(users) || users.length === 0) {
            context.res = {
                status: 400,
                body: "Invalid input. Please provide a valid task and a non-empty
list of users."
            };
            return;
        }
        const taskDifficulty = evaluateTaskDifficulty(task);
        const points = assignPoints(taskDifficulty);
        const recommendedUser = recommendUser(users);
        const response = {
            points,
            recommendedUser
        };
        context.res = {
            status: 200,
            body: response
        };
    } catch (error) {
        context.log.error(error);
        context.res = {
            status: 500,
            body: "Internal Server Error"
        };
    }
};
function evaluateTaskDifficulty(task) {
    // Example: Calculate difficulty based on task properties
    // You can replace this with your own logic
    const { description, technologies } = task;
    const difficulty = description.length + technologies.length;
    return difficulty;
}
function assignPoints(difficulty) {
    // You can customize this array based on your point scale
    const pointScale = [0, 1, 2, 3, 5, 8, 13, 21];
```

```
    // Example: Assign points based on difficulty (you can customize this logic)
    if (difficulty < 10) {
        return pointScale[0];
    } else if (difficulty < 20) {
        return pointScale[1];
    } else if (difficulty < 30) {
        return pointScale[2];
    } else if (difficulty < 40) {
        return pointScale[3];
    } else if (difficulty < 50) {
        return pointScale[4];
    } else if (difficulty < 60) {
        return pointScale[5];
    } else if (difficulty < 70) {
        return pointScale[6];
    } else {
        return pointScale[7];
    }
}
function recommendUser(users) {
    // Example: Recommend a user based on workload and experience
    // You can replace this with your own logic
    const availableUsers = users.filter(user => user.taskLoad < 5); // Adjust
workload threshold as needed

    // Sort available users by experience in descending order
    availableUsers.sort((a, b) => b.experience - a.experience);

    // Return the most experienced available user
    return availableUsers[0];
}
```

fuzzAnalyze.js

```
const fuzz = require('fuzzball');

module.exports = async function (context, req) {
    try {
        context.log('JavaScript HTTP trigger function processed a request.');

        // Parse the request body
        const { task, users } = req.body;

        // Validate input
        if (!task || !users || !Array.isArray(users) || users.length === 0) {
            context.res = {
                status: 400,
                body: "Invalid input. Please provide a valid task and a non-empty
list of users."
            };
            return;
```

```
        }

        // Evaluate task difficulty and assign points
        const taskDifficulty = evaluateTaskDifficulty(task);
        const points = assignPoints(taskDifficulty);
        const recommendedUser = recommendUserFuzzyLogic(task, users);

        // Prepare the response
        const response = {
            points,
            recommendedUser
        };

        // Return the response
        context.res = {
            status: 200,
            body: response
        };
    } catch (error) {
        context.log.error(error);
        context.res = {
            status: 500,
            body: "Internal Server Error"
        };
    }
};
// Function to recommend a user based on fuzzy logic
function recommendUserFuzzyLogic(task, users) {
    const { position, techStack, taskLoad, experience } = task;
    const userScores = users.map(user => {
        const positionScore = fuzz.ratio(position, user.position);
        const techStackScore = fuzz.token_sort_ratio(techStack, user.techStack);
        const taskLoadScore = 100 - Math.abs(taskLoad - user.taskLoad);
        const experienceScore = fuzz.ratio(experience, user.experience);
        const weightedScore = (positionScore * 0.2) +
                              (techStackScore * 0.3) +
                              (taskLoadScore * 0.2) +
                              (experienceScore * 0.3);
        return {
            user,
            score: weightedScore
        };
    });
    userScores.sort((a, b) => b.score - a.score);
    return userScores[0].user;
}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ УПРАВЛІННЯ ПРОЕКТАМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ AZURE

Виконав: студент 2-го курсу,
групи 2КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Кривенко І. І.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Богач І.В.
(прізвище та ініціали)
«07» _____12_____ 2023 р.

Вінниця ВНТУ - 2023 рік

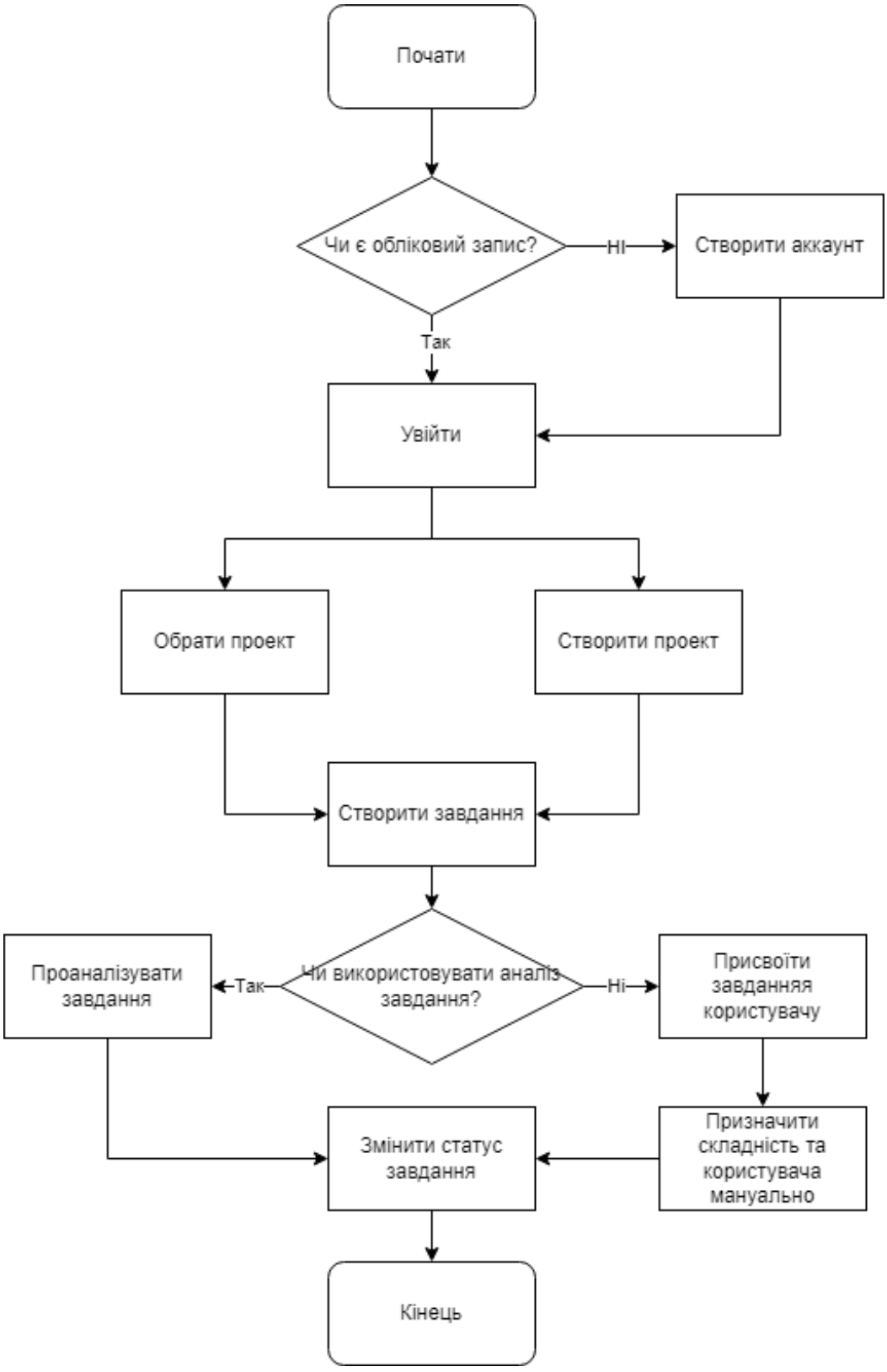

Рисунок В.1 – Алгоритм інформаційної технології роботи веб-додатку

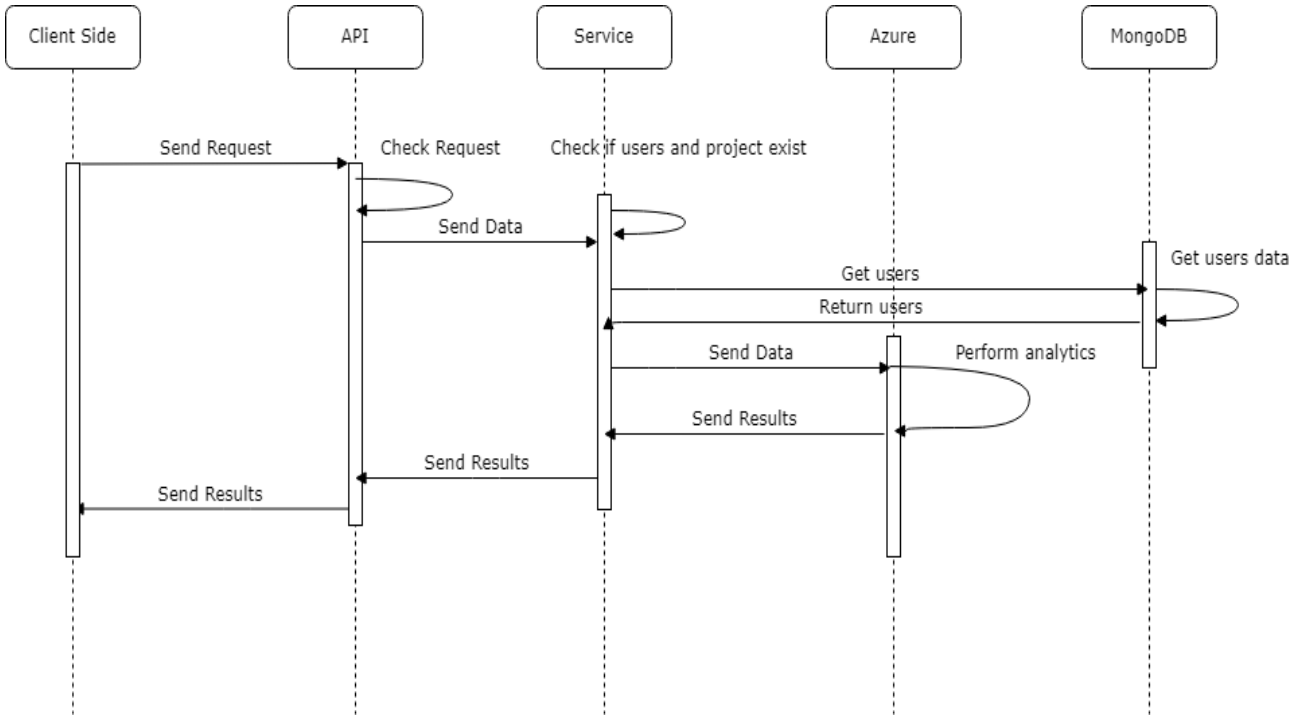

Рисунок В.2 – UML діаграма роботи веб-додатку

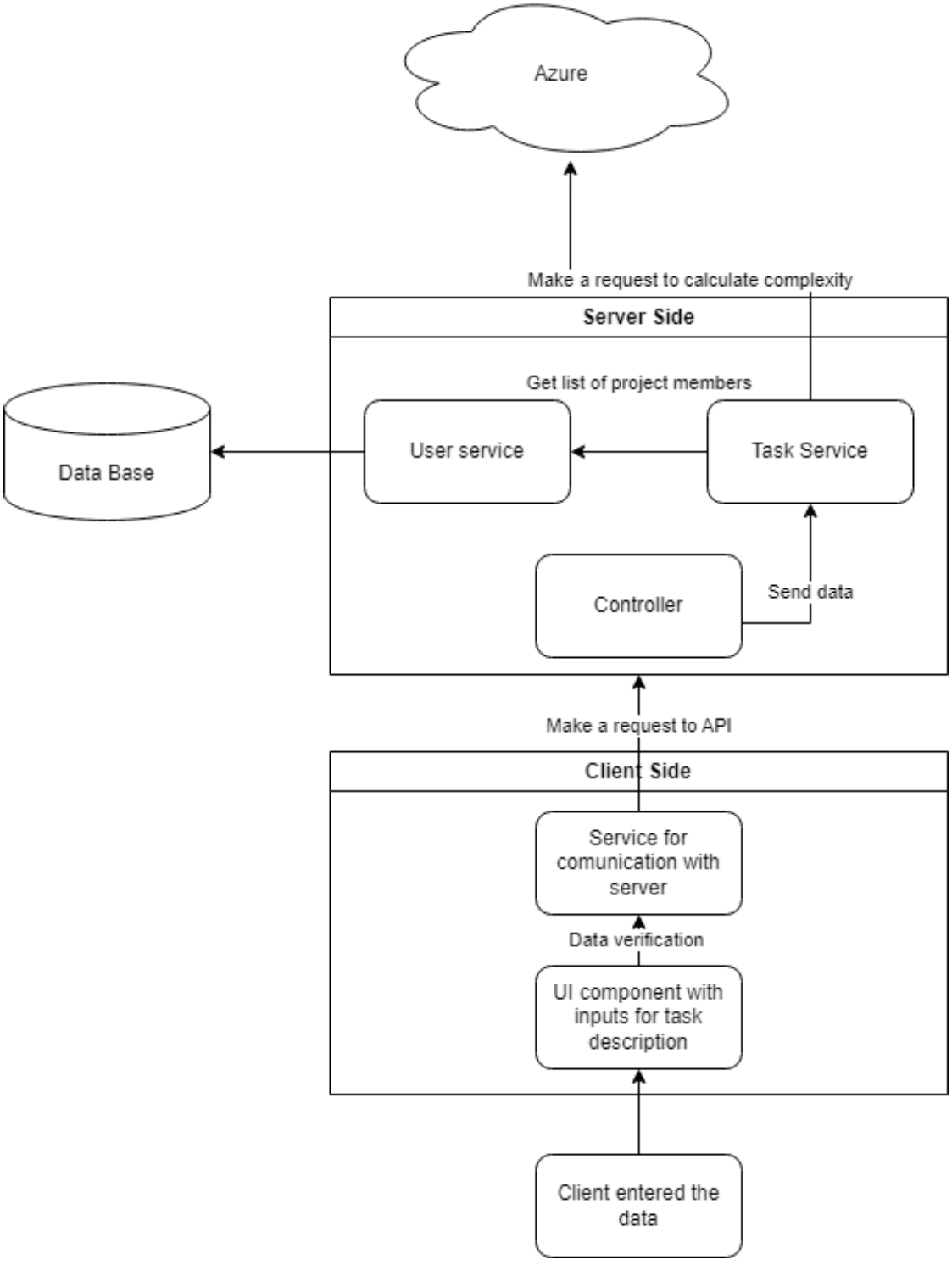

Рисунок В.3 – Архітектура додатку

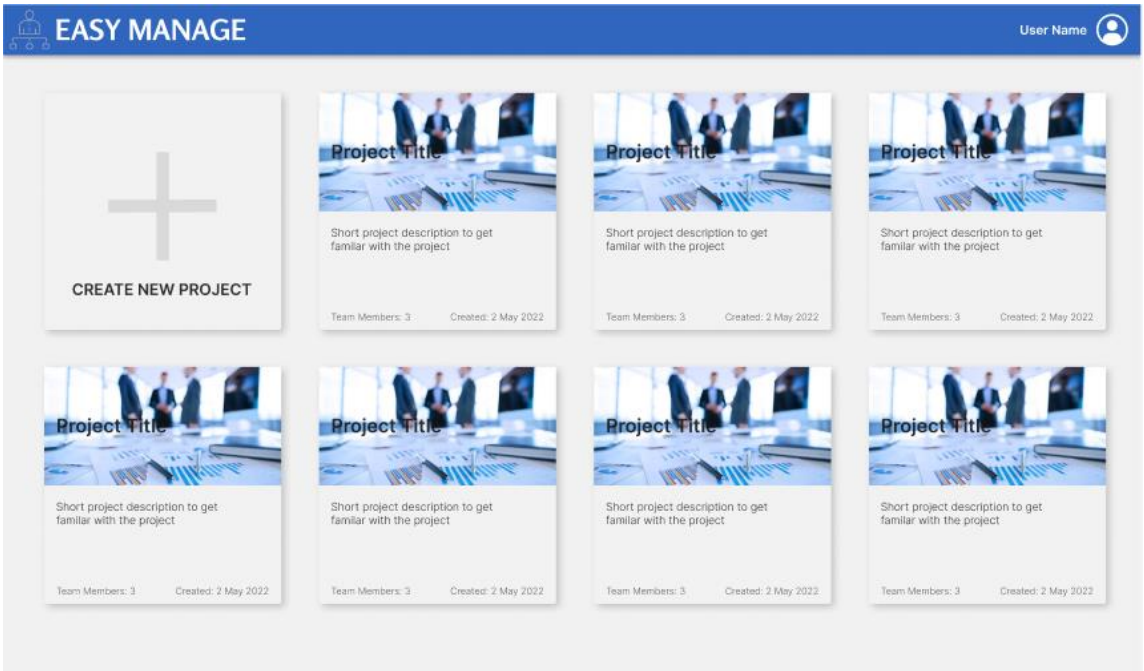

Рисунок В.4 – Вікно зі списком проектів доступних користувачу

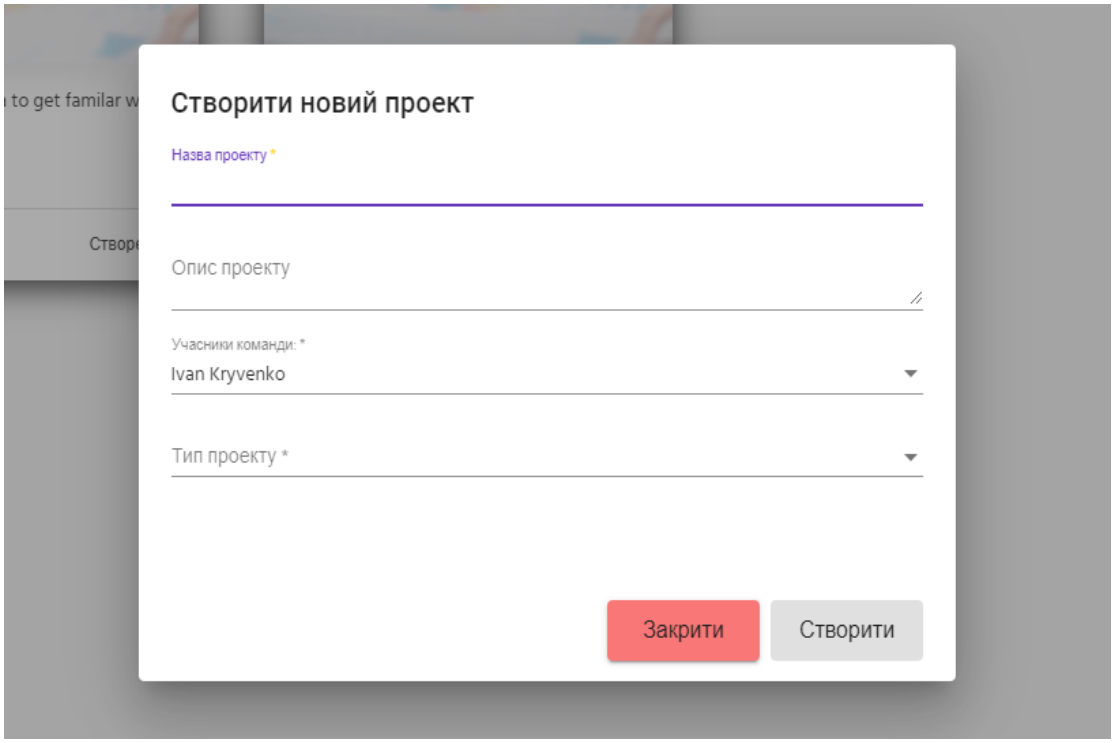

Рисунок В.5 – Модальне вікно створення проекту

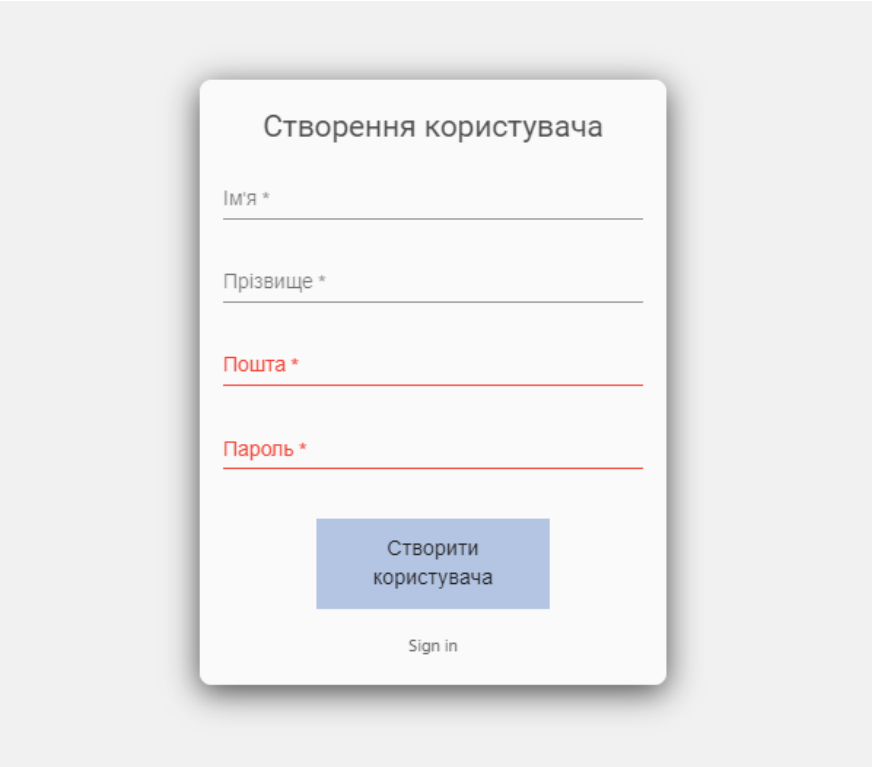

Рисунок В.6 – Вікно реєстрації користувача

Додаток Г (обов'язковий)

Інструкція користувача

1. Для початку роботи з додатком необхідно ввійти в систему, якщо користувач вже існує, або створити нового користувача, вікно створення нового користувача зображено на рисунку нижче:

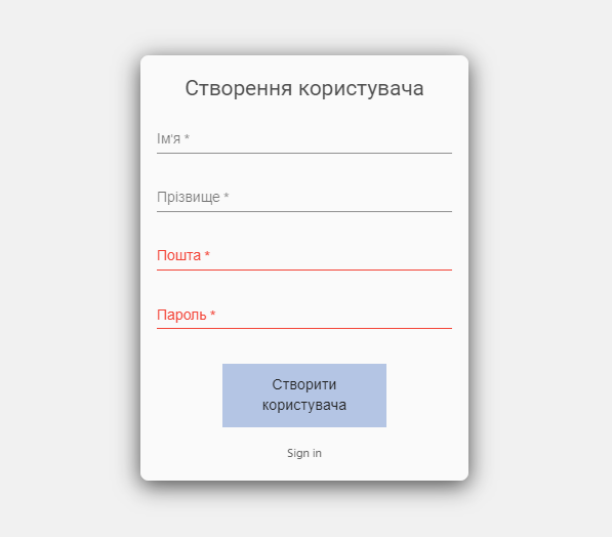

Рисунок Г.1 – Вікно створення нового користувача

2. Після потрапляння до головного меню, перед користувачем буде список проектів, в яких він бере участь, але якщо користувач новий, в нього буде пустий екран.

3. Користувач може створити новий проект або бути запрошеним до існуючого проекту.

4. Після того, як буде створений проект, користувачу необхідно обрати його зі списку проектів, список проектів зображений на рисунку Г.2.

5. Після вибору проекту, користувач може створити задання або обрати одне з наявних завдань, які знаходяться на екрані проекту.

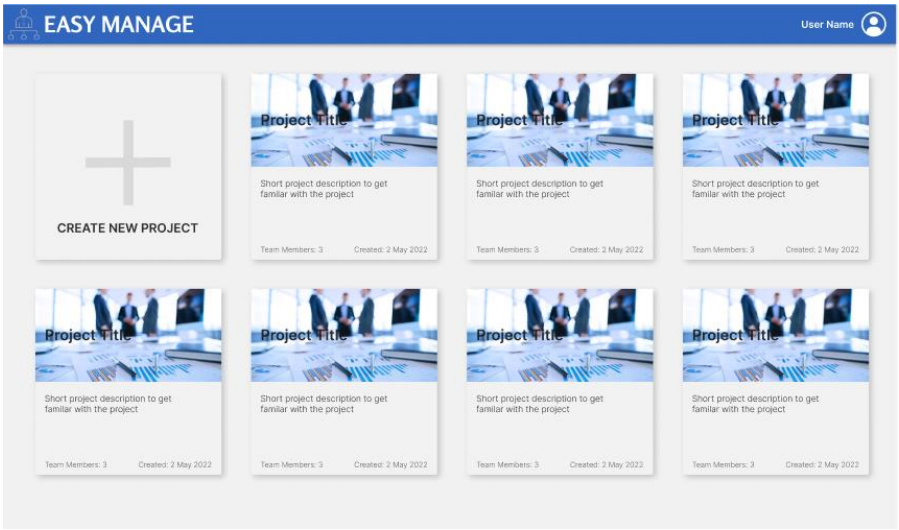

Рисунок Г.2 – Вікно зі списком проектів користувача

6. Для створення нового завдання необхідно натиснути на знак «+».

7. Потім необхідно додати короткий опис завдання.

8. Для активізації функціоналу аналізу даних, необхідно натиснути на кнопку «Аналізувати завдання», після цього пройде невеликий проміжок часу, під час якого, додаток проаналізує складність завдання та порекомендує працівника, що найкраще підходить для виконання цього завдання.

9. Після створення завадння, користувач може переміщати його між колонками, що відповідають статусу завдання.

10. Для виходу з системи, необхідно натиснути на значок користувача в верхньому правому кутку екрану і обрати опцію «Вийти».