Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі

Виконав: студент II курсу
групи ___3ПІ−22м спеціальності
121 – Інженерія програмного забезпечення
Ковальський С. В.

Керівник: к.т.н., асистент каф. ПЗ Тужанський С. Є.
«11» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Кожем'яко А. В.
«11» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«11» грудня 2023 р.

ВНТУ – 2023

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 19 » вересня 2023 р.

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

Ковальському Сергію Валерійовичу

1. Тема роботи – метод і програмний засіб оптимізації ресурсів на проектах у ІТ.

Керівник роботи: Тужанський Станіслав Євгенович, к.т.н., асистент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи - __5 грудня 2023 р.__

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова програмування TypeScript, операційна система Windows, використання фреймворку NestJS, фреймворку ReactJS та існуючих методів оптимізації ресурсів.

4. Зміст текстової частини: вступ, аналіз предметної області, аналіз існуючих методів оптимізації ресурсів, аналіз існуючих програмних засобів оптимізації ресурсів, порівняння існуючих висновки та постановка задачі, проектування методу для оптимізації ресурсів на проектах, вибір методу автентифікації у програмному засобі, обґрунтування вибору технологій та середовища розробки, проектування бази даних, проектування серверної частини, проектування користувацького інтерфейса, висновки, впровадження сервісу автентифікації, розробка серверної частини програмного засобу, розробка інтерфейсу користувача і тестування, висновки, перелік посилань, додатки.

5. Перелік ілюстративного матеріалу: блок-схема алгоритму; діаграми залежностей, зображення вікон програмного засобу, тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Тужанський С. Є., д.т.н., асистент кафедри ПЗ	19.09.23	05.12.23
4	Причепа І.В., к.е.н., доцент каф. ЕПВМ	20.11.23	04.12.23

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітк
1	Аналіз предметної області	20.09.2023 – 10.10.2023	вик
2	Проектування програмного засобу для оптимізації ресурсів	10.10.2023 – 30.10.2023	вик
3	Програмна реалізація та впровадження методу оптимізації	01.11.2023 – 30.11.2023	вик
4	Економічна частина	21.11.2023 – 01.12.2023	вик

Студент _____ **Ковальський С.В.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Тужанський С.Є.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.912.032.26

Ковальський С. В. Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі.

Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023, 148 с.

На укр. мові. Бібліогр.: 28 назв; рис.: 40; табл. 13.

У магістерській кваліфікаційній роботі наведено детальний аналіз методів оптимізації ресурсів і програмних засобів управління проектами у ІТ. Обгрунтовано доцільність створення методу оптимізації та програмного засобу для його реалізації, який оптимізує залучення ресурсів на проєкти.

Магістерська робота містить аналіз алгоритмів оптимізації ресурсів з різних сфер, а також існуючих програмних засобів.

У роботі розроблено програмний засіб, який використовує розроблений метод оптимізації.

Наукова новизна дослідження методу, спеціально адаптованого до потреб управління ресурсами на проектах у ІТ сфері. Цей метод включає в себе нові підходи до обрання та використання ресурсів та моніторингу ресурсів.

Отримані в магістерській кваліфікаційній роботі результати можна впровадити у ІТ компаніях для оптимізації ресурсів на проектах.

Ключові слова: оптимізація ресурсів, ресурси компанії, програмний засіб, веб технології.

ANNOTATION

Kovalskyi S. V. Method and software tool for optimizing resources on IT projects. Vinnysia: VNTU, 2023, 148 p.

In Ukrainian language. Bibliographer: 28 titles; fig.: 40; tabl. 13.

The master's qualification work provides a detailed analysis of resource optimization methods and IT project management software tools.

The expediency of creating an optimization method and a software tool for its implementation, which optimizes the attraction of resources to projects, is substantiated.

The master's thesis contains an analysis of resource optimization algorithms from various fields, as well as existing software tools.

In the work, a software tool was developed that uses the developed optimization method.

The scientific novelty of researching a method specially adapted to the needs of resource management on projects in the IT field. This method includes new approaches to resource selection and use and resource monitoring.

The results obtained in the master's qualification work can be implemented in IT companies to optimize resources on projects.

Keywords: resource optimization, company resources, software tool, web technologies.

ЗМІСТ

ВСТУП

Актуальність теми. Інформаційні технології (ІТ) в сучасному світі відіграють вирішальну роль у розвитку бізнесу та суспільства. Проекти в ІТ галузі здійснюються різноманітними компаніями для розробки програмного забезпечення, створення веб-сайтів, імплементації інформаційних систем тощо. Однак, управління ресурсами на таких проектах завжди є складним завданням, виконання якого вимагає ефективних методів та програмних засобів для оптимізації витрат і забезпечення успішності їх реалізації.

Великі ІТ проекти вимагають значних інвестицій у людські ресурси, обладнання та інфраструктуру. Здатність оптимізувати такі ресурси може значно зменшити витрати і збільшити прибутковість проектів. Отже, розробка методів та програмних засобів для оптимізації ресурсів у ІТ галузі має велике економічне значення.

ІТ компанії постійно шукають способи покращити ефективність та знизити витрати через високу конкрурентність на ринку. Оптимізація ресурсів може надати перевагу, отже є завжди актуальною.

Дослідження з оптимізації ресурсів на ІТ проектах має структурований характер і може включати аналіз методів управління проектами, ефективності використання людських ресурсів, використання технологій автоматизації і моніторингу, а також розробку програмних рішень. Ця структурованість дозволяє здійснювати дослідження системно та послідовно.

Результати дослідження можуть мати практичний вплив на ІТ індустрію, допомагаючи підприємствам поліпшити ефективність управління проектами, знизити ризики та витрати, а також збільшити якість та швидкість виконання проектів.

Таким чином, тема магістерської кваліфікаційної роботи "Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі" є актуальною і перспективною для підвищення конкурентоспроможності сучасних компаній у ІТ секторі.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності управління проектами у ІТ галузі за рахунок удосконалення методів оптимізації використання ресурсів у таких проектах.

Основними завданнями дослідження є:

- провести аналіз досліджень та публікацій, що стосуються управління ресурсами у ІТ проектах, існуючих методів та інструментів оптимізації ресурсів;
- удосконалити метод оптимізації використання ресурсів у ІТ проектах з урахуванням сучасних потреб і вимог, сформованих на основі проведеного опитування фахівців та керівників проектів.
- розробити програмний засіб для аналізу та оптимізації використання ресурсів ІТ проекту на основі запропонованого методу;
- провести тестування розробленого програмного засобу на реальних проектах в ІТ галузі, оцінити його ефективність та продуктивність за допомогою ключових показників;
- проаналізувати отримані результати тестування та зробити висновки щодо ефективності та перспектив використання розробленого методу та програмного засобу;
- на основі отриманих результатів розробити рекомендації для підприємств ІТ галузі щодо оптимізації ресурсів у проектах перспективи подальших досліджень в цій області.

Об’єкт дослідження – процес оптимізації використання ресурсів (людські ресурси, бюджет) у проектах, які здійснюються в галузі ІТ.

Предмет дослідження – метод і програмні засоби оптимізації використання ресурсів у ІТ проектах.

Методи дослідження. У роботі використовувались емпіричні методи досліджень, лінійна алгебра, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, теорії кейс-навчання та тестування програмних засобів.

Новизна роботи. Дістав подальшого розвитку метод оптимізації ресурсів у IT проектах з використанням жадібних алгоритмів, автоматизованого планування та моніторингу, який за відповідними критеріями (тип задачі, перелік технологій, дати виконання) дозволяє більш точно розподіляти людські ресурси в межах таких проектів, зменшуючи витрати і ризики при їх реалізації.

Практичне значення одержаних результатів. Розроблений програмний засіб може бути впроваджений у IT компанії для покращення управління людськими ресурсами, що у свою чергу оптимізує бюджетні надходження та зменшує час на опрацювання командного складу.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародній науково-практичній інтернет-конференції "Електронні інформаційні ресурси: створення, використання, доступ 2023".

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Управління IT-ресурсами передбачає управління ресурсами організації, які включають апаратне забезпечення, програмне забезпечення, програми та мережі [1].

Управління IT-ресурсами є критично важливим компонентом управління IT-послугами (ITSM) і включає в себе кілька процесів, таких як керування активами та конфігураціями послуг, управління потужністю та доступністю.

З розвитком дистанційного навчання як у школах так і у закладах вищої освіти з'явилось багато навчальних онлайн ресурсів, які позитивно впливають на розвиток фахівців через доступність великої кількості матеріалів у вільному доступі мережі інтернет, в тому числі вузькоспеціалізовані освітні проекти. Отже, витрачаючи невеликі кошти, компанія може значно покращити своє становище на загальному ринку праці для створення більш цінних і конкурентних проектів [2].

Основною метою управління IT-ресурсами є оптимізація використання IT-ресурсів і забезпечення їх відповідності цілям і завданням організації.

Це включає визначення та визначення пріоритетів IT-проектів, розподіл ресурсів, а також моніторинг і контроль витрат.

Програмні застосунки і різні методи оптимізації ресурсів на проектах та загалом у IT галузі у сучасному світі технологій є невід'ємною частиною їх життєвого циклу та збільшення прибутковості компаній.

Оскільки задача по оптимізації ресурсів на проектах у IT галузі є поширеною проблемою багатьох компаній слід проаналізувати доступні програмні застосунки які пропонуються у мережі інтернет.

Аналіз може включати вибір та оцінку різних програмних інструментів для моніторингу та управління ресурсами проектів, такі як інструменти для відстеження робочого часу, аналізу витрат, та вирішення завдань усередині команд. Наприклад, використання відслідковування часу працівника може допомогти знизити непродуктивний час, залучити працівника на проекти, де

не потрібна повна зайнятість тим самим збільшити прибутковість проектів та загалом самої компанії. Окрім цього, слід розглянути можливість впровадження штучного інтелекту та аналітики даних для покращення процесів управління ресурсами на проектах та у IT галузі загалом. Використання інноваційних рішень може стати ключовим фактором досягнення успіху та динамічного розвитку з розширення штату та збільшення прибутковості у найкоротші терміни. Розглянемо залежність на рисунку 1.1.

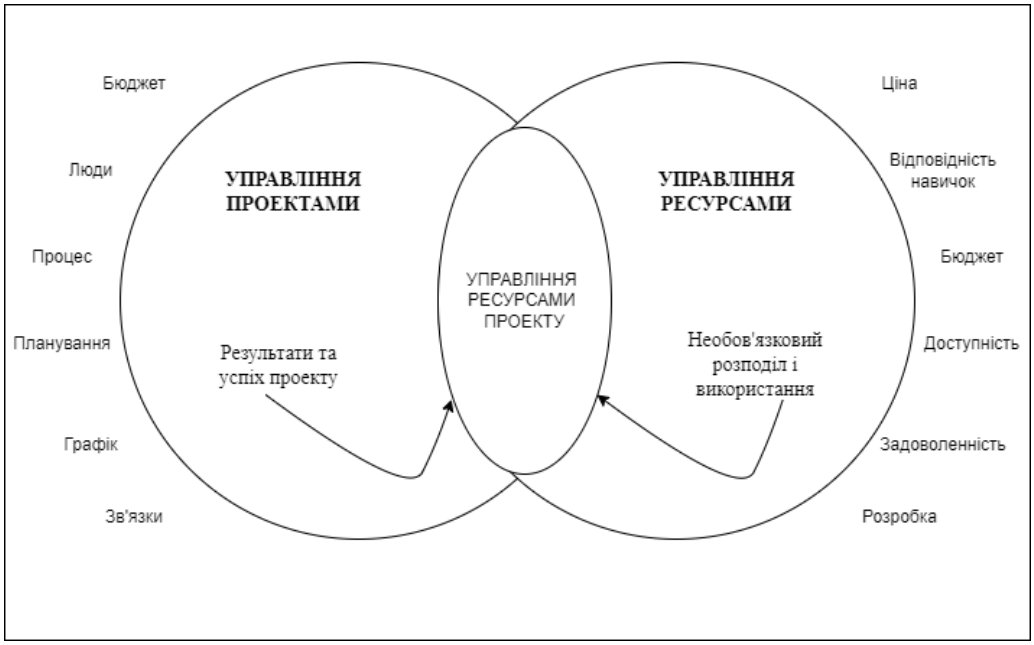


Рисунок 1.1 – Залежність розподілення ресурсів

Однією з головних передумов ефективного залучення нових проектів є аналіз попиту на ресурси з наявною пропозицією та вчасне вжиття заходів для їх узгодження. Іншими словами, перш ніж додати новий проект у своє середовище, ви повинні переконатися, що доступних ресурсів достатньо для виконання роботи. Відсутність планування ресурсів може призвести до затримок або необґрунтованого найму, що призведе до додаткових витрат.

Щоб ефективно керувати проектами та їхніми ресурсами, потрібно працювати з різноманітними даними: наприклад, компетенціями співробітників, доступністю, спроможністю, продуктивністю ресурсів, прогресом роботи, використанням бюджету тощо. Також важливо

відстежувати споживання матеріальних ресурсів якщо вони необхідні для завершення проекту. Якщо ці дані не централізовані (зберігаються в різних місцях), регулярно оновлюються та легко доступні за потреби, оптимізувати використання ресурсів буде досить важко.

1.1 Аналіз методів оптимізації ресурсів

Зі стрімким розвитком технологій змінюється і попит на навички працівників. Відповідно до Звіту про майбутнє робочих місць [3], до 2025 року 85 мільйонів робочих місць можуть бути витіснені внаслідок зміни розподілу праці між людьми та машинами, і можуть з’явитися 97 мільйонів нових ролей, які будуть більш адаптовані до нового розподілу праці між людьми. машини та алгоритми. Це означає, що потреба в перекваліфікації та/або підвищенні кваліфікації співробітників зростає експоненціально.

IT галузь є однією з найбільш швидкозростаючих і конкурентоспроможних галузей економіки. Для того, щоб бути успішним у цій галузі, підприємствам необхідно оптимізовувати свої людські та технічні ресурси.

Методи оптимізації можна розділити на дві категорії:

- оптимізація людських ресурсів;
- оптимізація технічних ресурсів.

Оптимізація людських ресурсів у IT галузі передбачає ефективне використання навичок і знань працівників. Для цього підприємства можуть використовувати такі методи:

- планування кар’єри або roadmap. Підприємства за допомогою менторства мають розробляти програми планування розвитку для своїх працівників, щоб визначити можливості для росту спеціаліста у своїй галузі;
- оцінка ефективності праці. Під оцінкою ефективності можна розглядати можливості для перерозподілення або залучення ресурсів до нових чи існуючих проектів;
- розвиток персоналу. Інвестування в навчання та розвиток працівників,

щоб вони могли залишитися конкурентоспроможними на швидкозростаючому ринку праці.

Оптимізація технічних ресурсів передбачає ефективне використання програмного забезпечення, апаратного забезпечення та інших технічних ресурсів. Для цього підприємства можуть використовувати такі методи:

- вибір оптимального обладнання. Необхідно ретельно обирати обладнання, яке відповідає потребам співробітників на тих чи інших проєктах відносно їх величини, використання технічних ресурсів обладнання задля оптимізації їх повсякденних рутинних процесів на проектах;
- впровадження автоматизації. Компанії можуть автоматизовувати деякі завдання, щоб звільнити час для працівників;
- забезпечення безпеки. Компанія повинна впровадити заходи безпеки для захисту своїх технічних ресурсів від злому та інших загроз.

Окремо можна виділити два найбільш використовуваних методів оптимізації ресурсів:

Вирівнювання ресурсів (resource smoothing) - це процес, у якому кількість ресурсів є фіксованою, але дати початку та завершення проекту змінюються відповідно до потреб для отримання якісних результатів. Цей метод оптимізації ресурсів допомагає керівникам проектів збалансовано, враховуючи попит на певний ресурс і його доступність. Це також допомагає зменшити виснаження співробітників і розподілити завдання, для виконання яких вони мають кваліфікацію та досвід.

До переваг вирівнювання ресурсів можна віднести:

- його здатність мінімізувати дефіцит;
- ефективне використання ресурсів;
- запобігає надмірному розподілу;
- забезпечує успішне надання якісних результатів.
- до недоліків можна віднести:
- може спричинити затримки в інших проектах;
- перевиконання бюджету;

- може вимагати додаткових ресурсів.

Другим є метод згладжування ресурсу (resource leveling). Як метод згладжування так і саме вирівнювання, спрямовані на досягнення цілей проекту при збереженні певних обмежень. Але вони принципово відрізняються підходом.

Переваги згладжування ресурсу:

- успішне та своєчасне виконання проекту;
- це ефективний і рентабельний спосіб використання ресурсів.

Недоліки згладжування ресурсу:

- менеджерам проектів доведеться відкласти певну роботу, щоб надати першочерговий пріоритет конкретному проекту;
- щоб запобігти затримкам, графік зазвичай мало гнучкий.

Використовуючи наведені методів та їх характеристики побудуємо порівняльну таблицю з зазначенням їх певних особливостей і відмінностей за критеріями.

Деякі з основних відмінностей включають (і не обмежуться ними):

Таблиця 1.1 - Використання методів оптимізації ресурсів

Параметр	Вирівнювання ресурсів	Згладжування ресурсів
Тривалість проекту	Фіксована	Чітко не визначена
Доступність ресурсів	Безлімітно	Фіксовано або лімітовано
Порядок виникнення	Виконується після вирівнювання ресурсів	Спочатку планується
Зміна у критичному шляху	Не дозволяється	Дозволяється
Коли необхідний	Ресурси розподілені нерівномірно	Ресурси розподілені надмірно

Для загальної оцінки методів розглянемо метод зворотного розподілу ресурсів можна використовувати з двома описаними вище методами, але передбачає підхід до нього з іншого боку. Ця техніка працюватиме у

зворотному порядку, тобто ви починаєте з останнього завдання або найважливішого завдання та повертаєте свій розклад назад.

Ця техніка працює, коли проект має фіксований крайній термін або є важливі завдання, які мають виконуватися у встановлені дати. Наприклад, якщо у вас є проект, який потребує експерта для одного завдання на критичному шляху, ви призначите завдання так, щоб експерт компанії міг працювати в цей конкретний день над заданим проектом.

Щоб подолати всі виклики та забезпечити ефективне впровадження методів оптимізації ресурсів, перш за все необхідно усвідомити, що не існує єдино вірного рішення, яке могло б однаково працювати для усіх підприємств. Оптимальний розподіл ресурсів - це процес заснованих на вимогах, який також триває природньо. Постійні дослідження можуть призвести до забезпечення вдосконалення впроваджених методів оптимізації ресурсів.

1.2 Аналіз програмних засобів оптимізації ресурсів

Під програмними засобами можна розглядати застосунки для десктопу, мобільних пристроїв які мають у собі інформацію про співробітників, проєкти, фінансування, найм нових співробітників, інше. Такі програми називаються ERP системами.

ERP системи (Enterprise Resource Planning) - це комплексні програмні продукти, які автоматизують основні бізнес-процеси підприємства [4]. Вони дозволяють об'єднати в єдину систему всі дані та процеси, пов'язані з виробництвом, продажем, фінансами, управлінням персоналом та іншими сферами діяльності.

На ринку існує безліч ERP систем від продуктових компаній, які пропонують універсальні рішення для вирішення проблем компаній і являють собою великі системи з функціональністю яка надається пакетним рішенням і часто не має потреби у настільки комплексному рішенні. Однією з проблем є конфіденційність внутрішніх даних співробітників, їх зарплати або оплати виконаних робіт у залежності від способу співпраці компанії та співробітника,

інформації про проєкти та їх фінансового обліку. Тому великі компанії можуть віддати перевагу розробці внутрішньої ERP системи своїми силами витрачаючи більші кошти на ці потреби віддаючи перевагу захищеності над фінансовою вигодою.

Розглянемо одну з доступних на ринку ERP систему від компанії Gauzy. Вперше зайшовши у демо одразу видно перенасиченість пунктів меню, які мають у собі ще пункти підменю у яких дуже легко заплутатись навіть при чіткому розділенні на певні модулі управління компанією. На рисунку 1.2 зображено головний екран системи Gauzy.

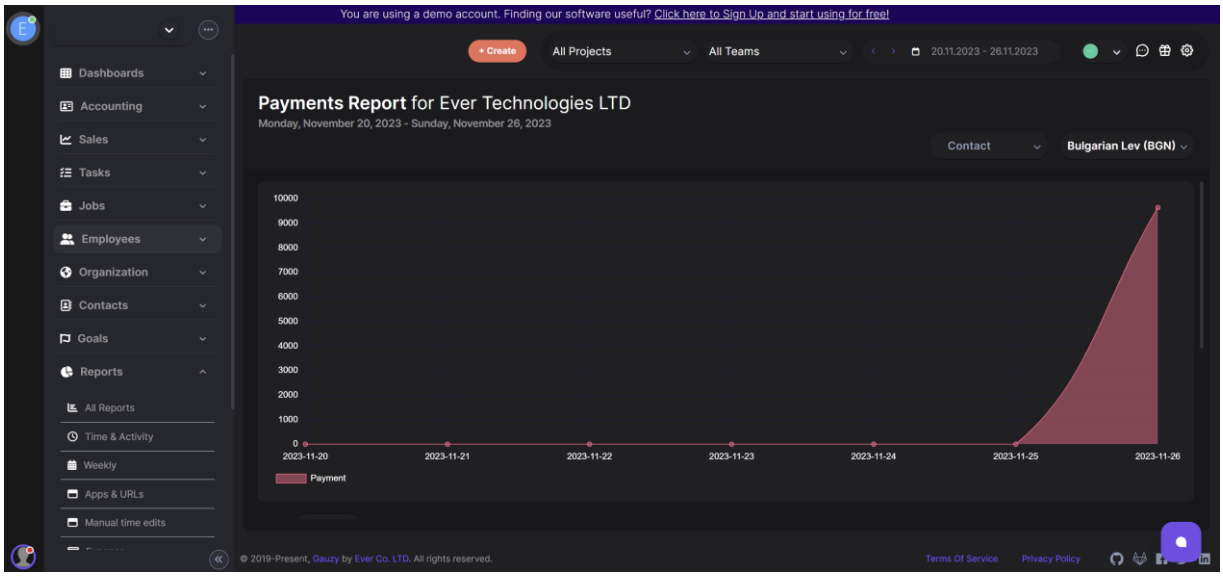


Рисунок 1.2 – Скріншот ERP системи Gauzy

Розглянемо такий пункт як управління співробітниками, їх додавання у систему та операції з фінансами зображені на рисунку 1.3.

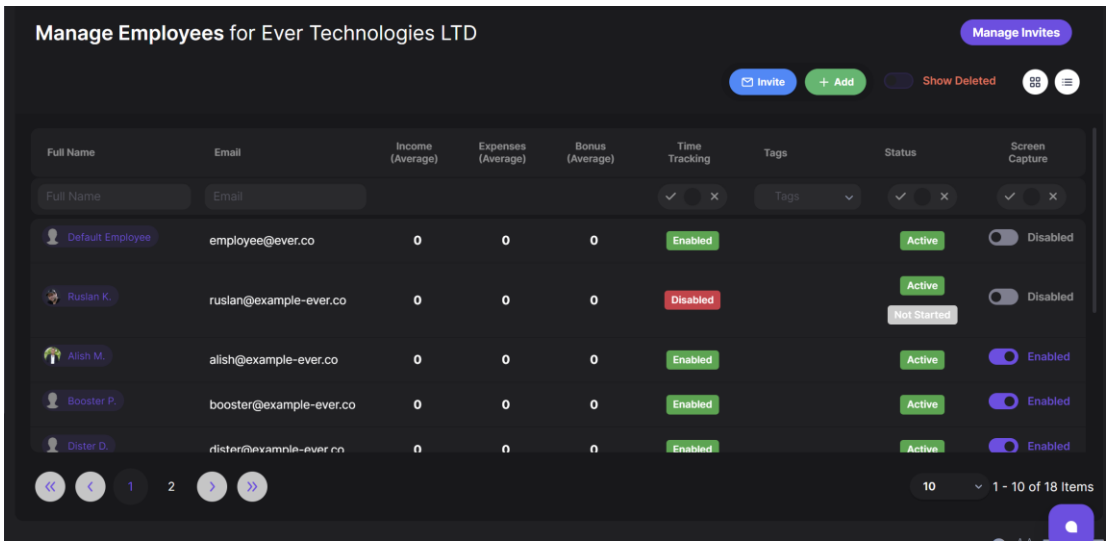


Рисунок 1.3 – Список адміністрування співробітників Gauzy

Детально вивчивши всі пункти меню управління, можна зробити висновок, що схожого або наближеного функціоналу щодо оптимізації ресурсів немає. Загалом система представлена, як потужний інструмент управління компанією загалом, але все побудовано навколо стандартних даних, які потрібні для функціонування компанії.

Розглянемо, ще одну з систем під назвою Project Manager. На рисунку 1.4 зображено приклад створення завдань, які потрібно вирішити для данного проєкта.

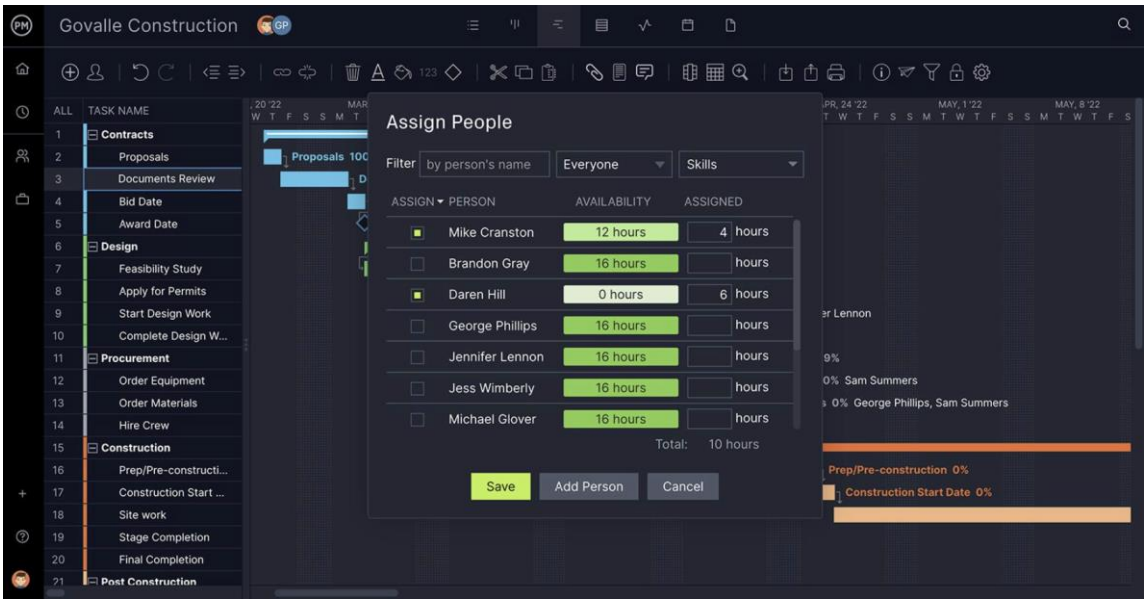


Рисунок 1.4 – Скріншот системи Project Manager

Можна побачити детальне налаштування проєкту, але це вже проєкт який має компанія, та підписані всі контракти з замовником. Ця сторінка відповідає вже за наповнення відомого проєкта, з розподіленими задачами чітко за годинами та між певною кількістю співробітників, які залучення до даного проєкта.

На рисунку 1.5 зображена алокація ресурсів по задачах і має вигляд відслідковування часу який витратив залучений співробітник на кожну з задач. Система пропонує ручне залучення співробітників та відслідковування виконання тих чи інших поставлених завдань.

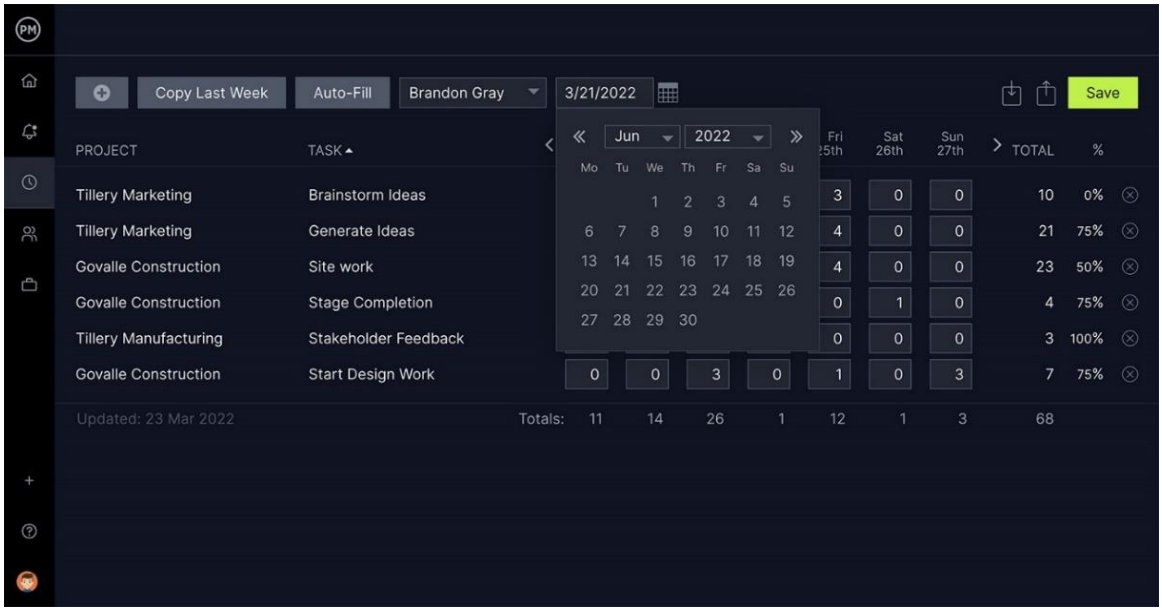


Рисунок 1.5 – Скріншот системи Project Manager

У підсумку маємо декілька систем, які мають багато зручного функціоналу для адміністрування компанією, але не мають функціональності, яка могла б рекомендувати по заданим параметрам задач кого з співробітників залучити на новостворений проєкт та візуалізувати потенційну пропозицію по використанню ресурсів.

1.3 Порівняння існуючих алгоритмів оптимізації ресурсів

Оптимізація ресурсів - це процес максимізації використання обмежених ресурсів (таких як час, гроші, енергія, обладнання тощо) для досягнення бажаного результату. Існує багато алгоритмів та стратегій для оптимізації ресурсів у різних сферах, таких як виробництво, логістика, інформаційні технології, транспорт, тощо [5]. Наведемо деякі загальні типи алгоритмів та їх характеристики:

- генетичні алгоритми - вони моделюють процес еволюції, використовуючи поняття генетичного коду та операторів як еволюційних механізмів. Робочий принцип включає створення початкової популяції, вибір батьків для створення нових особин, застосування операторів кросовера та мутації для створення нових потомків;

- алгоритми оптимізації зграї частинок. Алгоритм моделює поведінку зграї частинок, які рухаються в просторі пошуку з метою знаходження оптимального рішення. Кожна частинка зберігає своє власне найкраще рішення та найкраще рішення в межах зграї, а також швидкість та позицію;

- лінійне програмування вирішує задачу оптимізації, де функція, яку потрібно максимізувати чи мінімізувати, є лінійною функцією цільової функції. Обмеження також є лінійними нерівностями чи рівностями;

- градієнтні методи використовують градієнт (вектор похідних) цільової функції для керування процесом оптимізації. Починаючи з початкової точки, алгоритм рухається вздовж антиградієнта з метою зменшення значення цільової функції;

$$grad\ F = (\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \dots \frac{\partial F}{\partial x_n}) \qquad (1.1)$$

- еволюційні стратегії базуються на ідеї еволюції та використовують поняття генетичних операцій для оптимізації. Включають генерацію популяції рішень, оцінку їхньої придатності та вибір нових рішень для створення

наступної популяції;

- методи оптимізації зграї частинок. Зараження натхненням від поведінки мурах у природі, де мурахи залишають феромони для маркування шляхів до їжі. Мурашки обирають шляхи з більшою концентрацією феромонів, збільшуючи ймовірність вибору цих шляхів іншими мурашками;

- жадібні алгоритми приймають локально оптимальне рішення на кожному кроці, сподіваючись, що це призведе до глобально оптимального результату. Часто використовуються в задачах, де необхідно максимізувати або мінімізувати якісний показник.

$$E_i = E[C1 + \cdots + Cn] = \sum_{i=1}^{n} = 1/i \qquad (1.2)$$

По вище наведений алгоритмам зробимо порівняльну таблицю переваг і недоліків деяких алгоритмів. Одним з основних параметрів обрання буде простота його використання і впровадження у розроблений метод та відповідність результатів з 100% вірогідністю не є проблемою для вирішення поставленої задачі. Порівняльна характеристика наведена у таблиці 1.2.

Таблиця 1.2 – Порівняння переваг і недоліків алгоритмів оптимізації ресурсів

Критерій	Жадібний алгоритм	Генетичні алгоритми	Лінійне програмування	Градієнтні методи
Переваги	Простота і швидкість виконання	Здатність пристосовуватися до складних функцій	Ефективність для лінійних завдань, простота	Ефективність для гладких функцій, швидкість збіжності
		Здатність уникати застрягання в локальних мінімумах	Можливість розв'язання широкого спектру задач	Збіжність до локального оптимуму, застосування в різних сферах
		Випадкова генерація нових рішень	Використання методів симплексу для розв'язання	
Недоліки	Не завжди гарантує	Велика обчислювальна	Обмежена застосовність для	Чутливість до початкових умов,

	найкращий глобальний результат	складність	нелінійних задач	можливість застрягання в локальних мінімумах
		Вимагає великої кількості обчислень	Чутливість до даних вхідних параметрів	Чутливість до початкових умов, можливість застрягання в локальних мінімумах
		Може бути неефективним для простих задач	Обмежена ефективність для складних неструктурованих задач	Висока обчислювальна складність

Жадібні алгоритми можуть бути вкрай ефективними у випадках, коли вони забезпечують достатньо задовільний результат за короткий час. Вони є найкращим вибором для оптимізації ресурсів на проектах у IT сфері, зокрема через їхню простоту і швидкість виконання. Вони є ефективними в ситуаціях, коли глобальна оптимальність не є критичною, і можна отримати достатньо чіткі результати шляхом локального вибору оптимальних рішень на кожному етапі.

1.4 Висновки та постановка задачі

Провівши аналіз у результаті маємо таку мету роботи: розробка програмного засобу на основі досліджених методів оптимізації ресурсів, спрямованих на оптимізацію використання ресурсів у IT.

Для досягнення мети з підвищення ефективності управління проектами в IT галузі шляхом максимально можливого оптимального розподілення людських ресурсів по проектах, зменшення витрат і ризиків, а також забезпечення успішності виконання проектів.

У даному розділі було здійснено аналіз існуючих і широко використовуваних методів оптимізації ресурсів. Детально розглянуті їх відмінності і способи запровадження у компаніях, наведено їх переваги та недоліки.

Було проведено дослідження по доцільності розробки спеціалізованого програмного забезпечення з використанням таких методів як цільових у оптимізації ресурсів.

У підсумку, досягнення поставленої мети є актуальною проблемою IT галузі і має високий практичних рівень застосування для компаній з різною кількістю ресурсів. Тому необхідно розробити метод оптимізації ресурсів та спроектувати основні модулі програмного засобу інтегрувавши розроблений метод у нього.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ РЕСУРСІВ

2.1 Проектування методу для оптимізації ресурсів на проектах

Оптимізація ресурсів на проектах є важливим завданням кожної компанії для забезпечення ефективного використання бюджету та ресурсів, що у підсумку впливає на загальне становище на ринку праці. Розглянемо декілька методів, які враховують взаємозв’язки різних ресурсів.

Варто переглянути завантаженість співробітників на проєктах та залучити ті ресурси які мають доступний час і додати їх на нові проєкти завдяки цьому збільшивши прибуток.

На оптимізацію ресурсів на проектах у сфері ІТ, впливає багато чинників як зовнішніх так і внутрішніх. Зазвичай, найбільша стаття витрат компаній є оплата праці.

Виходячи з цього можна запропонувати, можливі систематичні методі на основі “жадібних алгоритмів”.

Жадібні алгоритми – це ціле сімейство алгоритмів (можна назвати це як жадібний підхід або жадібне програмування). Основою їх є – вибрати оптимальне рішення на кожному конкретному кроці, незважаючи на кроки, які були зроблені або будуть зроблені після. Іншими словами, жадібний алгоритм робить локально оптимальний вибір, що призводить до глобального оптимального вибору.

Вхідними даними являються введені дані нового проекту у яких зазначено фінансові ресурси у вигляді бюджету, які виділені для реалізації, включаючи технічні вимоги у вигляді задач, термінів їх виконання та інше. Вхідні дані є ключовими для подальшого успіху проекту, оскільки визначають основні параметри, на основі яких буде розроблено метод і програмний засіб оптимізації ресурсів.

Наступна дія розподілення задач по напрямках фокусується на ефективному розподілі задач проекту серед команди відповідно до їхніх компетенцій та бази знань.

Зазначимо наступним залучення ресурсу для тестування відповідно до кількості розробників, він буде перераховуватись автоматично.

Далі потрібно отримати усіх доступних співробітників по заданим напрямках включаючи тестувальників, вони не будуть включені, як окремою задачею на тестування.

Перевіряючи кожен наступний напрямок по задачі потрібно заповнити його ресурсом, тим хто буде його реалізовувати по заданим критеріям беручи до уваги його позицію і пріоритетність розробника чи іншого спеціаліста.

Головним завдання є коректний розрахунок фінансового блоку на основі годин, які потрібно витрати для реалізації задачі та погодинної ставки потенційного співробітника. У випадку вільного місця у бюджеті і часу на задачі буде додано більше спеціалістів одного напрямку.

Перевіряючи кількість задач і залишок бюджету буде наповнюватись команда співробітників для виконання цього проекту згідно поставлених задач. При заповненні всіх позицій буде видано результат з підрахунком кожного співробітника і залишку бюджету, якщо такий є.

Блок-схема є графічним зображенням послідовності кроків у вигляді блоків різних форм, які з'єднані лініями. Це інструмент, що забезпечує зрозуміле відображення операцій та їх послідовності в системі. Кожен блок відображає конкретний етап або дію, яку слід виконати у процесі реалізації алгоритму (рис. 2.1).

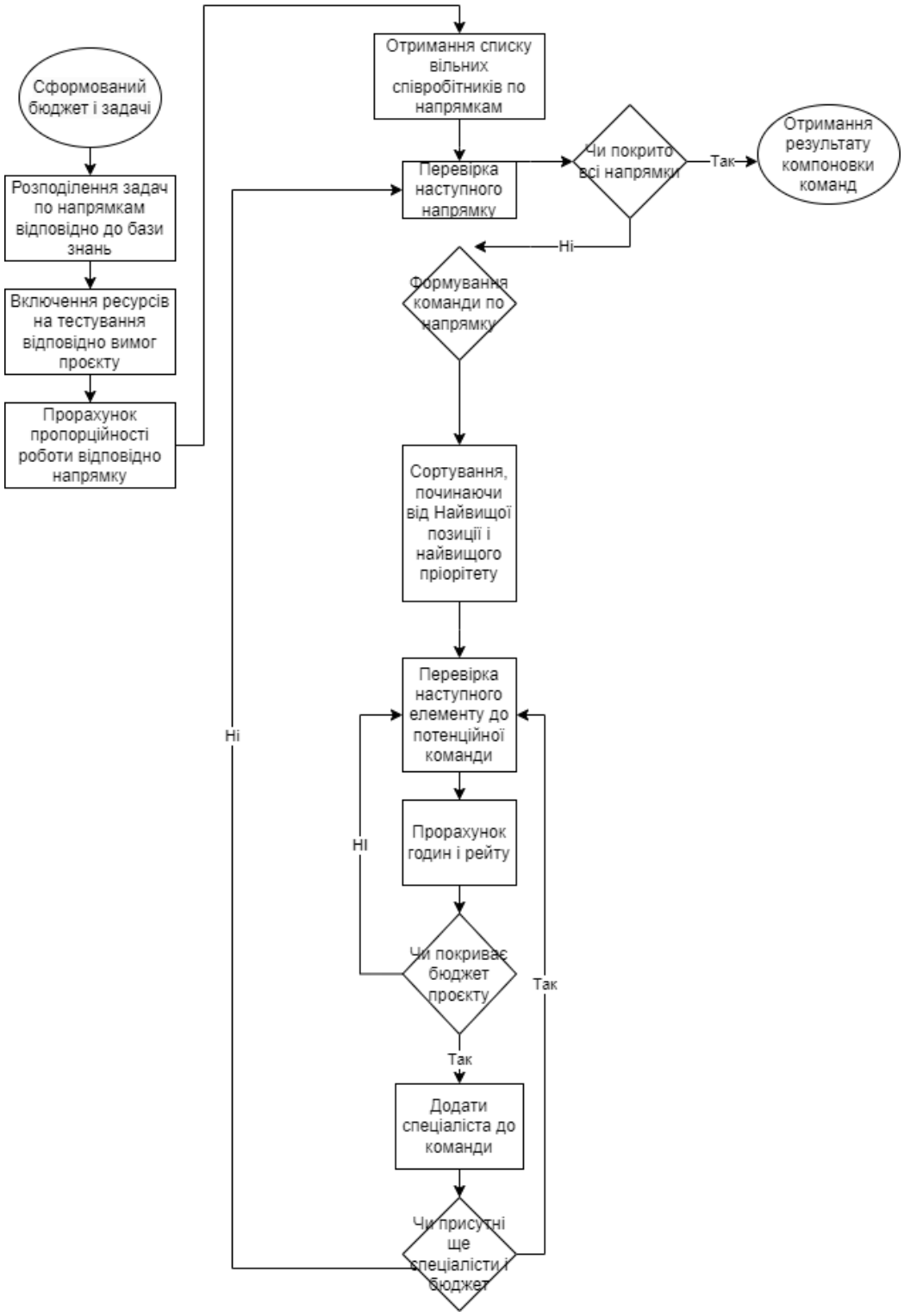


Рисунок 2.1 – Блок-схема алгоритму оптимізації ресурсів

Зображена схема показує покроковість дій, які виконуються для підбору оптимального командного складу по для залучення на проєкт на основі введених даних і існуючих даних співробітників. Можна виділити такі основні етапи:

- початок – сформований бюджет проєкту і його задачі по типам;
- розбиття задач по напрямках та пропорційний розрахунок навантаженості;
- отримання доступних співробітників по напрямках та підбір відносно задач;
- сортування по критеріям та перевірка чи закритий напрям;
- додавання співробітника, якщо він відповідає критеріям;
- кінець – завершення алгоритму і отримання результату.

2.2 Вибір методу автентифікації у програмному засобі

Важливою частиною таких програмних засобів є збереження конфіденційності даних, доступ обмеженому переліку осіб які скоріш за все відносяться до менеджменту та які тісно пов’язані з обігом доходів компанії. Тому важливим кроком є вибір правильного і зручного методу автентифікації користувача до програмного засобу.

Автентифікація — це процес, який використовується для перевірки ідентичності користувача, системи або програми, щоб переконатися, що вони є тими, за кого себе видають. Це важливий аспект інформаційної безпеки та відіграє життєво важливу роль у захисті конфіденційних даних і ресурсів. Автентифікація зазвичай передбачає використання імен користувачів і паролів, токенів, біометричних даних або інших методів для підтвердження особи користувача [6].

Переваги автентифікації:

- безпека: Захищає від несанкціонованого доступу та порушень даних;
- відповідальність користувача: Допомагає відстежувати дії користувачів і вести контрольний слід;
- цілісність даних: гарантує, що дані залишаються точними та не змінюються неавторизованими користувачами;
- відповідність: Допомагає виконувати нормативні вимоги та вимоги відповідності.

Недоліки автентифікації:

- складність: впровадження систем автентифікації може бути складним і може потребувати ретельного проектування та обслуговування;
- перешкоди для користувачів: деякі методи автентифікації, особливо з високим рівнем безпеки, можуть спричинити перешкоди для користувачів, що ускладнить процес входу;
- ризики безпеці: якщо системи аутентифікації не реалізовані належним чином, вони можуть стати вразливими до різних загроз безпеці, таких як атаки грубої сили, фішинг тощо.

З доступних захищений і простих у впровадженні варіантів розглянемо такі рішення:

- Firebase Authentication;
- AWS Cognito.

Варіанти з Firebase Authentication та AWS Cognito є дешевшими з точки зору ціни на розробку, більш захищеними і зберігають багато часу на реалізацію.

Firebase Authentication – це служба, яка надається платформою Google Firebase і спрощує процес аутентифікації для розробників [7]. Ось деякі переваги використання автентифікації Firebase:

- проста інтеграція: Аутентифікація Firebase пропонує легку інтеграцію з такими популярними постачальниками ідентифікаційної інформації, як Google, Facebook, X (Twitter) тощо;
- багатофакторна автентифікація (MFA): Підтримує багатофакторну автентифікацію для додаткового рівня безпеки;
- масштабованість: Firebase — це безсерверна платформа, що дозволяє розробникам зосередитися на створенні програми, а Firebase піклується про масштабованість.

Автентифікація Firebase має щедрий безкоштовний рівень, і з вас стягується плата залежно від кількості перевірок і щомісячних активних користувачів (MAU) поза безкоштовним рівнем. Деталі ціни можуть

змінюватися, тому важливо перевіряти сторінку цін Firebase, щоб отримати найновішу інформацію.

Amazon Cognito — це повністю керована служба керування ідентифікацією та доступом, яку надає Amazon Web Services (AWS) [8]. Він розроблений для обробки автентифікації користувачів, авторизації та керування користувачами для веб- і мобільних програм.

У порівнянні з AWS Cognito обидва сервіси пропонують схожі функції, але вибір може залежати від конкретних випадків використання, знайомства з платформою та інших факторів. AWS Cognito є частиною ширшої екосистеми AWS, яка пропонує низку послуг, тоді як Firebase забезпечує більш спрощений досвід для розробників, які зосереджені на мобільних і веб-додатках.

AWS Cognito часто використовується в поєднанні з іншими службами AWS для створення безсерверних програм або традиційних веб- і мобільних програм. Це спрощує реалізацію безпечної аутентифікації користувачів і контролю доступу, дозволяючи розробникам більше зосереджуватися на створенні функцій, а не на керуванні складністю керування користувачами.

Порівняємо Firebase Authentication з AWS Cognito у таблиці 2.1.

Таблиця 2.1 – Порівняння Firebase Authentication з AWS Cognito

Особливість	Firebase Authentication	AWS Cognito
Автентифікація користувача	Підтримує електронну пошту/пароль, телефон, Google, Facebook, Twitter тощо.	Підтримує різні постачальники ідентифікаційної інформації, включаючи постачальників соціальних ідентифікаційних даних і власних постачальників ідентифікаційних даних.
Підтримка кількох платформ	iOS, Android, Web, Unity, C++	Розширена підтримка веб-, мобільних і серверних програм.
Інтеграція	Проста інтеграція SDK і бібліотеки інтерфейсу	Частина ширшої екосистеми AWS вимагає налаштування й конфігурації AWS.

Продовження таблиці 2.1

Постачальники ідентифікаційної інформації	Підтримує загальні постачальники соціальних даних (Google, Facebook тощо)	Широка підтримка різних постачальників ідентифікаційної інформації, включаючи соціальних постачальників і SAML.
Захищеність	Забезпечує безпечне зберігання паролів, відновлення облікових записів і захист від типових загроз безпеці.	Забезпечує різні функції безпеки, включаючи багатофакторну автентифікацію, шифрування тощо.
Стан автентифікації в реальному часі	Моніторинг і керування станом автентифікації в реальному часі.	Функції синхронізації в реальному часі з іншими сервісами AWS.
Ціна	Плата стягується на основі кількості перевірок і щомісячних активних користувачів за межами безкоштовного рівня.	Плата стягується на основі щомісячних активних користувачів (MAU) та інших факторів.
Кастомізація	Пропонує деякі налаштування, але є більш самовпевненим.	Забезпечує більшу гнучкість для налаштування відповідно до конкретних випадків використання.
Розширюваність	Firebase автоматично масштабується.	Частина AWS, успадковує масштабованість і надійність AWS.

Отже, порівнявши усі особливості оберемо Firebase Authorization. Додатково виходячи з інформації, що багато компаній мають власну корпоративну пошту з Gmail, є додаткова можливість заблокувати доступ тільки з корпоративної пошти.

2.3 Обґрунтування вибору технологій та середовища розробки

Вибір технологій для розробки є критично важливим, оскільки визначає продуктивність майбутнього програмного засобу, системи чи інструменту, масштабованість, надійність і швидкість розробки.

Для реалізації програмного засобу було обрано мову програмування TypeScript у зв’язці з NestJS фреймворком серверної частини який у свою чергу під собою використовує NodeJS та для клієнтської частини було обрано фреймворк ReactJS.

TypeScript – це мова програмування, яка компілюється у JavaScript [9-10] і була розроблена компанією Microsoft. Вона додає статичну типізацію до JavaScript, що робить код більш надійним і який легше підтримувати. Запозичує TypeScript найкраще з мови C# тим самим роблячи його більш універсальним інструментом як для клієнтської частини, так і для серверної.

Переваги у порівнянні з JavaScript:

- статична типізація: Однією з основних переваг TypeScript є можливість визначати типи для змінних, параметрів функцій і інших об'єктів. Це дозволяє виявляти помилки на етапі компіляції, забезпечуючи більш високий рівень надійності коду та полегшуючи рефакторинг;
- підтримка оголошень типів та інтерфейсів: TypeScript дозволяє оголошувати власні типи та інтерфейси, що полегшує розробку та забезпечує чіткіше визначення структури даних у коді;
- об'єктно-орієнтований підхід: TypeScript надає можливості об'єктно-орієнтованого програмування, включаючи класи, інтерфейси, успадкування та інші концепції, що спрощують структуру коду та його повторне використання;
- автоматичне визначення типів: TypeScript може автоматично визначати типи для деяких значень, що полегшує роботу з кодом та зменшує необхідність в явному указанні типів у деяких випадках;
- легша рефакторизація: Завдяки статичній типізації та великій підтримці інструментів розробки, рефакторизація коду в TypeScript може бути більш безпечною та ефективною;

- сумісність з JavaScript: TypeScript є розширенням JavaScript, тому всі існуючі бібліотеки та фреймворки, написані на JavaScript, можна використовувати у TypeScript без будь-яких змін.

NodeJS – це платформа, яка дозволяє створювати швидкі та масштабовані серверні додатки [11]. Ця платформа є надзвичайно популярною у сучасному світі розробки, так як має малий поріг входження, детальну документацію та велике комьюніті яке підтримує і розширює цю платформу.

Обравши NodeJS як основу серверу для спрощення роботи з ним і розширення існуючого інструментарію для швидкого створення і розширення серверної частини було обрано NestJS фреймворк який надає структурованість і певний зручний, шаблонний підхід для розробки, що у свою чергу впливає на покращення продуктивності, забезпечення безпеки та підвищує зручність розширюваності програмного засобу [12].

Ключові концепції NestJS:

- модулі: NestJS сприяє розбиттю додатку на модулі. Кожен модуль є найменшою одиницею в додатку та об'єднує в собі компоненти, контролери, сервіси тощо. Це сприяє високій ступеню модульності та перевикористання коду;
- контролери: Контролери обробляють HTTP-запити та визначають, які сервіси повинні бути викликані для виконання певної операції;
- сервіси: Сервіси містять логіку бізнес-логіки та обробку даних. Вони можуть бути викликані контролерами для виконання певних завдань;
- middleware: NestJS підтримує middleware, які можуть втручатися в обробку запитів та відповідей перед їх передачею контролерам;
- пайпи (Pipes): Пайпи використовуються для обробки та валідації даних перед їх передачею в контролери.

Фреймворк має досить детальну документацію, підтримує важливі аспекти, такі як dependency injection, валідація даних, тестування та багато іншого. З допомогою NestJS ви можете побудувати масштабовані та легко тестовані серверні застосунки на основі TypeScript.

Описавши основні складові серверної частини, потрібно мати базу даних, яка покриває потреби розробки.

Існує багато різних СУБД, серед яких: MSSQL, MySQL, MongoDB, PostgeSQL та інші. Оскільки програмний засіб буде мати різні відношення, одне з яких, наприклад, співробітник-проєкт, то база даних має бути реляційною.

В якості бази даних було обрано PostgeSQL [13]. PostgreSQL (або "Postgres" для короткості) - це потужна об'єктно-реляційна система керування базами даних (СКБД), яка заснована на мові SQL. Вона є відкритою програмою і має велику активну спільноту розробників. Ось деякі ключові аспекти PostgreSQL:

- архітектура та функціональність. PostgreSQL використовує мультиверсійний підхід до управління транзакціями, що робить його ефективним у великих інтенсивних обсягах даних. Підтримує ACID (Atomicity, Consistency, Isolation, Durability) властивості для забезпечення надійності та цілісності даних;
- мова SQL. PostgreSQL повністю сумісний з SQL і надає багатий набір функцій, таких як JOIN, GROUP BY, та інші;
- розширення та модульність. Дозволяє користувачам створювати свої власні функції, типи та операції, розширюючи функціонал бази даних;
- типи даних. Підтримує розширений набір вбудованих типів даних, включаючи числові, текстові, географічні, JSON і багато інших;
- JSON та JSONB. PostgreSQL має вбудовану підтримку для обробки JSON-даних, включаючи можливість зберігати та запитувати JSON-об'єкти;
- реплікація та відновлення. Підтримує велику кількість режимів реплікації для створення резервних копій та розподілення навантаження;
- індексація. PostgreSQL має розширені можливості індексації, включаючи B-tree, Hash, GiST (Generalized Search Tree) та GIN (Generalized Inverted Index);
- розширення SQL. Дозволяє використовувати SQL-розширення, такі як

PL/pgSQL, PL/Tcl, PL/Perl, PL/Python, та інші, для реалізації збережених процедур та тригерів;

- активна спільнота. PostgreSQL має широко розгалужену та активну спільноту розробників, що сприяє постійному вдосконаленню та розвитку системи.

Postgres SQL гарантує високий рівень захисту шляхом впровадження мережевої безпеки, яка інтегрується з сервером безпеки. Доступ користувачів до записів обмежений відповідно до їхніх привілеїв, забезпечуючи максимальну безпеку. Дані знаходяться на відокремленому сервері з обмеженням несанкціонованого доступу, що додатково забезпечує їхню безпеку.

Для створення клієнтської частини було обрано бібліотеку ReactJS ReactJS - це бібліотека JavaScript, розроблена компанією Facebook для побудови інтерфейсів користувача. Основна ідея React полягає в створенні компонентів, які можна повторно використовувати та легко управляти їх станом [14]. Він сприяє розробці односторінкових застосунків, де весь код завантажується один раз, а зміни сторінок відбуваються без повторної завантаження сторінки.

Наведемо переваги використання ReactJS для розробки клієнтського застосунку:

- компонентна архітектура. React дозволяє розбивати інтерфейс на невеликі, самостійні компоненти. Це полегшує управління кодом, робить його читабельнішим і сприяє повторному використанню коду;

- віртуальний DOM. React використовує віртуальний DOM для ефективного оновлення інтерфейсу. Замість безпосередньої зміни реального DOM, React спочатку змінює віртуальний DOM, порівнює його з реальним DOM і виконує лише необхідні оновлення;

- розширення за допомогою бібліотек і фреймворків. Існує багато бібліотек і фреймворків, які побудовані поверх React, такі як Redux для управління станом, React Router для навігації, і Material-UI для готових

компонентів з матеріальним дизайном;

- активна спільнота. ReactJS має велику та активну спільноту розробників. Це означає, що ви можете легко знайти відповіді на свої питання, а також розбудовувати належний рівень знань та навичок;
- підтримка великих проектів: React може ефективно використовуватися для розробки великих та складних проектів, забезпечуючи структуру коду та організацію проекту.

Ці переваги роблять ReactJS популярним вибором для розробки веб-інтерфейсів, особливо для односторінкових застосунків та проектів, де важлива швидкість та ефективність розробки.

Середовище розробки має бути зручним і легко налаштовуваним. Однією з переваг одного середовища розробки над іншим є його варіант розповсюдження, підтримка розширень, споживання ресурсів системи.

У даному випадку було обрано Visual Studio Code там як вона розповсюджується безкоштовно, має широку підтримку серед розробників які утворюють велику кількість розширень під різні потреби які так само розповсюджуються безкоштовно. VSCode споживає мало ресурсів системи відносно інших його конкурентів таких як сімейства IDE від компанії JetBrains, показано на рисунку 2.2. У свою чергу продукти JetBrains [15] розповсюджуються по системі підписки, тобто не є безкоштовними хоч і мають тестовий період у 30 діб. Один з можливих варіантів використання продукту по студентській ліцензії, яка потребує додаткових кроків для використання, таких як реєстрація акаунту з використанням пошти від університету або надсилання студентського квитка. Проблема тут у використанні продуктів тільки у некомерційних цілях, а тільки для навчання.

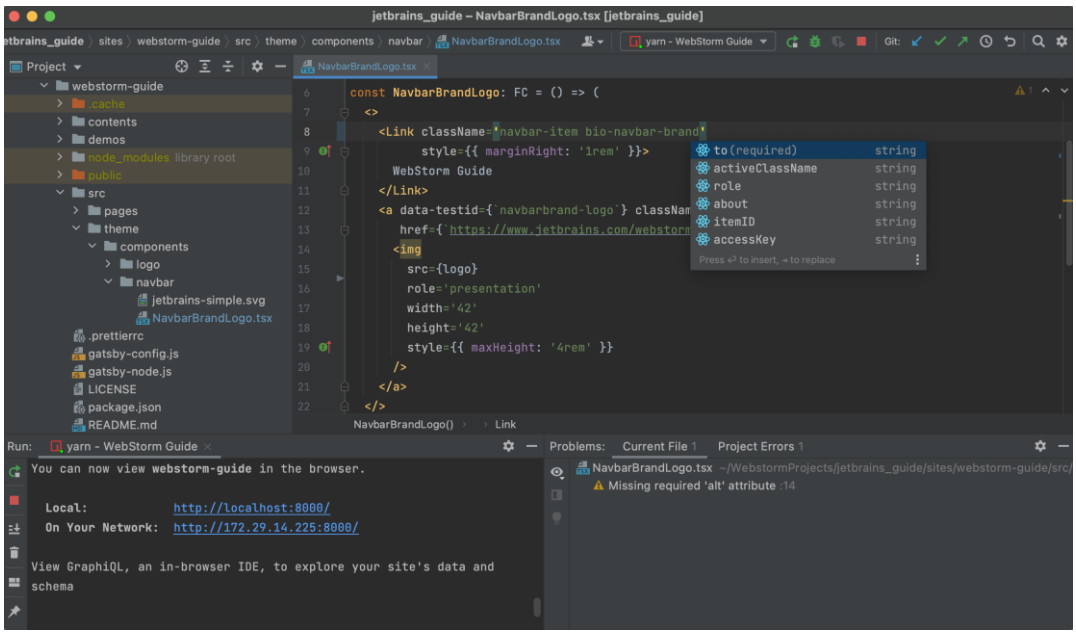


Рисунок 2.2 – Скріншот WebStorm від JetBrains

Visual Studio Code має простий неперевантажений інтерфейс у якому легко і швидко розібратись та налаштувати під свої потреби у разі виникнення таких [16]. Схема розповсюдження даного середовища розробки є безкоштовною без реєстрації акаунту, тощо. Інтерфейс файлової системи та робочої області представлено на рисунку 2.3.

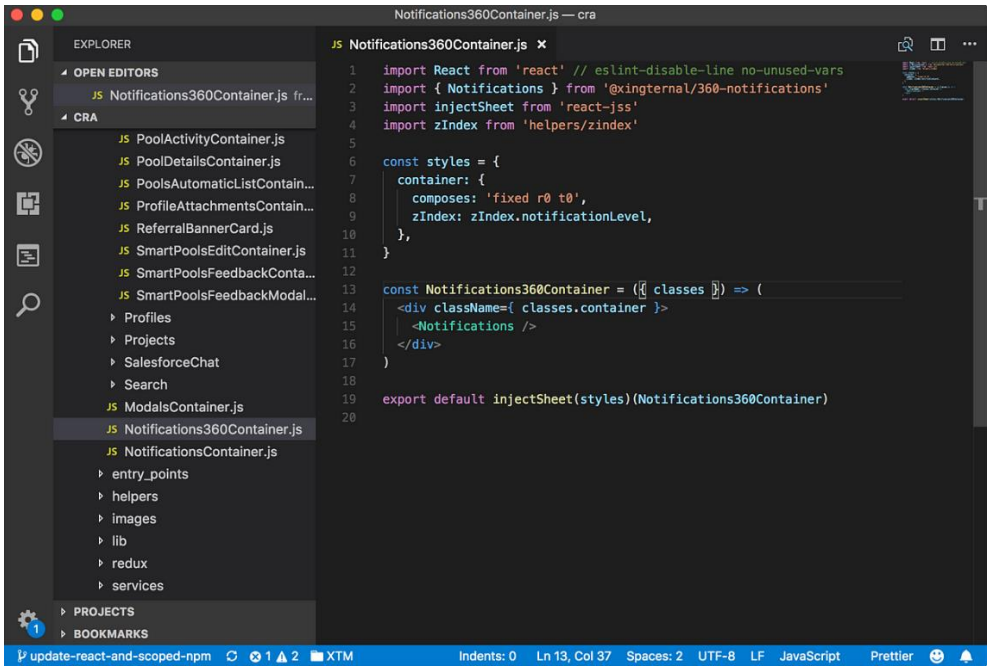


Рисунок 2.3 – Скріншот робочої області VSCode

У ньому є розширення якими розробник без жодних проблем може налаштувати своє середовище з його особистих вподобань, яке не буде перевантажувати систему. На рисунку 2.4 показана вкладка розширень які можна встановити власноруч однією кнопкою.

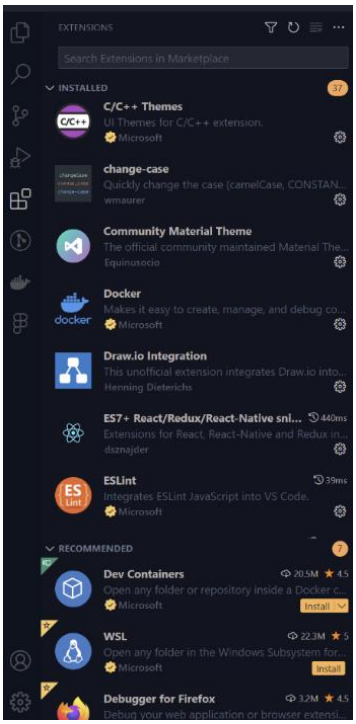


Рисунок 2.4 – Скріншот розширень VSCode

Порівняємо описані середовища розробки по критеріям (табл. 2.2), які напряму впливають на зручність використання, швидкість розробки будь-якого програмного засобу будь-якою мовою програмування та доступність використання на різних операційних системах.

Таблиця 2.2 – Порівняння середовищ розробки

Критерії	VSCode	WebStorm
Вартість	Безкоштовно	Підписка
Підтримка мов і технологій	Широкий спектр розширень через маркетплейс	Вузьконапревлений на певні технології

Продовження таблиці 2.2

Швидкодія та продуктивність	Легкий і швидкий	Важкий і менш швидкий через велику к-сть вбудовиних функцій
Вбудовані функції	Редагування тексту, відлагодження, контроль версій, підтримка базових мов веб-розробки	Багато вбудованих інструментів для веб-розробки
Спільнота і підтримка	Широке коло спільноти та підтримка Microsoft	JetBrains
Інтеграція з різними продуктами як Git, Docker	Наявна	Наявна
Кросплатформеність	Windows, MacOS, Linux та веб-версія доступна звідусіль	Windows, MacOS і Linux

У таблиці наведено порівняльну характеристику середовищ розробки, та на основі отриманих результатів оцінки критерій було прийнято рішення про використання Visual Studio Code, як середовища розробки для створення програмного засобу для оптимізації ресурсів у ІТ.

2.4 Проектування бази даних

Гарно спроєктована база даних є важливою частиною успішності кожного проєкту. Правильне проєктування гарантує швидкодію, відповідність даних та їх розширення.

У проектованій системі необхідно зберігати дані співробітників, проектів, бюджетів проектів, витрати на співробітників, такі як оплата праці за певний період. Всі ці залежності мають бути тісно пов’язані між собою задля цілісності системи і коректного обрахунку витрат компанії по всім категоріям проектів і співробітників.

Реляційна база даних – це формат зберігання даних, де інформація структурована у вигляді таблиць з рядками і стовпцями. У цій системі дані

організовані у вигляді записів, де кожен рядок відображає конкретний об'єкт, а кожен стовпець визначає певну характеристику чи атрибут. [17].

Згідно проведеного аналізу постає задача збереження даних і для цього доцільно обрати реляційну базу даних.

Проаналізуємо сутності збереження даних. Необхідно зберігати дані співробітників, проектну інформацію, навички співробітників.

Розділимо проектування проведення відношень на декілька етапів використовуючи метод сутність-зв’язок [18].

По-перше, визначимо головні сутності даних. В основні частини програмного застосунку можна визначити такі головні сутності:

- СПІВРОБІТНИК;
- ПРОЄКТ;
- ДЕПАРТАМЕНТ;
- НАВИЧКА.

По-друге, визначимо зв’язки між визначеними сутностями. Зв’язком являється з’єднання між двомі і більше сутностями:

- СПІВРОБІТНИК має ПРОЄКТ;
- СПІВРОБІТНИК має ДЕПАРТАМЕНТ;
- СПІВРОБІТНИК має НАВИЧКУ;
- ДЕПАРТАМЕНТ має НАВИЧКУ.

У сутностей є атрибути. Атрибутом виступає властивість сутності. У даному випадку атрибути у кожної сутності будуть такі:

- СПІВРОБІТНИК (<Ідентифікатор співробітника>, Ім’я, Прізвище, Місто, Позиція, Рівень, email, Вага навички, Погодинна ставка, Погодинна ставка продажу, початок співпраці, кінець співпраці, Дата створення, Дата оновлення, Дата видалення);
- ДЕПАРТАМЕНТ (<Ідентифікатор департамента>, Назва);
- НАВИЧКА (<Ідентифікатор навички>, Назва);
- ПРОЄКТ (<Ідентифікатор проєкта>, Назва, Опис, Початок, Кінець, Валюта, Ім’я клієнта, Бюджет, Дата створення, Дата оновлення, Дата

видалення).

Наступним етапом у проектування бази даних є визначення типу зв’язків між сутностями. Важливо відзначити, що ступені зв’язків для сутностей бази даних формуються протягом усього її існування. Визначення зв’язків представлено у вигляді ER-діаграм. ER-діаграма – це тип діаграми, яка показує зв’язок сутностей у базі даних. Іншими словами допомагає пояснти логічну структуру бази даних.

Існують такі типи зв’язків сутностей:

- один до одного (One-to-One – 1:1) – кожен запис у таблиці відповідає одному запису у іншій таблиці;
- один до багатьох (One-to-Many – 1:N) – кожен запис в одній таблиці відповідає багатьом записам у іншій таблиці;
- багато до багатьох (Many-to-Many – N:M) – кожен запис в одній таблиці відповідає багатьом записам в іншій таблиці і навпаки. Зазвичай для такого типу зв’язку використовується проміжна таблиця, яка зберігає у собі ідентифікатори обох таблиць.

Визначення ступеня зв’язку та класу належності сутностей «СПІВРОБІТНИК» та «ДЕПАРТАМЕНТ» показано на рисунку 2.5.

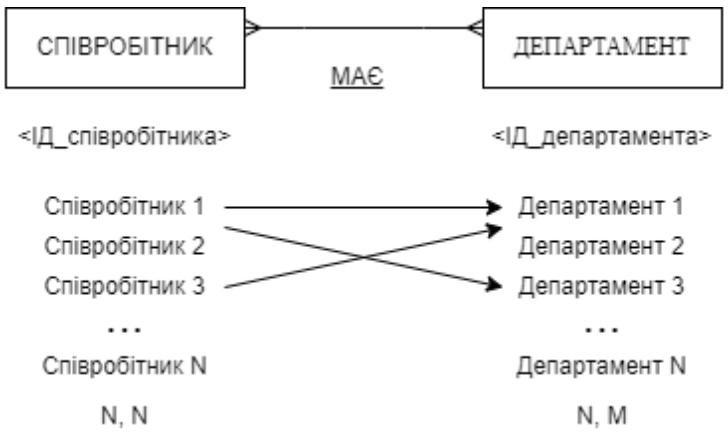


Рисунок 2.5 – Визначення ступеня зв’язку та класу належності сутностей «СПІВРОБІТНИК» та «ДЕПАРТАМЕНТ»

Отже, ступінь зв’язку між сутностями N:M. Відповідно отримуємо такі відношення:

- СПІВРОБІТНИК (<Ідентифікатор співробітника>, Ім’я, Прізвище, Місто, Позиція, Рівень, email, Вага навички, Погодинна ставка, Погодинна ставка продажу, початок співпраці, кінець співпраці, Дата створення, Дата оновлення, Дата видалення);
- ДЕПАРТАМЕНТ (<Ідентифікатор департамента>, Назва);
- СПІВРОБІТНИК - ДЕПАРТАМЕНТ (<Ідентифікатор співробітника>, <Ідентифікатор департамента>).

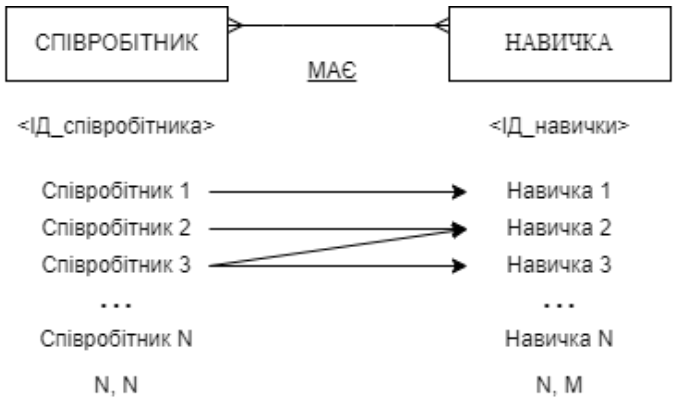


Рисунок 2.6 – Визначення ступеня зв’язку та класу належності сутностей «СПІВРОБІТНИК» та «НАВИЧКА»

Отже, ступінь зв’язку між сутностями N:M (рис. 2.6). Відповідно отримуємо такі відношення:

- СПІВРОБІТНИК (<Ідентифікатор співробітника>, Ім’я, Прізвище, Місто, Позиція, Рівень, email, Вага навички, Погодинна ставка, Погодинна ставка продажу, початок співпраці, кінець співпраці, Дата створення, Дата оновлення, Дата видалення);
- НАВИЧКА (<Ідентифікатор навички>, Назва);
- СПІВРОБІТНИК - НАВИЧКА (<Ідентифікатор співробітника>, <Ідентифікатор навички>).

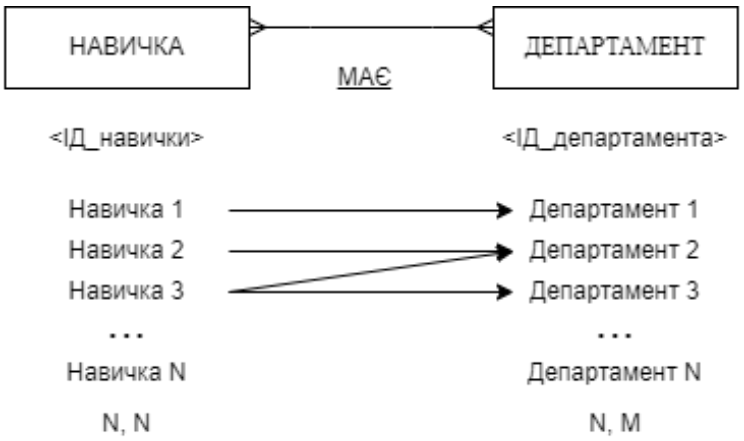


Рисунок 2.7 - Визначення ступеня зв’язку та класу належності сутностей «НАВИЧКА» та «ДЕПАРТАМЕНТ»

Отже, ступінь зв’язку між сутностями N:M(рис. 2.7). Відповідно отримуємо такі відношення:

- НАВИЧКА (<Ідентифікатор навички>, Назва);
- ДЕПАРТАМЕНТ (<Ідентифікатор департамента>, Назва);
- НАВИЧКА - ДЕПАРТАМЕНТ (<Ідентифікатор навички>, <Ідентифікатор департамента>).

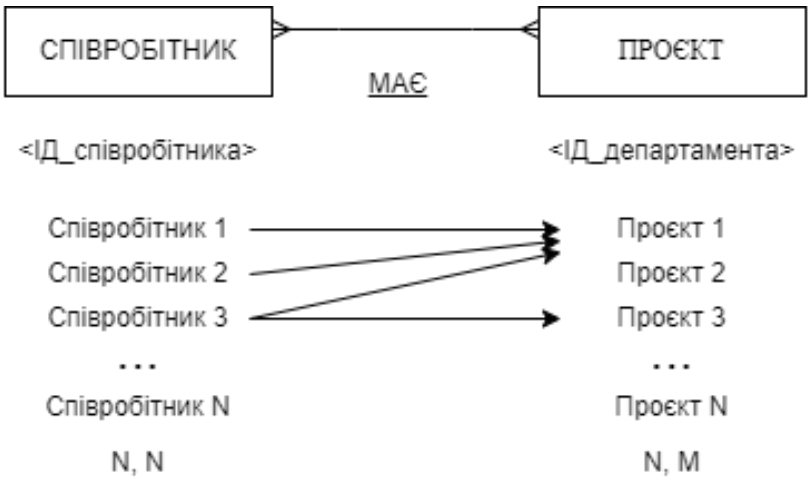


Рисунок 2.8 - Визначення ступеня зв’язку та класу належності сутностей «СПІВРОБІТНИК» та «ПРОЄКТ»

Отже, ступінь зв’язку між сутностями N:M(рис. 2.8). Відповідно отримуємо такі відношення:

- СПІВРОБІТНИК (<Ідентифікатор співробітника>, Ім’я, Прізвище, Місто, Позиція, Рівень, email, Вага навички, Погодинна ставка, Погодинна ставка продажу, початок співпраці, кінець співпраці, Дата створення, Дата оновлення, Дата видалення);

- ПРОЄКТ (<Ідентифікатор проєкта>, Назва, Опис, Початок, Кінець, Валюта, Ім’я клієнта, Бюджет, Дата створення, Дата оновлення, Дата видалення);

- СПІВРОБІТНИК - ПРОЄКТ (<Ідентифікатор співробітника>, <Ідентифікатор проєкта>).

Визначивши усі зв’язки між сутностями ми побачили, що більшість сутностей пов’язані між собою відносинами багато-до-багатьох, що потребує додаткових табличок, які будуть зберігати унікальні ідентифікатори обох сутностей. Використання такої асоціативної таблиці дозволяє ефективно виражати і обробляти багато-до-багатьох відносини, так як вона дозволяє легко визначити, які дані необхідно включати в основні таблиці.

Спроєктуємо базову структуру для візуалізації зв’язків, кількості полів у кожній сутності, їх типи [19]. Спроєктована база даних показана на рисунку 2.9.

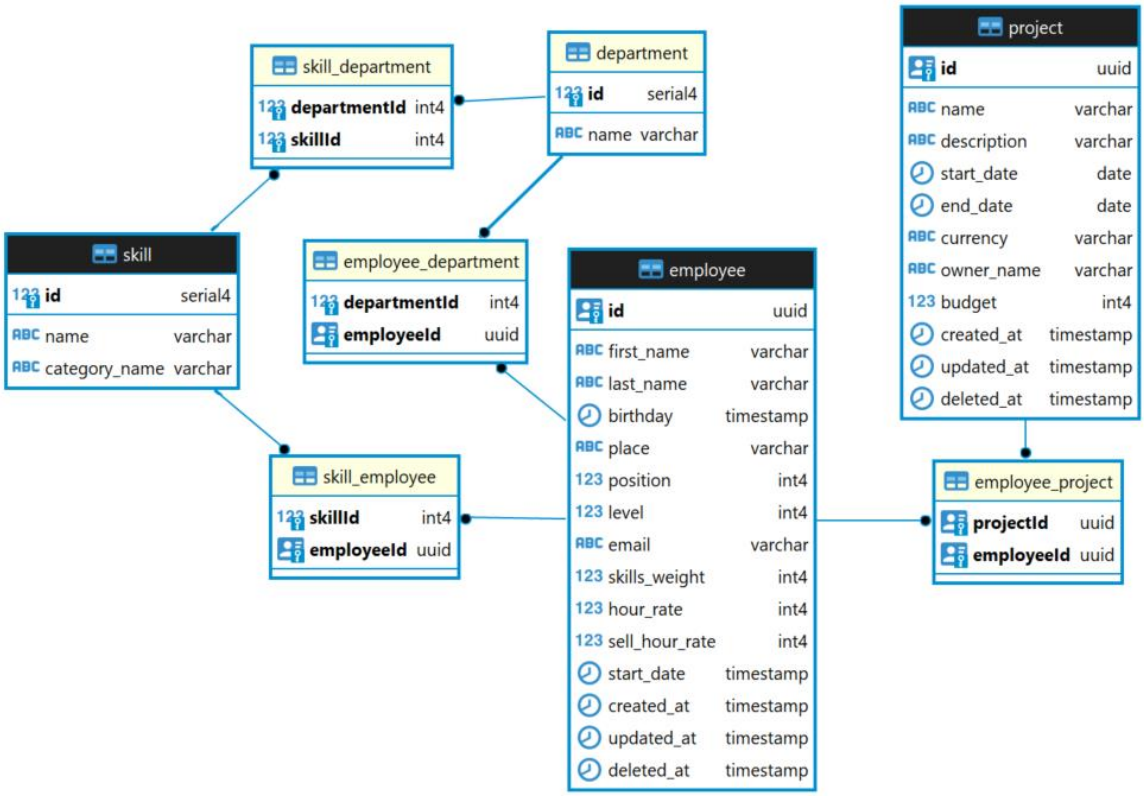


Рисунок 2.9 – Діаграма бази даних

Дана діаграма використовується для моделювання бази даних та візуалізує усі дані та зв’язки у простій візуальній формі.

Спроєктована реляційна база даних має усі необхідні таблиці, атрибути та зв’язки між таблицями необхідні для даного програмного засобу і коректної роботи метода.

2.5 Проектування серверної частини

Проектуючи серверну частину слід відповідально відноситись до побудови архітектури і всіх залежностей, так як від цього залежить подальша зручність розширення і додавання нового функціоналу. Найрозповсюдженішим типом є чиста архітектура [20]. Ця архітектура намагається об’єднати деякі провідні сучасні архітектури, такі як гексагональна архітектура, цибулева або тришарова архітектура та кричуща архітектура в одну основну архітектуру.

На рисунку 2.10 показана модель діаграми чистої архітектури.

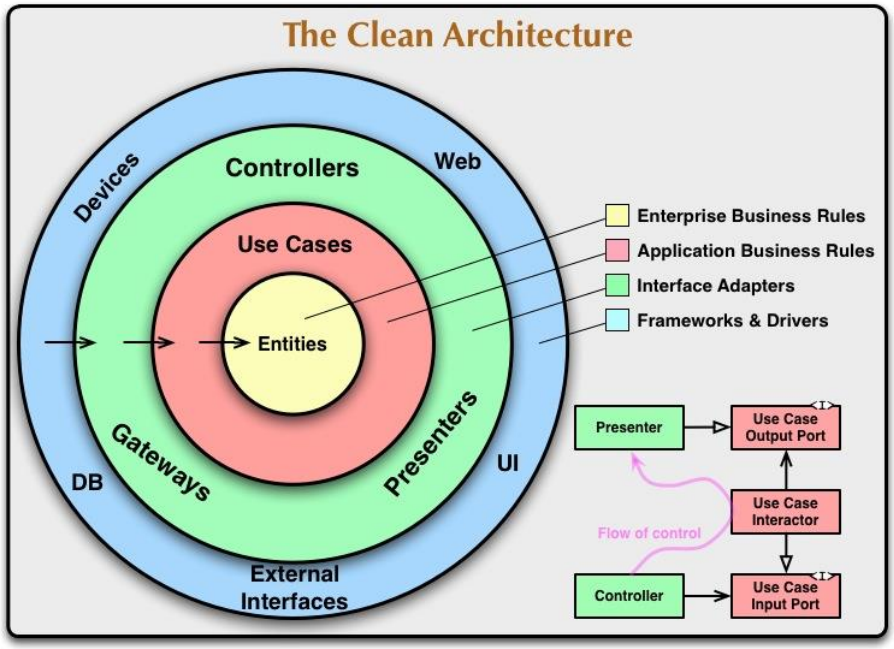


Рисунок 2.10 – Діаграма чистої архітектури

Розглянемо основні концепції та особливості патернів проектування MVC, MVP і MVVM:

MVC (Model-View-Controller)

- Model (Модель): Відповідає за обробку даних та бізнес-логіку. Може включати в себе збереження даних, роботу з базою даних тощо;
- View (Вигляд): Відповідає за відображення даних користувачеві. Отримує дані від моделі та відображає їх;
- Controller (Контролер): Приймає вхідні дані від користувача через взаємодію з виглядом та обробляє їх, взаємодіючи з моделлю для оновлення даних.

MVP (Model-View-Presenter):

- Model (Модель): Аналогічно до MVC;
- View (Вигляд): Відповідає за відображення даних, але не прямо взаємодіє з моделлю. Взаємодія з моделлю відбувається через презентера;
- Presenter (Презентер): Отримує введені дані від користувача від виду, обробляє їх та оновлює модель. Також оновлює вигляд згідно зі змінами в моделі.

MVVM (Model-View-ViewModel):

- Model (Модель): Також відповідає за дані та бізнес-логіку;
- View (Вигляд): Відображення даних користувачу, аналогічно до MVC та MVP;
- ViewModel (Модель-Вигляд): Це проміжний шар між моделлю та видом. Відповідає за підготовку даних для відображення та обробку введення користувача. Взаємодіє як з моделлю, так і з видом.

Обравши NestJS як фреймворк для розробки серверної частини у контексті паттернів проектування буде використаний MVC. Розглянемо схематичне зображення програмного засобу розділено на шари на рисунку 2.11.

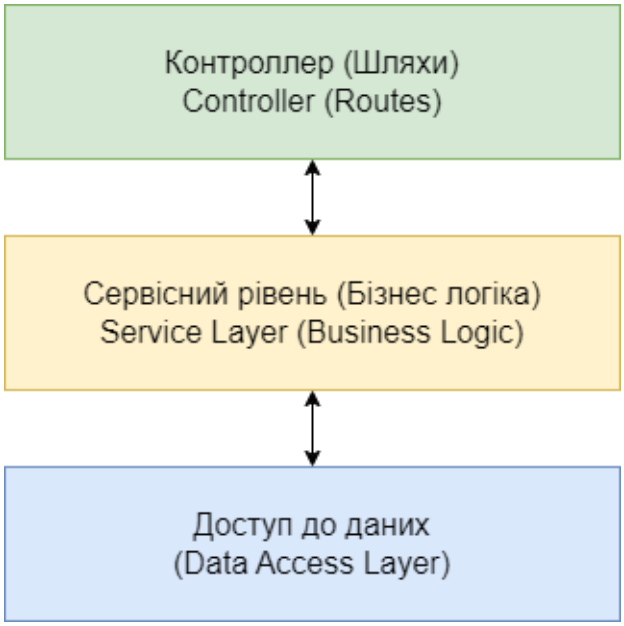


Рисунок 2.11 – Розподілення системи на шари

- контролери: єдина мета контролера — отримувати запити до програми та працювати з маршрутами;
- сервісний рівень: ця частина блоку має містити лише бізнес-логіку. Наприклад, усі операції та методи CRUD, щоб визначити, як можна створювати, зберігати й оновлювати дані;
- рівень доступу до даних: цей рівень забезпечує доступ до даних, що зберігаються в певному постійному сховищі, і забезпечує логіку доступу.

Важливою частною для захищеності серверної частини є використання HTTPS-протоколу з’єднання та використання SSL. Далі розглянемо кожну частину у деталях.

Протокол HTTPS (Hypertext Transfer Protocol Secure) є розширенням стандартного HTTP, яке додає безпеку шляхом зашифрування обміну даними між користувачем і веб-сайтом. Зазвичай використовується для забезпечення конфіденційності та цілісності даних під час їх передачі через Інтернет.

Принципи роботи:

- шифрування даних: HTTPS використовує криптографічні протоколи, такі як TLS (Transport Layer Security) або його попередник SSL (Secure Sockets Layer), для захисту від перехоплення та читання інформації під час передачі між користувачем і сервером. Це робить надійним захищеним способом обміну конфіденційною інформацією, такою як паролі, особисті дані та банківська інформація;
- автентифікація сервера: Процес взаємного визнання, що гарантує, що користувач з'єднується із справжнім сервером, а не з імітацією або зловмисною стороною. Це виконується за допомогою цифрових сертифікатів, які видані надійними центрами сертифікації;
- цілісність даних: Цей аспект гарантує, що дані не були змінені або пошкоджені під час їх передачі. Це досягається за допомогою хеш-функцій та цифрових підписів, які підтверджують, що дані залишилися недоторканими;
- HTTPS і SEO: З 2014 року Google враховує наявність HTTPS як фактор для підвищення рейтингу в пошукових результатах. Використання HTTPS може позитивно вплинути на видимість вашого веб-сайту;
- застосування веб-браузерів: Більшість сучасних веб-браузерів позначають небезпечні або незахищені з'єднання, надаючи користувачам індикацію про те, що веб-сайт використовує HTTPS або ні. Це може впливати на довіру користувачів до вашого веб-сайту.

Захист HTTPS може вимагати додаткових ресурсів для обробки шифрування та аутентифікації. У внутрішніх системах, де багато

обчислювальних ресурсів знаходиться в одній локальній мережі, це може бути менш критичним фактором порівняно з публічними веб-сайтами.

З точки зору захисту і доступу розроблюваний програмний засіб має використовуватись у межах внутрішньої мережі або з обмеженим доступом. В межах внутрішньої мережі використання HTTPS може бути доцільним, якщо важливі дані передаються між серверами чи сервісами. Такий підхід забезпечить додатковий шар захисту від можливого перехоплення або зміни інформації.

Загалом, використання HTTPS є важливим для захисту конфіденційної інформації та забезпечення безпеки під час перегляду веб-сайтів. Особливо це актуально в сучасному Інтернеті, де зловмисники можуть намагатися використовувати незахищені з'єднання для отримання доступу до особистої інформації користувачів.

Розібравшись з HTTPS, потрібно врахувати одну з вижливих частин його використання, це SSL-сертифікати. SSL (Secure Sockets Layer) - це протокол безпеки, який забезпечує захищене з'єднання між клієнтом і сервером через Інтернет. SSL-сертифікати використовуються для шифрування даних, які передаються між користувачем та веб-сайтом, забезпечуючи конфіденційність та цілісність цих даних. SSL і HTTPS (Hypertext Transfer Protocol Secure) тісно пов'язані між собою. HTTPS є застосуванням протоколу SSL або його більш сучасного варіанту, TLS (Transport Layer Security).

Отже, HTTPS є підмножиною протоколу SSL/TLS, і коли ви використовуєте HTTPS, ви насправді використовуєте шифрування, яке надається протоколом SSL або TLS.

Наступною важливою складовою захищеності системи є CORS. CORS, або Cross-Origin Resource Sharing (Спільний доступ до ресурсів між доменами), є безпековою політикою, що обмежує, як веб-сторінки або веб-додатки можуть запитувати ресурси з інших доменів, ніж той, з якого була завантажена сама сторінка. Це заходи безпеки, призначені для захисту

користувачів від атак, які можуть виникнути внаслідок взаємодії з ресурсами на інших доменах.

Коли веб-сторінка або веб-додаток намагається зробити запит на ресурс (наприклад, запит на API) на інший домен, браузер надсилає HTTP-запит з заголовками CORS для перевірки того, чи дозволено здійснення цього запиту. Якщо сервер, на якому знаходиться ресурс, підтримує CORS і налаштований на прийняття запитів з даного домену, браузер дозволяє здійснити запит і отримати відповідь.

Користь CORS з точки зору безпеки системи полягає в тому, що він запобігає атакам на основі міжсайтового сценарію (Cross-Site Scripting, XSS) та іншим видам атак, які можуть виникнути при взаємодії з ресурсами інших доменів. За допомогою CORS можна обмежити доступ до ресурсів лише для певних відомих та довірених доменів, що підвищує загальний рівень безпеки веб-додатків.

2.6 Проектування користувацького інтерфейса

Інтерфейс користувача представляє певний набір засобів для обробки та відображення інформації та спрощує взаємодію з певними даними певної системи.

Процес проектування сучасного ПЗ передбачає вирішення ряду задач, зокрема: зниження витрат на проектування, скорочення термінів проектування, покращення якості пропонованих рішень, забезпечення нескладного в освоєнні та використанні ПЗ, вивчення та впровадження нових технологій та засобів, досягнення кращих результатів в порівнянні з конкурентами.

Можна виділити такі найпоширеніші типи користувацьких інтерфейсів як:

- CLI (command line interface – командний рялок) – один із найстаріших типів, який приймає введення тексту для виконання функцій операційної системи;

- GUI (graphical user interface – графічний інтерфейс) – це цифровий інтерфейс, у якому користувач взаємодіє з графічними компонентами. Ці компоненти передають важливу інформацію для користувача, а також які дії він може виконати;

- VUI (voice user interface – голосовий інтерфейс) – це інтерфейс який дозволяє людині взаємодіяти з комп'ютерами за допомогою голосу.

Можна виділити ще інтерфейс жестів, де користувач взаємодіє з графічним інтерфейсом за допомогою дотиків та рухів.

Загалом розробку графічного користувацького інтерфейсу можна поділити на 2 важливі області: UI/UX.

UI (user interface – інтерфейс користувача) стосується екранів, кнопок, перемикачів, значків та інших візуальних елементів, з якими взаємодіє користувач під час використання веб-сайту, програми чи електронного пристрою [21].

UX (user experience – досвід користувача) означає те як користувач взаємодіє з веб-сайтом, програмою або пристроєм [22]. Тобто зручність розташування елементів інтерфейсу, як вони взаємодіють між собою і яким чином буде побудована сама взаємодія користувача і машини. На рисунку 2.12 преставлено графічне зображення поділу відповідальності кожної з частин.

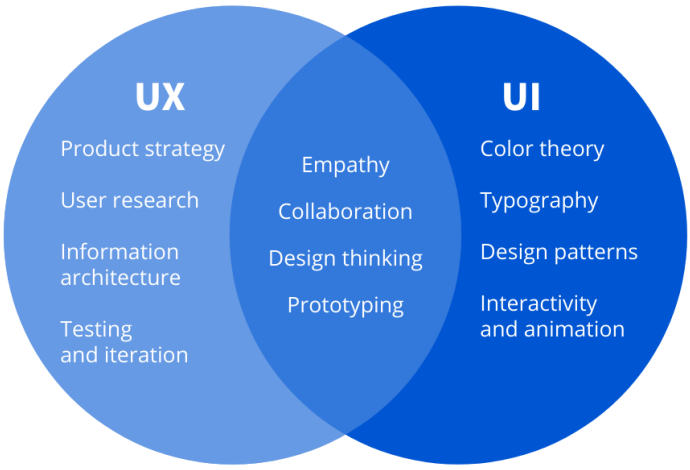


Рисунок 2.12 – Відповідність користувацького інтерфейсу до його взаємодії з користувачем

Головною вимогою до даної системи є зручне використання та відображення даних по співробітниках, проектах та рекомендаціях для оптимізації ресурсів і зменшення кількості кліків миші для відображення потрібної інформації для користувача.

Виходячи з цього будуємо інтерфейс інтуїтивно простим і зрозумілим. Використовуючи табличний варіант подання інформації про співробітників або проєкти буде найзручнішим, тому що можна швидко шукати інформацію по декільком полям і негайно її обробити за потреби.

Розроблені прототипи зображені на рисунках 2.13-2.14.

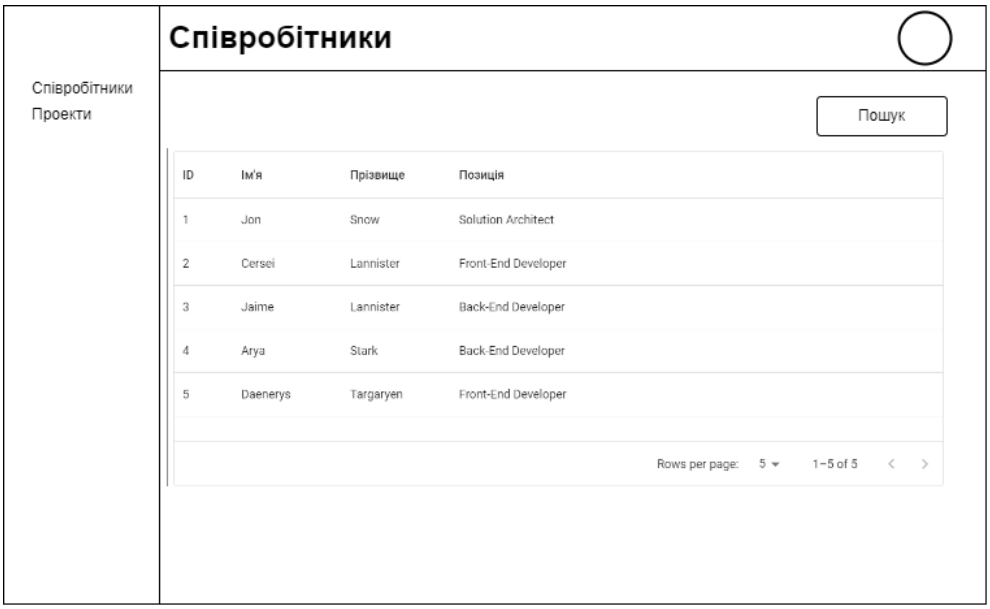


Рисунок 2.13 – Прототип списку співробітників

Зображення даних у виді таблиці дозволяє вивести якумога більше інформації по обраному розділу у компактному вигляді з максимальною кількістю доступних для показу даних. Додатково повинна бути змога шукати по таблиці по будь-якій колонці, для цього нам потрібен елемент пошуку на сторінці. Зображати будемо дані по проєктам відповідно до зображення співробітників у вигляді таблиці з пошуком.

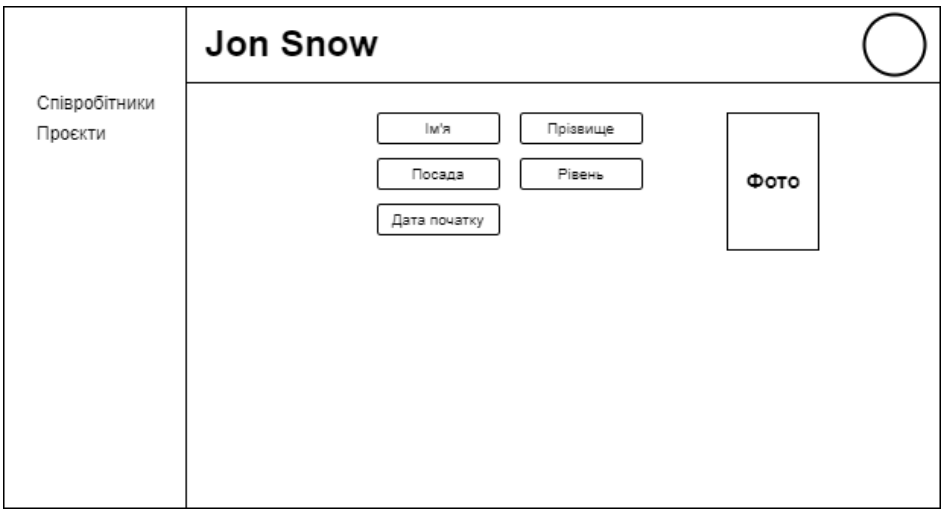


Рисунок 2.14 – Прототип профіля співробітника

Основою будь-якого інтерфейсу користувача по роботі з даними є зручне редагування одного запису та його перегляд. Тому потрібно мати сторінку одного запису для кожної окремої сутності. Сторінки одного запису співробітника або проєкта має мати елементи введення тексту, вибору з декількох варіантів, тощо. Розміщення елементів редагування запису буде структуроване відповідно до даних які потрібно заповнити або редагувати (рис. 2.10).

Створивши прототипи можна проаналізувати доцільне розміщення елементів на сторінці і зрозуміти який деталей, елементів, структурних блоків для коректного відображення інформації не вистачає або які елементи на сторінці можуть бути лишніми.

Для створення екрану для створення проекту розіб’ємо його на три кроки на одній сторінці виділивши такі кроки, як:

- Заповнення інформації по проєкту;
- Додавання даних по задачах для проєкта та відображення їх за допомогою діаграми Ганта;
- Показ інформації по оптимальному підбору співробітників по заданим критеріям;

Важливою частиною є відображення даних у візуальній формі з допомогою діаграми Гантта, яка допомагає відчути масштабність і

довготривалість проєкту візуально.

Діаграма Ганта (Gantt chart) — це вид діаграми, який використовується для візуалізації плану проекту. Він складається з горизонтальних планок (рядків), які представляють завдання чи задачі проекту, та вертикальних стрілок (стовпців), які представляють тривалість цих завдань. Кожна планка має довжину, яка відображає тривалість виконання завдання. Завдяки Гант-діаграми можна легко визначити, коли кожне завдання повинно бути виконане та як вони взаємодіють між собою.

Переваги використання діаграми Ганта у плануванні завдань для проектів у IT сфері включають:

- Візуалізація графіка проекту. Надає чіткий та легко зрозумілий спосіб відображення графіка проекту. Це дозволяє всім учасникам проекту швидко зрозуміти, як розподілені завдання в часі.
- Управління ресурсами. Допомагає визначити, які ресурси (людські, фінансові, матеріальні) необхідні для виконання кожного завдання та коли ці ресурси потрібні.
- Виявлення залежностей між завданнями. Може вказувати залежності між різними завданнями, що допомагає уникнути конфліктів та забезпечити логічний порядок виконання завдань.
- Оптимізація графіка проекту. Можна оптимізувати графік проекту, розподіляючи завдання ефективно та мінімізуючи затримки.
- Керування та моніторинг прогресу. Дозволяє відстежувати виконання завдань та вносити корективи у графік проекту при необхідності.

Загалом, діаграма Ганта є потужним інструментом для планування та відстеження проектів у IT сфері, допомагаючи командам ефективно керувати часом, ресурсами та завданнями.

2.7 Висновки

У даному розділі було спроектовано основні функції програмного засобу для оптимізації ресурсів у IT. Було порівняно особливості різних варіантів

автентифікації за допомогою сторонніх провайдерів, таких як Firebase Authentication та AWS Cognito. Розглянуто різні типи інтерфейсів і на основі даних було обрано використання графічного інтерфейсу, який може бути використаний на ноутбуках, персональних комп’ютерах і мобільних девайсах.

Було обґрунтовано вибір мови програмування та інструментарію для розробки програмного засобу, обрано мову програмування TypeScript, для розробки серверної частини було обрано NestJS фреймворк, для створення інтерфейсу користувача і взаємодії з серверною частниною – ReactJS та середовищем розробки редактор коду Visual Studio Code.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ МЕТОДУ У ПРОГРАМНИЙ ЗАСІБ

3.1 Впровадження сервісу автентифікації

Для того, щоб використовувати Firebase Authentication сервіс у застосунку, потрібно створити Google Account або використати вже існуючих який буде адміністратором у даній системі. Після того, як буде авторизованим адміністратор у системі firebase (рис. 3.1), буде доступний екран з кнопкою створення проекту який буде відповідати за автентифікацію користувача системи.

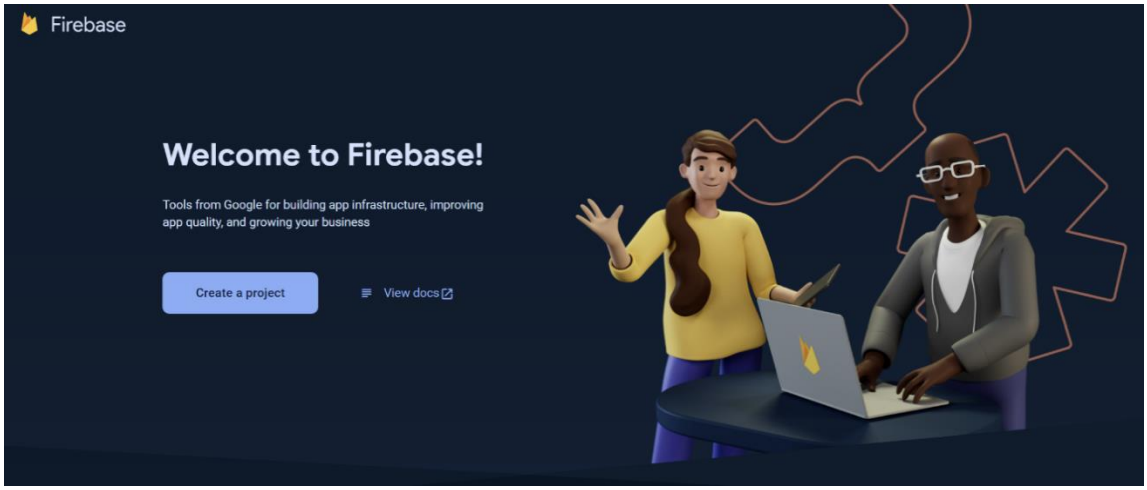


Рисунок 3.1 – Створення проекту у Firebase

Далі потрібно визначитись з назвою проекту і задати її зрозумілою так як вона буде фігурувати у посиланнях і налаштуваннях цього сервісу авторизації (рис. 3.2).

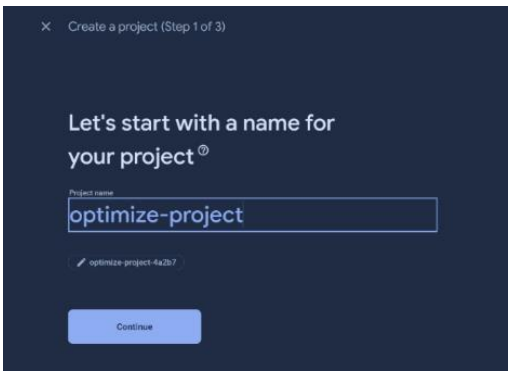


Рисунок 3.2 – Створення назви проекту у Firebase

Наступний крок є одним з найважливіших, так як пропонується спосіб встановлення пакету з методами обробки інформації для авторизації у firebase системі та додавання конфігурації налаштування саме до цього аккаунту, де будуть зберігатись дані користувачів системи (рис. 3.3).

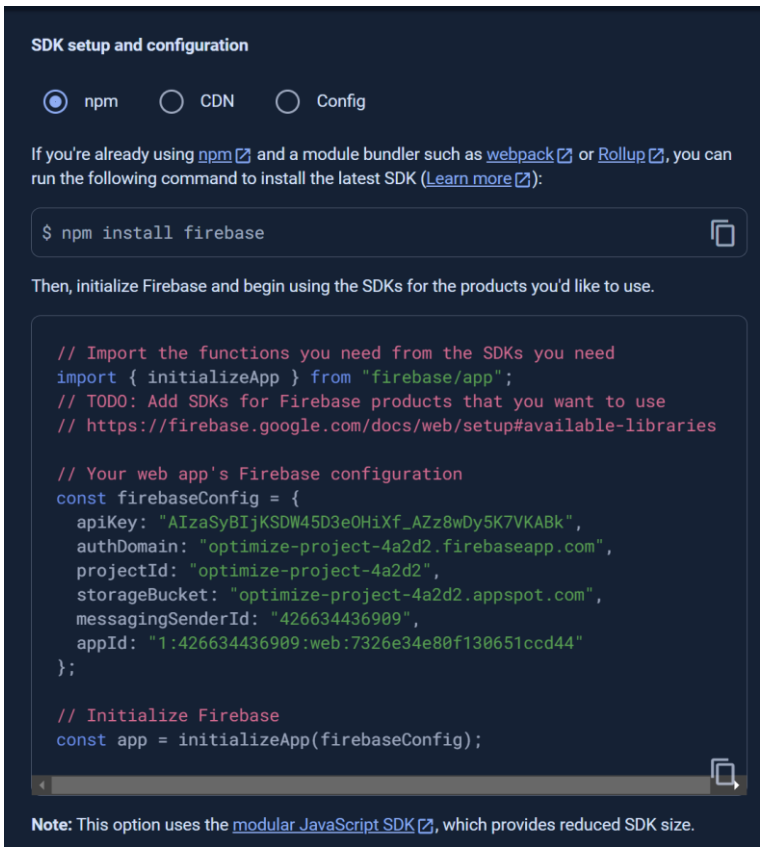


Рисунок 3.3 - Налаштування конфігурації для проекту

Наступним кроком після встановлення всіх необхідних залежностей у застосунок, перевірки на помилки зареєструємо через панель управління

firebase користувача який буде мати доступ до програмного засобу (рис. 3.4). Кількість створених користувачів обмежується тільки потребою у кількості доступів, які зазвичай видаються певній ланці менеджменту який відповідає за людські ресурси у компанії та має доступ до фінансів. У даному випадку користувач створюється безпосередньо через панель управління, так як створювати форму реєстрації не є доцільним, так як програмний засіб має бути закритим і доступним тільки для певного переліку користувачів, які буде заведені через панель управління адміністратором.

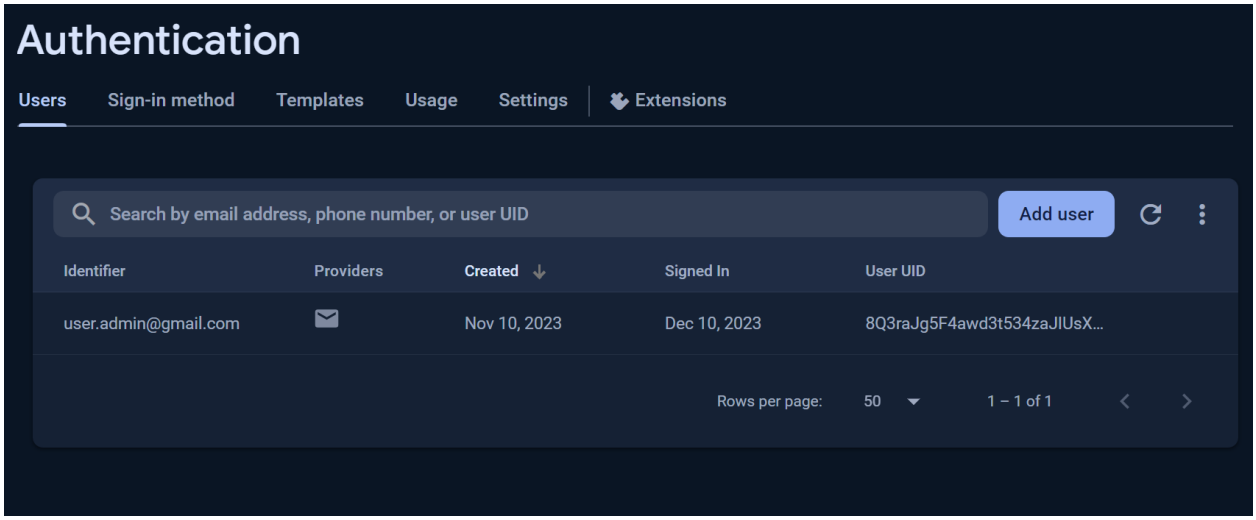


Рисунок 3.4 – Панель автентифікації з користувачами та конфігурацією

Так як у програмному засобі буде відсутня форма реєстрації користувачів задля закритості системи маємо якимось чином додавати користувачів, які будуть мати доступ до даних і змінювати їх. Для цього потрібно зареєструвати користувача через вікно додавання користувача ввівши дані для автентифікації (рис. 3.5).

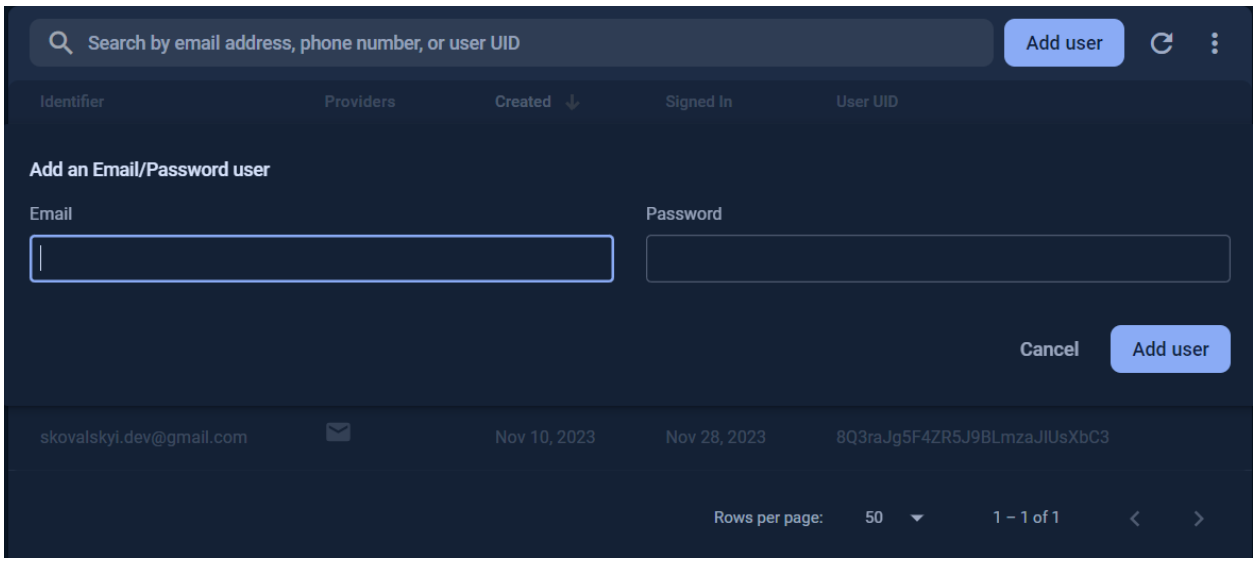


Рисунок 3.5 – Вікно створення нового користувача

Панель управління має швидкі дії для роботи з користувачами системи. У декілька кліків на потрібному користувачі можна змінити пароль і потрібному користувачеві на електронну пошту прийде лист з описом як це зробити, заблокувати акаунт, щоб тимчасово забрати доступ до системи та звісно видалення користувача з системи повністю. Меню управління користувачем показано на рисунку 3.6.

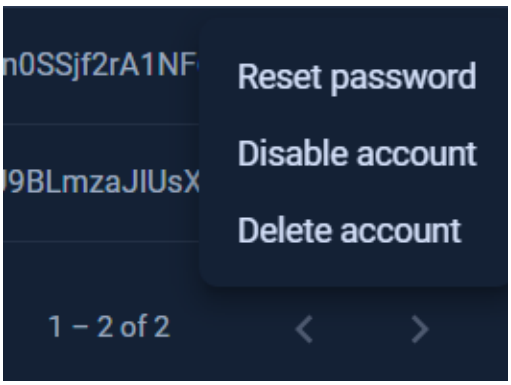


Рисунок 3.6 – Меню управління користувачем

Отже, було налаштовано адміністративний акаунт у Firebase, у якому було створено проєкт, який буде відповідати за автентифікацію і який має функціональність для додавання, блокування, зміна пароля і видалення

користувачів. Отримано конфігурацію доступу до підключення на стороні клієнта для автентифікації користувача у системі.

3.2 Розробка серверної частини програмного засобу

Термін “трирівнева архітектура” зазвичай відноситься до шаблону проектування, який зазвичай використовується в розробці програмного забезпечення, особливо в контексті створення веб-додатків.

Вона розділяє компоненти програми на три взаємопов’язані рівні, кожен з яких має певний набір обов’язків. Ці три шари:

- рівень представлення (Presentation Layer). Це найвищий рівень, який взаємодіє з кінцевим користувачем. Він відповідає за представлення інформації користувачеві та обробку введених даних;
- рівень бізнес-логіки (Business Logic Layer). Середній рівень, також відомий як рівень бізнес-логіки, містить основні функції програми та бізнес-правила. Він обробляє та керує даними, виконує обчислення та реалізує специфічну бізнес-логіку програми. Він не залежить від інтерфейсу користувача та механізмів зберігання даних, що робить його більш модульним і простим у обслуговуванні;
- рівень доступу до даних (Data Access Layer). Нижній рівень займається зберіганням і пошуком даних. Він відповідає за взаємодію з базою даних або будь-яким іншим механізмом зберігання даних для виконання операцій CRUD (створення, читання, оновлення, видалення). Рівень доступу до даних абстрагує базові деталі зберігання даних від рівня бізнес-логіки. Це розділення дозволяє змінювати технологію бази даних, не впливаючи на решту програми.

Виходячи з вищеописаного було побудовано і розбито структуру на окремі модулі, такі як:

- співробітник (Employee);
- проєкт (Project);

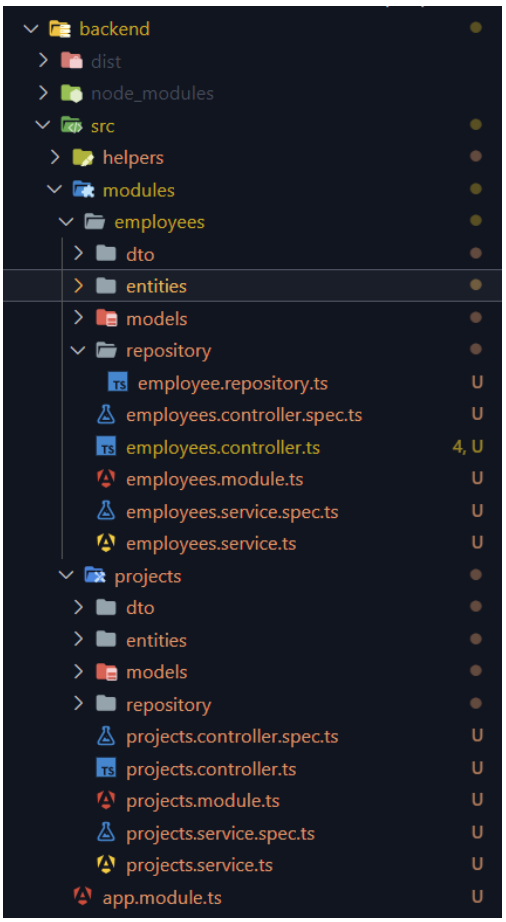


Рисунок 3.7 – Структура серверної частини

Модуль “Співробітник” має у собі три сутності, які зв’язані між собою і не потребують наявності окремих модулів для цих сутностей. Сутності “Співробітник”, “Навичка” та “Департамент” будуть розміщені у одному модулі.

Кожен модуль має репозиторій (repository), який відповідає за роботу з базою даних обраних сутностей.

```
async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);
  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      transform: true,
      forbidNonWhitelisted: true,
    }),
  );
  app.useGlobalFilters(new HttpExceptionFilter());
  app.enableCors();

  const config = new DocumentBuilder()
    .setTitle('OptimizeIT Docs')
    .setDescription('The OptimizeIT API description')
    .setVersion('1.0')
    .build();
  const options: SwaggerDocumentOptions = {
    operationIdFactory: (controllerKey: string, methodKey: string) => methodKey,
  };
  const document = SwaggerModule.createDocument(app, config, options);
  SwaggerModule.setup('swagger', app, document);

  await app.listen(process.env.API_PORT || 3001);
}
bootstrap();
```

Рисунок 3.8 – Конфігурація запуску сервера

На рисунку 3.8 показано конфігурацію для запуску сервера, де зазначено, що застосунок буде використовувати Express фреймворк.

Express – відомий мінімалістичний фреймворк для NodeJS. Він має велику кількість ресурсів та широку підтримку спільноти. Він використовується як стандартний при використанні NestJS.

Можна використовувати як чистий Express-код, так і використовувати функціональності NestJS в тих випадках, коли це необхідно. Це робить фреймворк гнучким та пристосованим до різних вимог проекту.

Увімкнувши глобальну фільтрацію маємо змогу застосувати фільтр логування помилок у системі.

Важливо мати глобальну конфігурацію змінних, які будуть використовуватись для різних залежностей, наприклад, для підключення до бази даних, обрання порту на якому буде запущений сервер та інших важливих конфігурованих сутностей. Вони мають бути правильно оголошені, з чіткою назвою за що кожна з них відповідає та легко змінюватись при потребі. Для

цього оголосимо у головному модулі модуль ConfigModule, який буде доступний глобально для використання у будь-якому модулі.

Невід'ємною складовою є спілкування сервера з базою даних. Оголосимо конфігурацію підключення до бази даних за допомогою TypeORM (рис. 3.9).

```
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    TypeOrmModule.forRoot({
      type: 'postgres',
      username: process.env.DB_USER,
      password: String(process.env.DB_PASSWORD),
      database: process.env.DB_NAME,
      port: Number(process.env.DB_PORT),
      host: process.env.DB_HOST,
      autoLoadEntities: true,
      logging: true,
      migrationsTableName: `migrations`,
      synchronize: true,
      useUTC: true,
      migrationsRun: true,
    }),
    EmployeesModule,
    ProjectsModule,
  ],
})
export class AppModule {}
```

Рисунок 3.9 – Підключення до БД та додавання модулів

TypeORM - це об'єктно-реляційний мапер (ORM) для TypeScript та JavaScript. Він забезпечує зручний спосіб взаємодії з базами даних за допомогою об'єктів та класів, замість безпосереднього використання мови SQL [23].

Основні можливості TypeORM включають:

- підтримка багатьох типів баз даних. TypeORM підтримує роботу з різними типами баз даних, такими як MySQL, PostgreSQL, SQLite, та інші;

- створення таблиць з класів. Ви можете визначити класи для представлення таблиць у базі даних. TypeORM дозволяє вам використовувати

декоратори для визначення відносин між таблицями та інших параметрів;

- міграції. TypeORM надає можливість використовувати міграції для збереження та автоматичного оновлення схеми бази даних;

- запити до бази даних. Ви можете використовувати вищий рівень абстракції для виконання запитів до бази даних, замість того, щоб писати рівень SQL;

- підтримка асинхронності. Враховуючи ріст популярності асинхронного програмування в JavaScript та TypeScript, TypeORM добре підтримує асинхронні запити до бази даних.

Як було зазначено вище про використання глобальних фільтрів, потрібно застосунку дати зрозуміти, що саме і коли має валідуватись. Для кожного API шляху, які мають тіло запиту або параметри запиту, потрібно проводити їх валідацію при кожному їх виклику. З допомогою class-validator плагіна опишемо моделі валідації для кожного запиту, де мають бути вхідні дані або параметри. Важливо вказати правильні вбудовані декоратори валідації, так як він них буде залежати коректність даних, які будуть збережні у базі даних (рис 3.10).

```
export class CreateEmployeeDto {
  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  firstName: string;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  lastName: string;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  birthday: Date;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  place: string;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  phoneNumber: string;
```

Рисунок 3.10 – Створення класів валідації DTO

Для кожного такого запиту створено валідаційні класи, які називаються DTO. DTO або Data Transfer Object (Об'єкт передачі даних) - це патерн проектування програмного забезпечення, який використовується для передачі даних між прошарками застосунку. Основна ідея полягає в тому, щоб упакувати набір даних у окремий об'єкт, який може бути переданий через мережу чи іншими механізмами передачі даних.

DTO може включати в себе лише ті дані, які необхідні для певного завдання чи операції, щоб зменшити обсяг переданих даних та забезпечити ефективність мережевої передачі.

У контексті валідації, DTO використовується для забезпечення та контролю правильності передачі даних між різними шарами системи. Валідаційні класи, пов'язані з DTO, дозволяють перевіряти правильність формату та значень переданих даних, забезпечуючи таким чином консистентність і надійність обміну інформацією між компонентами.

Завдяки його невеликому обсягу та спрощеному визначенню структури, DTO допомагає покращити ефективність роботи системи та зменшити ймовірність помилок при передачі даних.

Крім того, використання DTO дозволяє визначити чіткий контракт між відправником і отримувачем даних, що сприяє розширюваності та підтримці коду системи в майбутньому. Все це робить патерн Data Transfer Object важливим інструментом для розробників при створенні ефективних та надійних систем обробки даних.

Спілкування клієнт-сервер буде відбуватись за допомогою JWT (JSON Web Token) під час аутентифікації користувача і доступу до здійснення запитів. JWT - це стандарт передачі заявок (claims) між двома сторонами в форматі, який може бути перевірений та підписаний. Це зазвичай використовується для аутентифікації та обміну інформацією між веб-серверами і клієнтами. JWT складається з трьох частин: заголовка (Header), заявки (Payload) та підпису (Signature).

Використання JWT у спілкуванні клієнт-сервер має кілька переваг. Одна з найважливіших - це легкість розпізнавання та перевірки токена, оскільки весь необхідний контекст міститься у самому токені. Крім того, вони дозволяють ефективно керувати правами доступу та уникати необхідності постійної перевірки ідентифікаторів користувачів у базі даних.

Виконуючи спілкування клієнт-сервер будемо слідувати принципам REST [24]:

- першим принципом є концепція ресурсів (Resources), де будь-що, що може бути ідентифіковано за допомогою URI, розглядається як ресурс. Ресурси можуть представляти конкретні сутності або колекції даних;
- другий принцип - це представлення (Representation). Кожен ресурс може мати різні представлення, такі як XML, JSON або HTML. Це дає можливість клієнту вибирати той формат, який найбільше відповідає його потребам;
- стан (Stateless) - третій принцип REST вказує на те, що кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для обробки запиту, і сервер не повинен зберігати стан клієнта між запитами.

REST надає універсальний інтерфейс для взаємодії з ресурсами, що дозволяє взаємодіяти з різними типами ресурсів без необхідності змінювати сам протокол. Це забезпечує простоту використання і розширюваності системи.

Принцип гіпермедіа вказує на те, що сервер повинен надсилати клієнту не тільки дані, але й посилання на інші можливі дії або ресурси. Це дозволяє клієнтові динамічно взаємодіяти з сервером, інтерпретуючи отримані дані.

REST рекомендує використання шарованої архітектури, де компоненти системи розташовані на різних рівнях. Це дозволяє визначати окремо логіку представлення, бізнес-логіку і управління даними.

Слідуючи цим принципам для кожного модуля потрібно маршрутизація за допомогою HTTP методів, до прикладу:

- GET – використовується для отримання сутності/сутностей. Одним з прикладів буде отримання списку співробітників - GET “/employee”;

- POST – використовується для створення нової сутності. До прикладу, створення нового співробітника - POST “/employee”;
- PUT – використовується для модифікації сутності. До прикладу, створення нового співробітника - PUT “/employee/:id”;
- DELETE – використовується для повного видалення або софт видалення сутності. Наприклад, видалення співробітника, який вже не працює у компанії – DELETE “/employee/:id”.

Використовуючи принципи REST спрощується розробка API запитів, кожен виклик відповідає тільки за певну сутність, легкість у розширенні і зручність спілкування за допомогою уніфікації запитів клієнт-сервер.

Кожен модуль має свою зону відповідальності, може розроблятись окремо один від одного, полегшується тестування і масштабованість за рахунок незалежності один від одного (рис. 3.7).

3.3 Розробка інтерфейсу користувача і тестування

Розробка інтерфейсу користувача є важливою частиною кожного застосунку для тестування серверної частини і самого методу. Створюючи застосунки з використанням ReactJS, важливим кроком є коректна побудова структури папок розбиваючи їх на логічні частини системи. При побудові структури папок застосовуються найкращі практики організації ReactJS застосунків (рис. 3.11).

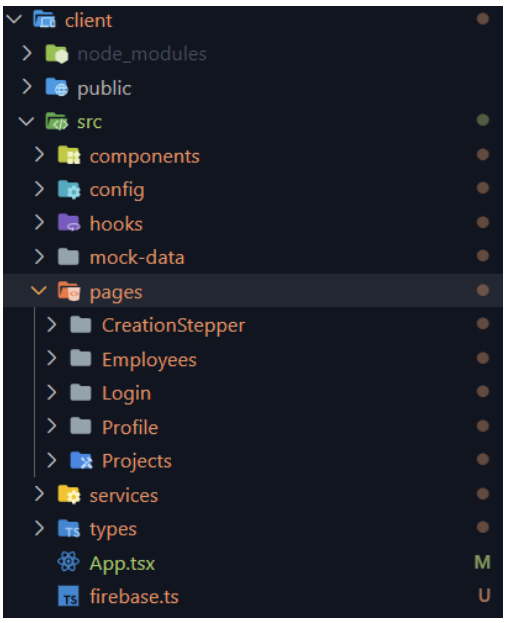


Рисунок 3.11 – Структура папок клієнта

Важливою і невід'ємною частиною є npm-пакети, які встановлюються за допомогою вбудованих команд з зазначенням потрібного імені пакета у автоматичному режимі. Переглянути усі встановленні залежності можна у файлі package.json. Також у ньому зазначаються скрипти за допомогою яких можна запустити клієнтську частину, тестування та режим відлагодження застосунку (рис. 3.12).

Рисунок 3.12 – Скрипти запуску застосунку та його залежності

Першим кроком створення інтерфейсу користувача потрібно створити сторінку автентифікації через, яку користувач буде потрапляти у систему. Вона буде мати поля email та пароль, які будуть надіслані до Firebase Authentication за допомогою вбудованих методів залежності firebase викликаючи signInWithEmailAndPassword() з вказаними параметрами (рис 3.13).

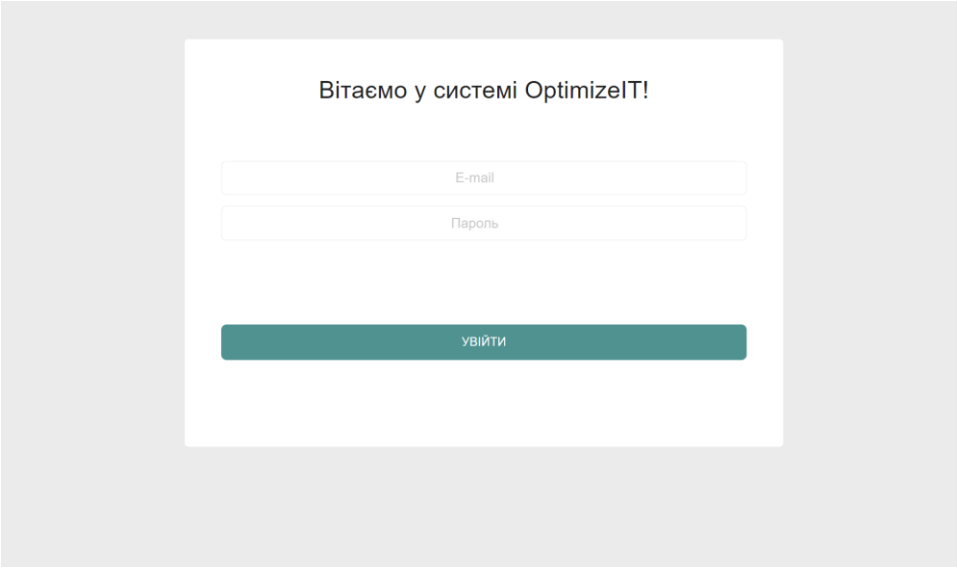


Рисунок 3.13 – Сторінка автентифікація користувача

Після автентифікації користувач потрапляє на сторінку з співробітниками, де вони відображені у вигляді таблиці, як було спроєктовано. Розроблений інтерфейс взаємодії зі списком співробітників зображено на рисунку 3.14. Таблиця має пагінацію, область, яка відповідає за кількість сторінок записів у таблиці та переходу по ним для зручного відображення записів по сторінках. Вона виконується на стороні клієнта, так як, кількість записів не буде великою і дана таблиця підтримує віртуалізацію великої кількості записів для оптимізації відображення. Дані для користувачів отримуються викликаючи запит GET «/employee».

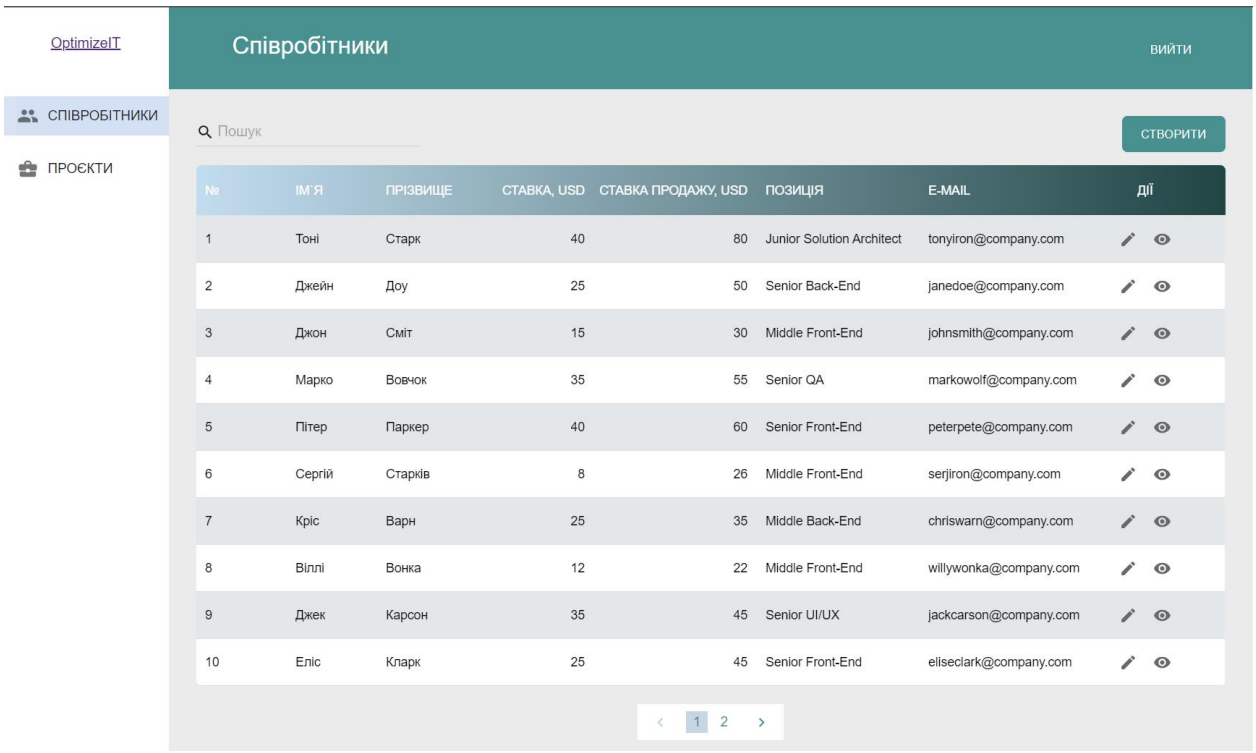


Рисунок 3.14 – Інтерфейс списка користувачів

При натисканні кнопки «Створити» користувач буде перенаправлений на сторінку з формою створення нового співробітника з зручними полями для введення інформації про нього (рис. 3.15). При натисканні на кнопку «Зберегти» введені дані будуть відправлені на зберігання за допомогою виклику команди з сервісу POST для створення і PUT для оновлення «/employees», де у вигляді JSON у тілі запита будуть відправлені на сервер.

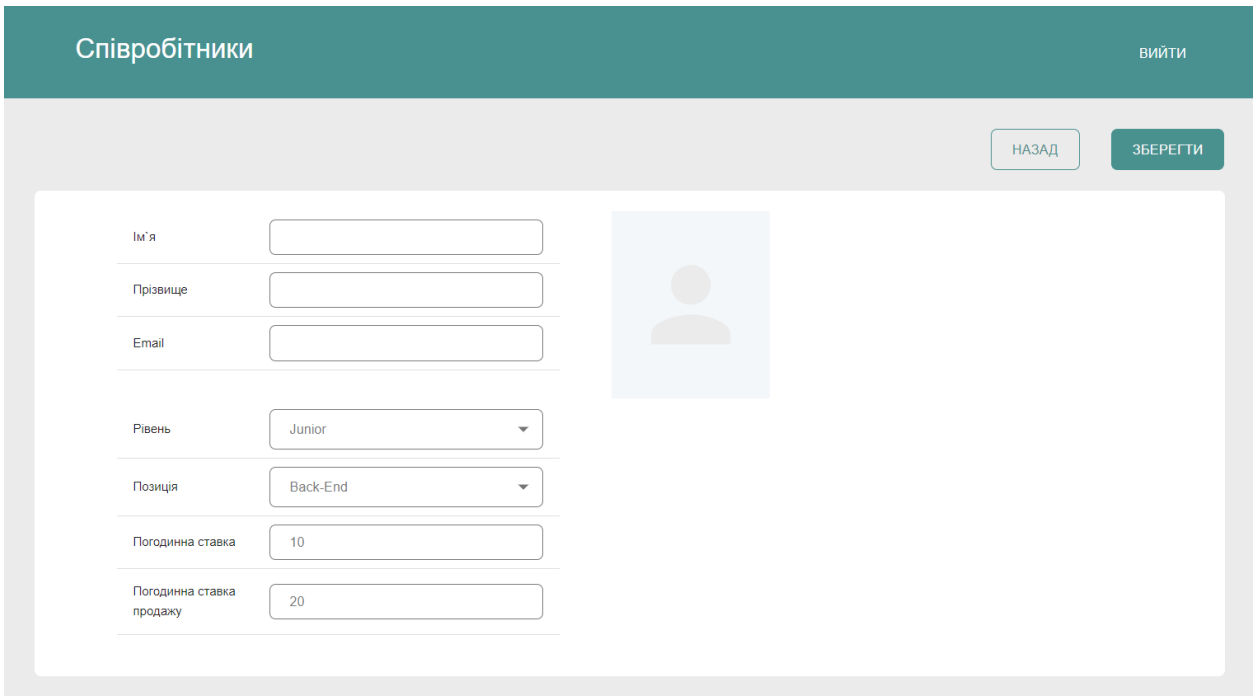


Рисунок 3.15 – Інтерфейс додавання/редагування користувача

Далі розглянемо сторінку з проектами на рисунку 3.16. Вона розроблена по такому самому принципу, що і сторінка списку співробітників.

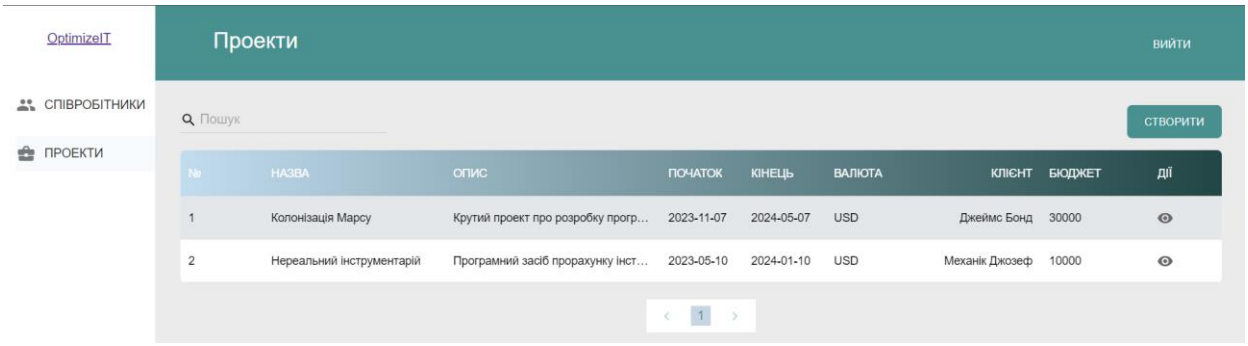


Рисунок 3.16 – Інтерфейс списку проектів

При натисканні на кнопку іконки “Око” користувач буде перенаправлений на перегляд, а при натисканні “Створити” користувач буде перенаправлений на сторінку створення нового проекта, де і буде відбуватись калькуляція за допомогою розробленого метода.

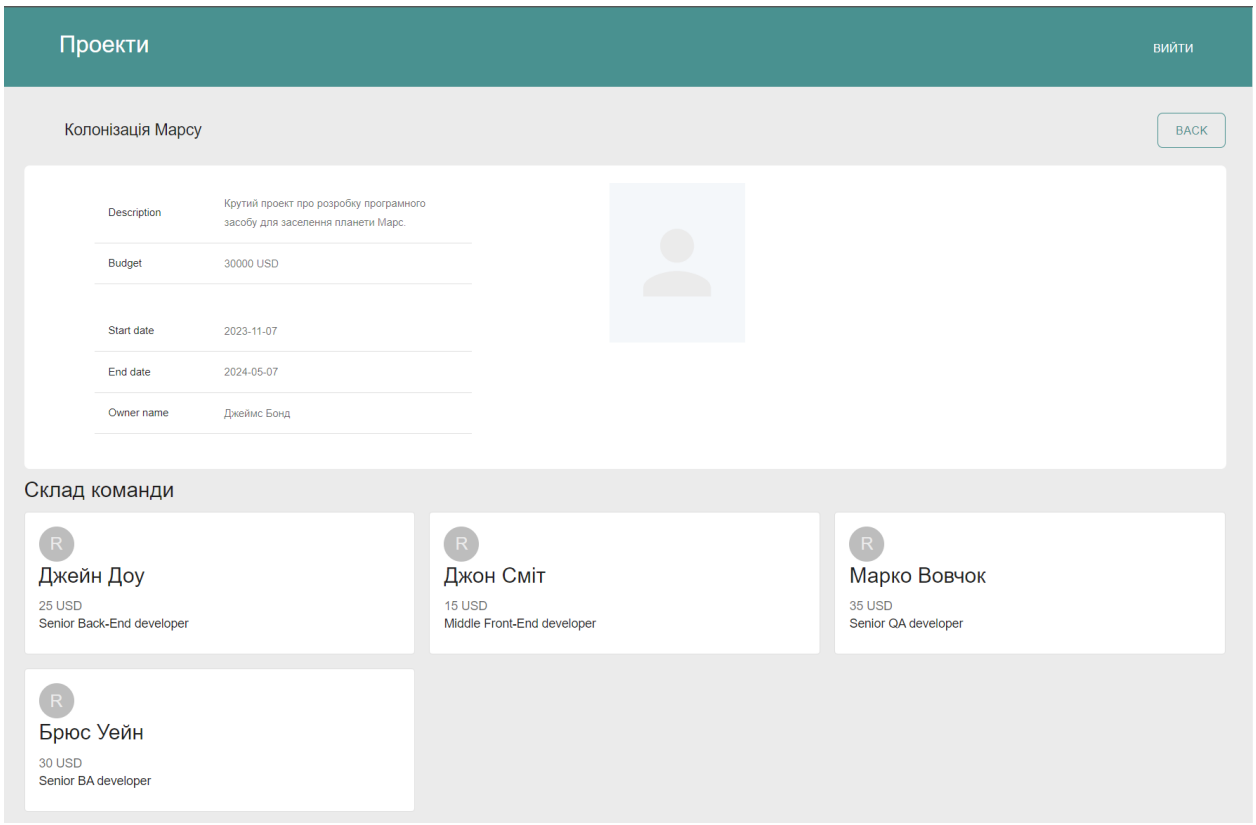


Рисунок 3.17 - Інтерфейс перегляду проекта

Далі розглянемо основну частину програмного засобу, який і є основою проекту для оптимізації ресурсів. На рисунку 3.18 можна побачити сторінку створення проєкта, яка розділена на кроки для структуризації даних. Усі кроки позначені відповідними іконками та кольорами для простого орієнтування і розумінням, що система буде потребувати на кожному з кроків. Перший крок містить поля для заповнення основної інформації про проект, включаючи його назву, короткий опис, бюджет та дати початку і кінця, які є одними зі складових у оптимізації ресурсів. Зазначення дати початку і кінця впливають на наступний крок, так як ставлять часові рамки протягом яких можна буде зазначати задачі. По натисканні кнопки «Далі» користувач буде перенаправлений на наступний крок, а попередньо введена інформація буде збережена у формі.

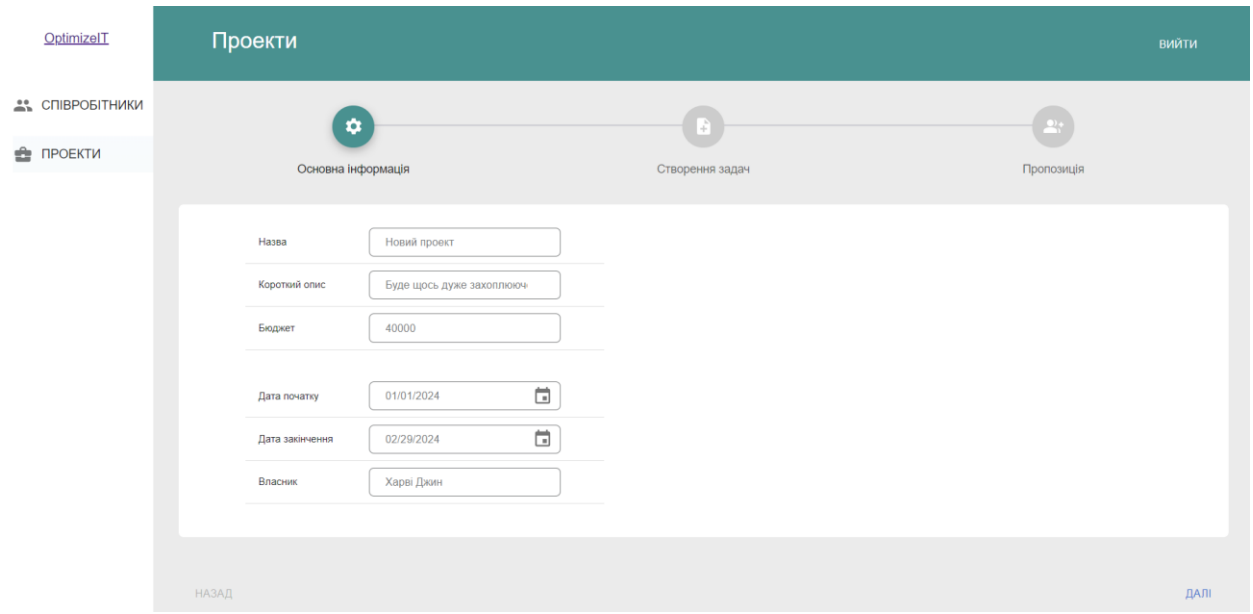


Рисунок 3.18 – Інтерфейс першого кроку створення проекту

Другим кроком є введення інформації по кожній з задач, які мають реалізуватись у даному новому проекті. Від правильності введених даних буде залежати оптимальний результат який ми отримуємо після опрацювання інформації розробленим методом оптимізації для отримання найкращого складу команди, яку можна залучити на даних проект (рис. 3.19). При натисканні «Додати задачу» з’являється новий пустий рядок з елементами для введення даних. Як було попередньо зазначено у полях дат початку і закінчення задачі стоять рамки відносно зазначених дат початку і закінчення проекту. Кожен рядом за потреби можна видалити.

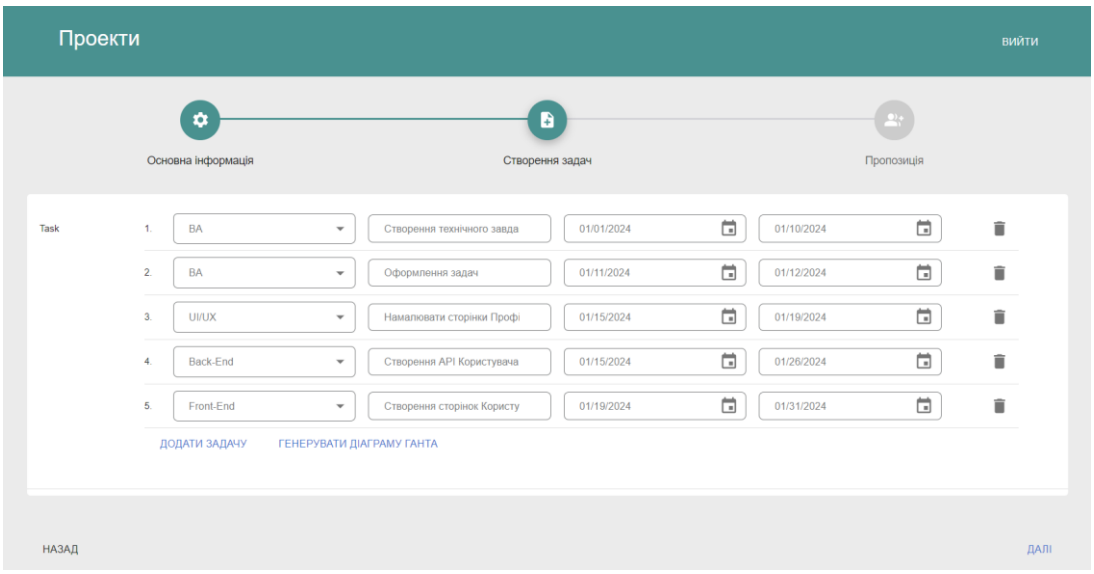


Рисунок 3.19 – Інтерфейс другого кроку введення інформації про задачі

На рисунку 3.20 наведено згенеровану діаграму Ганта, яка дозволяє візуально оцінити потенційні задачі і їх строки і у випадку неспівпадіння – відредагувати їх. Вона показується після заповнення хоча б однієї задачі та натисканні кнопки «Генерувати діаграму Ганта». При додаванні або зміні інформації будь-якої задачі та повторному натисканні на кнопку «Генерувати діаграму Ганта» діаграма буде перемальнована відповідно до змін.

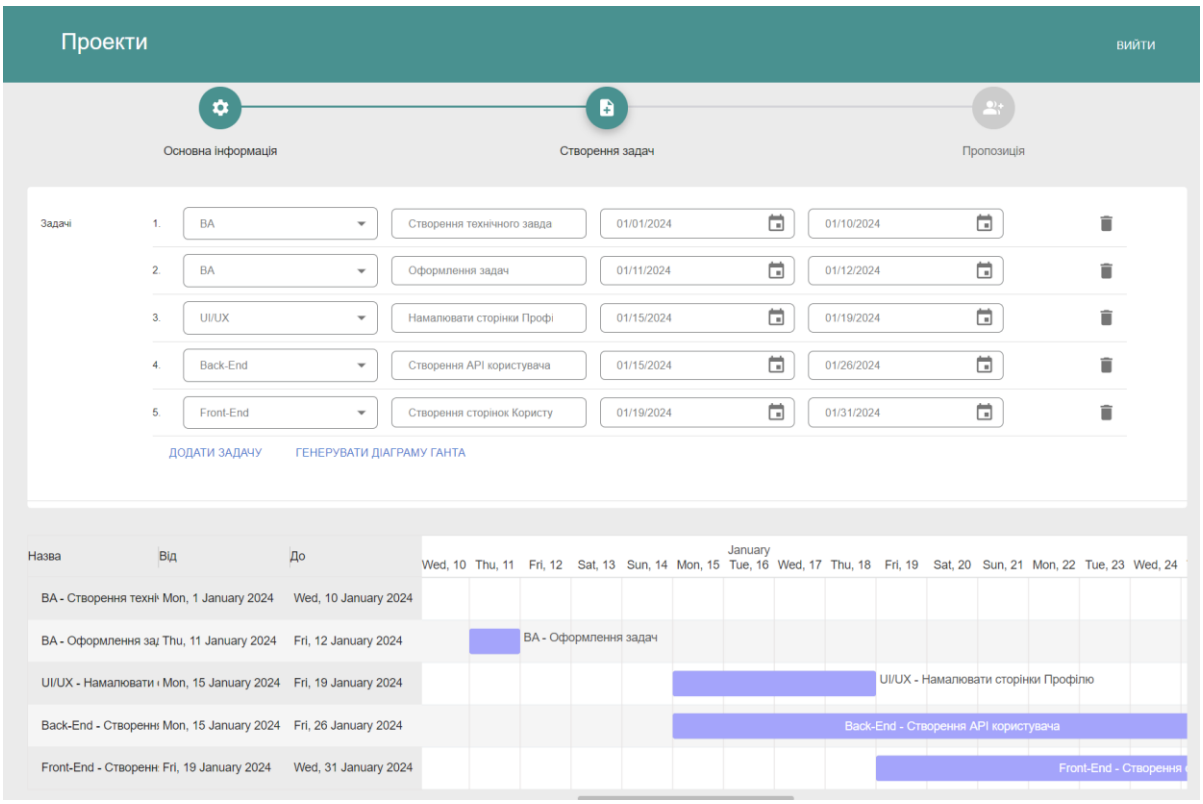


Рисунок 3.20 – Інтерфейс другого кроку з генерацією діаграми Ганта на основі введених задач

Далі, щоб закінчити наше створення проєкту і отримати результат за допомогою розробленого методу потрібно натиснути на кнопку «Отримати результати».

На основі усіх введених даних (рис. 3.21) було згенеровано оптимізовану пропозицію ресурсів системи засновану на розробленому методі оптимізації ресурсів. Під кожний департамент зазначено перелік співробітників, їх години, які потрібні для виконання задач та калькульовану ставку для співробітника та компанії.

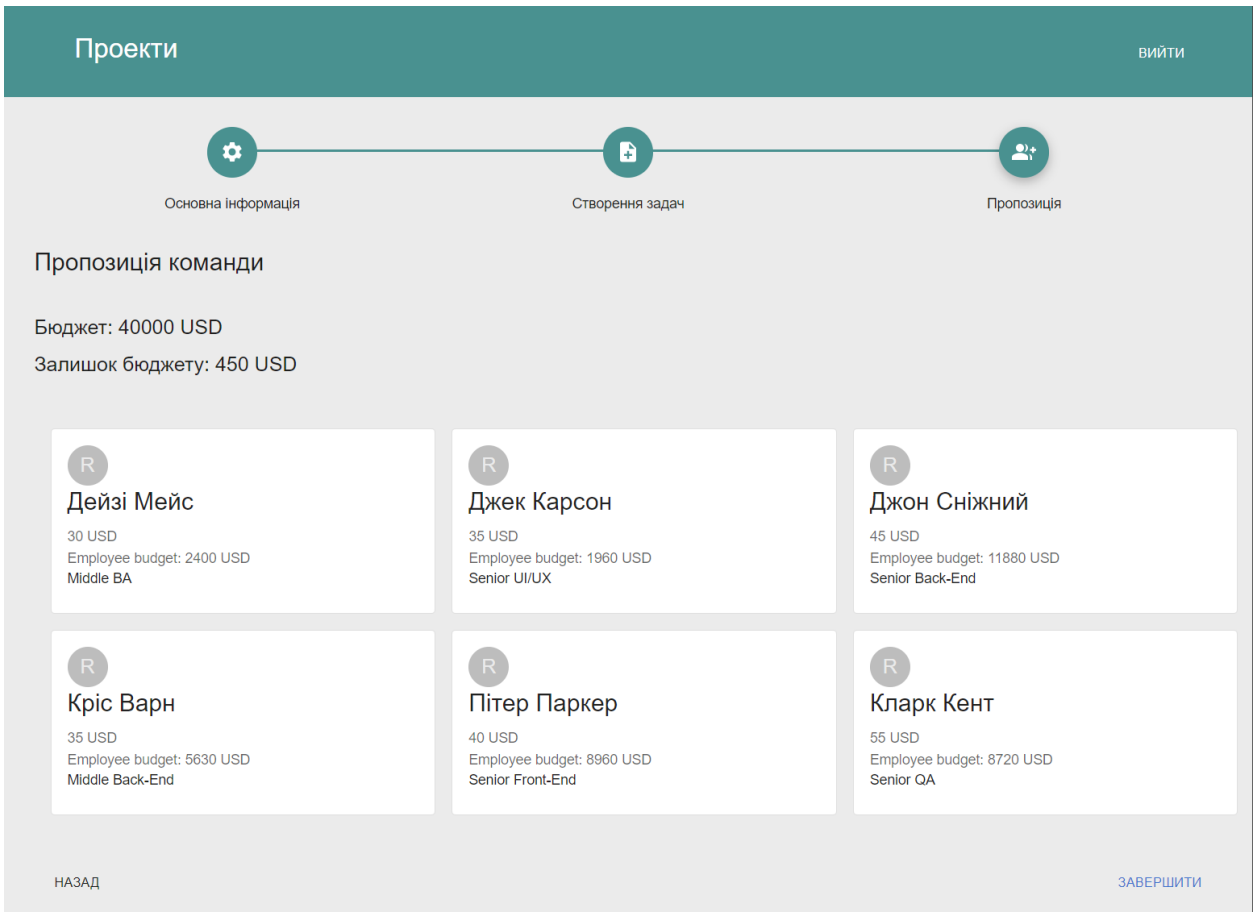


Рисунок 3.21 – Інтерфейс пропозиції на основі розробленого метода

Кожен програмний засіб потребує тестування для виявлення помилок розроблених засобів, його логічних частин та невідповідності зазначених даних. За допомогою тестування розроблений засіб буде приведений відповідно до вимог у коректно працюючий стан без помилок.

Крім того, важливим етапом тестування програмного засобу є визначення його продуктивності та масштабованості. Тестування продуктивності дозволяє оцінити швидкість та ефективність роботи програми при різних навантаженнях та умовах. Методи тестування продуктивності можуть включати в себе аналіз часу відповіді, обсяг використаної пам'яті, а також оптимізацію ресурсів.

Також, важливим етапом є тестування на безпеку. Забезпечення високого рівня безпеки програмного засобу є критично важливим, оскільки це захищає дані користувачів і запобігає несанкціонованому доступу. Тестування на безпеку може включати в себе аналіз вразливостей, тестування на витоки

даних, аутентифікації та авторизації.

Для забезпечення високої якості тестування використовуються різні підходи, такі як автоматизоване тестування, ручне тестування, а також комбіновані методи. Важливо враховувати, що тестування – це неперервний процес, і воно варто проводити на всіх етапах розробки програмного продукту для виявлення та виправлення помилок якнайраніше.

У випадку такого програмного засобу підійде такий метод тестування як “Сірі склянки”.

Тестування “Сірих склянок” поєднує у собі елементи як “Чорного ящика”, так і “Білого ящика” тестування [25]. Тестувальник має частковий доступ до внутрішніх деталей системи, що дозволяє йому створювати тестові випадки, які враховують як зовнішню, так і внутрішню поведінку програми. Це дозволяє покрити усі модулі системи, які з них є незалежними або залежними від іншого модуля.

На основі створеного інтерфейсу користувача та API, складаємо порядок дій тестування засобу. Це тестування охопить кожен модуль системи, API запити та метод оптимізації ресурсів. Розглянемо такий кейс використання:

- потрапляння на сторінку співробітників, після автентифікації;
- створення інформації про співробітника;
- редагування інформації у разі зазначення неправильних даних;
- перехід на сторінку проєктів і створення проєкта;
- визначення основної інформації про проєкт, введення коректних дат початку і закінчення та бюджет;
- перехід на наступний крок та створення переліку задач по датам і департаментам;
- перехід на останній крок та отримання оптимального складу команди для проєкту на основі розробленого метода.

Сформований сценарій покриває основні складові системи та надає змогу слідкувати за змінами у системі.

3.4 Висновки

В результаті проектування та розробки програмного засобу, включаючи метод оптимізації ресурсів, були розглянуті і описані основні частини розробки.

Розробка програмного засобу є не простою задачею, яка має включати багато чинників від яких залежить потенційна дохідність від проектів і розвиток компаній у IT галузі загалом. Окрім, програмний засіб має забезпечити конфіденційність та безпеку персональних даних співробітників, їх збереження та обробку, що було враховано при розробці програмного засобу.

Виходячи з цього, результатом є розроблений програмний засіб, який складається з серверної частини та інтерфейсу користувача, які використовують розроблений метод оптимізації ресурсів, необхідний для підбору оптимальних ресурсів для залучення на відповідний проект.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Метод і програмний засіб оптимізації ресурсів у ІТ» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод і програмний засіб оптимізації ресурсів у ІТ» 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. 5.1 [27-28].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і 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	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
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
Практична здійсненність					

Продовження таблиці 5.1

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 використовують ся у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію	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 звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	3	3	4
2. Ринкові переваги (наявність аналогів)	4	3	3
3. Ринкові переваги (ціна продукту)	4	3	3
4. Ринкові переваги (технічні властивості)	3	3	3
5. Ринкові переваги (експлуатаційні витрати)	3	4	3
6. Ринкові перспективи (розмір ринку)	3	3	4
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	3	3	4
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	4
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	38	37	41
Сума балів	3	3	4
Середньоарифметична сума балів $СБ_c$	**38,7**		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [27-28].

Таблиця 5.3 – 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 «Метод і програмний засіб оптимізації ресурсів у ІТ» становить 38,7 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод і програмний засіб оптимізації ресурсів у ІТ», 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 (Зо) розраховуємо у відповідності до посадових окладів працівників, за формулою [27-28]:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_i}{Т_р}, \qquad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

Мпі – місячний посадовий оклад конкретного дослідника, грн;

ti – число днів роботи конкретного дослідника, дні;

Тр – середнє число робочих днів в місяці, Тр=22 дні.

Зо = 30000,00 · 68 / 22 = 92727,27 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	30000	1363,64	68	92727,27
Інженер-розробник програмного забезпечення	25000	1136,36	68	77272,73
Консультант	12000	545,45	10	5454,55
Всього				175454,55

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{4.2}$$

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

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

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

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

де ММ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо Мм=6700,00 грн;

Кі – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [27-28];

Кс – 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.

Тр – середнє число робочих днів в місяці, приблизно Тр = 22 дні;

tзм – тривалість зміни, год.

С1 = 6700,00 · 1,10 · 1,65 / (22 · 8) = 69,09 грн.

Зр1 = 69,09 · 8,00 = 552,75 грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця дослідника	8	2	1,1	69,09	552,75
Інсталяція програмного забезпечення	3	5	1,7	106,78	320,34
Тестування	40	5	1,7	106,78	4271,25
Всього					5144,34

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 Ндод – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (175454,55+ 5144,34) · 11 / 100% = 19865,88грн.

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 Нзп – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (175454,55+ 5144,34+ 29096,04) · 22 / 100% = 44102,25грн.

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.

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

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

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

n – кількість видів матеріалів;

Цj – вартість матеріалу j-го найменування, грн/кг;

Кj – коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15);

Вj – маса відходів j-го найменування, кг;

Цвj – вартість відходів j-го найменування, грн/кг.

М1 = 3 · 210,00 · 1,1 - 0,000 · 0,00 = 693,0 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір	210	3	0	0	693
Канцелярське приладдя (набір офісного працівника)	175	2	0	0	385
Flesh-пам'ять	130	1	0	0	143
Всього					1221

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі (Кв), які використовують при проведенні НДР на тему «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі» відсутні.

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на «Спецустаткування для наукових (експериментальних) робіт» в роботі на тему «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ» відсутні.

4.2.6 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:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.7)$$

де Ціпрг – ціна придбання одиниці програмного засобу даного виду, грн;

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

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

k – кількість найменувань програмних засобів.

Впрг = 11600,00 · 1 · 1,1 = 12992 грн.

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	1	11600	12992
Прикладний пакет Microsoft Office 2019	1	5500	6160
Всього			19152

4.2.7 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:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.8)$$

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

tвик – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

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

Аобл = (30000,00 · 3) / (3 · 12) = 2500 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук ASUS Vivobook 15 X1504ZA-BQ359 (90NB1021-M01250)	30 000	3	3	2500,00
Робоче місце дослідника	210000	5	3	10500,00
Оргтехніка	6000	4	3	375,00
ОС Windows 11	12992	1	3	3248,00
Прикладний пакет Microsoft Office 2019	6160	2	3	770,00
Всього				17393,00

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (Ве) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i}, \qquad (4.9)$$

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

ti – тривалість роботи обладнання на етапі дослідження, год;

Це – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо Це = 7,50 грн;

Квпі – коефіцієнт, що враховує використання потужності, Квпі <1;

ηі – коефіцієнт корисної дії обладнання, ηі<1.

Ве = 0,5 · 544,0 · 7,50 · 0,95 / 0,97 = 1997,94 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
HP 205 G8 Starry White (6D452EA)-2шт	0,5	544	1997,94
Робоче місце дослідника	0,15	544	599,38
Оргтехніка	0,45	20	66,11
Всього			2663,43

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи 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:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.10)$$

де Нсв – норма нарахування за статтею «Службові відрядження», приймемо Нсв = 20%.

Всв = (175454,55+ 5144,34) · 20 / 100% = 36119,78 грн.

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми

основної заробітної плати дослідників та робітників за формулою:

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

де Нсп – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо Нсп= 35%.

Всп = (175454,55+ 5144,34) · 35 / 100% = 63209,61 грн.

4.2.11 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:

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

де Нів – норма нарахування за статтею «Інші витрати», приймемо Нів = 50%.

Ів = (175454,55+ 5144,34) · 50 / 100% = 90299,44 грн.

4.2.12 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:

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

де Ннзв – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо Ннзв = 120%.

Внзв = (175454,55+ 5144,34) · 120 / 100% = 216 718,67 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + І_в + B_{нзв}. \quad (5.14)$$

Взаг = 785943,94 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=0,8.

ЗВ = 785943,94/ 0,9 = 982429,93 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 передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 4000 компаній користувачів/рік;

2-й рік – 5000 компаній користувачів/рік;

3-й рік – 3500 компаній користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 20000 компаній користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 75000,00 грн /за рік користування;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 1500,00 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 [27-28]:

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

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ=30%;

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

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(20000,00·1500,00+76500·4000)·0,83·0,3·(1-0,18/100%)= 68852448 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(20000,00·1500,00+76500·(4000+5000))·0,83·0,3·(1-0,18/100%)= 147233583 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(20000,00·1500,00+76500·(4000+5000+3500))·0,83·0,3·(1-0,18/100%)= 202100377,5 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=0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором

додаткових чистих прибутків у цьому році.

$ПП =$ 68852448/(1+0,25)1+147233583/(1+0,25)2+202100377,5/(1+0,25)3=252786844,80 грн.

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

$$PV = k_{інв} \cdot ЗВ, \tag{4.18}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=4;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 982429,93 грн.

$PV = k_{інв} \cdot ЗВ = 4 \cdot 982429,93 = 3929719,722.$

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

$$E_{абс} = ПП - PV \tag{4.19}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 252786844,80грн;

PV – теперішня вартість початкових інвестицій, 3929719,722грн.

$E_{абс} = ПП - PV$ = 252786844,80- 3929719,722= 248857125,08грн.

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, 248857125,08грн;

PV – теперішня вартість початкових інвестицій, 3929719,722грн;

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

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+248857125,08/3929719,722)^{1/3} = 3,01.$$

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 =0,11;

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

$\tau_{мін}$ = 0,11+0,25 = 0,36 < 3,01 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 «Метод і програмний засіб оптимізації ресурсів у ІТ» доцільно.

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

$$T_{ок} = \frac{1}{E_в}, \quad (4.22)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок} = 1 / 3{,}01 = 0{,}33$ року.

$T_{ок} <$ 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.

4.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі» становить 38,7 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 0,33 р., 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 «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі».

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було розроблено метод і програмний засіб оптимізації ресурсів на проектах у IT галузі. Створений метод був включений у розроблений застосунок для оптимізації ресурів на проєктах у IT. Для розробки програмного засобу було обрано середивище розробки Visual Studio Code. Робота оформлена згідно методичних вказівок [28].

Виконавши аналіз методів оптимізації ресурсів та програмних засобів управління співробітниками було визначено мету проєкту, яка полягає у розробці методу та програмного засобу на основі досліджених методів оптимізації ресурсів, спрямованих на оптимізацію використання ресурсів на IT проектах. Основною метою є підвищення ефективності управління проектами в IT галузі, збільшення прибутковості компанії.

Розробивши усі складові програмного засобу було проведено його тестування. Важливим кроком у цьому було забезпечення засобу правильними даними у достатній кількості, щоб можна було провести якісне тестування.

Результатом проведеного аналізу, проєктування та розробки є створений метод, який допомагає оптимізувати ресурси на проєктах та його застосування у програмному засобі. Розроблений метод є актуальним і має високе практичне використання у сфері IT та потребує подальшого вдосконалення для досягнення найкращих результатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Human Resources Management System Vol 5, Jie Cai, Kehinde David Folarin, 2021. 112p.

2. Ковальський С.В., Тужанський С.Є. Оцінювання та вимірювання успіху освіти з використанням цифрових інструментів – 2023, матеріали Міжнародної науково-практичної Інтернет-конференції “Електронні інформаційні ресурси: створення, використання, доступ 2023”

3. The Future of Jobs Report 2020. [Електронний ресурс]. URL: https://www.weforum.org/reports/the-future-of-jobs-report-2020/digest/

4. What is ERP. URL: https://www.sap.com/ukraine/products/erp/what-is-erp.html

5. Introduction to Algorithms. Fourth Edition. The MIT Press, 2023. 1412 p.

6. Identity and Data Security for Web Development: Best Practices. Jonathan LeBlanc & Tim Messerschmidt, 2016. 201 p.

7. Firebase Authentication. [Електронний ресурс]. URL: https://firebase.google.com/docs/auth.

8. AWS Cognito. [Електронний ресурс]. URL: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html

9. JavaScript. The Definitive Guide. Master the World’s Most-Used Programming Language 7th Edition. By David Flanagan. O`Reilly Publishing, 2020. 706 p.

10. What is TypeScript and how to use it. [Електронний ресурс]. URL: https://www.typescriptlang.org/

11. Introduction to NodeJS. [Електронний ресурс]. URL: https://nodejs.dev/en/learn/

12. Nest.js: A Progressive Node.js Framework. By Greg Magolan, Patric Housley Birmingham: Packt Publishing, 2019. 317 p.

13. PostgreSQL Overview – Introduction. [Електронний ресурс]. URL: https://www.tutorialspoint.com/postgresql/postgresql_overview.htm

14. Learning React. Modern Pattern by Alex Banks, Eve Porcello, 2020. 307 p.

15. JetBrains WebStorm IDE. [Електронний ресурс]. URL:

https://www.jetbrains.com/webstorm/

16. Visual Studio Code. Code editing. Redefined. [Електронний ресурс]. URL: https://code.visualstudio.com/

17. Конноллі Т. Бази даних. Проектування, реалізація та супровід. Теорія та практика. 3-тє видання / Т. Конноллі, К. Бегг.

18. Relational Decomposition. [Електронний ресурс]. URL: https://www.geeksforgeeks.org/properties-of-relational-decomposition/.

19. Michael J. Hernandez. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. Boston: Addison-Wesley, 2003. 611 p.

20. Чиста архітектура. Друге видання – Роберт Мартін. Фабула, 2019. 318 ст.

21. Intuitive Web Design: Hot to make your website intuitive to use. [Електронний ресурс]. URL: https://cxl.com/blog/intuitive-web-design-how-to-make-your-website-intuitive-to-use.

22. The Definition of User Experience (UX). [Електронний ресурс]. URL: https://nngroup.com/articles/definition-user-experience.

23. TypeORM Documentation. [Електронний ресурс]. URL: https://typeorm.io/

24. RESTful Web API Patterns and Practices Cookbook – Mike Amundsen. O`Reilly Publishing, 2022. 468 ст.

25. Myers G. The Art of Software Testing – Chicago, 2012. 178 p.

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

27. Кавецький В. В**.** Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

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

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"19" вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
асистент кафедри ПЗ, к.т.н.
С.Є.Тужанський
"__19__"_____вересня_____2023 р.
Виконав:
студент гр.3ПІ-22м С.В. Ковальський
"__19__"_____вересня_____2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Метод і програмний засіб оптимізації ресурсів на проектах у IT галузі».

Галузь застосування – IT компанії.

2. Підстава для розробки

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки

Метою роботи є підвищення ефективності управління проектами в IT галузі, збільшення дохідності компанії, а також забезпечення проектами усіх співробітників.

Призначення роботи – розробка метода і програмного засоба оптимізації ресурсів на проектах у IT.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Introduction to Algorithms. Fourth Edition. The MIT Press, 2023. 1412 p.
2. Nest.js: A Progressive Node.js Framework. By Greg Magolan, Patric Housley Birmingham: Packt Publishing, 2019. 317 p.
3. Learning React. Modern Pattern by Alex Banks, Eve Porcello, 2020. 307 p.

4. Технічні вимоги

Вхідні дані – інформації про співробітників, інформації про проєкти, типи департаментів; Вихідні дані – оптимізований список ресурсів з даними для нового проекту.

5. Конструктивні вимоги

Конструкція застосунків серверної і клієнтської частин мають відповідати найкращим практикам та використовувати загально відомі шаблони проектування. Код має збиратись без помилок.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз предметної області	20.09.2023 – 10.10.2023
2	Проектування програмного засобу для оптимізації ресурсів	10.10.2023 – 30.10.2023
3	Програмна реалізація та впровадження методу оптимізації	01.11.2023 – 30.11.2023
4	Економічна частина	21.11.2023 – 01.12.2023

9. Порядок контролю та прийняття

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

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: «Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі»

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ, ЗПІ-22м

Науковий керівник: к.т.н., асистент каф. ПЗ Тужанський С. Є.

Unicheck	
Оригінальність	**97%**
Схожість	3%

Аналіз звіту подібності

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

2. 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.

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

Особа, відповідальна за перевірку _____ Черноволик Г.О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Ковальський С. В.

Керівник роботи _____ Тужанський С. Є.

Додаток В

Лістинг коду серверної частини

```
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { AppModule } from './app.module';
import { NestExpressApplication } from '@nestjs/platform-express';
import { HttpExceptionFilter } from './helpers/http-exception.filter';
import {
 DocumentBuilder,
 SwaggerDocumentOptions,
 SwaggerModule,
} from '@nestjs/swagger';

async function bootstrap() {
 const app = await NestFactory.create<NestExpressApplication>(AppModule);
 app.useGlobalPipes(
  new ValidationPipe({
   whitelist: true,
   transform: true,
   forbidNonWhitelisted: true,
  }),
 );
 app.useGlobalFilters(new HttpExceptionFilter());
 app.enableCors();

 const config = new DocumentBuilder()
  .setTitle('OptimizeIT Docs')
  .setDescription('The OptimizeIT API description')
  .setVersion('1.0')
  .build();
 const options: SwaggerDocumentOptions = {
  operationIdFactory: (controllerKey: string, methodKey: string) => methodKey,
 };
 const document = SwaggerModule.createDocument(app, config, options);
 SwaggerModule.setup('swagger', app, document);

 await app.listen(process.env.API_PORT || 3001);
}
bootstrap();

# Use these environment variables only for local development

DB_HOST=localhost
DB_USER=postgres
DB_PASSWORD=
DB_NAME=
DB_PORT=5432
APP_PORT=3001

import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { EmployeesModule } from './modules/employees/employees.module';
import { ProjectsModule } from './modules/projects/projects.module';

@Module({
 imports: [
  ConfigModule.forRoot({
   isGlobal: true,
  }),
  TypeOrmModule.forRoot({
   type: 'postgres',
   username: process.env.DB_USER,
```

```
      password: String(process.env.DB_PASSWORD),
      database: process.env.DB_NAME,
      port: Number(process.env.DB_PORT),
      host: process.env.DB_HOST,
      autoLoadEntities: true,
      logging: true,
      migrationsTableName: `migrations`,
      synchronize: true,
      useUTC: true,
      migrationsRun: true,
    }),
    EmployeesModule,
    ProjectsModule,
  ],
})
export class AppModule {}

import {
  ExceptionFilter,
  Catch,
  ArgumentsHost,
  HttpException,
  Logger,
  HttpStatus,
} from '@nestjs/common';
import { Request, Response } from 'express';

@Catch(HttpException)
export class HttpExceptionFilter implements ExceptionFilter {
  private readonly logger: Logger;

  constructor() {
    this.logger = new Logger();
  }

  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const request = ctx.getRequest<Request>();

    const statusCode =
      exception instanceof HttpException
        ? exception.getStatus()
        : HttpStatus.INTERNAL_SERVER_ERROR;

    const message =
      exception instanceof HttpException
        ? exception.getResponse()['message']
          ? exception.getResponse()['message']
          : exception['message']
        : HttpStatus.INTERNAL_SERVER_ERROR;

    const devErrorResponse: any = {
      statusCode,
      timestamp: new Date().toISOString(),
      path: request.url,
      method: request.method,
      message: message,
    };

    const prodErrorResponse: { statusCode: number; message: string } = {
      statusCode,
      message,
    };

    this.logger.log(
```

```
      `Request method: ${request.method}, request url: ${request.url}`,
      JSON.stringify(devErrorResponse),
    );

    const environment = 'development';

    response
      .status(statusCode)
      .send(
        environment === 'development' ? devErrorResponse : prodErrorResponse,
      );
  }
}

import {
  Column,
  Entity,
  PrimaryGeneratedColumn,
  JoinTable,
  ManyToMany,
} from 'typeorm';
import { Skill } from './skills.entity';
import { Employee } from './employee.entity';

@Entity()
export class Department {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @ManyToMany(() => Employee, (emp) => emp.departments, {
    onUpdate: 'CASCADE',
    onDelete: 'CASCADE',
  })
  @JoinTable({ name: 'employee_department' })
  employees?: any[];

  @ManyToMany(() => Skill, (skill) => skill.departments, {
    onUpdate: 'CASCADE',
    onDelete: 'CASCADE',
  })
  @JoinTable({ name: 'skill_department' })
  skills?: any[];
}

import {
  Column,
  CreateDateColumn,
  Entity,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
  DeleteDateColumn,
  JoinTable,
  ManyToMany,
} from 'typeorm';
import { Project as ProjectEntity } from '../../../modules/projects/entities/project.entity';
import { Project } from '../../../modules/projects/models/project.model';
import { Skill } from './skills.entity';
import { Department } from './department.entity';

@Entity()
export class Employee {
  @PrimaryGeneratedColumn('uuid')
  id: string;
```

```
@Column({ name: 'first_name' })
firstName: string;

@Column({ name: 'last_name' })
lastName: string;

@CreateDateColumn({
  type: Date,
  nullable: true,
  name: 'birthday',
})
birthday: Date;

@Column({
  type: 'int4',
})
position: number;

@Column({
  type: 'int4',
})
level: number;

@Column()
email: string;

// from 1 to 10
@Column({ name: 'skills_weight', nullable: true })
skillsWeight: number;

@Column({ name: 'hour_rate' })
hourRate: number;

@Column({ name: 'sell_hour_rate', nullable: true })
sellHourRate: number;

@CreateDateColumn({
  type: Date,
  default: () => 'CURRENT_TIMESTAMP',
  name: 'start_date',
})
startDate: Date;

@CreateDateColumn({
  type: Date,
  default: () => 'CURRENT_TIMESTAMP',
  name: 'created_at',
})
createdAt: Date;

@UpdateDateColumn({
  type: Date,
  default: () => 'CURRENT_TIMESTAMP',
  name: 'updated_at',
})
updatedAt: Date;

@DeleteDateColumn({
  type: Date,
  name: 'deleted_at',
})
deletedAt: Date;

/**
 * Employee Skills
```

```
   */
  @ManyToMany(() => Skill, (sk) => sk.employees, {
    onUpdate: 'CASCADE',
    onDelete: 'CASCADE',
  })
  skills: ISkill[];

  @ManyToMany(() => ProjectEntity, (pr) => pr.employees, {
    onUpdate: 'CASCADE',
    onDelete: 'CASCADE',
  })
  projects?: Project[];

  @ManyToMany(() => Department, (dp) => dp.employees, {
    onUpdate: 'CASCADE',
    onDelete: 'CASCADE',
  })
  departments?: IDepartment[];
}

import {
  Column,
  Entity,
  PrimaryGeneratedColumn,
  ManyToMany,
  JoinTable,
} from 'typeorm';
import { Department } from './department.entity';
import { Employee } from './employee.entity';

@Entity()
export class Skill {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column({ name: 'category_name' })
  categoryName: string;

  @ManyToMany(() => Employee, (dep) => dep.skills)
  @JoinTable({ name: 'skill_employee' })
  employees?: Employee[];

  @ManyToMany(() => Department, (dep) => dep.skills)
  departments?: IDepartment[];
}

import { ApiProperty } from '@nestjs/swagger';
import {
  IsString,
  IsDateString,
  IsEmail,
  IsNumber,
  IsNotEmpty,
  IsInt,
} from 'class-validator';

export class CreateEmployeeDto {
  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  firstName: string;

  @IsString()
```

```
  @IsNotEmpty()
  @ApiProperty()
  lastName: string;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  birthday: Date;

  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  phoneNumber: string;

  @IsInt()
  @IsNotEmpty()
  @ApiProperty()
  position: number;

  @IsInt()
  @IsNotEmpty()
  @ApiProperty()
  level: number;

  @IsDateString()
  @IsNotEmpty()
  @ApiProperty()
  startDate: Date;

  @IsEmail()
  @IsNotEmpty()
  @ApiProperty()
  email: string;

  @IsNumber()
  @IsNotEmpty()
  @ApiProperty()
  hourRate: number;
}

export type EmployeeDto = {
  id: string;
  firstName: string;
  lastName: string;
  birthday: string;
  place: string;
  phoneNumber: string;
  position: number;
  level: number;
  startDate: string;
  email: string;
  hourRate: number;
};

import { PartialType } from '@nestjs/mapped-types';
import { CreateEmployeeDto } from './create-employee.dto';

export class UpdateEmployeeDto extends PartialType(CreateEmployeeDto) {}

import { Injectable } from '@nestjs/common';
import { EmployeeDto } from '../dto/employee.dto';
import { Employee as EmployeeEntity } from '../entities/employee.entity';
import { CreateEmployeeDto } from '../dto/create-employee.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository, UpdateResult } from 'typeorm';
import { UpdateEmployeeDto } from '../dto/update-employee.dto';
```

```
@Injectable()
export class EmployeeRepository {
 constructor(
  @InjectRepository(EmployeeEntity)
  private readonly employeeRepository: Repository<EmployeeEntity>,
 ) {}

 async createEmployee(employee: CreateEmployeeDto): Promise<EmployeeEntity> {
  const newEmployee = this.employeeRepository.create(employee);

  return this.employeeRepository.save<EmployeeEntity>(newEmployee);
 }

 async updateEmployee(updateData: UpdateEmployeeDto): Promise<EmployeeEntity> {
  return await this.employeeRepository.save(updateData);
 }

 async findAll() {
  return this.employeeRepository.find({
   withDeleted: true,
  });
 }

 async findOne(id: string) {
  return this.employeeRepository.findOne({
   where: {
    id,
   },
   withDeleted: true,
  });
 }

 async findEmployeeByPosition(position: number) {
  return this.employeeRepository.find({
   where: {
    position: position,
   },
   withDeleted: true,
  });
 }

 softDelete(id: string): Promise<UpdateResult> {
  return this.employeeRepository
   .createQueryBuilder('employees')
   .softDelete()
   .where('id = :id', { id })
   .execute();
 }

 restoreSoftDelete(id: string): Promise<UpdateResult> {
  return this.employeeRepository
   .createQueryBuilder('employees')
   .where('id = :id', { id })
   .restore()
   .execute();
 }
}

import {
 Controller,
 Get,
 Post,
 Body,
 Patch,
 Param,
```

```
  Delete,
  Put,
} from '@nestjs/common';
import { EmployeesService } from './employees.service';
import { CreateEmployeeDto } from './dto/create-employee.dto';
import { UpdateEmployeeDto } from './dto/update-employee.dto';
import { ApiOperation, ApiResponse, ApiTags } from '@nestjs/swagger';

@Controller('employees')
export class EmployeesController {
  constructor(private readonly employeesService: EmployeesService) {}

  @Post()
  @ApiTags('Employees')
  @ApiOperation({ summary: 'Create a new employee' })
  @ApiResponse({
    status: 201,
    description: 'The created employee',
  })
  @ApiResponse({ status: 400, description: 'Bad request' })
  @ApiResponse({ status: 401, description: 'Unauthorized' })
  @ApiResponse({ status: 500, description: 'Internal server error' })
  create(@Body() createEmployeeDto: CreateEmployeeDto) {
    return this.employeesService.create(createEmployeeDto);
  }

  @Get()
  @ApiTags('Employees')
  findAll() {
    return this.employeesService.findAll();
  }

  @Get(':id')
  @ApiTags('Employees')
  findOne(@Param('id') id: string) {
    return this.employeesService.findOne(id);
  }

  @Patch(':id')
  @ApiTags('Employees')
  update(
    @Param('id') id: string,
    @Body() updateEmployeeDto: UpdateEmployeeDto,
  ) {
    return this.employeesService.update(id, updateEmployeeDto);
  }

  @Delete(':id')
  @ApiTags('Employees')
  remove(@Param('id') id: string) {
    return this.employeesService.remove(id);
  }

  @Put(':id/restore')
  @ApiTags('Employees')
  restoreEmployee(@Param('id') id: string) {
    return this.employeesService.restoreEmployee(id);
  }
}

import { Module } from '@nestjs/common';
import { EmployeesService } from './employees.service';
import { EmployeesController } from './employees.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Employee } from './entities/employee.entity';
import { EmployeeRepository } from './repository/employee.repository';
```

```
import { Skill } from './entities/skills.entity';
import { Department } from './entities/department.entity';

@Module({
 imports: [TypeOrmModule.forFeature([Employee, Skill, Department])],
 controllers: [EmployeesController],
 providers: [EmployeesService, EmployeeRepository],
})
export class EmployeesModule {}

import {
 Injectable,
 NotFoundException,
 InternalServerErrorException,
} from '@nestjs/common';
import { CreateEmployeeDto } from './dto/create-employee.dto';
import { UpdateEmployeeDto } from './dto/update-employee.dto';
import { Employee as EmployeeEntity } from './entities/employee.entity';
import { EmployeeRepository } from './repository/employee.repository';

@Injectable()
export class EmployeesService {
 constructor(private readonly employeeRepository: EmployeeRepository) {}

 async create(createEmployeeDto: CreateEmployeeDto) {
  const createdEmployee: EmployeeEntity =
   await this.employeeRepository.createEmployee(createEmployeeDto);

  return createdEmployee;
 }

 findAll() {
  return this.employeeRepository.findAll();
 }

 findOne(id: string) {
  return this.employeeRepository.findOne(id);
 }

 async update(id: string, updateEmployeeDto: UpdateEmployeeDto) {
  const employeeToUpdate: EmployeeEntity =
   await this.employeeRepository.findOne(id);

  if (!employeeToUpdate) {
   throw new NotFoundException(
    'Not found. The employee with the provided ID does not exist.',
   );
  }

  const updateData = {
   ...employeeToUpdate,
   ...updateEmployeeDto,
  };

  try {
   await this.employeeRepository.updateEmployee(updateData);

   return await this.employeeRepository.findOne(id);
  } catch {
   throw new InternalServerErrorException();
  }
 }

 async remove(id: string) {
  const story = await this.employeeRepository.findOne(id);
```

```
    if (!story) {
      throw new NotFoundException(
        `Not found. The employee with the provided ID does not exist`,
      );
    }
    return await this.employeeRepository.softDelete(id);
  }

  async restoreEmployee(id: string) {
    const story = await this.employeeRepository.findOne(id);

    if (!story) {
      throw new NotFoundException(
        `Not found. The employee with the provided ID does not exist`,
      );
    }
    return await this.employeeRepository.restoreSoftDelete(id);
  }
}

import { Employee } from 'src/modules/employees/entities/employee.entity';
import { EmployeeType } from '../../../modules/employees/models/employee.model';
import {
  Column,
  CreateDateColumn,
  Entity,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
  DeleteDateColumn,
  Index,
  ManyToMany,
  JoinTable,
} from 'typeorm';

@Entity()
export class Project {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Index()
  @Column()
  name: string;

  @Column({ nullable: true })
  description: string;

  @Column({
    type: 'date',
    name: 'start_date',
    nullable: true,
  })
  startDate: string;

  @Column({
    type: 'date',
    name: 'end_date',
    nullable: true,
  })
  endDate: string;

  @Column({ nullable: true, default: 'USD' })
  currency: string;

  @Column({ nullable: true, name: 'owner_name' })
  ownerName: string;
```

```
  @Column({ nullable: true })
  budget: number;

  @ManyToMany(() => Employee, (employee) => employee.projects)
  @JoinTable({ name: 'employee_project' })
  employees?: EmployeeType[];

  @CreateDateColumn({
    type: Date,
    default: () => 'CURRENT_TIMESTAMP',
    name: 'created_at',
  })
  createdAt: Date;

  @UpdateDateColumn({
    type: Date,
    default: () => 'CURRENT_TIMESTAMP',
    name: 'updated_at',
  })
  updatedAt: Date;

  @DeleteDateColumn({
    type: Date,
    name: 'deleted_at',
  })
  deletedAt: Date;
}

import { ApiProperty } from '@nestjs/swagger';
import {
  IsString,
  IsNumber,
  IsNotEmpty,
  IsOptional,
  IsDateString,
} from 'class-validator';

export class CreateProjectDto {
  @IsString()
  @IsNotEmpty()
  @ApiProperty()
  name: string;

  @IsString()
  @IsOptional()
  @ApiProperty()
  description?: string;

  @IsDateString()
  @IsOptional()
  @ApiProperty()
  startDate: string;

  @IsString()
  @IsOptional()
  @ApiProperty()
  endDate: string;

  @IsString()
  @IsOptional()
  @ApiProperty()
  currency: string;

  @IsString()
  @IsOptional()
  @ApiProperty()
```

```
  ownerName: string;

  @IsNumber()
  @IsOptional()
  @ApiProperty()
  budget?: number;
}

import { PartialType } from '@nestjs/mapped-types';
import { CreateProjectDto } from './create-project.dto';

export class UpdateProjectDto extends PartialType(CreateProjectDto) {}

import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository, UpdateResult } from 'typeorm';
import { Project as ProjectEntity } from '../entities/project.entity';
import { UpdateProjectDto } from '../dto/update-project.dto';
import { CreateProjectDto } from '../dto/create-project.dto';

@Injectable()
export class ProjectRepository {
  constructor(
    @InjectRepository(ProjectEntity)
    private readonly projectRepository: Repository<ProjectEntity>,
  ) {}

  async createProject(project: CreateProjectDto): Promise<ProjectEntity> {
    const newProject = this.projectRepository.create(project);

    return await this.projectRepository.save<ProjectEntity>(newProject);
  }

  async updateProject(updateData: UpdateProjectDto): Promise<ProjectEntity> {
    return await this.projectRepository.save(updateData);
  }

  async findAll() {
    return this.projectRepository.find({
      withDeleted: true,
    });
  }

  async findOne(id: string) {
    return this.projectRepository.findOne({
      where: {
        id,
      },
      withDeleted: true,
    });
  }

  async findProjectByName(name: string) {
    return this.projectRepository.find({
      where: {
        name: name,
      },
      withDeleted: true,
    });
  }

  softDelete(id: string): Promise<UpdateResult> {
    return this.projectRepository
      .createQueryBuilder('projects')
      .softDelete()
      .where('id = :id', { id })
```

```
    .execute();
  }

  restoreSoftDelete(id: string): Promise<UpdateResult> {
    return this.projectRepository
      .createQueryBuilder('projects')
      .where('id = :id', { id })
      .restore()
      .execute();
  }
}

import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  Put,
} from '@nestjs/common';
import { ProjectsService } from './projects.service';
import { CreateProjectDto } from './dto/create-project.dto';
import { UpdateProjectDto } from './dto/update-project.dto';
import { ApiOperation, ApiResponse, ApiTags } from '@nestjs/swagger';

@Controller('projects')
export class ProjectsController {
  constructor(private readonly projectsService: ProjectsService) {}

  @Post()
  @ApiTags('Projects')
  @ApiOperation({ summary: 'Create a new project' })
  @ApiResponse({
    status: 201,
    description: 'The created project',
  })
  @ApiResponse({ status: 400, description: 'Bad request' })
  @ApiResponse({ status: 401, description: 'Unauthorized' })
  @ApiResponse({ status: 500, description: 'Internal server error' })
  create(@Body() createProjectDto: CreateProjectDto) {
    return this.projectsService.create(createProjectDto);
  }

  @Get()
  @ApiTags('Projects')
  findAll() {
    return this.projectsService.findAll();
  }

  @Get(':id')
  @ApiTags('Projects')
  findOne(@Param('id') id: string) {
    return this.projectsService.findOne(id);
  }

  @Patch(':id')
  @ApiTags('Projects')
  update(@Param('id') id: string, @Body() updateProjectDto: UpdateProjectDto) {
    return this.projectsService.update(id, updateProjectDto);
  }

  @Delete(':id')
  @ApiTags('Projects')
  remove(@Param('id') id: string) {
```

```
    return this.projectsService.remove(id);
  }

  @Put(':id/restore')
  @ApiTags('Projects')
  restoreProject(@Param('id') id: string) {
    return this.projectsService.restoreProject(id);
  }
}

import { Module } from '@nestjs/common';
import { ProjectsService } from './projects.service';
import { ProjectsController } from './projects.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Project } from './entities/project.entity';
import { ProjectRepository } from './repository/project.repository';

@Module({
  imports: [TypeOrmModule.forFeature([Project])],
  controllers: [ProjectsController],
  providers: [ProjectsService, ProjectRepository],
})
export class ProjectsModule {}

import {
  Injectable,
  NotFoundException,
  InternalServerErrorException,
} from '@nestjs/common';
import { CreateProjectDto } from './dto/create-project.dto';
import { UpdateProjectDto } from './dto/update-project.dto';
import { ProjectRepository } from './repository/project.repository';
import { Project as ProjectEntity } from './entities/project.entity';

@Injectable()
export class ProjectsService {
  constructor(private readonly projectRepository: ProjectRepository) {}

  async create(createProjectDto: CreateProjectDto) {
    const createdProject: ProjectEntity =
      await this.projectRepository.createProject(createProjectDto);

    return createdProject;
  }

  findAll() {
    return this.projectRepository.findAll();
  }

  findOne(id: string) {
    return this.projectRepository.findOne(id);
  }

  async update(id: string, updateProjectDto: UpdateProjectDto) {
    const projectToUpdate: ProjectEntity =
      await this.projectRepository.findOne(id);

    if (!projectToUpdate) {
      throw new NotFoundException(
        'Not found. The project with the provided ID does not exist.',
      );
    }

    const updateData = {
      ...projectToUpdate,
      ...updateProjectDto,
```

```
    };

    try {
      await this.projectRepository.updateProject(updateData);

      return await this.projectRepository.findOne(id);
    } catch {
      throw new InternalServerErrorException();
    }
  }

  async remove(id: string) {
    const story = await this.projectRepository.findOne(id);

    if (!story) {
      throw new NotFoundException(
        `Not found. The project with the provided ID does not exist`,
      );
    }
    return await this.projectRepository.softDelete(id);
  }

  async restoreProject(id: string) {
    const story = await this.projectRepository.findOne(id);

    if (!story) {
      throw new NotFoundException(
        `Not found. The project with the provided ID does not exist`,
      );
    }
    return await this.projectRepository.restoreSoftDelete(id);
  }
}
```

Додаток Г

Лістинг коду клієнтської частини

```
import React from 'react';
import { Outlet } from 'react-router-dom';
import { ThemeProvider } from '@mui/material/styles';
import { Theme } from './theme';
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
import { AdapterDateFns } from '@mui/x-date-pickers/AdapterDateFns';
import { ToastProvider } from './components/toaster/toasterProvider';

const App: React.FC = () => {
 return (
  <div className='App'>
   <LocalizationProvider dateAdapter={AdapterDateFns}>
    <ThemeProvider theme={Theme}>
     <ToastProvider delay={3000}>
      <Outlet />
     </ToastProvider>
    </ThemeProvider>
   </LocalizationProvider>
  </div>
 );
};

export { App };



import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';

const firebaseConfig = {
 apiKey: 'AIzaSyBIjKoCCF2D3eOHiXf_AZz8wDy5K7VKABk',
 authDomain: 'optimize-project-4a2b7.firebaseapp.com',
 projectId: 'optimize-project-4a2b7',
 storageBucket: 'optimize-project-4a2b7.appspot.com',
 messagingSenderId: '426653136909',
 appId: '1:426654336909:web:dc870f7a17215c654bbd44',
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth(app);

export const employeeService = {
 findAll: async () => {
  const requestOptions = {
   method: 'GET',
   headers: { 'Content-Type': 'application/json' },
  };
  try {
   const result = await fetch(`http://localhost:3001/employees`, requestOptions);

   if (!result.ok) {
    throw new Error(`HTTP error! Status: ${result.status}`);
   }

   return (await result.json()) as any[];
  } catch (error) {
   console.error('An error occurred:', error);
   return [];
  }
 },
```

```
  findByName: async (name: string) => {
    try {
      const result = await fetch(
        'http://localhost:3001/employees?' +
          new URLSearchParams({
            name: name,
          }),
      );
      return (await result.json()) as any[];
    } catch (error) {
      return [];
    }
  },
  getById: async (id: string): Promise<any> => {
    const requestOptions = {
      method: 'GET',
      headers: { 'Content-Type': 'application/json' },
    };
    const result = await fetch(`http://localhost:3001/employees/${id}`, requestOptions);
    return await result.json();
  },
  updateEmployeeDetails: async (id: string, updateData: any) => {
    const requestOptions = {
      method: 'PATCH',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(updateData),
    };
    return await fetch(`http://localhost:3001/employees/${id}`, requestOptions);
  },
};

export const projectsService = {
  findAll: async () => {
    try {
      const result = await fetch(`http://localhost:3001/projects`);

      if (!result.ok) {
        throw new Error(`HTTP error! Status: ${result.status}`);
      }

      return (await result.json()) as any[];
    } catch (error) {
      console.error('An error occurred:', error);
      return [];
    }
  },
  findByName: async (name: string) => {
    try {
      const result = await fetch(
        'http://localhost:3001/projects?' +
          new URLSearchParams({
            name: name,
          }),
      );
      return (await result.json()) as any[];
    } catch (error) {
      return [];
    }
  },
  getById: async (id: string): Promise<any> => {
    const result = await fetch(`http://localhost:3001/projects/${id}`);
    return (await result.json()) as any;
  },
  updateProjectDetails: async (id: string, updateData: any) => {
    const requestOptions = {
      method: 'PUT',
```

```
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(updateData),
   };
   return await fetch(`http://localhost:3001/projects/${id}`, requestOptions);
  },
};

import { ReactElement } from 'react';
import PeopleIcon from '@mui/icons-material/People';
import BusinessCenterIcon from '@mui/icons-material/BusinessCenter';

export interface INavLink {
  name?: string;
  title?: string;
  link: string;
  icon: ReactElement;
  tooltipText?: string;
}

export enum NavTypesEnum {
  MAIN = 'main',
}

const MainNavigationList: INavLink[] = [
  { name: 'Співробітники', title: 'Співробітники', link: '/employees', icon: <PeopleIcon /> },
  { name: 'Проекти', title: 'Проекти', link: '/projects', icon: <BusinessCenterIcon /> },
];

export const getNavigationList = (type: NavTypesEnum): INavLink[] => {
  switch (type) {
    case NavTypesEnum.MAIN:
      return MainNavigationList;
    default:
      return MainNavigationList;
  }
};



import * as React from 'react';
import Box from '@mui/material/Box';
import Stack from '@mui/material/Stack';
import Stepper from '@mui/material/Stepper';
import Step from '@mui/material/Step';
import StepLabel from '@mui/material/StepLabel';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';

import { ProfileDataForm } from '../Projects/CreateProjectPage';
import { ProposalPage } from '../Projects/ProposalPage';
import { GanttTasksPage } from '../Projects/GanttTasksPage';
import { ColorlibConnector, ColorlibStepIcon } from './CustomSteps';

export default function CreationStepper() {
  const [activeStep, setActiveStep] = React.useState(0);

  const steps = [
    { label: 'Основна інформація', element: <ProfileDataForm /> },
    { label: 'Створення задач', element: <GanttTasksPage /> },
    { label: 'Пропозиція', element: <ProposalPage /> },
  ];

  const handleNext = () => {
    setActiveStep((prevActiveStep) => prevActiveStep + 1);
  };
```

```
 const handleBack = () => {
  setActiveStep((prevActiveStep) => prevActiveStep - 1);
 };

 const handleReset = () => {
  setActiveStep(0);
 };

 return (
  <Stack sx={{ width: '100%' }} spacing={4}>
   <Stepper sx={{ mb: '30px' }} activeStep={activeStep} alternativeLabel connector={<ColorlibConnector />}>
    {steps.map((item, index) => {
     const stepProps: { completed?: boolean } = {};
     const labelProps: {
      optional?: React.ReactNode;
     } = {};
     return (
      <Step key={item.label} {...stepProps}>
       <StepLabel StepIconComponent={ColorlibStepIcon} {...labelProps}>
        {item.label}
       </StepLabel>
      </Step>
     );
    })}
   </Stepper>
   {activeStep === steps.length ? (
    <React.Fragment>
     <Typography sx={{ mt: 2, mb: 1 }}>Усі кроки завершено - ви завершили</Typography>
     <Box sx={{ display: 'flex', flexDirection: 'row', pt: 2 }}>
      <Box sx={{ flex: '1 1 auto' }} />
      <Button onClick={handleReset}>Скинути</Button>
     </Box>
    </React.Fragment>
   ) : (
    <React.Fragment>
     {steps[activeStep].element}
     <Box sx={{ display: 'flex', flexDirection: 'row', pt: 2 }}>
      <Button color='inherit' disabled={activeStep === 0} onClick={handleBack} sx={{ mr: 1 }}>
       Назад
      </Button>
      <Box sx={{ flex: '1 1 auto' }} />
      <Button onClick={handleNext}>{activeStep === steps.length - 1 ? 'Завершити' : 'Далі'}</Button>
     </Box>
    </React.Fragment>
   )}
  </Stack>
 );
}

import styled from '@emotion/styled';
import StepConnector, { stepConnectorClasses } from '@mui/material/StepConnector';
import { StepIconProps } from '@mui/material/StepIcon';
import SettingsIcon from '@mui/icons-material/Settings';
import GroupAddIcon from '@mui/icons-material/GroupAdd';
import NoteAddIcon from '@mui/icons-material/NoteAdd';

export const ColorlibConnector = styled(StepConnector)(() => ({
 [`&.${stepConnectorClasses.alternativeLabel}`]: {
  top: 22,
 },
 [`&.${stepConnectorClasses.active}`]: {
  [`& .${stepConnectorClasses.line}`]: {
   background: '#009490',
  },
 },
 [`&.${stepConnectorClasses.completed}`]: {
```

```
    [`& .${stepConnectorClasses.line}`]: {
      background: '#009490',
    },
  },
  [`& .${stepConnectorClasses.line}`]: {
    height: 3,
    border: 0,
    backgroundColor: '#d3d3d8',
    borderRadius: 1,
  },
}));

const ColorlibStepIconRoot = styled('div')<{
  ownerState: { completed?: boolean; active?: boolean };
}>(({ ownerState }) => ({
  backgroundColor: '#ccc',
  zIndex: 1,
  color: '#fff',
  width: 50,
  height: 50,
  display: 'flex',
  borderRadius: '50%',
  justifyContent: 'center',
  alignItems: 'center',
  ...(ownerState.active && {
    background: '#009490',
    boxShadow: '0 4px 10px 0 rgba(0,0,0,.25)',
  }),
  ...(ownerState.completed && {
    background: '#009490',
  }),
}));

export function ColorlibStepIcon(props: StepIconProps) {
  const { active, completed, className } = props;

  const icons: { [index: string]: React.ReactElement } = {
    1: <SettingsIcon />,
    2: <NoteAddIcon />,
    3: <GroupAddIcon />,
  };

  return (
    <ColorlibStepIconRoot ownerState={{ completed, active }} className={className}>
      {icons[String(props.icon)]}
    </ColorlibStepIconRoot>
  );
}

import { useNavigate, useRouteLoaderData } from 'react-router-dom';
import CustomDataGrid from '../../components/CustomDataGrid/CustomDataGrid';
import { GridActionsCellItem, GridRowParams } from '@mui/x-data-grid';
import VisibilityIcon from '@mui/icons-material/Visibility';
import ModeEditIcon from '@mui/icons-material/ModeEdit';
import { Levels } from '../../types/Level';
import { Employee } from '../../types/Employee';
import { Position } from '../../types/Position';

export default function EmployeesPage() {
  const navigate = useNavigate();
  const employees: Employee[] = useRouteLoaderData('employees') as Employee[];

  const columns = [
    {
      field: 'id',
      headerName: '№',
```

```
    type: 'string',
   },
   {
    field: 'firstName',
    headerName: 'Ім`я',
   },
   {
    field: 'lastName',
    headerName: 'Прізвище',
   },
   {
    field: 'hourRate',
    headerName: 'Ставка, USD',
    type: 'number',
    width: 140,
   },
   {
    field: 'sellHourRate',
    headerName: 'Ставка продажу, USD',
    type: 'number',
    width: 180,
   },
   {
    field: 'position',
    headerName: 'Позиція',
    type: 'string',
    width: 180,
    valueGetter: (params: any) => `${Levels[params.row.level]} ${Position[params.row.position]}`,
   },
   {
    field: 'email',
    headerName: 'E-Mail',
    type: 'string',
    width: 200,
   },
   {
    field: 'actions',
    headerName: 'Дії',
    type: 'actions',
    getActions: (params: GridRowParams) => [
     <GridActionsCellItem
      icon={<ModeEditIcon htmlColor='dimgrey' />}
      onClick={(e) => navigate(`/employees/${params.row.id}`)}
      label='Редагувати профіль користувача'
     />,
     <GridActionsCellItem
      icon={<VisibilityIcon htmlColor='dimgrey' />}
      onClick={(e) => navigate(`/employees/${params.row.id}`)}
      label='Переглянути профіль користувача'
     />,
    ],
   },
  ];

  return <CustomDataGrid rows={employees || []} columns={columns} />;
}

import LoginForm from '../../components/LoginForm/LoginForm';

export default function LoginPage() {
  return <LoginForm />;
}

import { Avatar, Button, Grid } from '@mui/material';
import React, { useEffect } from 'react';
```

```
import List from '@mui/material/List';
import ListItem from '@mui/material/ListItem';
import ListItemText from '@mui/material/ListItemText';
import Paper from '@mui/material/Paper';
import Stack from '@mui/material/Stack';
import { useForm } from 'react-hook-form';
import ControlledTextField from '../../components/Elements/ControlledTextfield/ControlledTextfield';
import { employeeService } from '../../services/EmployeeService';
import { Employee } from '../../types/Employee';
import { useToast } from '../../components/toaster/useToaster';
import { useNavigate, useRouteLoaderData } from 'react-router-dom';
import ControlledSelect from '../../components/Elements/ControlledSelect/ControlledSelect';
import { levelsList } from '../../types/Level';
import { positionList } from '../../types/Position';

const listStyle = {
  padding: 0,
  width: '100%',
  maxWidth: 430,
  bgcolor: 'background.paper',
  marginBottom: '30px',
};

const paperStyle = {
  marginBottom: '8px',
  borderRadius: '6px',
  overflow: 'hidden',
  padding: '20px 115px 10px 80px',
};

export default function ProfilePage() {
  const employee: Employee = useRouteLoaderData('profile') as Employee;
  const toast = useToast();
  const navigate = useNavigate();
  const [values, setValues] = React.useState<Employee>();

  const { control, getValues, handleSubmit } = useForm<Employee>({
    defaultValues: {
      firstName: '',
      lastName: '',
      position: undefined,
      level: 1,
      email: '',
      hourRate: 0,
    },
    values,
  });

  useEffect(() => {
    setValues(employee);
  }, []);

  const saveProfile = async () => {
    const formData = getValues();
    if (values) {
      const { id, createdAt, updatedAt, deletedAt, ...dataToUpdate } = formData;
      await employeeService.updateEmployeeDetails(id, dataToUpdate);
      toast.open(`Updated`, { type: 'success' });
    }
  };

  const handleCancel = () => {
    navigate('/employees/list', { replace: true });
  };

  return (
```

```
  <>
   <form onSubmit={handleSubmit(saveProfile)}>
    <Grid container alignItems={'center'} sx={{ marginBottom: '20px' }}>
     <Grid item paddingLeft={'46px'} xs={8}></Grid>
     <Grid item xs={4} gap={5}>
      <Stack spacing={'30.5px'} direction='row' alignItems='center' justifyContent='flex-end'>
       <Button variant='outlined' onClick={handleCancel}>
        Назад
       </Button>
       <Button variant='contained' type='submit' disableElevation>
        Зберегти
       </Button>
      </Stack>
     </Grid>
    </Grid>
    <Paper elevation={0} sx={paperStyle}>
     <Grid container>
      <Grid item xs={6}>
       <List sx={listStyle} aria-label=''>
        <ListItem divider>
         <ListItemText primary='Ім`я' />
         <ControlledTextField name={`firstName`} control={control} />
        </ListItem>
        <ListItem divider>
         <ListItemText primary='Прізвище' />
         <ControlledTextField name={`lastName`} control={control} />
        </ListItem>
        <ListItem divider>
         <ListItemText primary='Email' />
         <ControlledTextField name={`email`} control={control} />
        </ListItem>
       </List>
       <List sx={listStyle} aria-label=''>
        <ListItem divider>
         <ListItemText primary='Рівень' />
         <ControlledSelect name={`level`} control={control} emptyValueText='' listData={levelsList} />
        </ListItem>
        <ListItem divider>
         <ListItemText primary='Позиція' />
         <ControlledSelect name={`position`} control={control} emptyValueText='' listData={positionList} />
        </ListItem>
        <ListItem divider>
         <ListItemText primary='Погодинна ставка' />
         <ControlledTextField name={`hourRate`} control={control} />
        </ListItem>
        <ListItem divider>
         <ListItemText primary='Погодинна ставка продажу' />
         <ControlledTextField name={`sellHourRate`} control={control} />
        </ListItem>
       </List>
      </Grid>
      <Grid item>
       <Avatar
        variant='square'
        sx={{ width: '154px', height: '182px', backgroundColor: 'rgba(195, 215, 229, 0.20)' }}
        alt='User Avatar'
       />
      </Grid>
     </Grid>
    </Paper>
   </form>
  </>
 );
}

import React, { useState } from 'react';
```

```
import { Gantt, Task } from 'gantt-task-react';
import { Controller, useFieldArray, useFormContext } from 'react-hook-form';
import List from '@mui/material/List';
import ListItem from '@mui/material/ListItem';
import ListItemText from '@mui/material/ListItemText';
import FormControl from '@mui/material/FormControl';
import Button from '@mui/material/Button';
import Paper from '@mui/material/Paper';
import IconButton from '@mui/material/IconButton';
import DeleteIcon from '@mui/icons-material/Delete';
import ControlledTextField from '../../components/Elements/ControlledTextfield/ControlledTextfield';
import { DatePicker } from '@mui/x-date-pickers';
import { parseISO } from 'date-fns';
import ControlledSelect from '../../components/Elements/ControlledSelect/ControlledSelect';
import { Position } from '../../types/Position';

const listStyle = {
  padding: 0,
  width: '100%',
  maxWidth: '85%',
  bgcolor: 'background.paper',
  marginBottom: '30px',
};

export function GanttTasksPage() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const { control } = useFormContext();

  const newEmptyRow = {
    startDate: '',
    endDate: '',
    name: '',
    id: '',
    progress: 100,
    taskType: '',
    type: 'task',
    displayOrder: null,
  };

  const { fields, append, remove } = useFieldArray({
    control: control,
    name: 'tasks',
  });

  const updateTasks = () => {
    const mappedTasks = control._formValues.tasks.map((item: Task, index: number) => {
      return {
        ...item,
        id: index,
        displayOrder: index + 1,
      };
    });
    setTasks(mappedTasks as Task[]);
  };

  const formatTasks = () => {
    return tasks.map((task: any) => {
      const taskIndex = taskTypes.findIndex((taskType: any) => Position[task.taskType] === task.taskType);
      return {
        ...task,
        start: new Date(task.startDate),
        end: new Date(task.endDate),
        name: `${taskTypes[taskIndex].name} - ${task.name}`,
      };
    });
  };
```

```
const disableWeekends = (date: any) => {
  return date.getDay() === 0 || date.getDay() === 6;
};

return (
  <>
    <Paper elevation={0}>
      <List>
        <ListItem sx={{ alignItems: 'baseline', paddingRight: '0' }} divider>
          <ListItemText sx={{ width: '100%', maxWidth: 150 }} primary='Task' />
          <List sx={listStyle}>
            {fields.length ? (
              fields.map((value: any, index: number) => {
                return (
                  <React.Fragment key={index}>
                    <ListItem
                      disablePadding
                      divider
                      secondaryAction={
                        <IconButton edge='end' aria-label='remove' onClick={() => remove(index)}>
                          <DeleteIcon />
                        </IconButton>
                      }
                    >
                      <ListItemText sx={{ maxWidth: '28px !important' }} primary={`${index + 1}.`} />
                      <FormControl sx={{ m: 1, minWidth: 230, maxWidth: 230 }} size='small'>
                        <ControlledSelect
                          name={`tasks.${index}.taskType`}
                          control={control}
                          emptyValueText={'Тип задачі'}
                          listData={taskTypes}
                        />
                      </FormControl>
                      <FormControl sx={{ m: 1, minWidth: 230, maxWidth: 230 }} size='small'>
                        <ControlledTextField name={`tasks.${index}.name`} control={control} placeholder={'Назва'} />
                      </FormControl>
                      <FormControl sx={{ m: 1, minWidth: 230, maxWidth: 230 }} size='small'>
                        <Controller
                          control={control}
                          name={`tasks.${index}.startDate`}
                          render={({ field }) => {
                            const parsedDate = parseISO(field.value) ?? '';
                            return (
                              <DatePicker
                                {...field}
                                value={parsedDate}
                                minDate={control._formValues.project.startDate}
                                maxDate={control._formValues.project.endDate}
                                disableHighlightToday={true}
                                shouldDisableDate={disableWeekends}
                              />
                            );
                          }}
                        />
                      </FormControl>
                      <FormControl sx={{ m: 1, minWidth: 230, maxWidth: 230 }} size='small'>
                        <Controller
                          control={control}
                          name={`tasks.${index}.endDate`}
                          render={({ field }) => {
                            const parsedDate = parseISO(field.value) ?? '';
                            return (
                              <DatePicker
                                {...field}
                                value={parsedDate}
```

```
                        minDate={control._formValues.project.startDate}
                        maxDate={control._formValues.project.endDate}
                        shouldDisableDate={disableWeekends}
                      />
                    );
                  }}
                />
              </FormControl>
            </ListItem>
            {(!fields.length || index + 1 === fields.length) && fields.length < 10 && (
              <>
                <Button onClick={() => append(newEmptyRow)}>Додати задачу</Button>
                <Button onClick={() => updateTasks()}>Генерувати діаграму Ганта</Button>
              </>
            )}
          </React.Fragment>
        );
      })
    ) : (
      <Button onClick={() => append(newEmptyRow)}>Додати задачу</Button>
    )}
  </List>
 </ListItem>
</List>
</Paper>
{tasks.length ? <Gantt tasks={formatTasks()} /> : null}
</>
);
}

import { FormControl, Grid } from '@mui/material';

import List from '@mui/material/List';
import ListItem from '@mui/material/ListItem';
import ListItemText from '@mui/material/ListItemText';
import Paper from '@mui/material/Paper';
import { Controller, FormProvider, useForm, useFormContext } from 'react-hook-form';
import ControlledTextField from '../../components/Elements/ControlledTextfield/ControlledTextfield';
import { useToast } from '../../components/toaster/useToaster';
import { useNavigate } from 'react-router-dom';
import 'gantt-task-react/dist/index.css';
import CreationStepper from '../CreationStepper/CreationStepper';
import { DatePicker } from '@mui/x-date-pickers';
import { parseISO } from 'date-fns';

const listStyle = {
 padding: 0,
 width: '100%',
 maxWidth: 430,
 bgcolor: 'background.paper',
 marginBottom: '30px',
};

const paperStyle = {
 marginBottom: '8px',
 borderRadius: '6px',
 overflow: 'hidden',
 padding: '20px 115px 10px 80px',
};

export function ProfileDataForm() {
 const { control } = useFormContext();

 const disableWeekends = (date: any) => {
  return date.getDay() === 0 || date.getDay() === 6;
 };
```

```
return (
 <Paper elevation={0} sx={paperStyle}>
  <Grid container>
   <Grid item xs={6}>
    <List sx={listStyle} aria-label=''>
     <ListItem divider>
      <ListItemText primary='Назва' />
      <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
       <ControlledTextField name={`project.name`} control={control} placeholder={''} />
      </FormControl>
     </ListItem>
     <ListItem divider>
      <ListItemText primary='Короткий опис' />
      <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
       <ControlledTextField name={`project.description`} control={control} placeholder={''} />
      </FormControl>
     </ListItem>
     <ListItem divider>
      <ListItemText primary='Бюджет' />
      <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
       <ControlledTextField name={`project.budget`} control={control} placeholder={''} />
      </FormControl>
     </ListItem>
    </List>
    <List sx={listStyle} aria-label=''>
     <ListItem divider>
      <ListItemText primary='Дата початку' />
      <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
       <Controller
        control={control}
        name={`project.startDate`}
        render={({ field }) => {
         const parsedDate = parseISO(field.value) ?? null;
         return (
          <DatePicker
           {...field}
           value={parsedDate}
           disableHighlightToday
           shouldDisableDate={disableWeekends}
          />
         );
        }}
       />
      </FormControl>
     </ListItem>
     <ListItem divider>
      <ListItemText primary='Дата закінчення' />
      <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
       <Controller
        control={control}
        name={`project.endDate`}
        render={({ field }) => {
         const parsedDate = parseISO(field.value) ?? null;
         return (
          <DatePicker
           {...field}
           value={parsedDate}
           disableHighlightToday
           shouldDisableDate={disableWeekends}
          />
         );
        }}
       />
      </FormControl>
     </ListItem>
```

```
          <ListItem divider>
            <ListItemText primary='Власник' />
            <FormControl sx={{ minWidth: 230, maxWidth: 230 }} size='small'>
              <ControlledTextField name={`project.ownerName`} control={control} placeholder={''} />
            </FormControl>
          </ListItem>
        </List>
        <List sx={listStyle} aria-label=''></List>
      </Grid>
    </Grid>
  </Paper>
 );
}

export default function CreateProjectPage() {
 const toast = useToast();
 const navigate = useNavigate();

 const methods = useForm<any>({
  defaultValues: {
   project: {
    name: 'Новий проект',
    budget: 40000,
    ownerName: 'Харві Джин',
    currency: 'USD',
    description: 'Буде щось дуже захоплююче і технічно сучасне',
    startDate: '',
    endDate: '',
   },
   tasks: [
    {
     startDate: '',
     endDate: '',
     name: null,
     taskType: '',
     id: '1',
     progress: 100,
     type: 'task',
     hideChildren: false,
     displayOrder: 0,
    },
   ],
  },
 });

 const onSubmit = (data: any) => console.log(data);

 return (
  <>
   <FormProvider {...methods}>
    <form onSubmit={methods.handleSubmit(onSubmit)}>
     <CreationStepper />
    </form>
   </FormProvider>
  </>
 );
}
import { GridActionsCellItem, GridRowParams } from '@mui/x-data-grid';
import CustomDataGrid from '../../components/CustomDataGrid/CustomDataGrid';
import VisibilityIcon from '@mui/icons-material/Visibility';
import { useNavigate, useRouteLoaderData } from 'react-router-dom';

export default function ProjectsPage() {
 const navigate = useNavigate();
 const { projects } = useRouteLoaderData('projects');
 const columns = [
```

```
    {
      field: 'id',
      headerName: '№',
      type: 'string',
    },
    {
      field: 'name',
      headerName: 'Назва',
      minWidth: 220,
    },
    {
      field: 'description',
      headerName: 'Опис',
      minWidth: 260,
    },
    {
      field: 'startDate',
      headerName: 'Початок',
      type: 'Date',
    },
    {
      field: 'endDate',
      headerName: 'Кінець',
      type: 'Date',
    },
    {
      field: 'currency',
      headerName: 'Валюта',
      type: 'string',
    },
    {
      field: 'ownerName',
      headerName: 'Клієнт',
      type: 'number',
      minWidth: 160,
    },
    {
      field: 'budget',
      headerName: 'Бюджет',
      type: 'string',
    },
    {
      field: 'actions',
      headerName: 'Дії',
      type: 'actions',
      getActions: (params: GridRowParams) => [
        <GridActionsCellItem
          icon={<VisibilityIcon htmlColor='dimgrey' />}
          onClick={(e) => navigate(`/projects/${params.row.id}`)}
          label='Переглянути деталі проекту'
        />,
      ],
    },
  ];
  return <CustomDataGrid rows={projects || []} columns={columns} />;
}

import { DataGrid, GridApi, GridPaginationModel, useGridApiRef } from '@mui/x-data-grid';
import { CustomPagination } from '../Pagination/Pagination';
import React, { useEffect } from 'react';
import { CustomGridToolbar } from '../CustomGridToolbar/CustomGridToolbar';

const DataGridStyles = {
  '--DataGrid-overlayHeight': '100%',
  '& .MuiDataGrid-main': {
    borderRadius: '6px',
```

```
  },
  '& .MuiDataGrid-columnHeaderTitleContainer': {
    lineHeight: 1.5,
  },
  '.MuiDataGrid-columnHeaderTitle': {
    fontSize: '14px',
    fontStyle: 'normal',
    fontWeight: 500,
    lineHeight: '19.5px',
    textTransform: 'uppercase',
  },
  '.MuiDataGrid-cellContent': {
    fontSize: '14px',
    fontStyle: 'normal',
    fontWeight: 400,
    lineHeight: '22px',
    color: '#000',
  },
  '.MuiDataGrid-columnSeparator': { display: 'none' },
  '&.MuiDataGrid-root': { border: 'none' },
  '.MuiDataGrid-columnHeaders': {
    background: 'linear-gradient(90deg, #BDE1F4 -1.77%, #034846 99.3%, #034846 101.37%)',
    color: 'white',
  },
  '.MuiDataGrid-row:nth-of-type(even)': {
    backgroundColor: 'white',
  },
  '.MuiDataGrid-row:nth-of-type(odd)': {
    backgroundColor: 'rgba(195, 215, 229, 0.2)',
  },
  '.MuiDataGrid-columnHeader:focus': {
    outline: 'none',
  },
  '.MuiDataGrid-cell:focus': {
    outline: 'none',
  },
  '.MuiIconButton-root': {
    color: 'white',
  },
  '& .MuiDataGrid-toolbarContainer': {
    marginBottom: '15px',
    padding: '0',
    gap: '20px',
  },
  '.MuiDataGrid-footerContainer': {
    justifyContent: 'center',
    marginTop: '14px',
    border: 'none',
  },
  '.MuiDataGrid-columnHeader:focus-within': {
    outline: 'none',
  },
  '.MuiDataGrid-cell:focus-within': {
    outline: 'none',
  },
};

export default function CustomDataGrid(props: any) {
  const apiRef = useGridApiRef<GridApi>();
  const [paginationModel, setPaginationModel] = React.useState<GridPaginationModel>({ page: 0, pageSize: 10 });

  useEffect(() => {
    setPaginationModel({ page: 0, pageSize: 10 });
  }, [props.columns]);

  const handlePaginationChange = (paginationModel: { page: number; pageSize: number }) => {
```

```
    setPaginationModel(paginationModel);
  };

  return (
    <DataGrid
      apiRef={apiRef}
      columns={props.columns}
      rows={props.rows}
      pagination
      pageSizeOptions={[10]}
      paginationModel={paginationModel}
      onPaginationModelChange={handlePaginationChange}
      sx={DataGridStyles}
      slots={{
        pagination: CustomPagination,
        toolbar: CustomGridToolbar,
      }}
      slotProps={{
        toolbar: {
          showQuickFilter: true,
          selectedTabIndex: props.selectedTabIndex,
        },
      }}
    />
  );
}

import TextField from '@mui/material/TextField';
import { Controller } from 'react-hook-form';

type ControlledTextFieldProps = {
  name: string;
  placeholder?: string;
  control: any;
  multiline?: boolean;
  minRows?: number;
  type?: string;
};

export default function ControlledTextField(props: ControlledTextFieldProps) {
  return (
    <Controller
      name={props.name}
      control={props.control}
      render={({ field: { ref, name, ...field }, fieldState: { error } }) => (
        <TextField
          {...field}
          placeholder={props.placeholder ?? ''}
          type={props.type ?? 'text'}
          multiline={props.multiline ?? false}
          minRows={props.minRows ?? 1}
          error={!!error}
          fullWidth
          size='small'
          variant='outlined'
        />
      )}
    />
  );
}

import MenuItem from '@mui/material/MenuItem';
import Select from '@mui/material/Select';
import { Controller } from 'react-hook-form';

type ControlledSelectProps = {
```

```
  listData: { value: number; name: string }[];
  name: string;
  emptyValueText: string;
  control: any;
};

export default function ControlledSelect(props: ControlledSelectProps) {
  return (
    <Controller
      name={props.name}
      control={props.control}
      defaultValue=''
      render={({ field }) => (
        <Select
          {...field}
          style={{ width: '100%' }}
          displayEmpty
          renderValue={(value) =>
            value ? (
              props.listData.find((item: any) => item.value === value)?.name
            ) : (
              <span style={{ opacity: 0.5 }}>{props.emptyValueText}</span>
            )
          }
        >
          {props.listData.map((item: any, index: number) => (
            <MenuItem key={index} value={item.value}>
              {item.name}
            </MenuItem>
          ))}
        </Select>
      )}
    />
  );
}
```

Додаток Д

ІЛЮСТРАТИВНА ЧАСТИНА

«Метод і програмний засіб оптимізації ресурсів на проектах у ІТ галузі»

Вінницький національний технічний університет
Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: Метод і програмний засіб
оптимізації ресурсів на проектах у IT галузі

Виконав:
студент групи ЗПІ-22м Ковальський С.В.

Науковий керівник:
к.т.н., асистент кафедри ПЗ Тужанський С.Є.

Мета, предмет та об'єкт дослідження

- **Метою роботи** є підвищення ефективності управління проектами у IT галузі за рахунок удосконалення методів оптимізації використання ресурсів у таких проектах.
- **Об'єктом дослідження** є процес оптимізації використання ресурсів (людські ресурси, бюджет) у проектах, які здійснюються в галузі IT.
- **Предметом дослідження** є метод і програмні засоби оптимізації використання ресурсів у IT проектах.

Наукова новизна розробки та практичне значення одержаних результатів

- Дістав подальшого розвитку метод оптимізації ресурсів у IT проектах з використанням жадібних алгоритмів, автоматизованого планування та моніторингу, який на відміну від існуючих методів за відповідними критеріями (бюджет, тип задачі, перелік технологій, дати виконання) дозволяє більш точно розподіляти людські ресурси в межах таких проектів, зменшуючи витрати і ризики при їх реалізації.

- Розроблений програмний засіб може бути впроваджений у IT компанії для покращення управління людськими ресурсами, що у свою чергу оптимізує бюджетні надходження та зменшує час на опрацювання командного складу.

Актуальність роботи

Інформаційні технології (IT) в сучасному світі відіграють вирішальну роль у розвитку бізнесу та суспільства. Проекти в IT галузі здійснюються різноманітними компаніями для розробки програмного забезпечення, створення веб-сайтів, імплементації інформаційних систем тощо. Однак, управління ресурсами на таких проектах завжди є складним завданням, виконання якого вимагає ефективних методів та програмних засобів для оптимізації витрат і забезпечення успішності їх реалізації.

Методи оптимізації ресурсів

Оптимізація людських ресурсів

Оптимізація людських ресурсів у IT галузі передбачає ефективне використання навичок і знань працівників.

Оптимізація технічних ресурсів

Оптимізація технічних ресурсів передбачає ефективне використання програмного забезпечення, апаратного забезпечення та інших технічних ресурсів.

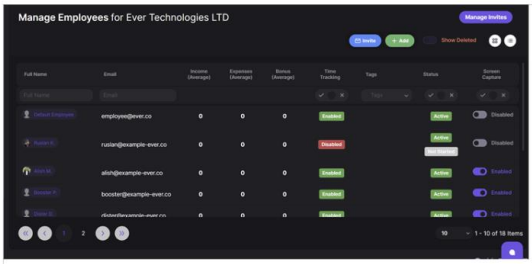
Постановка задачі

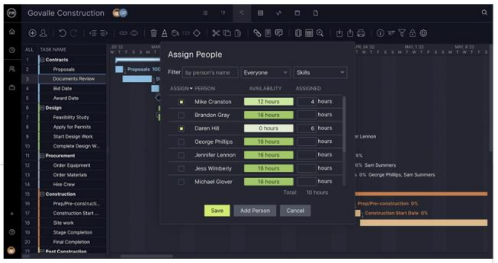
- провести аналіз досліджень та публікацій, що стосуються управління ресурсами у IT проектах, існуючих методів та інструментів оптимізації ресурсів;
- удосконалити метод оптимізації використання ресурсів у IT проектах з урахуванням сучасних потреб і вимог, сформованих на основі проведеного опитування фахівців та керівників проектів.
- розробити програмний засіб для оптимізації використання ресурсів IT проекту на основі запропонованого методу;
- на основі тестування зробити висновки, щодо перспектив використання розробленого методу та програмного засобу;

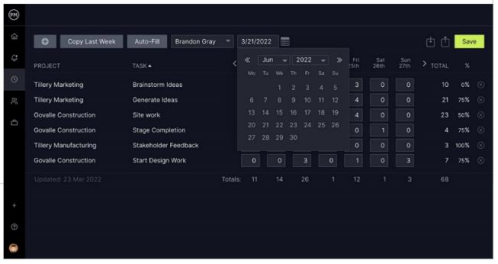
Порівняння алгоритмів оптимізації ресурсів

Критерій	Жадібний алгоритм	Генетичні алгоритми	Лінійне програмування	Градієнтні методи
Переваги	Простота і швидкість виконання	Здатність пристосовуватися до складних функцій	Ефективність для лінійних завдань, простота	Ефективність для гладких функцій, швидкість збіжності
		Здатність уникати застрягання в локальних мінімумах	Можливість розв'язання широкого спектру задач	Збіжність до локального оптимуму, застосування в різних сферах
		Випадкова генерація нових рішень	Використання методів симплексу для розв'язання	
Недоліки	Не завжди гарантує найкращий глобальний результат	Велика обчислювальна складність	Обмежена застосовність для нелінійних задач	Чутливість до початкових умов, можливість застрягання в локальних мінімумах
		Вимагає великої кількості обчислень	Чутливість до даних вхідних параметрів	Чутливість до початкових умов, можливість застрягання в локальних мінімумах
		Може бути неефективним для простих задач	Обмежена ефективність для складних неструктурованих задач	Висока обчислювальна складність

Аналіз аналогів

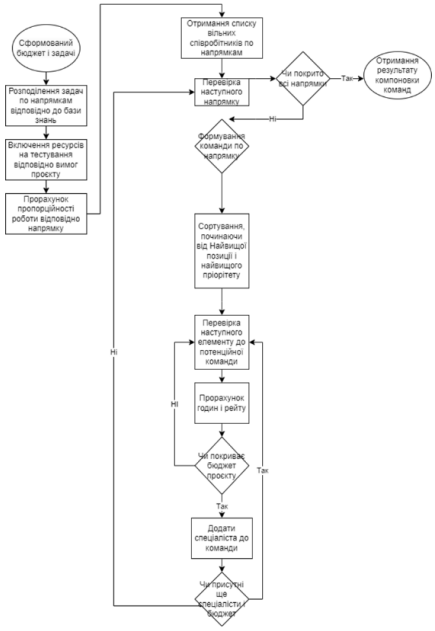






Проектування методу оптимізації ресурсів на проектах у IT

- формування бюджет проєкту і його задач по типам;
- розбиття задач по напрямках та пропорційний розрахунок навантаженості;
- отримання доступних співробітників по напрямках та підбір відносно задач;
- сортування по критеріям та перевірка чи закритий напрям задачі;
- додавання співробітника, якщо він відповідає критеріям;
- завершення алгоритму і отримання результату.



Вибір методу автентифікації

Firebase Authentication

VS

AWS Cognito

Обрання засобів розробки

Мова програмування TypeScript

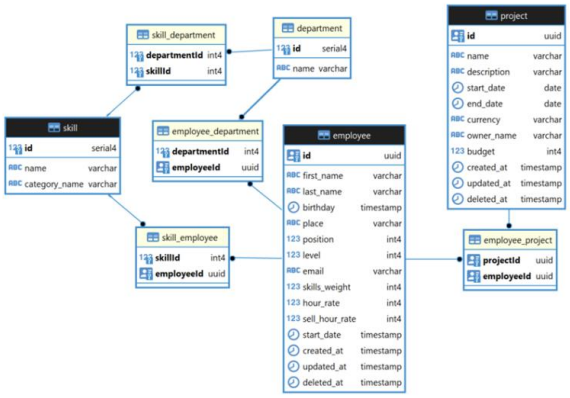
База даних Postgres

Серверна частина NestJS, ExpressJS

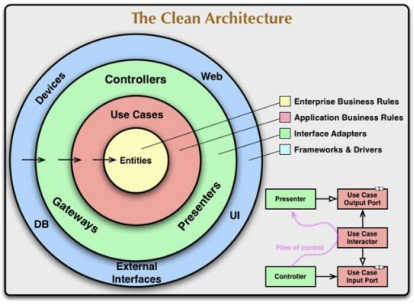
Користувацький інтерфейс ReactJS, SCSS, HTML

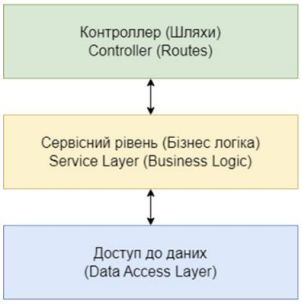
Середовище розробки Visual Studio Code

Проектування бази даних

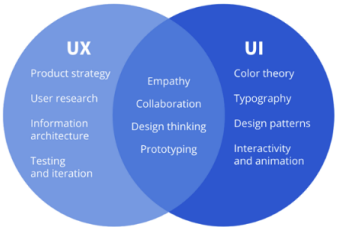


Проектування серверної частини

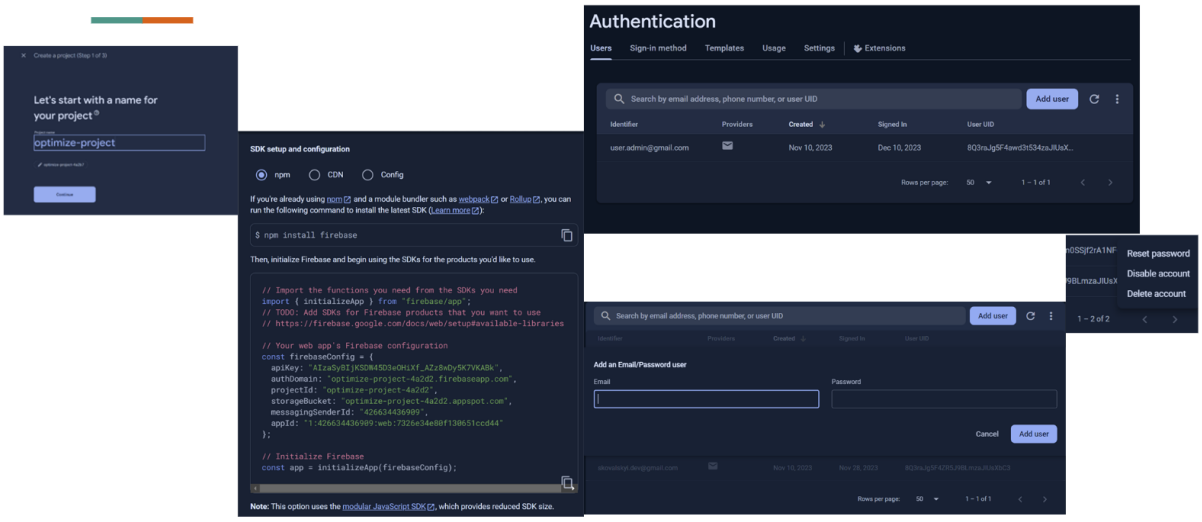




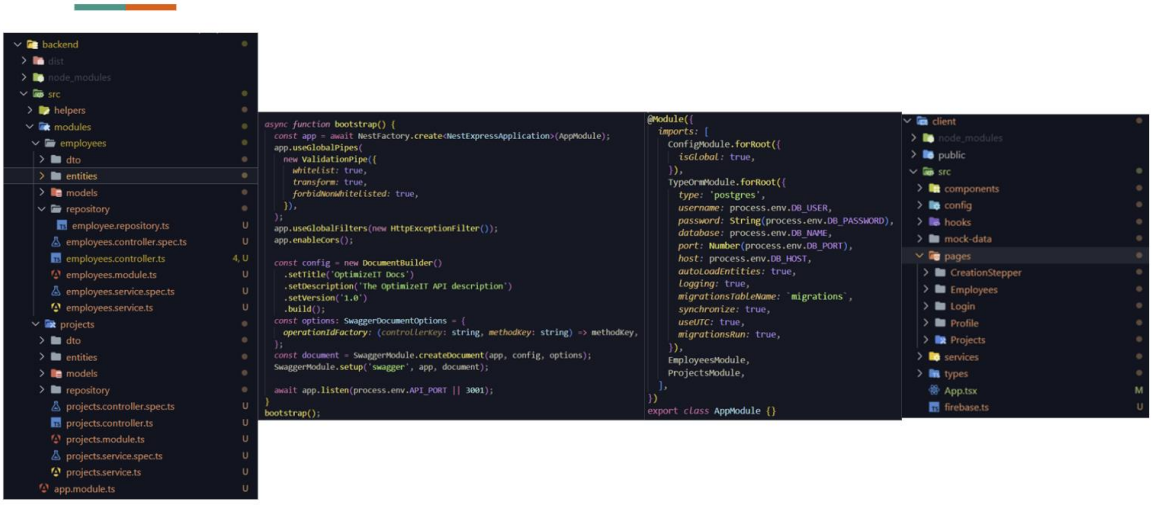
Проектування інтерфейсу користувача



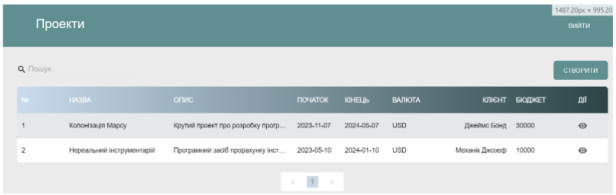
Впровадження Firebase Authentication

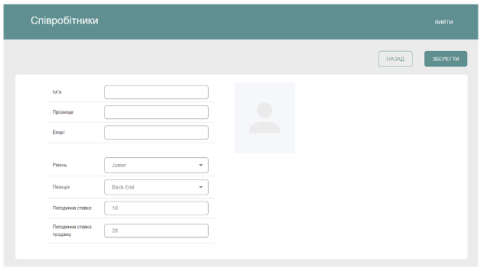


Структура проектів

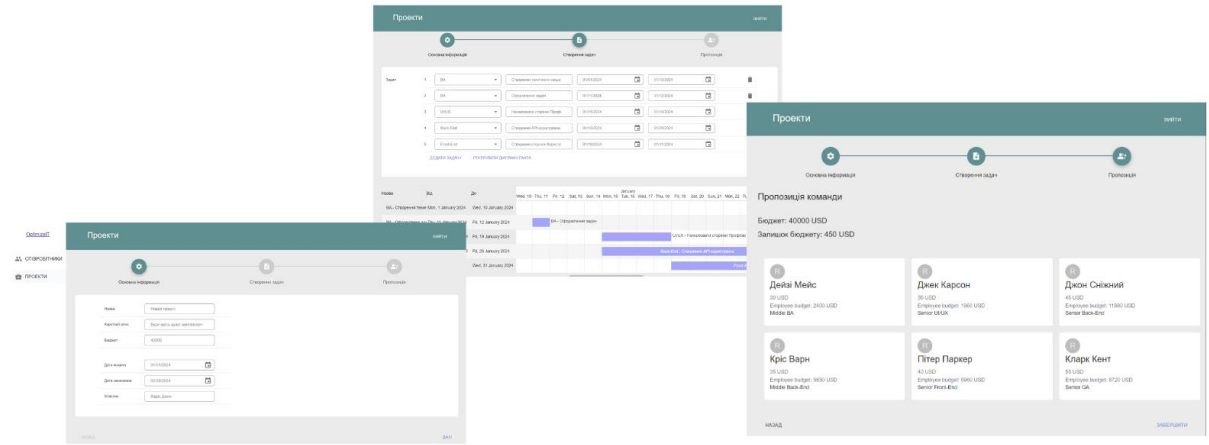


Робота застосунку: список проектів та співробітників. Створення співробітника





Робота застосунку: створення проекту і застосування методу оптимізації ресурсів



Апробації та публікації

Результати магістерської кваліфікаційної роботи доповідались та обговорювались на міжнародній науково-практичній Інтернет-конференції “Електронні інформаційні ресурси: створення, використання, доступ 2023” - Ковальський С. В., Тужанський С. Є.

Оцінювання та вимірювання успіху освіти з використанням цифрових інструментів – 2023

Висновки

- Проведено аналіз методів оптимізації та програмних засобів по оптимізації ресурсів
- Проведено порівняння існуючих алгоритмів оптимізації ресурсів
- Спроектовано метод для оптимізації ресурсів на проектах
- Обґрунтовано вибір технологій та середовища розробки
- Реалізовано програмний засіб, який складається з серверної частини та інтерфейсу користувача та використано розроблений метод оптимізації ресурсів
- Проведено тестування

Дякую за увагу!