Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка системи менеджменту IT-компанії, з використанням Entity Framework та Angular

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Баб'як Н.І. (прізвище та ініціали)
Керівник	(підпис)	Цуприк Г.Б. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	Матійчук Л.П. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка системи менеджменту IT-компанії, з використанням Entity Framework та Angular» виконана Баб'яком Назарієм Ігоровичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41.

Баб'як Н. І. Розробка вебзастосунку для управління діяльністю IT-компанії : кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю „121 — інженерія програмного забезпечення“ / Назарій Ігорович Баб'як. — Тернопіль: ТНТУ, 2025. — 61 с.

Відомості про обсяг: сторінок — 61, рисунків — 25, таблиць — 3, частин — 3, додатків — 2, посилань — 26.

Метою роботи є створення вебзастосунку, що забезпечує централізоване управління бізнес-процесами IT-компанії. Система автоматизує облік працівників, проєктів, задач, ролей та фінансових операцій, підвищує ефективність внутрішньої організації, зменшує обсяг ручної роботи та сприяє прозорому розподілу завдань.

Система створена з використанням Angular на клієнтській частині та ASP.NET Core з Entity Framework на серверній стороні. Такий підхід забезпечує масштабованість, високу продуктивність і зручність підтримки програмного продукту.

Результатом роботи є сучасний вебзастосунок, що відповідає вимогам функціональності, безпеки, зручності у використанні та придатності до розгортання в умовах реального підприємства.

ABSTRACT

Bachelor's qualification thesis on the topic “ Development of an IT company management system using Entity Framework and Angular ” was completed by Nazarii Ihorovych Babyak, a student of the Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41.

Babyak N. I. Development of a Web Application for IT Company Management: Bachelor's qualification thesis for obtaining the degree of Bachelor in specialty “121 — Software Engineering” / Nazarii Ihorovych Babyak. — Ternopil: TNTU, 2025. — 61 p.

Thesis details: pages — 61, figures — 25, tables — 3, sections — 3, appendices — 2, references — 26.

The aim of this thesis is to develop a web application that enables centralized management of an IT company's business processes. The system automates employee management, project and task tracking, role-based access control, and financial operations, enhancing internal organizational efficiency, reducing manual workload, and promoting transparency in task distribution.

The system was developed using Angular on the client side and ASP.NET Core with Entity Framework on the server side. This technology stack ensures scalability, high performance, and maintainability of the software product.

The result of the work is a modern web application that meets the requirements of functionality, security, usability, and is suitable for deployment in a real business environment.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій та глобалізації бізнес-процесів IT-компанії стикаються з необхідністю ефективної організації внутрішніх процесів, управління проєктами, співробітниками, завданнями та клієнтською базою. В умовах жорсткої конкуренції на ринку послуг інформаційних технологій саме здатність до швидкої адаптації, чіткої організації та прозорого контролю процесів визначає успіх компанії. Одним із важливих інструментів у досягненні цих цілей є спеціалізоване програмне забезпечення для внутрішнього менеджменту.

Сьогодні існує велика кількість SaaS-продуктів для управління бізнесом, однак багато з них є занадто загальними або дорогими, не забезпечують гнучкості для потреб окремої компанії або не враховують специфіку IT-індустрії. Саме тому виникає потреба у створенні власного вебзастосунка, орієнтованого на потреби конкретного підприємства, з урахуванням індивідуальних вимог до процесів управління персоналом, задачами, проєктами та відстеження прогресу.

Дана робота присвячена розробці вебзастосунка для менеджменту IT-компанії, реалізованого з використанням сучасного стеку технологій: Angular для фронтенду та ASP.NET Core з Entity Framework Core для бекенду. Така комбінація дозволяє створити швидкий, масштабований та зручний у використанні застосунок із чітким розподілом відповідальностей між клієнтською та серверною частинами.

Розроблена система забезпечує можливості управління:

- користувачами та ролями;
- проєктами та завданнями;
- статусами виконання задач;
- аутентифікацією та авторизацією користувачів;
- адміністративною панеллю та аналітикою.

Актуальність теми полягає в необхідності забезпечення ефективного управління діяльністю IT-компанії за допомогою індивідуального, гнучкого та масштабованого програмного забезпечення, яке враховує специфіку сучасних бізнес-процесів у сфері розробки програмного забезпечення.

Метою даної роботи є проєктування та розробка системи менеджменту для IT-компанії, яка дозволить автоматизувати внутрішні процеси керування персоналом і проєктами, забезпечить прозору взаємодію між учасниками команди та сприятиме підвищенню продуктивності роботи.

Для досягнення мети необхідно реалізувати наступні завдання:

- провести аналіз предметної області та вимог до системи;
- спроєктувати архітектуру програмного забезпечення;
- реалізувати вебінтерфейс з використанням Angular;
- створити серверну частину з використанням ASP.NET Core та Entity Framework;
- виконати тестування та верифікацію основних функцій системи;
- описати процес розгортання системи та її подальшого використання.

Об’єктом дослідження є внутрішні процеси управління IT-компанією.

Предметом дослідження є методи, інструменти та технології розробки корпоративного вебзастосунка для менеджменту проєктів, персоналу та задач із використанням Angular, ASP.NET Core та Entity Framework.

1. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1. Аналіз вимог до програмної системи

Аналіз вимог є одним з найважливіших етапів розробки програмної системи, адже саме на цьому етапі визначаються цілі створення системи, її функціональні можливості, основні користувачі та сценарії взаємодії. Якісний аналіз дозволяє уникнути непорозумінь між замовником і розробниками, сформувати єдине бачення очікуваного результату та закласти основу для подальшого проєктування [1].

Цей підрозділ охоплює п’ять послідовних кроків. Спершу виконується аналіз предметної області, що дозволяє зрозуміти специфіку роботи IT-компанії, її бізнес-процеси та проблеми, які потребують автоматизації. Далі формується постановка задачі — описується мета створення програмного забезпечення та основні функції, які повинна реалізовувати система.

Наступним етапом є пошук актантів та варіантів використання, тобто виявлення всіх ролей, що взаємодіятимуть із системою, а також можливих сценаріїв їхньої роботи. Потім на основі цього виконується опис ключових варіантів використання, в яких розглядаються типові дії користувачів у межах системи [2].

Завершальним кроком є моделювання словника системи, де формуються визначення основних термінів, понять та сутностей, які використовуються в системі. Це забезпечує уніфікацію термінології та спрощує спілкування між усіма учасниками розробки.

1.1.1. Аналіз предметної області

Управління діяльністю ІТ-компанії охоплює низку важливих процесів, серед яких ключовими є облік персоналу, управління проєктами, призначення та контроль виконання задач, а також адміністрування прав доступу. Ефективне функціонування ІТ-компанії потребує злагодженої взаємодії між співробітниками різних ролей, чіткого визначення відповідальностей та забезпечення контролю за виконанням поставлених завдань [2].

Основними об’єктами предметної області є: співробітники, проєкти, задачі, ролі, статуси задач, а також права доступу. Співробітники можуть бути залучені до кількох проєктів одночасно, а задачі можуть мати різні пріоритети та строки виконання. Контроль за цими об’єктами зазвичай здійснюється менеджерами, які відповідають за розподіл задач, моніторинг прогресу та коригування планів відповідно до потреб компанії.

У реальних умовах такі процеси часто реалізуються за допомогою різнорідних інструментів — таблиць, окремих сервісів для спілкування, сторонніх таск-трекерів. Це призводить до дублювання інформації, втрати контексту, складнощів у підтримці єдиної структури управління та ускладненого контролю ефективності.

Актуальною є потреба в уніфікації зазначених процесів в межах єдиної інформаційної системи, яка б забезпечувала:

- централізоване зберігання даних про співробітників, проєкти та задачі;
- підтримку ієрархії ролей та прав доступу;
- можливість простежити життєвий цикл задач;
- ефективну взаємодію між усіма учасниками команди.

Таким чином, предметна область включає процеси управління внутрішніми ресурсами ІТ-компанії, формалізацію задач і ролей, взаємодію між учасниками команд і контроль за виконанням проєктних робіт.

1.1.2 Постановка задачі

Метою розробки програмної системи є створення вебзастосунку для управління діяльністю ІТ-компанії, який забезпечить централізовану автоматизацію основних бізнес-процесів: облік працівників, проєктів, задач, ролей та прав доступу. Такий застосунок має підвищити ефективність управління компанією, зменшити кількість ручної роботи та забезпечити прозорість і контроль над виконанням задач [1].

Для досягнення цієї мети необхідно реалізувати такі основні функціональні можливості системи:

- Управління користувачами: створення облікових записів співробітників, призначення ролей (адміністратор, менеджер, працівник) та налаштування рівнів доступу;
- Управління проєктами: створення, редагування та архівування проєктів із можливістю призначення відповідальних осіб;
- Управління задачами: створення задач у межах проєктів, призначення виконавців, встановлення дедлайнів, зміна статусу задачі;
- Ведення ролей і прав доступу: реалізація механізму авторизації, що обмежує доступ до певних дій залежно від ролі користувача;
- Візуалізація інформації: надання зручного інтерфейсу для перегляду задач, фільтрації та сортування, а також інформування про прогрес виконання.
- Ведення обліку фінансів: розрахунок кошторису об’єктів та нарахування заробітної плати працівникам.

Також система повинна бути розгорнута у вигляді SPA (single-page application) з використанням Angular на фронтенді та .NET Core з Entity Framework на бекенді, що забезпечить високу швидкодію, масштабованість і зручність у підтримці.

1.1.3 Пошук актантів та варіантів використання

У процесі аналізу предметної області було визначено три основні категорії акторів, які взаємодіють із системою: адміністратор, менеджер та монтажник. Кожен з них виконує специфічні функції, які відповідають його ролі в організації, що забезпечує комплексну підтримку управлінських і виробничих процесів.

Адміністратор є особою, що відповідає за повне управління користувачами та структурою системи. Він має можливість додавати нових користувачів, змінювати їхні дані, видаляти облікові записи, призначати ролі, а також скидувати паролі. Крім того, адміністратор виконує призначення робіт монтажникам, контролює статус їх виконання, генерує звіти, переглядає та редагує службові дані. Його функції забезпечують безперебійну роботу всієї системи, контроль за ресурсами та підтримку актуальності інформації.

Менеджер є посередником між адміністрацією та виконавцями. Він відповідає за контроль і облік виконаних робіт, перегляд персоналу, позначення завершених завдань, аналіз загальної статистики діяльності, а також розрахунок заробітної плати монтажників. Менеджер має доступ до функцій генерації аналітичних звітів і виконує ключову роль у плануванні, моніторингу та оцінці ефективності роботи персоналу, що напряму впливає на якість управління процесами.

Монтажник виконує функції безпосереднього виконавця завдань. Він отримує доступ до призначених йому робіт, відмічає їх виконання, стежить за власним прогресом, переглядає особисту інформацію та може змінювати пароль доступу до системи. Його роль полягає в реалізації поточних завдань, забезпеченні своєчасного звітування про виконання, що дозволяє вищим ролям здійснювати контроль та формувати необхідну звітність. Монтажник є джерелом первинних даних, що використовуються для нарахування заробітної плати та формування звітів.

Взаємодія між цими акторами дозволяє системі функціонувати як цілісний механізм, де кожна роль чітко визначена, а функціональні можливості відповідають практичним потребам користувачів у межах робочих процесів [2].

1.1.4 Опис ключових варіантів використання

У процесі побудови функціональної моделі системи було ідентифіковано основних користувачів (акторів), які взаємодіють із програмним забезпеченням, а також описано типові сценарії їхньої поведінки — варіанти використання. Для кращої структурованості й наочності ці сценарії було подано в таблиці 1.1, яка містить коди варіантів, назви акторів, короткі назви функцій та їх формулювання [4].

Таблиця 1.1 – Варіанти використання системи

Код	Актор	Найменування	Формулювання
A1	Адміністратор	Керування користувачами	Забезпечує можливість додавання, редагування, видалення користувачів, призначення ролей, скидання паролів.
A2	Адміністратор	Призначення робіт монтажникам	Надає змогу призначати завдання конкретним монтажникам.
A3	Адміністратор	Перегляд статусу виконаних робіт	Дає можливість бачити стан виконання робіт по кожному монтажнику.
A4	Адміністратор	Управління звітами	Дозволяє генерувати звіти по роботах, монтажниках, експортувати їх.

Продовження таблиці 1.1

A5	Адміністратор	Перегляд і редагування даних	Надання доступу до перегляду та редагування довідкової інформації (користувачів, робіт, звітів тощо).
M1	Монтажник	Перегляд призначених робіт	Дає змогу переглядати список завдань, призначених конкретному монтажнику.
M2	Монтажник	Позначення виконання робіт	Дозволяє монтажнику вказати, що певна робота виконана.
M3	Монтажник	Перегляд статусу виконаних робіт	Монтажник переглядає статус своїх завдань (виконано/не виконано).
M4	Монтажник	Перегляд особистої інформації	Дає можливість перегляду власного профілю.
M5	Монтажник	Зміна паролю	Дозволяє змінити пароль до облікового запису.
MN1	Менеджер	Перегляд списку монтажників	Перегляд усіх монтажників, які є в системі.
MN2	Менеджер	Позначення виконаних робіт	Менеджер може підтвердити чи вказати статус виконання робіт.
MN3	Менеджер	Підрахунок зарплатні монтажникам	Формує зарплатний звіт на основі обсягу виконаних робіт.
MN4	Менеджер	Генерація звітів по роботах	Формування загального звіту по роботах.
MN5	Менеджер	Перегляд загальної статистики	Перегляд аналітичної інформації щодо виконаних робіт.

Діаграма варіантів використання (UML Use Case Diagram) є важливим інструментом моделювання функціональності системи. Вона демонструє взаємодію зовнішніх акторів (користувачів або інших систем) із системою через

сценарії використання (use cases), які описують основні функції, що повинні бути реалізовані.

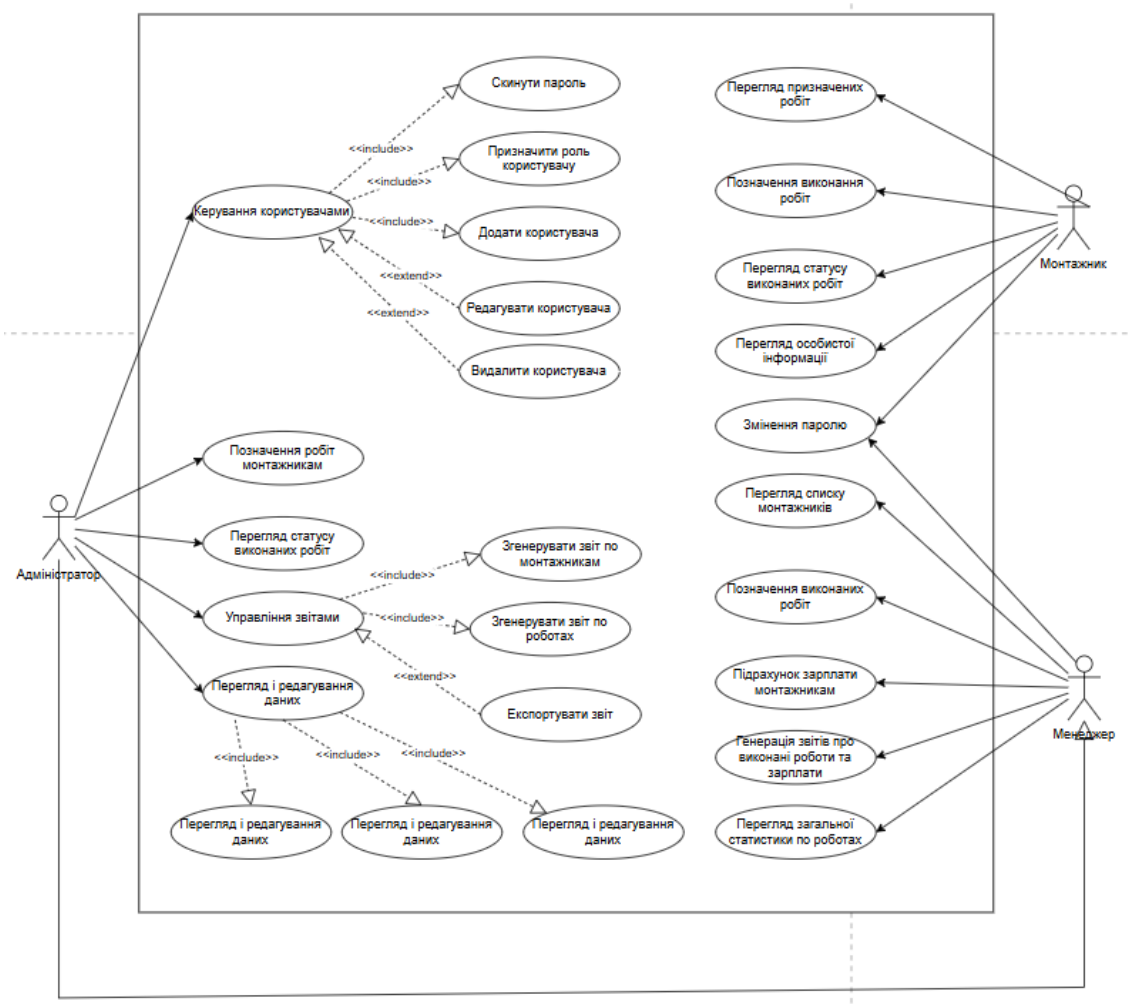

Рисунок 1.1 – Діаграма варіантів використання

1.1.5 Моделювання словника системи

Для забезпечення єдності термінології під час розробки, впровадження та супроводу програмної системи необхідно сформувати словник основних понять, які використовуються в межах проєкту. Такий словник дозволяє уникнути неоднозначностей у розумінні функцій системи, визначити сутності, які є ключовими для предметної області, а також встановити їх чіткі формулювання.

Нижче наведено основні терміни, які використовуються в межах інформаційної системи

Таблиця 1.2 – Словник системи

№	Термін	Визначення
1	Користувач	Особа, яка має обліковий запис у системі та взаємодіє з нею за допомогою інтерфейсу.
2	Обліковий запис	Персоналізований набір даних, що дозволяє автентифікувати та ідентифікувати користувача.
3	Проєкт	Організаційна одиниця, що об'єднує низку пов’язаних задач для досягнення спільної мети.
4	Задача	Оперативне завдання, що виконується в межах певного проєкту та має опис, термін і стан.
5	Статус задачі	Означення поточного стану виконання задачі (наприклад: нова, у процесі, завершена).
6	Кошторис	Попередній розрахунок витрат, пов’язаних із реалізацією проєкту чи виконанням задач.
7	Зарплатна відомість	Документ або набір даних, що містить інформацію про обчислену оплату за виконану роботу.
8	Аутентифікація	Процедура перевірки особистості користувача під час входу до системи.
9	Авторизація	Процедура перевірки прав користувача для доступу до певних функцій або даних.
10	Інтерфейс	Сукупність візуальних елементів, з якими взаємодіє користувач у процесі роботи із системою.

Продовження таблиці 1.2

11	Дані задачі	Описова інформація про завдання: назва, опис, термін, пов’язані об'єкти.
12	Аналітичний модуль	Підсистема, що здійснює обробку та візуалізацію статистичних і фінансових даних системи.

1.2. Проектування програмної системи

Проєктування визначає структуру майбутньої системи, її компоненти, взаємозв’язки та логіку функціонування. На цьому етапі відбувається перехід від аналізу вимог до технічної реалізації, що забезпечує фундамент для ефективної розробки, тестування та подальшої підтримки програмного продукту.

Проєктування включає в себе вибір моделі процесу розробки, побудову логічної та фізичної структури бази даних, створення UML-діаграм для візуалізації об’єктно-орієнтованої моделі системи, а також моделювання загальної архітектури програмного забезпечення. Такий підхід дозволяє забезпечити узгодженість між функціональними вимогами та технічною реалізацією, знизити ризики на етапі розробки та полегшити подальше масштабування системи.

1.2.1 Вибір процесу розробки

Під час розробки вебзастосунку Personal IT Management було проаналізовано кілька моделей життєвого циклу програмного забезпечення, серед яких каскадна модель, V-модель, інкрементна модель, гнучкі методології (Agile) тощо. Ураховуючи специфіку та вимоги проєкту — поступове нарощування функціональності, зміна вимог протягом розробки, необхідність постійного

зворотного зв’язку та активне тестування — було обрано гнучкий підхід Agile, а саме його реалізацію через Scrum-методологію [14].

Agile дозволяє вести розробку за ітеративною моделлю, коли програмний продукт створюється частинами, у вигляді функціональних модулів, які постійно тестуються та вдосконалюються. В цьому випадку система розроблялася у вигляді серії спринтів — коротких періодів (1–2 тижні), протягом яких формувались задачі, реалізовувались окремі блоки функціоналу (наприклад, авторизація, додавання задач, розрахунок заробітної плати), і проводилося їх тестування.

Процес розробки мав наступні характеристики:

- Ітеративність — кожна нова функція впроваджувалась поступово, у вигляді нових етапів (спринтів);
- Інкрементність — наявний функціонал постійно доповнювався новими модулями без потреби повної перебудови архітектури;
- Комунікація — на кожному етапі розробки здійснювався перегляд реалізованого функціоналу, обговорення змін, уточнення вимог;
- Гнучкість — враховувалися нові ідеї та потреби, які виникали в процесі реалізації, зокрема вдосконалення інтерфейсу, розширення ролей користувачів, реалізація модуля обліку зарплати;
- Контроль якості — після кожної реалізованої частини функціоналу проводилось модульне тестування, перевірка логіки та адаптація коду.

Scrum дозволив розділити розробку на чіткі етапи з визначеним обсягом роботи та дедлайнами. Це забезпечило керованість процесом, вчасне виявлення та усунення недоліків, а також максимальне наближення результату до потреб кінцевих користувачів [2].

Така методологія також сприяла продуктивному використанню інструментів спільної роботи (наприклад, GitHub для контролю версій), що особливо актуально в командній або індивідуальній розробці з активним використанням гілок, pull-запитів і code review.

Отже, вибір Agile як процесу розробки виявився оптимальним для реалізації даного програмного забезпечення, дозволяючи швидко, адаптивно й ефективно створити якісний вебзастосунок для управління діяльністю IT-компанії.

1.2.2 Побудова схеми бази даних

У межах цього проєкту було розроблено схему бази даних для системи керування замовленнями та канбан-дошкою. Вона реалізована з використанням технології Entity Framework Core, що забезпечує зручну роботу з реляційною базою даних. Схема містить кілька сутностей, які логічно пов’язані між собою та дозволяють зберігати всю необхідну інформацію для функціонування системи.

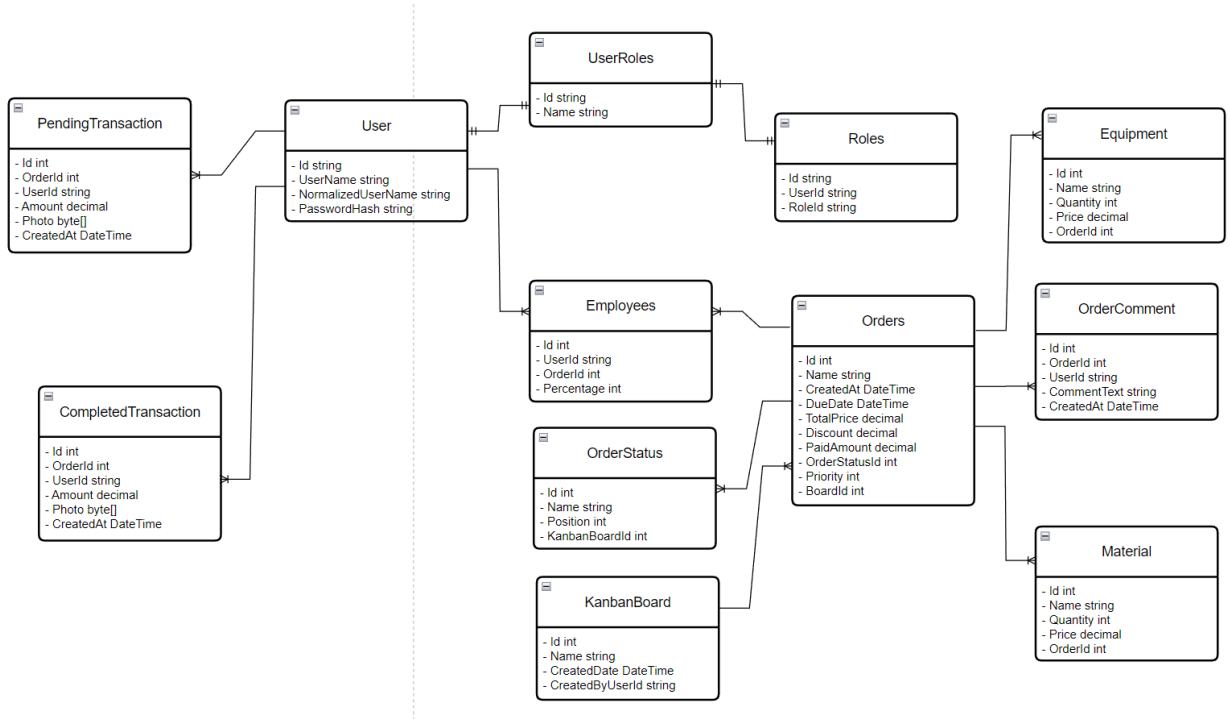

Рисунок 1.2 – Діаграма бази даних

Однією з основних сутностей є користувач, який може створювати замовлення, коментарі, а також здійснювати фінансові транзакції. Кожному

користувачеві може бути призначена одна або кілька ролей, наприклад адміністратор, менеджер або працівник. Зв’язок між користувачами та ролями реалізований через окрему таблицю для зв’язків, що дозволяє підтримувати гнучку систему прав доступу.

Замовлення є центральним елементом цієї системи. Воно містить інформацію про дату створення, термін виконання, загальну вартість, суму знижки та суму, яку вже оплачено. Кожне замовлення має свій статус, що дозволяє візуально контролювати його прогрес на канбан-дошці. Статуси можуть бути, наприклад, "Нове", "У процесі", "Завершено", й кожен з них належить до певної дошки.

Канбан-дошка служить для організації процесу роботи із замовленнями. Вона об’єднує всі статуси, які відображають етапи виконання замовлень, і дозволяє користувачам швидко орієнтуватися в поточному стані справ. Кожне замовлення прикріплене до певної дошки, а також до одного зі статусів, що представляють колонку на канбані.

До замовлень можуть бути прикріплені коментарі, які залишають користувачі, що дозволяє обговорювати деталі або залишати нотатки. Також кожне замовлення може містити інформацію про використані матеріали й обладнання — із зазначенням кількості та вартості. Це дозволяє вести облік витрат і краще планувати ресурси.

У системі реалізовано два типи фінансових транзакцій — очікувані та підтверджені. Очікувані транзакції створюються користувачами й можуть містити фотографію як підтвердження, а після перевірки адміністратором вони переводяться у завершені. Це дозволяє контролювати фінансові потоки в межах кожного замовлення.

Крім того, працівники, які беруть участь у виконанні замовлень, також фіксуються в базі даних. Для кожного працівника зберігається відсоток його участі в конкретному замовленні, що в подальшому можна використати для розрахунку прибутку чи заробітної плати.

Таким чином, створена схема бази даних дозволяє ефективно організувати всі бізнес-процеси, пов’язані з управлінням замовленнями, контролем фінансів і ресурсів, а також взаємодією між користувачами системи [9].

1.2.3 Побудова UML-діаграми класів

На цьому етапі розробки системи було побудовано UML-діаграму класів, що відображає основні програмні компоненти, їхні атрибути та взаємозв’язки між собою. Діаграма класів є важливим інструментом моделювання, який дозволяє логічно структурувати програмну архітектуру ще до її реалізації.

Діаграма класів охоплює як моделі даних (наприклад, Orders, Employees, Material, Equipment), так і контролери, що забезпечують взаємодію клієнта з API. У центрі моделі знаходиться клас Orders, який має зв’язки з іншими ресурсами: обладнанням (Equipment), матеріалами (Material), працівниками (Employees) та коментарями (OrderComment). Кожен з цих ресурсів має окремий контролер, відповідальний за CRUD-операції (створення, отримання, оновлення, видалення).

Контролери, такі як OrdersController, EmployeesController, EquipmentController тощо, реалізують окремі функціональні модулі системи і забезпечують обробку запитів REST API. Також на діаграмі відображено взаємозв’язок між користувачами (User) та аутентифікаційним контролером (AuthController), який відповідає за реєстрацію, вхід у систему та перевірку сесій [10].

Крім того, на діаграмі присутній модуль KanbanBoard, що дозволяє організувати замовлення у вигляді задач на дошці з різними статусами, подібно до Kanban-систем [4].

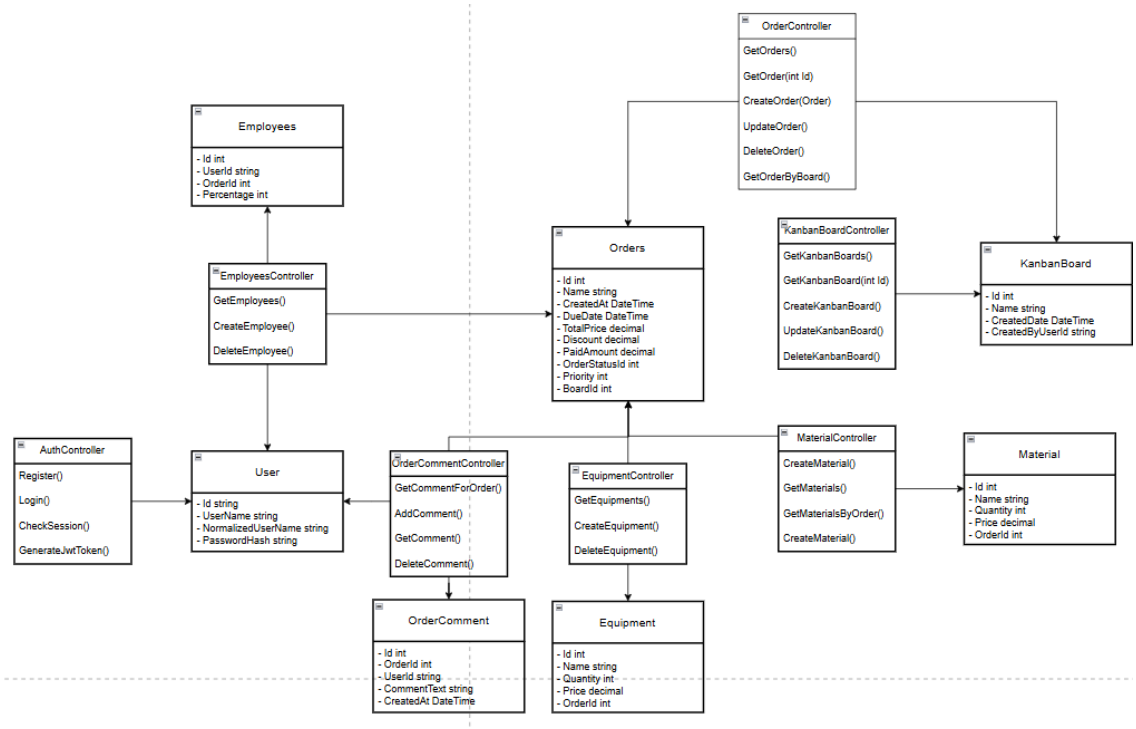

Рисунок 1.3 – Діаграма класів

1.2.4 Моделювання архітектури системи

У розробленій системі використано трьохрівневу архітектуру клієнт-сервер, яка логічно поділяє програму на три окремі рівні: представлення, бізнес-логіки та доступу до даних. Така архітектурна модель дозволяє досягти гнучкості, масштабованості, а також полегшує супровід та розвиток проєкту.

Перший рівень — це рівень представлення, який відповідає за взаємодію з користувачем або зовнішніми системами. У цьому додатку він реалізований у вигляді REST API-контролерів, таких як OrdersController, EmployeesController, AuthController тощо. Вони приймають HTTP-запити, викликають відповідну бізнес-логіку та повертають результат у вигляді відповіді.

Другий рівень — бізнес-логіка — реалізує основні правила функціонування системи. Тут знаходяться класи-сутності (моделі), що відображають ключові об’єкти предметної області, такі як Order, Employee, Material, Equipment,

KanbanBoard. Саме на цьому рівні визначено взаємозв’язки між сутностями, обробку даних, а також специфічну поведінку, пов’язану з виконанням замовлень, призначенням працівників тощо.

Третій рівень — доступ до даних. У системі використовується ORM-технологія Entity Framework Core, яка абстрагує роботу з базою даних. Усі операції зі зчитування, збереження та оновлення інформації реалізуються через контекст бази даних ApplicationDbContext. Таким чином, робота з фізичним сховищем даних ізольована від решти системи [9].

Завдяки такому розмежуванню архітектура системи є чіткою, модульною та легко підтримуваною. Кожен рівень виконує лише свою відповідальність, що дає змогу змінювати або оновлювати його без істотного впливу на інші частини. Це особливо важливо у складних інформаційних системах, які мають рости та адаптуватися до змін з часом [5].

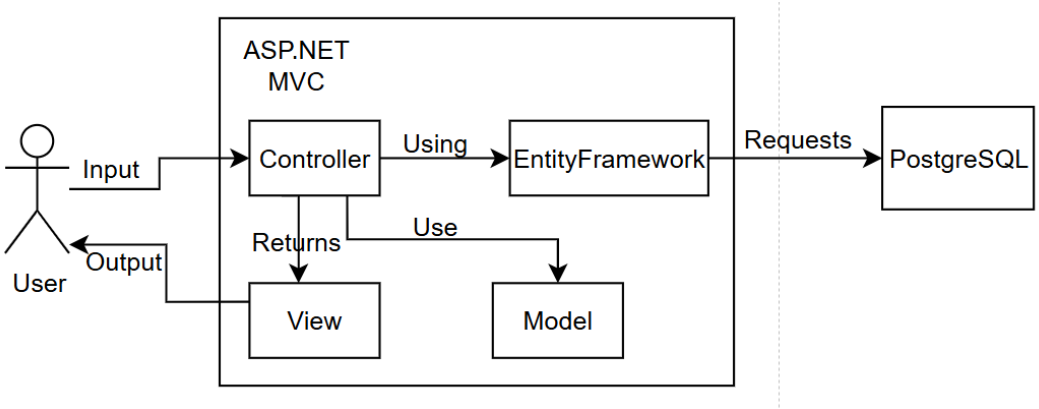

Рисунок 1.4 – Архітектура системи

1.3. Конструювання програмної системи

Після завершення етапів аналізу вимог та проектування, наступним кроком є безпосередня реалізація програмної системи. Конструювання охоплює вибір відповідних технологій, середовища розробки, системи керування базами даних, а

також реалізацію основних модулів, класів і методів, що забезпечують функціональність додатку. На цьому етапі відбувається втілення архітектурних рішень у вигляді програмного коду та налаштування інфраструктури, необхідної для роботи системи [1].

Цей розділ містить опис обраної мови програмування та середовища розробки, обґрунтування вибору СУБД, а також структуру основних програмних компонентів, реалізованих відповідно до поставлених вимог.

1.3.1 Вибір мови та середовища розробки

Під час розробки програмної системи було обрано низку сучасних мов програмування, фреймворків та середовищ, які забезпечують якісну реалізацію як клієнтської, так і серверної частини, гнучкість у розробці, легкість масштабування та зрозумілу підтримку коду.

Клієнтська частина вебзастосунку реалізована з використанням HTML, CSS, SCSS, JavaScript і TypeScript. HTML (HyperText Markup Language) використовується для створення структури сторінок, тобто для розміщення елементів інтерфейсу та контенту. Для стилізації зовнішнього вигляду застосовується CSS (Cascading Style Sheets), а також препроцесор SCSS (Sassy CSS), що є надбудовою над CSS. SCSS дозволяє використовувати змінні, вкладеність, міксини та інші інструменти, які значно покращують читабельність та масштабованість стилів, особливо при роботі з великим UI [11].

JavaScript забезпечує динамічну поведінку елементів на сторінці, обробку подій, інтерактивність інтерфейсу, а також взаємодію з API. TypeScript є надбудовою над JavaScript, яка додає статичну типізацію, що значно знижує ймовірність помилок і покращує автодоповнення та рефакторинг під час розробки. Ці мови активно застосовуються для управління станом інтерфейсу,

взаємодії з сервером через HTTP-запити, валідації форм та іншої бізнес-логіки клієнтської частини.

Серверна частина реалізована з використанням мови програмування C# в рамках фреймворку ASP.NET Core, що забезпечує побудову RESTful API. C# — це потужна об'єктно-орієнтована мова, яка широко застосовується у корпоративному програмуванні, а ASP.NET Core дозволяє ефективно обробляти HTTP-запити, реалізовувати маршрутизацію, керування авторизацією, автентифікацією та доступом до бази даних. Також серверна частина відповідає за реалізацію логіки керування користувачами, проєктами, задачами, обрахунком зарплат та кошторисів [10].

Основним середовищем розробки серверної частини обрано Visual Studio, яке забезпечує повний набір інструментів для розробки, налагодження, тестування, керування проєктами та використання пакетів NuGet. Visual Studio є офіційним середовищем для розробки на C# та .NET, і дозволяє ефективно працювати з багатошаровими архітектурами. Для клієнтської частини використовувалося легке, але потужне середовище Visual Studio Code, яке підтримує сучасні вебтехнології, має велику кількість розширень і забезпечує комфортну роботу з TypeScript, SCSS і HTML [2].

Такий вибір мов і середовищ дозволив реалізувати надійну, масштабовану, зрозумілу та ефективну програмну систему з чітким розділенням відповідальностей між клієнтською і серверною частинами.

1.3.2 Вибір СУБД та опис її фізичної моделі

Для зберігання даних у розробленій системі було обрано систему управління базами даних PostgreSQL. Такий вибір зумовлений кількома важливими чинниками. Насамперед, PostgreSQL — це потужна реляційна СУБД з відкритим вихідним кодом, яка активно використовується як у навчальних, так і у

комерційних проєктах. Вона надає широкий набір функціональних можливостей, включаючи підтримку транзакцій, зовнішніх ключів, перевірок цілісності даних, індексів, представлень, збережених процедур та тригерів, що повністю відповідає вимогам мого застосунку [12].

Окрім цього, PostgreSQL є кросплатформеним рішенням, що дозволяє легко розгортати її як на локальному сервері, так і в хмарному середовищі. Вона добре інтегрується з .NET-платформою, зокрема через постачальника Npgsql, що забезпечує зручне з'єднання з базою даних за допомогою засобів ORM, таких як Entity Framework Core. У процесі розробки це дало змогу швидко налаштувати підключення до бази даних, а також виконувати всі необхідні операції без написання великої кількості SQL-запитів вручну [13].

Ще одним важливим фактором вибору стало активне ком'юніті PostgreSQL, що забезпечує велику кількість документації, прикладів, розв'язань типових помилок, а також постійну підтримку та оновлення. Це значно спростило процес впровадження СУБД у проєкт і дозволило зосередитися безпосередньо на логіці додатку, не витрачаючи багато часу на конфігурацію та налагодження.

У результаті PostgreSQL було обрано як основну систему для зберігання всіх даних у проєкті, і вона повністю задовольнила поставлені вимоги як з технічного, так і з функціонального боку.

Проєктна база даних складається з чотирнадцяти таблиць, які логічно поділяються на кілька груп: таблиці для зберігання інформації про користувачів та їх ролі, таблиці для опису основних бізнес-сутностей (працівники, замовлення, матеріали, обладнання), таблиці для коментарів і статусів замовлень, а також таблиці для обліку фінансових транзакцій. Оскільки у застосунку передбачено автентифікацію користувачів, то частина таблиць була автоматично створена системою Identity, вбудованою у ASP.NET Core. Вони включають дані про користувачів, ролі, їхні логіни, токени, клейми та іншу службову інформацію, необхідну для безпечного управління доступом.

Фізична модель також охоплює сутності предметної області, що напряму пов'язані з логікою функціонування системи. Зокрема, це таблиці працівників, які

містять персональні дані, таблиці замовлень, що описують основні характеристики кожного замовлення, а також пов’язані з ними матеріали, обладнання, коментарі та статуси. Окремо реалізована таблиця канбан-дошок, що відображає структуру візуального управління замовленнями на інтерфейсі користувача [9].

Крім того, у фізичній моделі передбачено структуру для збереження інформації про фінансові операції — як завершені, так і заплановані. Це дозволяє враховувати витрати й надходження в межах кожного замовлення, що важливо для побудови звітності та аналізу рентабельності. Усі зв’язки між таблицями реалізовані за допомогою зовнішніх ключів, що гарантує цілісність даних. Наприклад, кожне замовлення може бути пов’язане з конкретним працівником, містити кілька коментарів, використовувати певні матеріали або обладнання, а також мати статус, що відповідає поточному етапу виконання.

Проєктування фізичної моделі виконувалося з дотриманням принципів нормалізації, що дозволило уникнути надлишковості даних і забезпечити зручну підтримку структури в майбутньому. Первинні ключі реалізовані у вигляді автоінкрементних цілочисельних ідентифікаторів. Сама база даних розроблена таким чином, щоб забезпечити масштабованість, що дає змогу при потребі розширити функціональність або додати нові сутності без порушення вже існуючої логіки.

1.3.3 Реалізація основних класів та методів

У даному підрозділі розглянемо реалізацію основних класів і методів, що забезпечують функціонування програмної системи.

На стороні серверної логіки основну роль відіграє клас OrderController, який відповідає за обробку замовлень: їх створення, оновлення, переміщення між

статусами, а також збереження відповідної інформації у базі даних. Реалізацію цього класу наведено на рис. 1.5.

```
[HttpGet]
0 references
public async Task<ActionResult<IEnumerable<OrderDTO>>> GetOrders()
{
    var orders = await _context.Orders
        .Include(o => o.Status)
        .Include(o => o.KanbanBoard)
        .Include(o => o.OrderComments)
        .Include(o => o.Employees)
        .Include(o => o.Equipments)
        .Include(o => o.Materials)
        .ToListAsync();

    var orderDTOs = orders.Select(MapToOrderDTO).ToList();
    return Ok(orderDTOs);
}

[HttpGet("{id}")]
1 reference
public async Task<ActionResult<OrderDTO>> GetOrder(int id)
{
    var order = await _context.Orders
        .Include(o => o.Status)
        .Include(o => o.KanbanBoard)
        .Include(o => o.OrderComments)
        .Include(o => o.Employees)
        .Include(o => o.Equipments)
        .Include(o => o.Materials)
        .FirstOrDefaultAsync(o => o.Id == id);

    if (order == null)
        return NotFound();

    return Ok(MapToOrderDTO(order));
}
```

Рисунок 1.6 – Код GET-запиту OrderController

Контролер OrderService обробляє HTTP-запити, що надходять з клієнтської частини. Він приймає запити на створення, редагування, переміщення замовлень і делегує відповідну логіку сервісному шару. Фрагмент реалізації контролера показано на рис. 1.6.

```typescript
@Injectable({providedIn: 'root'})
export class OrderService {
  private apiUrl = 'https://localhost:44394/api/orders';
  private statusUrl = 'https://localhost:44394/api/OrderStatus';
  private boardUrl = 'https://localhost:44394/api/KanbanBoards';

  constructor(private http: HttpClient) {
  }

  getOrdersByBoard(boardId: number): Observable<Order[]> {
    return this.http.get<Order[]>(`${this.apiUrl}/board/${boardId}`);
  }

  addOrder(order: Order): Observable<Order> {
    return this.http.post<Order>(this.apiUrl, order);
  }

  deleteOrder(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }

  updateOrderStatus(orderId: number, newStatusId: number): Observable<void> {
    return this.http.put<void>(`${this.apiUrl}/${orderId}/status/${newStatusId}`, {});
  }

  getStatusesByBoard(boardId: number): Observable<OrderStatus[]> {
    return this.http.get<OrderStatus[]>(`${this.statusUrl}/board/${boardId}`);
  }
}
```

Рисунок 1.6 – Код OrderService

На стороні клієнта головним компонентом є OrderBoardComponent, реалізований з використанням Angular. Він відповідає за візуалізацію канбан-дошки, а також за обробку drag-and-drop операцій та взаємодію з API. Реалізацію компонента показано на рис. 1.7, рис. 1.8.

```html
<div class="toolbar flex items-center gap-4 px-6 py-3 bg-white border-b border-gray-200 shadow-sm">
  <div class="flex items-center gap-2">
    <label class="text-sm font-medium text-gray-600" for="boardSelect">Борда:</label>
    <select
      (change)="onBoardChange()"
      [(ngModel)]="selectedBoardId"
      class="border border-gray-300 text-sm rounded-md px-3 py-1.5 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
      id="boardSelect"
    >
      <option *ngFor="let board of boards" [value]="board.id">{{ board.name }}</option>
    </select>
  </div>

  <button
    (click)="addStatusToCurrentBoard()"
    class="flex items-center gap-2 bg-blue-600 hover:bg-blue-700 text-white text-sm font-medium px-4 py-2 rounded-md shadow-sm transition"
  >
    <i class="lucide lucide-plus"></i>
    Додати статус
  </button>

  <button
    (click)="addNewBoard()"
    class="flex items-center gap-2 bg-green-600 hover:bg-green-700 text-white text-sm font-medium px-4 py-2 rounded-md shadow-sm transition"
  >
    <i class="lucide lucide-layout-dashboard"></i>
    Нова борда
  </button>
</div>
```

Рисунок 1.7 – Код HTML-реалізація Kanban-дошки

```typescript
loadBoardData(): void {
  if (!this.selectedBoardId) return;

  this.orderService.getStatusesByBoard(this.selectedBoardId).subscribe({
    next: (statuses) => {
      this.statuses = statuses;
      this.connectedDropLists = statuses.map(s => `status-${s.id}`);

      this.orderService.getOrdersByBoard(this.selectedBoardId).subscribe({
        next: (orders) => {
          this.orders = orders.filter(order =>
            statuses.some(s => s.id === order.orderStatusId)
          );
          this.groupOrdersByStatus();
        },
        error: (err) => console.error('Помилка завантаження замовлень:', err),
      });
    },
    error: (err) => console.error('Помилка завантаження статусів:', err),
  });
}
```

Рисунок 1.8 – Код функції для завантаження замовлень для Kanban-дошки

Окремий компонент OrderDialogComponent відкривається у вигляді модального вікна для створення або редагування замовлення. У ньому реалізовано реактивну форму, яка дозволяє вводити назву, опис, пріоритет, список працівників, матеріалів і обладнання. Приклад реалізації цього компонента представлено на рис. 1.9.

```
onSubmit(): void {
  if (this.orderForm.valid) {
    this.isLoading = true;
    const newOrder: Order = {
      id: 0,
      boardId: this.data.boardId,
      orderStatusId: this.data.orderStatusId,
      name: this.orderForm.get('name')?.value,
      address: this.orderForm.get('address')?.value,
      totalPrice: this.orderForm.get('totalPrice')?.value,
      discount: this.orderForm.get('discount')?.value,
      paidAmount: this.orderForm.get('paidAmount')?.value,
      remainingAmount: this.orderForm.get('remainingAmount')?.value
    };

    this.orderService.addOrder(newOrder).subscribe({
      next: (createdOrder) => {
        this.isLoading = false;
        this.snackBar.open('Замовлення успішно створено!', 'OK', {duration: 3000});
        this.dialogRef.close(createdOrder);
      },
      error: (err) => {
        this.isLoading = false;
        this.snackBar.open('Помилка створення замовлення', 'OK', {duration: 3000});
        console.error('Помилка створення ордеру:', err);
      }
    });
  }
}
```

Рисунок 1.9 – Код функції для додавання нового замовлення для Kanban-дошки

1.4. Використання програмної системи

Цей розділ описує практичні аспекти використання розробленої програмної системи, зокрема процес її розгортання, системні вимоги, типові сценарії роботи користувача та результати верифікації. Основна мета — продемонструвати, що система не лише реалізована згідно з технічним завданням, але й готова до ефективного використання кінцевими користувачами в реальних умовах. У

підрозділах буде наведено інструкції для налаштування та запуску проєкту, описано логіку користування ключовими функціями та перевірено відповідність поведінки системи очікуваним результатам.

1.4.1 Розгортання програмної системи та системні вимоги

Для того щоб запустити програмну систему, спочатку потрібно встановити всі необхідні компоненти. На стороні бекенду використовувався .NET 8, тому потрібен встановлений .NET SDK відповідної версії. Також необхідна база даних, яка в моєму випадку реалізована через Microsoft SQL Server. Для збереження налаштувань підключення до бази використовується файл конфігурації appsettings.json.

Фронтенд частина проєкту реалізована на Angular 17, тому потрібно мати встановлені Node.js (версії не нижче 18) та Angular CLI. Для запуску фронтенду достатньо виконати npm install, а потім ng serve [11].

Проєкт можна запускати локально або розгортати на сервері. Для локального запуску обидві частини запускаються окремо: серверна частина через Visual Studio або командою dotnet run, а клієнтська — через Angular CLI.

Мінімальні системні вимоги для розгортання:

Windows 10 або новіша (або Linux із відповідними компонентами)

.NET SDK 8.0

SQL Server або SQL Server Express

Node.js 18+

4 ГБ оперативної пам’яті

Браузер Google Chrome або аналогічний сучасний

Таким чином, для роботи з проєктом потрібно мати базові знання .NET, Angular і вміти працювати з базами даних. Усе необхідне програмне забезпечення є безкоштовним і доступним для скачування [16].

1.4.2 Опис типових схем використання системи

Система призначена для керування замовленнями в організації, тому більшість сценаріїв використання пов’язані з додаванням, переглядом, редагуванням або переміщенням замовлень між статусами. Найзручніше це робити через канбан-дошку, яка доступна на головній сторінці після авторизації.

Типовий сценарій виглядає так: користувач відкриває головну сторінку, бачить усі замовлення, розподілені за статусами. Якщо потрібно створити нове замовлення — натискає відповідну кнопку, заповнює поля у формі (назва, опис, пріоритет, матеріали, працівники, обладнання тощо) і підтверджує дію. Нове замовлення з’являється в колонці "Нові".

Під час виконання замовлення його можна перетягнути в колонку "У процесі", а коли завершене — в "Завершені". Усі зміни автоматично надсилаються на сервер і зберігаються в базі даних. Також є можливість переглядати або змінювати список працівників, матеріалів і обладнання, які використовуються для кожного замовлення.

На наступній діаграмі діяльності зображено типовий потік дій користувача під час роботи із замовленнями у системі. Вона відображає основні стани замовлення (створення, редагування, зміна статусу).

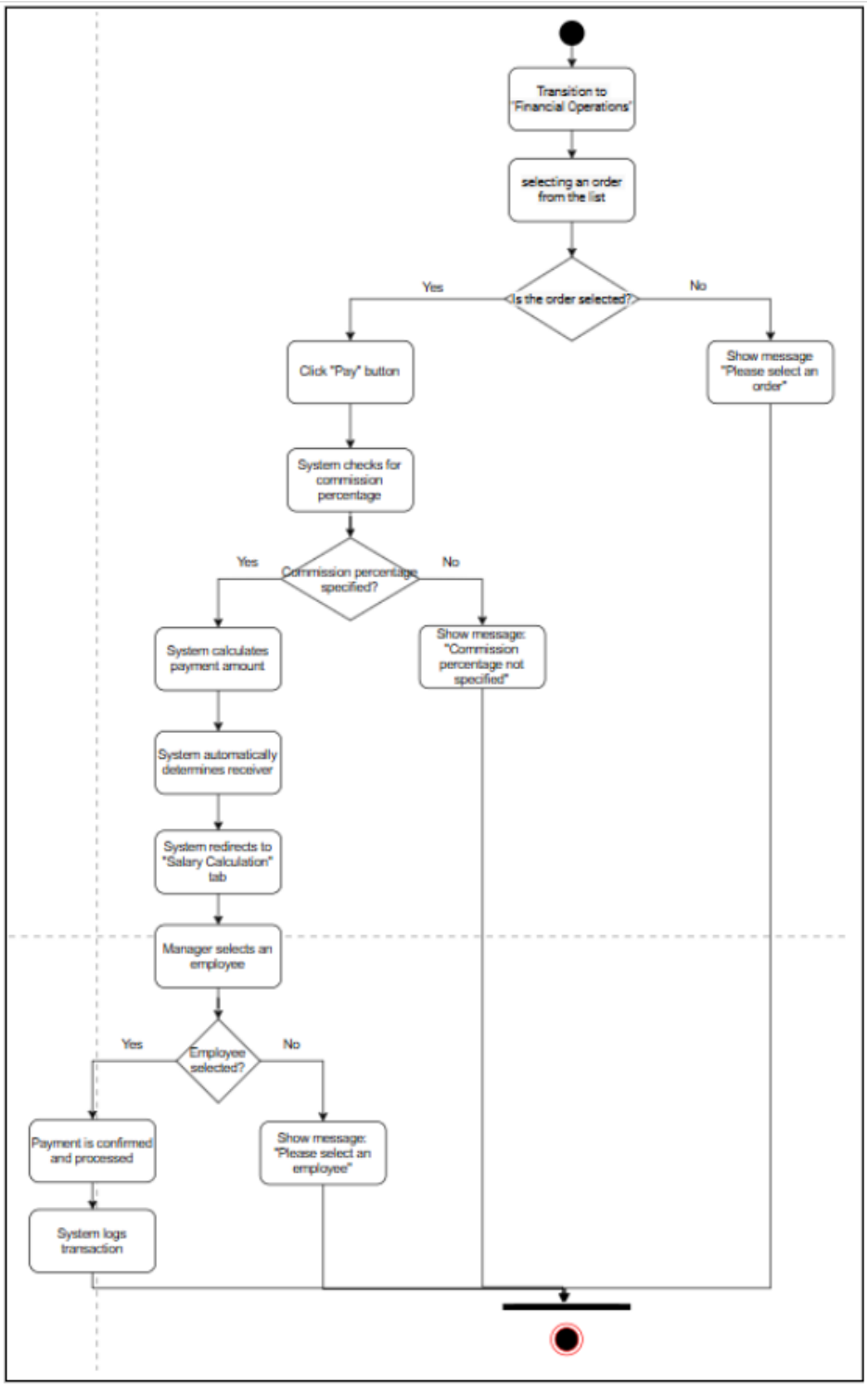

Рисунок 1.10 – Діаграма діяльності для обрахування зарплати

Наступна діаграма послідовності ілюструє процес взаємодії між чотирма основними компонентами інформаційної системи — користувачем (менеджером), клієнтським інтерфейсом (Frontend), серверною логікою (Backend) та базою даних (Database) — під час створення, доповнення та завершення замовлення. Менеджер

ініціює створення замовлення через клієнтський інтерфейс, який надсилає відповідний запит до серверної частини, де здійснюється збереження даних у базі даних із подальшим поверненням підтвердження успішного створення. Наступним кроком є передача матеріалів, які також обробляються через фронтенд та бекенд і зберігаються у базі даних із підтвердженням результату. Завершення процесу полягає в оновленні статусу замовлення до стану "завершено", що реалізується аналогічним чином через зміну даних у базі. Дана модель демонструє стандартизовану послідовність дій, що забезпечує цілісність та узгодженість інформаційних потоків між користувачем і всіма складовими системи.

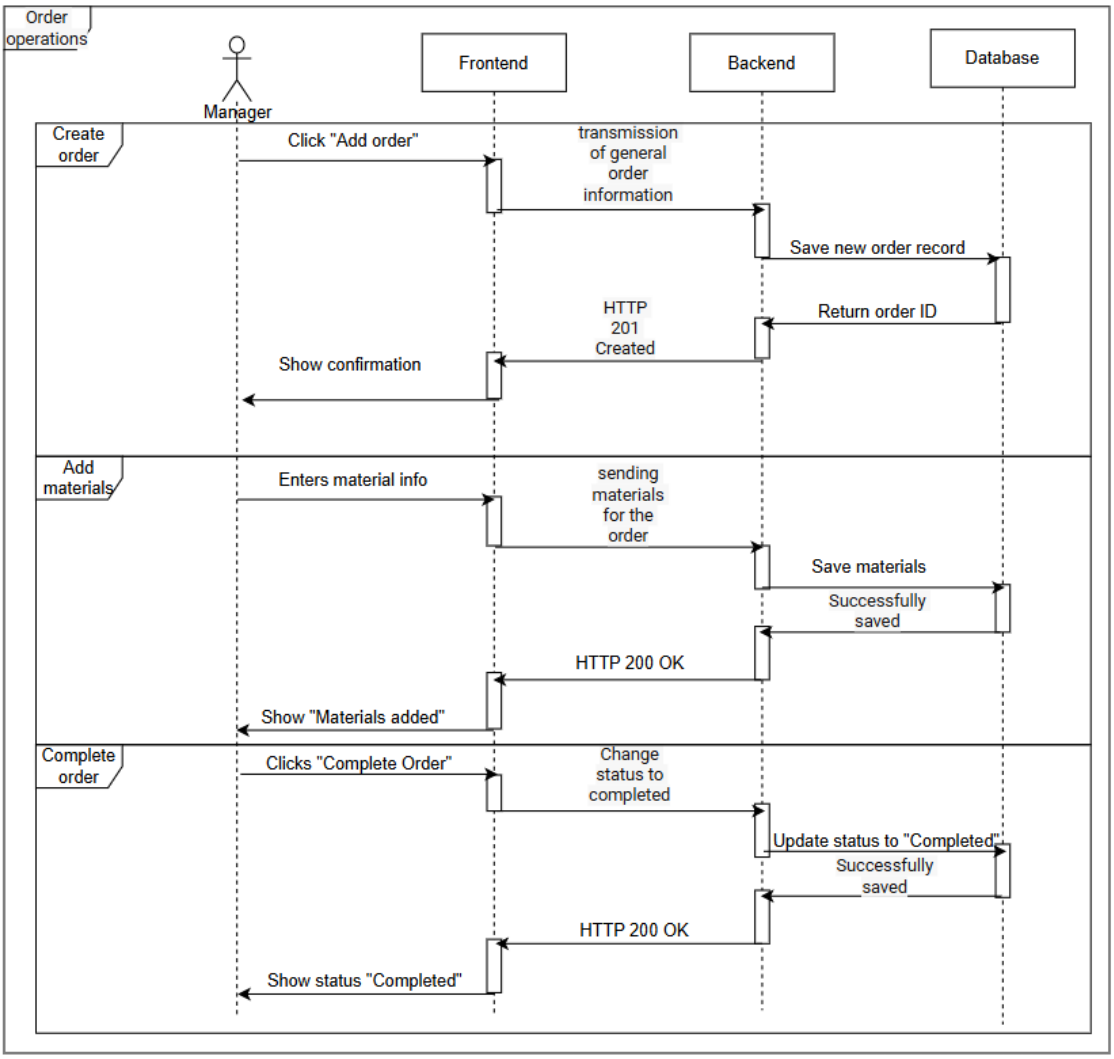

Рисунок 1.11 – Діаграма послідовності для операцій над замовленням

1.4.3 Верифікація програмної системи

Після завершення реалізації програмної системи було проведено її верифікацію — тобто перевірку, чи працює вона так, як задумувалося на етапі проєктування. Основна мета — переконатися, що всі основні функції працюють правильно, стабільно і без критичних помилок.

Перевірка відбувалась на основі заздалегідь визначених сценаріїв використання. Зокрема, було протестовано додавання, редагування, видалення та перегляд замовлень через канбан-дошку. Особливу увагу приділялось перетягуванню замовлень між різними статусами, як-от "Нові", "У процесі", "Завершені". Під час тестування спостерігалося, чи надсилається відповідний запит до сервера, чи змінюється статус, і чи оновлюється інтерфейс без помилок.

Також перевірялась поведінка системи в ситуаціях, коли користувач вводить некоректні або неповні дані. У таких випадках система правильно повідомляє про помилку й не дозволяє завершити дію. Важливо, що компоненти інтерфейсу коректно реагують на успішні й неуспішні дії, показуючи відповідні повідомлення.

На основі результатів верифікації можна зробити висновок, що програмна система реалізована відповідно до технічного завдання, а її функціональність відповідає очікуванням. Система готова до подальшого використання кінцевими користувачами.

2. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 План тестування

Тестування бекенду проводиться з метою перевірки коректної реалізації бізнес-логіки, стабільності REST API, обробки помилок, валідації вхідних даних та взаємодії з базою даних. Основними об’єктами тестування є REST-ендпоінти /api/Orders, /api/Employees, /api/Equipments, /api/Materials, /api/PendingTransactions, /api/OrderComments, для яких перевіряється повний набір CRUD-операцій. Тестуються як валідні сценарії , так і невалідні. Особливу увагу приділено відповідності HTTP-статусів, форматів відповідей, коректній роботі валідації та обробці виняткових ситуацій. Тестування виконується вручну через Postman.

Таблиця 2.1 – Тест-кейси для тестування системи

ID	Тип	Опис	Очікуваний результат
1	positive	Перевірка отримання списку всіх замовлень через GET /api/Orders	JSON-масив усіх замовлень, статус 200 OK
2	positive	Отримання одного замовлення за ID через GET /api/Orders/{id}	JSON-об’єкт замовлення з відповідним ID, статус 200 OK
3	negative	Спроба отримати замовлення з неіснуючим ID	Статус 404 Not Found, повідомлення про помилку
4	positive	Створення замовлення з валідними даними через POST /api/Orders	Статус 201 Created, повертається новостворене замовлення
5	negative	Створення замовлення з невалідними даними (відсутнє поле)	Статус 400 Bad Request, повідомлення про помилку
6	positive	Редагування замовлення через PUT /api/Orders/{id}	Статус 200 OK, повертається оновлене замовлення
7	negative	Спроба редагувати неіснуюче замовлення	Статус 404 Not Found, повідомлення про помилку

Продовження таблиці 2.1

8	positive	Видалення замовлення через DELETE /api/Orders/{id}	Статус 204 No Content
9	positive	Отримання списку працівників через GET /api/Employees	Список працівників, статус 200 OK
10	positive	Створення працівника з валідними даними через POST /api/Employees	Статус 201 Created, JSON-об’єкт нового працівника
11	negative	Спроба створити працівника без імені	Статус 400 Bad Request, повідомлення про помилку
12	positive	CRUD-операції для /api/Equipments (створення, читання, оновлення)	Відповідні статуси: 200/201/204 залежно від запиту
13	positive	CRUD-операції для /api/Materials	Успішне виконання всіх операцій, статуси 200/201/204
14	positive	Створення фінансової операції через POST /api/PendingTransactions	Статус 201 Created, JSON об’єкт транзакції
15	negative	Спроба створити транзакцію з некоректною сумою	Статус 400 Bad Request, повідомлення про помилку
16	positive	Створення коментаря до замовлення /api/OrderComments	Статус 201 Created, JSON коментар
17	negative	Створення коментаря без вмісту	Статус 400 Bad Request, повідомлення про помилку

Тестування фронтенду зосереджене на перевірці коректної роботи Angular-компонентів, які відповідають за відображення та керування сутностями, що надходять з бекенду через REST API. Перевіряється правильність відображення списків, функціональність форм додавання та редагування, стабільність інтерфейсу при зміні даних, обробка помилок, реакція на порожні або невалідні відповіді від сервера. Також перевіряється, чи коректно компоненти взаємодіють із сервісами, як змінюється інтерфейс після виконання операцій, і чи відображаються повідомлення про помилки або успіхи. Основна увага приділяється зручності взаємодії для користувача та відповідності поведінки очікуваним сценаріям [15].

2.2 Розробка тестів

Тестування проводилось вручну в Postman для перевірки основної роботи REST API, яке розгорнуте у проєкті. Було перевірено CRUD-операції для кількох сутностей, таких як замовлення, співробітники, обладнання, матеріали, коментарі до замовлень та фінансові транзакції. Нижче наведено опис проведених мною тестів.

1. Тестування GET /api/Orders – отримання всіх замовлень

За допомогою Postman було виконано GET-запит за адресою https://localhost:44394/api/Orders.

Отримано відповідь зі статусом 200 OK, у відповіді був масив із замовленнями. Це означає, що запит обробляється правильно.

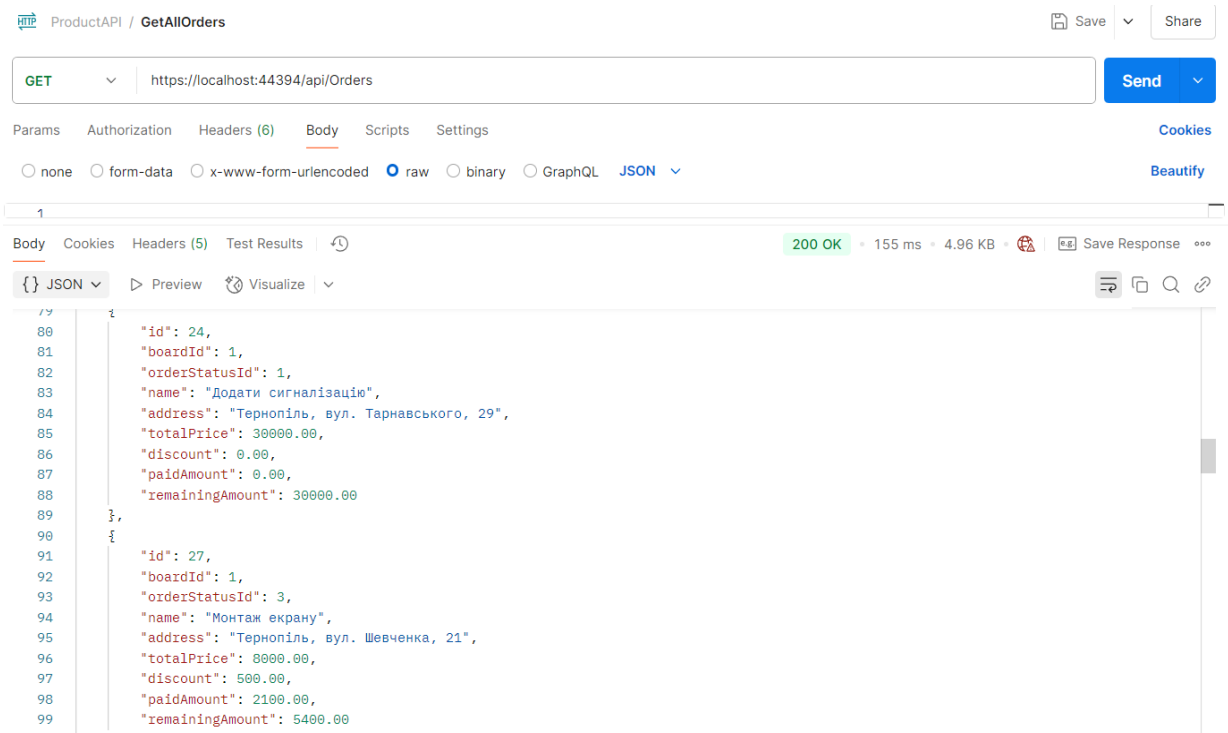

Рисунок 2.1 – Результат запиту на отримання всіх ордерів

2. Тестування GET /api/Orders/{id} – отримання одного замовлення

Використано ідентифікатор замовлення, який отримав у попередньому запиті, наприклад https://localhost:44394/api/Orders/24.

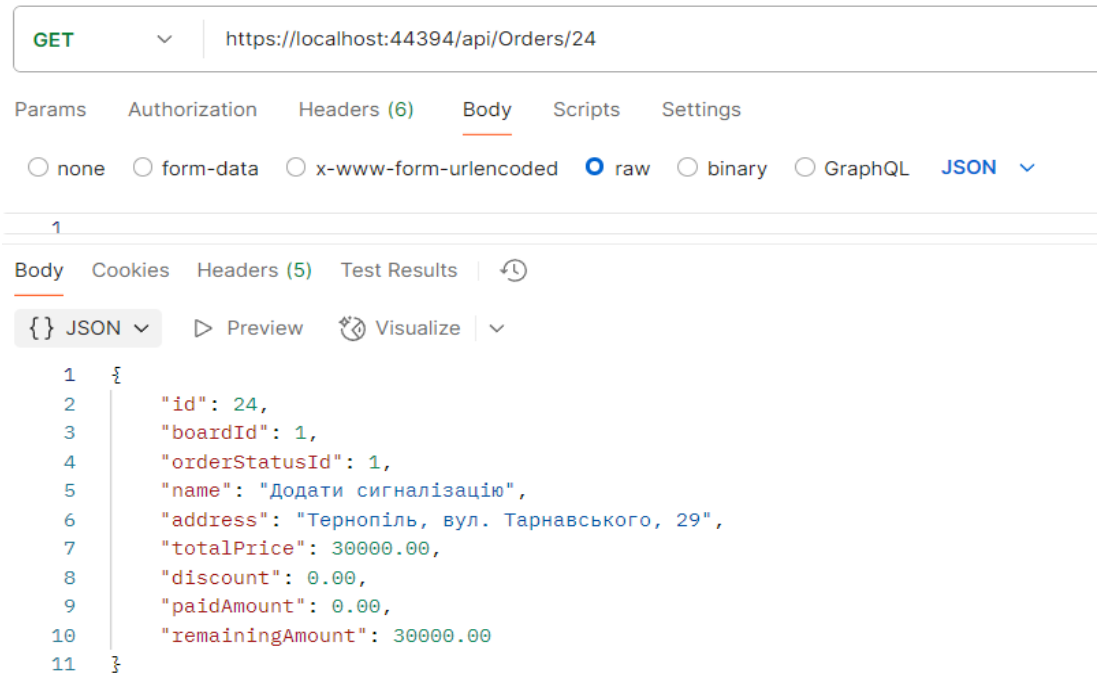

Рисунок 2.2 – Результат запиту на отримання ордера з певним id

Якщо ID існує – повертається об'єкт замовлення та статус 200 OK. Якщо вказується неіснуючий ID (наприклад, 9999) – отримано 404 Not Found.

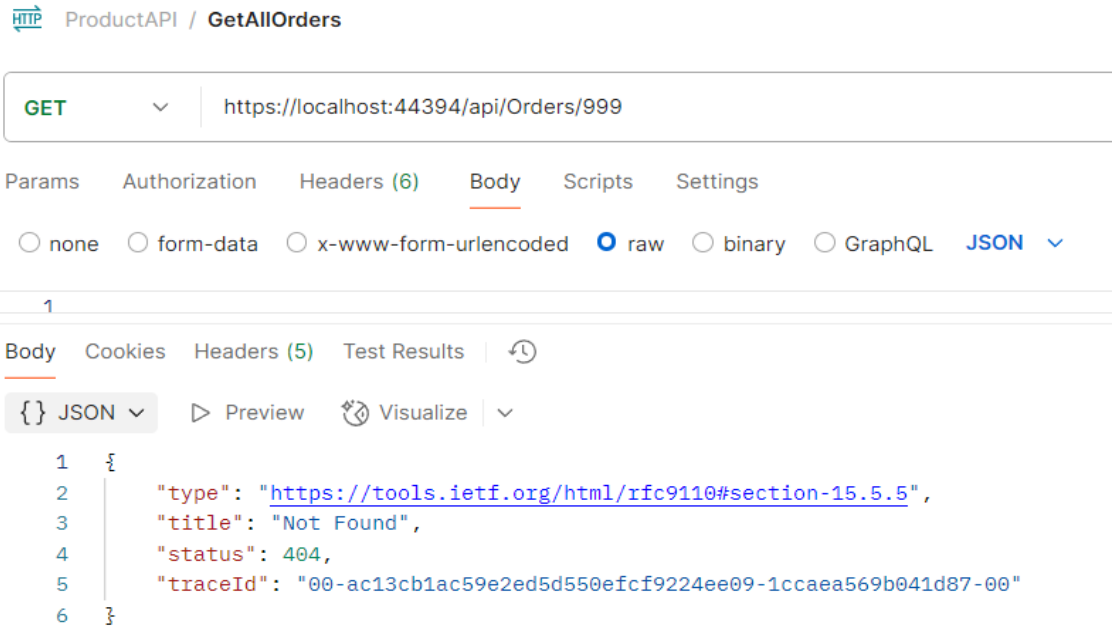

Рисунок 2.3 – Результат запиту на отримання ордера з неіснуючим id

3. Тестування POST /api/Orders – створення нового замовлення

У Postman вибрано метод POST, URL – https://localhost:44394/api/Orders. У вкладці Body вибрав формат JSON і введено валідні дані:

```
{
  "boardId": 67,
  "orderStatusId": 60,
  "name": "Монтаж екрану",
  "address": "м. Тернопіль, вул. Тарнавського, 28",
  "totalPrice": 30000,
  "discount": 5000,
  "paidAmount": 10000
}
```

Після натискання Send отримано відповідь 201 Created і в тілі відповіді – нове замовлення з присвоєним ID.

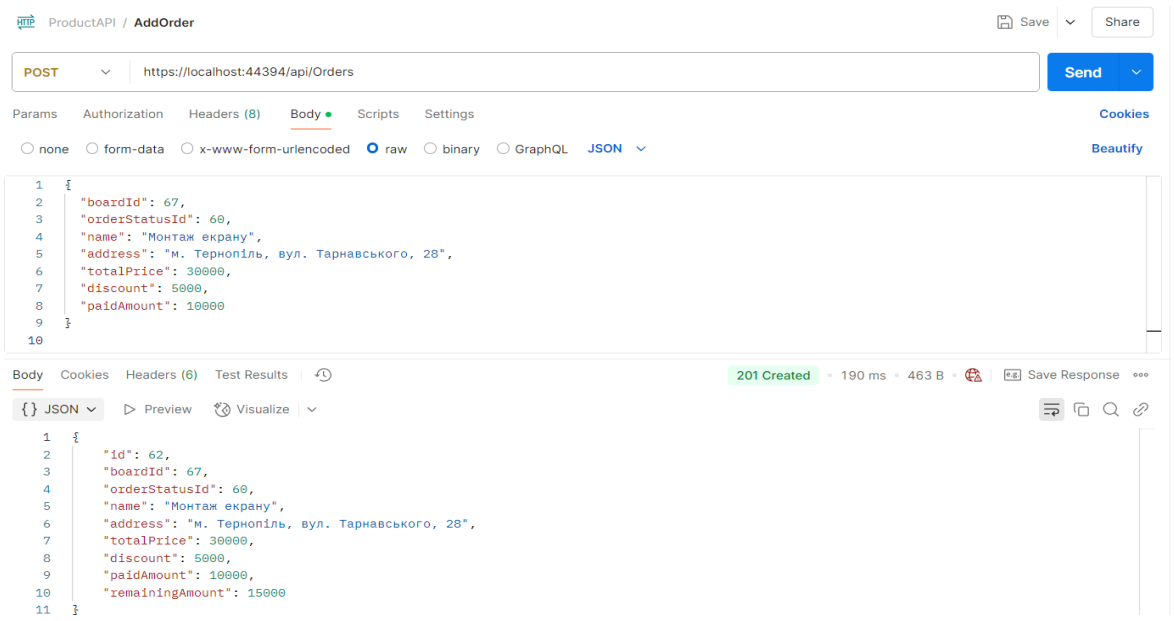

Рисунок 2.4 – Результат запиту на створення нового замовлення

4. Тестування POST /api/Orders – створення з помилкою

У цьому тесті було навмисно надіслано порожній об'єкт. Сервер відповів 400 Bad Request, з повідомленням про відсутність обов'язкових полів. Це значить, що валідація на бекенді працює правильно.

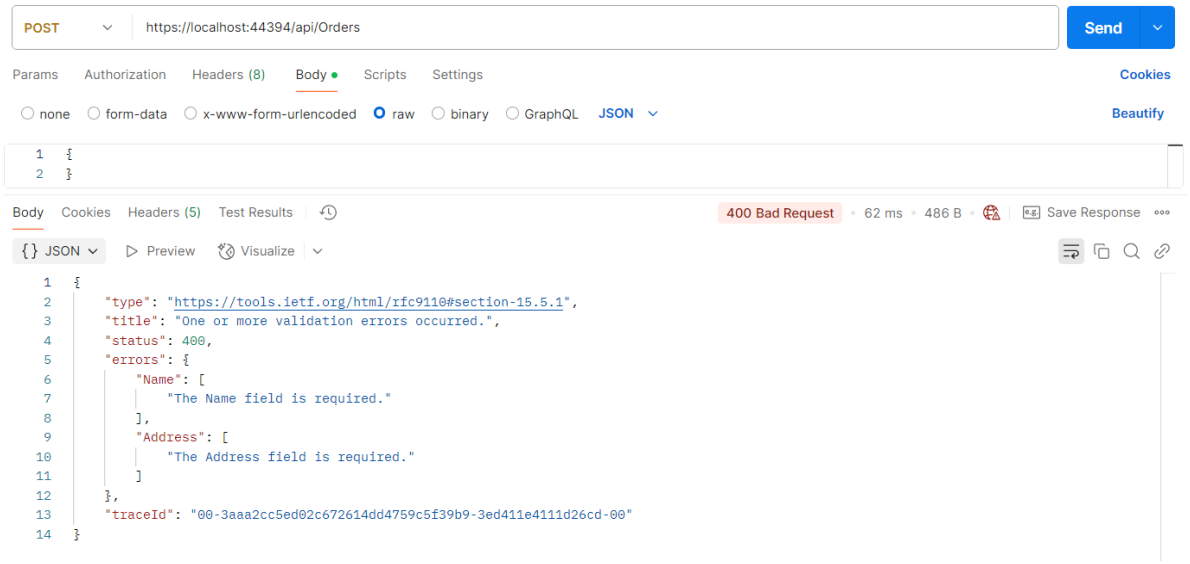

Рисунок 2.5 – Результат запиту на створення нового замовлення з пустим body

5. Тестування DELETE /api/Orders/{id} – видалення замовлення

Скопіював існуючий ID і надіслав DELETE-запит на https://localhost:44394/api/Orders/1.

Отримав статус 204 No Content. Після цього перевірив GET-запитом, що замовлення дійсно більше не існує (отримав 404).

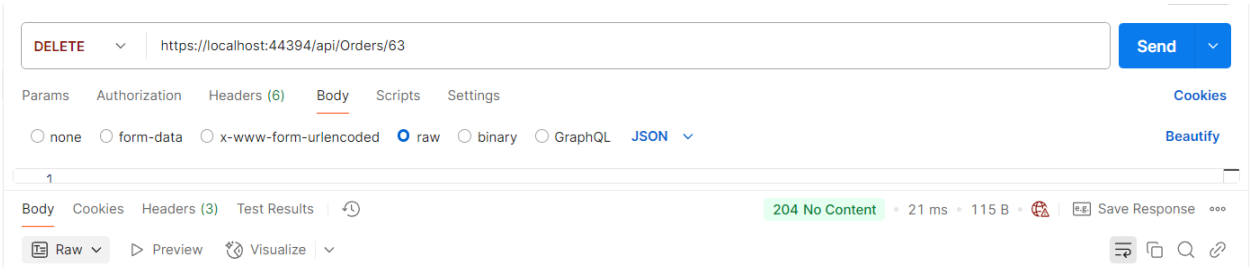

Рисунок 2.6 – Результат запиту на видалення замовлення

6. Аналогічні тести були виконані для:

/api/Employees, /api/Equipments, /api/Materials, /api/PendingTransactions,

/api/OrderComments

У кожному випадку було перевірено ті самі дії: перегляд усіх елементів, отримання одного за ID, створення, редагування і видалення. Також спеціально було вводено помилкові дані, щоб перевірити обробку помилок.

Отже, тестування показало, що бекенд працює стабільно. CRUD-операції виконуються правильно, статуси відповідають очікуваним, помилки обробляються адекватно. Валідація даних працює, запити з некоректними даними не проходять. При помилках сервер повертає зрозумілі повідомлення.

Було відкрито головну сторінку, де відображається канбан-дошка із замовленнями. Було перевірено через вкладку DevTools → Network, що за запитом /api/Orders повертається коректна відповідь у форматі JSON зі статусом 200 OK. Було переконанося, що всі стовпці (Нові, У процесі, Завершені) коректно відображаються на сторінці без порушення структури інтерфейсу. Було зафіксовано, що в кожній колонці відображаються відповідні картки замовлень, при цьому не виявлено жодних помилок або накладань елементів у UI.

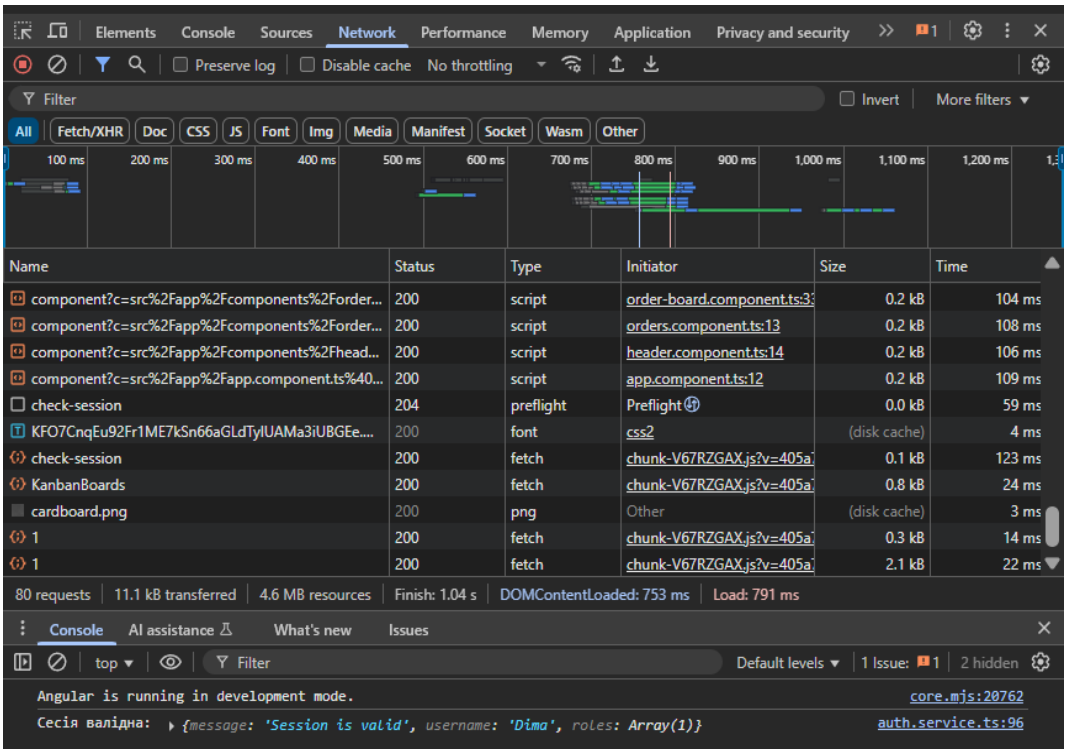

Рисунок 2.7 – Відображення канбан-дошки після завантаження сторінки у вікні Network

Було здійснено перетягування картки із колонки "У процесі" до колонки "Завершені". Було перевірено, що після цієї дії надсилається PUT-запит зі зміненим статусом замовлення, зокрема статус змінюється на "completed". Було зафіксовано, що сервер повернув відповідь зі статусом 200 OK, після чого інтерфейс автоматично оновився. Було переконанося, що картка зникла з попередньої колонки та з’явилась у новій, дублювання або помилок відображення не відбулося.

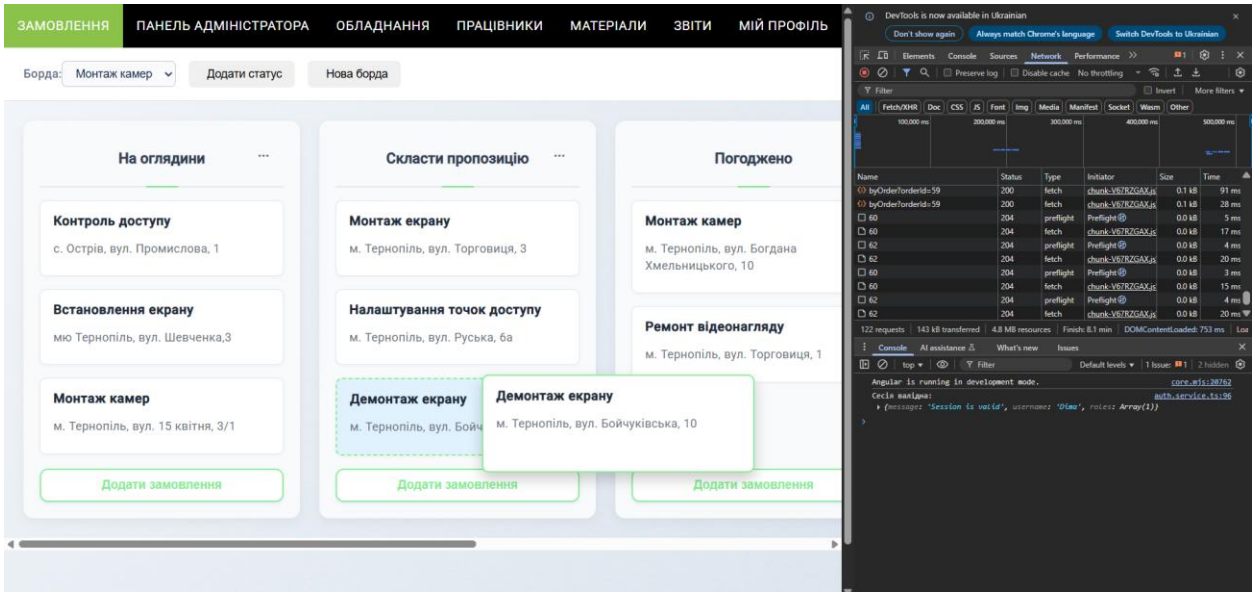

Рисунок 2.8 – Перетягування картки з колонки "Скласти пропозицію" до "Погоджено"

Було перевірено ситуацію з некоректним перетягуванням картки за межі канбан-дошки. Картку було спробувано перемістити в порожній простір поза колонки, у результаті чого вона повернулась на своє початкове місце. Було підтверджено, що при цьому не відправляється жодного запиту до сервера, а отже система правильно відкидає невалідні зміни й захищає дані від неконсистентного стану.

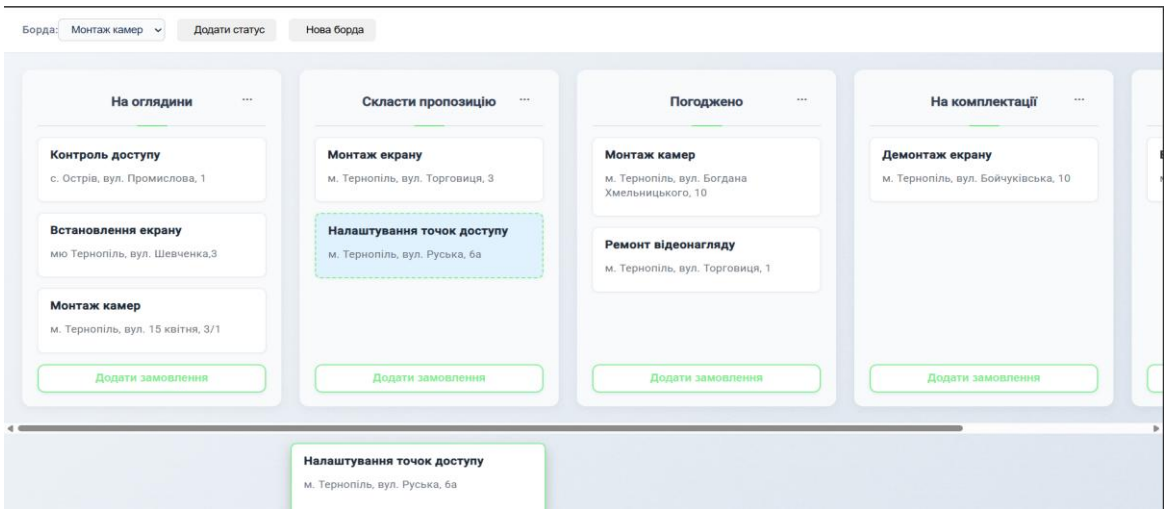

Рисунок 2.9 – Обробка некоректного перетягування картки за межі дошки

Було натиснуто кнопку “Додати статус” з метою додавання нового статусу для замовлень. У модальному вікні було введено назву нової колонки, “On Hold”. Після підтвердження було надіслано POST-запит на відповідний ендпоінт /api/OrderStatuses. Нова колонка з’явилась на дошці, її можна використовувати для перетягування карток. Також було перевірено поведінку при некоректному введенні, наприклад, при відправці форми з порожньою назвою — система видала повідомлення про помилку, що підтверджує правильну реалізацію валідації.

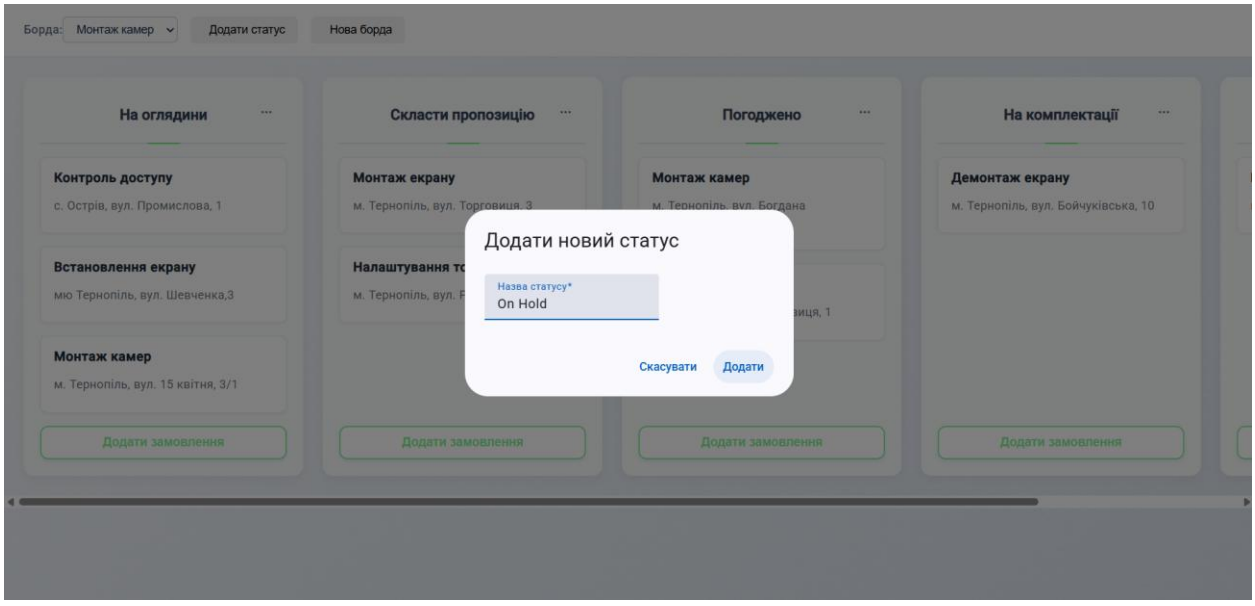

Рисунок 2.10 – Додавання нової колонки зі статусом "On Hold"

Було перевірено функцію видалення колонки з канбан-дошки. На прикладі щойно створеної колонки “On Hold” було натиснуто на значок видалення. З’явилось підтвердження операції, після погодження було надіслано запит на сервер. Було зафіксовано, що після отримання відповіді зі статусом 200 OK колонка зникла з інтерфейсу, а відповідний запис було видалено з бази. Інші елементи залишились стабільними, і жодних візуальних чи функціональних помилок не виникло.

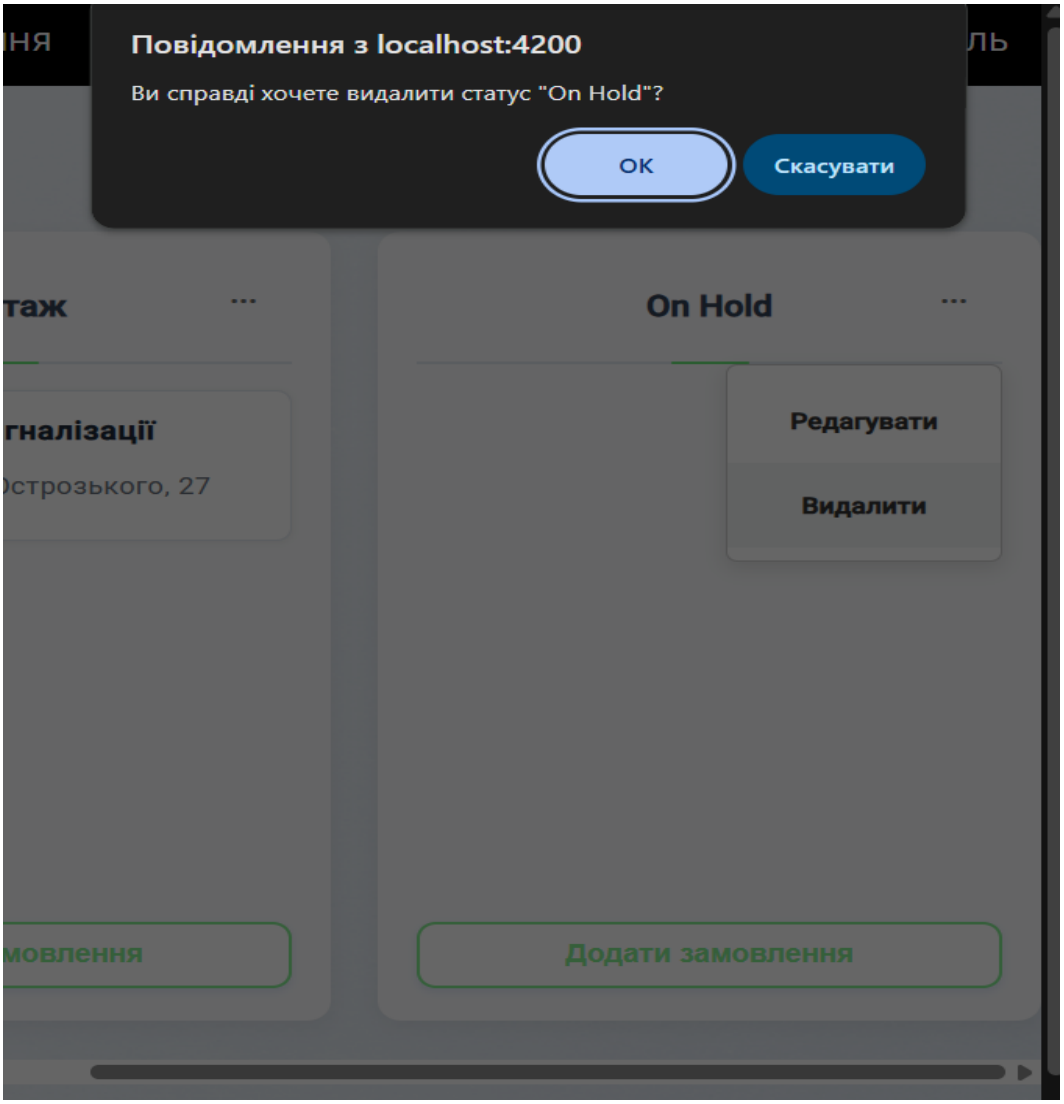

Рисунок 2.11 – Видалення доданої колонки зі статусом "On Hold"

3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1. Долікарська допомога при харчових отруєннях

Харчові отруєння — це гострі патологічні стани, які виникають внаслідок споживання їжі або напоїв, що містять біологічні чи хімічні токсиканти. Вони можуть бути спричинені патогенними мікроорганізмами, бактеріальними токсинами, вірусами, грибами, пестицидами, важкими металами, алергенами та іншими речовинами, які при потраплянні до організму порушують його життєво важливі функції [22]. Класифікація харчових отруєнь ґрунтується на етіологічному чиннику і включає: бактеріальні (сальмонельоз, ботулізм, стафілококові токсикоінфекції), вірусні (ротавірус, норовірус), грибкові (отруєння блідою поганкою, мухоморами), хімічні (нітрати, солі важких металів, миючі засоби, пестициди, побутова хімія), а також харчові алергії та інтолерантності (реакції на білки, глютен, горіхи, лактозу) [22, 26].

Клінічна картина харчових отруєнь може мати значну варіативність залежно від типу токсиканта, дози, індивідуальних особливостей організму та швидкості надання допомоги. Типовими симптомами є нудота, багаторазова блювота, діарея, спазматичний біль у животі, підвищення температури тіла, загальна слабкість, головний біль, ознаки інтоксикації й дегідратації. Залежно від інтенсивності симптомів виділяють три ступені важкості: легкий ступінь — одноразове блювання, короткочасна діарея, відсутність загальної інтоксикації; середній ступінь — повторне блювання, рясна водяниста діарея, температура до 38,5 °C, ознаки помірного зневоднення; важкий ступінь — неконтрольоване блювання, діарея до 10–15 разів на добу, гіпертермія понад 39 °C, порушення свідомості, судоми, важке зневоднення, шок [26].

Тривалість непрацездатності безпосередньо пов’язана з важкістю клінічного перебігу. У разі легкого отруєння зазвичай достатньо 2–3 днів амбулаторного лікування та відпочинку. При середньому ступені хворий потребує щонайменше

5–7 днів відновлення. У разі важкого отруєння необхідна госпіталізація, інтенсивна терапія та подальша реабілітація протягом 10–14 днів і більше [24].

Особливості надання долікарської допомоги залежать від характеру отруєння. При бактеріальних отруєннях доцільне промивання шлунка теплою водою або слабким розчином перманганату калію (тільки при збереженій свідомості), прийом сорбентів (активоване вугілля, ентеросгель, смекта), обов'язкова регідратація (Регідрон, фізіологічний розчин, вода з сіллю та цукром), спокій. При грибкових отруєннях слід якомога раніше викликати швидку допомогу, промивання можливе тільки за вказівкою лікаря, оскільки багато грибів мають повільну дію і специфічні антидоти. При хімічних отруєннях категорично забороняється викликати блювання, особливо при отруєнні кислотами, лугами, побутовою хімією, нафтопродуктами; необхідне негайне припинення контакту з токсичною речовиною, зрошення ротової порожнини водою, та обов'язковий виклик швидкої. При алергічних отруєннях слід дати антигістамінні препарати (лоратадин, димедрол), за наявності анафілактичної реакції — ввести адреналін та викликати швидку [25].

Алгоритм надання долікарської допомоги: припинити надходження токсину (відмовитись від підозрілої їжі, напоїв, продуктів), при збереженій свідомості — промити шлунок теплою водою (1–1,5 літра), дати сорбенти (вугілля 1 г/кг маси тіла), забезпечити регідратацію (розчини солі й цукру дрібними ковтками кожні 5–10 хвилин), надати спокій, забезпечити доступ повітря, викликати швидку допомогу. У випадках отруєння дітей, вагітних, осіб похилого віку чи за наявності тяжких симптомів — транспортувати до медичного закладу якнайшвидше [23].

3.2. Заходи з техніки безпеки при експлуатації обладнання

Техніка безпеки — це складова частина загальної системи охорони праці, що охоплює організаційні, технічні, індивідуальні та додаткові заходи,

спрямовані на попередження травматизму, аварій, нещасних випадків і професійних захворювань у процесі трудової діяльності [26]. Її основною метою є створення безпечних і здорових умов праці, мінімізація ризиків, пов’язаних із використанням машин, механізмів, електрообладнання та інших потенційно небезпечних засобів виробництва. Забезпечення безпеки працівників при експлуатації обладнання вимагає системного підходу, де всі заходи тісно взаємопов’язані та доповнюють одне одного [21].

Технічні заходи — це перш за все заходи інженерного характеру, які безпосередньо впливають на технічний стан обладнання та робочого середовища. До таких заходів належать: технічний контроль справності обладнання, своєчасна діагностика і профілактичне обслуговування машин і механізмів, використання надійних захисних кожухів, екранів, огороджень на рухомих частинах. Обов’язковим є заземлення електроустановок, встановлення автоматичних систем аварійного відключення у випадку перегріву, короткого замикання або перевантаження. Особливе значення мають системи примусової вентиляції для запобігання накопиченню шкідливих речовин, належне локалізоване освітлення робочих зон, захист від шуму й вібрації, а також встановлення датчиків контролю параметрів середовища. Важливо, щоб обладнання було оснащене системами блокування — наприклад, запуск пристрою неможливий при відкритому люку або наявності стороннього предмета у робочій зоні [23].

Організаційні заходи — це управлінські дії, які регламентують порядок виконання робіт та забезпечують систематичний контроль за дотриманням правил безпеки. До них належать розроблення, затвердження та впровадження інструкцій з охорони праці за кожним видом робіт, складання посадових інструкцій, визначення зон ризику. Усі працівники повинні проходити відповідні види інструктажу: вступний (при прийомі на роботу), первинний (безпосередньо на робочому місці), повторний (раз на пів року), позаплановий (при зміні умов праці чи впровадженні нового обладнання) і цільовий (перед виконанням небезпечних або рідкісних робіт). Працівники, які обслуговують складне обладнання, повинні бути офіційно призначені наказом керівництва та пройти відповідне навчання з

перевіркою знань. Організаційні заходи включають ведення облікової документації, журналів інструктажів, актів перевірок, протоколів тестування, а також планування заходів контролю та нагляду за умовами праці [26].

Індивідуальні заходи — це комплекс дій, спрямованих на забезпечення особистого захисту працівника. Основним засобом реалізації таких заходів є засоби індивідуального захисту (ЗІЗ), до яких належать: каски, захисні окуляри, респіратори, протигази, шумозахисні навушники, рукавиці, спецодяг, термостійке або кислотостійке взуття. Працівники мають бути навчено правильно використовувати ЗІЗ, здійснювати їх регулярну перевірку, догляд, заміну у разі пошкодження чи втрати захисних властивостей. Важливою складовою є ергономічне облаштування робочого місця — з урахуванням антропометричних особливостей працівника, забезпечення правильної робочої пози, освітлення, температурного режиму та рівня шуму. Окрему увагу слід приділити психофізіологічним факторам: втомі, монотонності, стресу, що можуть знижувати концентрацію уваги й підвищувати ризик нещасного випадку [21, 25].

Додаткові заходи мають на меті посилення ефективності системи техніки безпеки. Вони включають: розміщення попереджувальних і заборонних знаків на видимих місцях, розроблення евакуаційних схем, планів дій у разі аварій, пожеж, витоку токсичних речовин; наявність аптечок першої допомоги та справних вогнегасників; проведення тренувальних евакуацій; впровадження систем відеоспостереження й доступу; психофізіологічне тестування персоналу на придатність до роботи з підвищеною небезпекою. Доцільним є також впровадження систем управління ризиками, які дозволяють ідентифікувати, оцінити та контролювати потенційні загрози на виробництві [24].

Дотримання всіх вищезазначених заходів є обов’язковим згідно з чинним законодавством України та міжнародними стандартами. Нехтування технікою безпеки не лише загрожує здоров’ю працівників, а й тягне за собою адміністративну, а в окремих випадках — кримінальну відповідальність для керівників підприємств.

ВИСНОВКИ

У ході виконання даної дипломної роботи було здійснено повний цикл розробки вебзастосунка для внутрішнього менеджменту IT-компанії. Основною метою стало створення функціональної, масштабованої та адаптованої до потреб конкретного підприємства системи, яка автоматизує процеси керування персоналом, проєктами, задачами та прогресом їх виконання.

На початковому етапі було проведено аналіз предметної області, сформульовано вимоги до системи, виявлено основних актантів і побудовано ключові варіанти використання, що заклало основу для формування функціоналу системи.

У процесі проєктування було обрано гнучкий підхід до розробки, побудовано схему бази даних, UML-діаграму класів, а також архітектурну модель застосунка, що забезпечує розділення відповідальностей між клієнтською та серверною частинами.

Під час конструювання програмної системи було використано сучасні технології: Angular, HTML/SCSS, TypeScript для фронтенду, C#, ASP.NET Core та Entity Framework для бекенду, а також PostgreSQL як систему управління базами даних. Це дозволило досягти високої продуктивності, зручності обслуговування та адаптивності інтерфейсу.

На етапі розгортання були визначені системні вимоги, описано інфраструктуру для розміщення застосунка, а також наведено типові схеми його використання. Було виконано верифікацію основних функціональних сценаріїв, що підтвердило відповідність системи початковим вимогам.

Проведене тестування програмного забезпечення продемонструвало стабільність його роботи та коректність реалізації основних модулів. Застосунок відповідає вимогам щодо безпеки, зручності використання та масштабованості.

У результаті виконання дипломного проєкту досягнуто поставленої мети — розроблено власне, адаптоване програмне забезпечення для менеджменту IT-

компанії, яке сприяє ефективній організації внутрішніх процесів, покращує взаємодію між учасниками команди та підвищує загальну продуктивність роботи. Система охоплює ключові аспекти управління — від розподілу завдань і контролю їх виконання до генерації звітності та аналітики, що дозволяє компанії працювати більш злагоджено, прозоро й результативно. Розроблене рішення є масштабованим і здатним адаптуватися до змін у структурі команди чи бізнес-процесах, що робить його актуальним для використання в реальних умовах сучасного IT-середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pressman, R. S. Software Engineering: A Practitioner’s Approach. New York: McGraw-Hill, 2014. 928 p.

2. Sommerville, I. Software Engineering. 10th ed. Harlow: Pearson Education, 2015. 792 p.

3. Freeman, E., Robson, E. Head First Design Patterns. 2nd ed. Sebastopol: O'Reilly Media, 2020. 694 p.

4. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 395 p.

5. Martin, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.

6. Information System for Design of Thin Multilayer Film Processes Parameters Management Based on Diffusion. [Without author].

7. Petryk, M., Chyzh, V., Tsupryk, H., Petryk, O. Information Technologies: Theoretical and Applied Problems: Proc. 4th Int. Workshop ITTAP’2024 (Oct. 23–25, 2024, Ternopil, Ukraine – Opole, Poland). CEUR-WS, 2024. Vol. 3896. P. 486–493. URL: https://ceur-ws.org/Vol-3896/ (accessed: ...).

8. Yavorskyy, B., Yavorska, E., Tsupryk, H., Kinash, R. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli // Scientific Journal of TNTU. – Ternopil: TNTU, 2023. – Vol. 112, No. 4. – P. 82–90.

9. Fowler, M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.

10. Microsoft Docs – ASP.NET Core Documentation. URL: https://learn.microsoft.com/aspnet/core (accessed: ...).

11. Angular Official Documentation. URL: https://angular.io/docs (accessed: ...).

12. PostgreSQL Official Documentation. URL: https://www.postgresql.org/docs/ (accessed: ...).

13. Microsoft Docs – Entity Framework Core. URL: https://learn.microsoft.com/ef/core (accessed: ...).

14. Agile Alliance. What is Agile Software Development? URL: https://www.agilealliance.org/agile101 (accessed: ...).

15. Open Web Application Security Project (OWASP). URL: https://owasp.org (accessed: ...).

16. ISO/IEC 25010:2011. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models.

17. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications.

18. Bass, L., Clements, P., Kazman, R. Software Architecture in Practice. 3rd ed. Boston: Addison-Wesley, 2012. 624 p.

19. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329 (дата звернення: 08.06.2025).

20. Петрик, М. Р., Михалик, Д. М., Кінах, Я. І., Гладьо, С. В., Цуприк, Г. Б. Методичні вказівки до виконання дипломної роботи освітнього рівня – бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016. – 28 с.

21. Бедрій Я. І. Основи охорони праці : навч. посіб. / Я. І. Бедрій. – 4-е вид., перероб. і доп. – Тернопіль : Навчальна книга – Богдан, 2018. – 240 с.

22. Кодекс законів про працю України [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/322-08

23. Желібо Є. П. Безпека життєдіяльності : підручник / Є. П. Желібо, В. В. Зацарний. – Київ : Каравела, 2023. – 344 с.

24. ДСТУ ISO 6309:2007. Протипожежний захист. Знаки безпеки. – Київ : Держспоживстандарт України, 2007. – 11 с.

25. Типове положення про порядок проведення навчання з питань охорони праці : НПАОП 0.00-4.36-05. – Київ : Держпраці, 2005. – 26 с.

26. ДСТУ ISO/TS 16976-8:2021. Засоби індивідуального захисту органів дихання. Людські чинники. Частина 8. Ергономічні чинники. – Київ : Мінекономіки України, 2021. – 20 с.

ДОДАТКИ

ДОДАТОК А. ІЛЮСТРАЦІЇ ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ

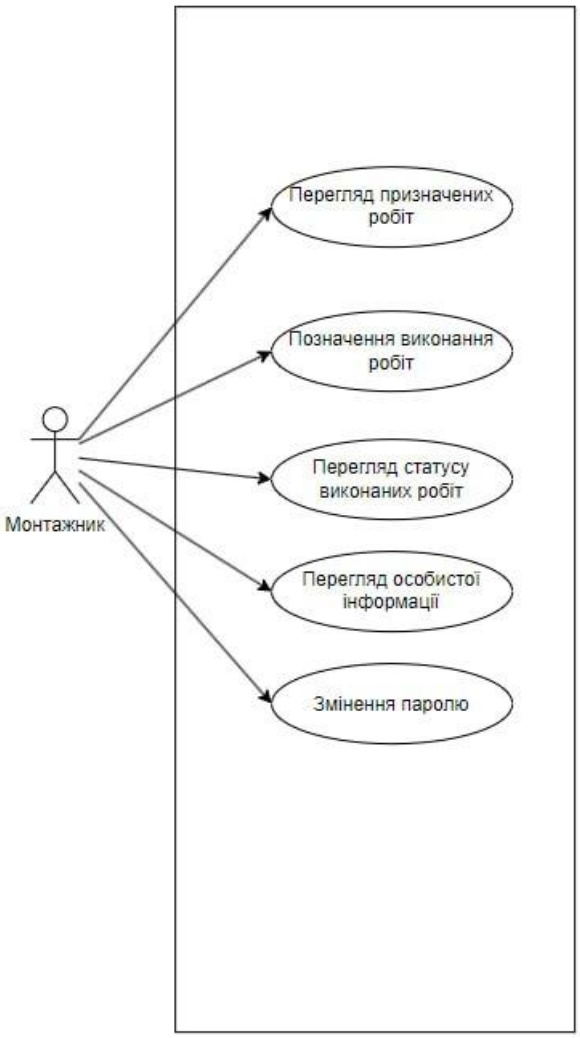

Рисунок А.1 – Діаграма варіантів використання для монтажника

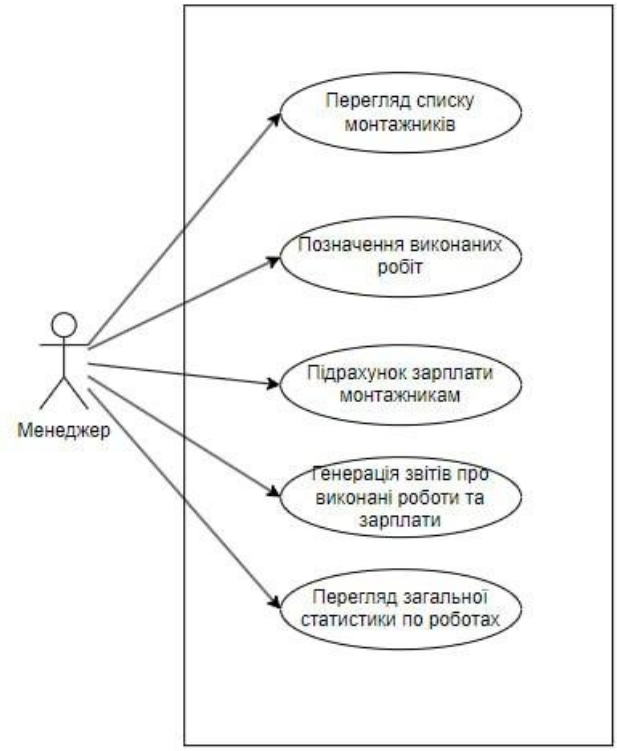

Рисунок А.2 – Діаграма варіантів використання для менеджера

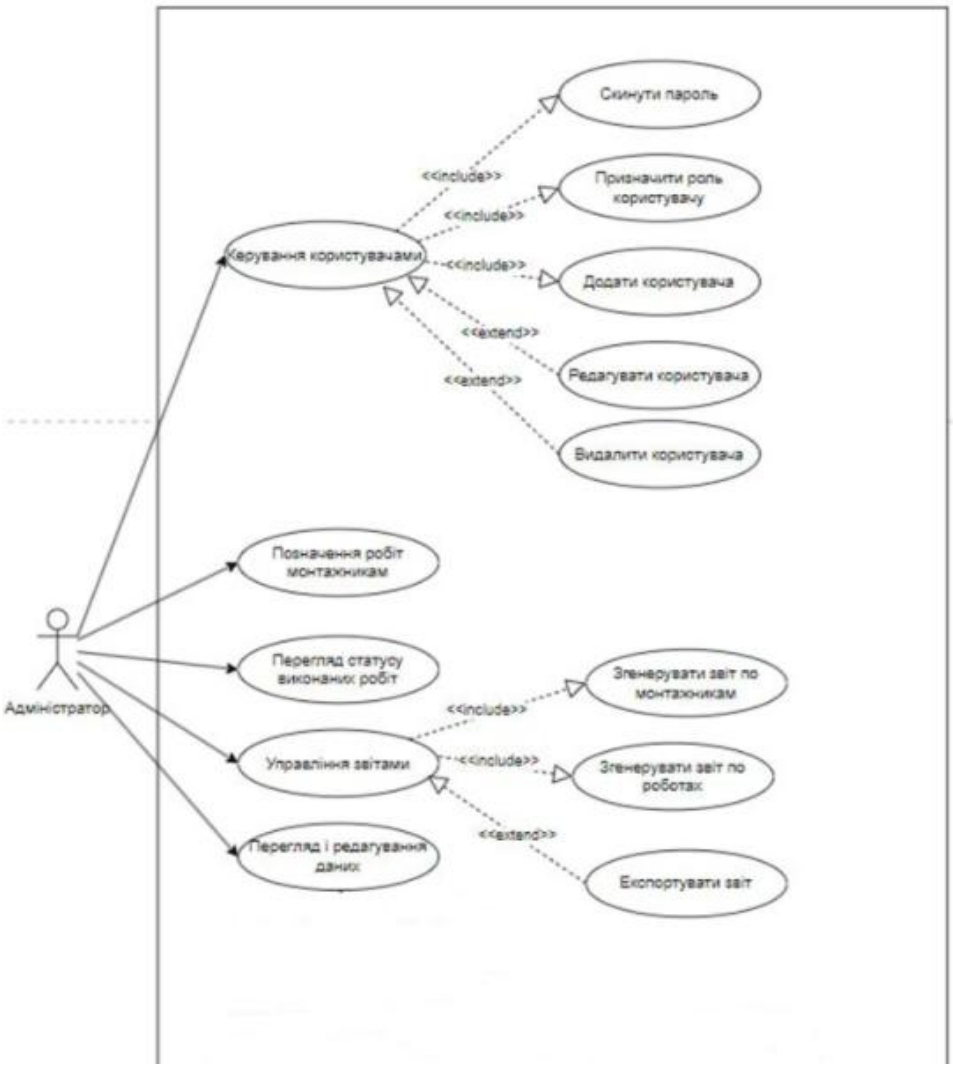

Рисунок А.3 – Діаграма варіантів використання для адміністратора

ДОДАТОК Б. CD З ПРОГРАМНИМ КОДОМ