

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО–КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інформаційна система моніторингу та управління
будівельними проєктами»**

Виконав студент групи ПЗ–22–3
ЄЛЕНЕЦЬКИЙ Віктор Вікторович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО–КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЄЛЕНЕЦЬКИЙ Віктор Вікторович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Інформаційна система моніторингу та управління будівельними проєктами»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: Об'єктом дослідження є процеси розробки програмного забезпечення для управління інформаційними потоками та робочими процесами при реалізації будівельних проєктів. Предметом дослідження є методи, моделі та програмні засоби створення веборієнтованих інформаційних систем моніторингу та управління будівельними проєктами.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	01.03.2026р	01.03.2026р
2	Лобанчикова Н.М.	01.04.2026р	01.04.2026р
3	Лобанчикова Н.М.	01.05.2026р	01.05.2026р

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з інформаційної системи моніторингу та управління будівельними проектами і пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 58 сторінок, 40 ілюстрацій та 24 таблиці. Загальний обсяг 83 сторінки.

Метою даної дипломної роботи є розробка та програмна реалізація інформаційної системи моніторингу та управління будівельними проектами, що автоматизує повний цикл виконання будівельного завдання – від первинного планування менеджером до технічного приймання інженером на основі фотодоказів. Система усуває розпорошеність інформації між месенджерами і таблицями, надаючи учасникам будівельного процесу єдине середовище з чітким розподілом доступу за ролями.

Архітектура системи побудована на принципі чіткого розмежування відповідальностей між шарами, що забезпечує незалежність бізнес-логіки від інфраструктурних деталей. Клієнтська і серверна частини розроблені як незалежні компоненти з взаємодією через програмний інтерфейс. У роботі детально описано доменну модель із автоматом станів будівельного завдання, алгоритми взаємодії між компонентами системи та структуру бази даних. Коректність реалізованої логіки підтверджена модульними і інтеграційними тестами. Розгортання системи повністю автоматизовано через конвеєр безперервної інтеграції та доставки.

КЛЮЧОВІ СЛОВА: БУДІВЕЛЬНИЙ ПРОЄКТ, ВЕБСИСТЕМА, CLEAN ARCHITECTURE, CQRS, .NET.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інформаційна система моніторингу та управління будівельними проектами Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Сленецький В.В.						
Керівник		Лобанчикова Н.М					4	83
Рецензент		Морозов А.В.				Житомирська політехніка, група ІПЗ-22-3		
Зав. каф		Вакалюк Т.А.						

ABSTRACT

The Bachelor's qualification work consists of an information system for monitoring and managing construction projects and an explanatory note. The explanatory note to the qualification work contains 58 pages, 40 illustrations, and 24 tables. The total volume is 83 pages.

The purpose of this diploma work is the development and software implementation of an information system for monitoring and managing construction projects, which automates the full cycle of executing a construction task — from initial planning by a manager to technical acceptance by an engineer based on photographic evidence. The system eliminates the fragmentation of information across messengers and spreadsheets, providing participants in the construction process with a unified environment featuring strict role-based access control.

The system architecture is built on the principle of a clear separation of concerns between layers, ensuring the independence of business logic from infrastructural details. The client and server parts are developed as independent components interacting via an Application Programming Interface. The work describes in detail the domain model with a state machine for construction tasks, the algorithms for interaction between system components, and the database structure. The correctness of the implemented logic is confirmed by unit and integration tests. The deployment of the system is fully automated through a continuous integration and continuous delivery (CI/CD) pipeline.

KEYWORDS: CONSTRUCTION PROJECT, WEB SYSTEM, CLEAN ARCHITECTURE, CQRS, .NET.

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		5

ЗМІСТ

ВСТУП	8
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	11
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ.....	12
1.1 Постановка задачі.....	12
1.2 Аналіз аналогів програмного продукту	14
1.3 Визначення архітектури інформаційної платформи.....	19
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	24
Висновки до першого розділу	29
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ	30
2.1 Визначення варіантів використання та об'єктно–орієнтованої структури системи	30
2.2 Розробка бази даних системи.....	33
2.3 Проєктування та реалізація алгоритмів роботи системи	38
2.4 Реалізація інформаційної системи моніторингу та управління будівельними проєктами.....	40
Висновки до другого розділу	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІЙНОЮ СИСТЕМОЮ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ.....	43
3.1 Порядок встановлення та налаштування параметрів системи	43
3.2 Структура інтерфейсу інформаційної системи моніторингу та управління будівельними проєктами.....	47

3.3 Тестування роботи програмного продукту	58
Висновки до третього розділу	63
ВИСНОВКИ.....	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ	68

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Сучасне будівництво передбачає одночасну координацію значної кількості учасників – менеджерів, підрядників та інженерів, кожен з яких виконує окремі функції в загальному технологічному ланцюзі. На практиці взаємодія між ними часто організована через месенджери, електронні таблиці або паперові акти, що призводить до розбіжностей у даних, затримок та відсутності прозорості на всіх етапах виконання робіт.

З розвитком вебтехнологій автоматизація процесів управління будівельними проєктами стає реально досяжною навіть для компаній середнього розміру. Централізована інформаційна система дозволяє в режимі реального часу відстежувати стан задач, фіксувати виконані роботи з фотодоказами та забезпечувати контроль якості без необхідності виїзду на об'єкт. Це суттєво скорочує час на адміністративні процедури та зменшує ризик людських помилок.

Наявні на ринку рішення, такі як Jira, Microsoft Project, або Procore або орієнтовані на IT-процеси, або є надто дорогими і складними для впровадження в невеликих будівельних організаціях. Це обумовлює доцільність розробки спеціалізованої вебсистеми, адаптованої під потреби будівельної галузі.

Мета і завдання роботи. Метою кваліфікаційної роботи є розробка та програмна реалізація інформаційної системи для моніторингу і управління будівельними проєктами, яка забезпечить автоматизацію основних робочих процесів та підтримку рольової моделі доступу для всіх учасників будівництва. Для досягнення поставленої мети необхідно вирішити комплекс взаємопов'язаних завдань:

- проаналізувати бізнес-процеси координації у будівництві та дослідити недоліки наявних програмних рішень; спроектувати доменну модель, реляційну базу даних і загальну архітектуру системи з використанням підходів Clean Architecture та CQRS;

- програмно реалізувати серверну частину на базі платформи .NET і веборієнтований клієнтський інтерфейс за допомогою React;

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

– розробити рольову модель доступу для різних типів користувачів та інтегрувати функціонал фіксації виконаних робіт із фотодоказами, а також провести комплексне тестування розробленого програмного забезпечення для підтвердження його працездатності й відповідності поставленим вимогам.

Об'єктом дослідження є процеси розробки програмного забезпечення для управління інформаційними потоками та робочими процесами при реалізації будівельних проєктів.

Предметом дослідження є методи, моделі та програмні засоби створення веборієнтованих інформаційних систем моніторингу та управління будівельними проєктами.

Методи дослідження. У роботі використано системний аналіз для вивчення предметної області, об'єктно–орієнтоване проєктування – при побудові доменної моделі, та реляційне моделювання – при проєктуванні бази даних. Як джерела використано документацію .NET, React, PostgreSQL та матеріали з патернів Clean Architecture і CQRS.

Практичне значення. Розроблена інформаційна система має високу практичну цінність, оскільки вирішує реальну проблему відсутності єдиного інструменту для координації учасників будівельного процесу. Система забезпечує автоматизацію повного циклу виконання будівельного завдання - від планування менеджером до технічного приймання інженером на основі фотодоказів. Реалізована рольова модель доступу усуває ризики змішування відповідальностей, а фіксація виконаних робіт із прив'язкою до етапів підвищує прозорість та контроль якості.

Завдяки використанню сучасних архітектурних підходів (Clean Architecture, CQRS) та хмарної інфраструктури система є масштабованою, легко підтримуваною та придатною до впровадження в невеликих і середніх будівельних компаніях. Автоматизований CI/CD-конвеєр забезпечує стабільність оновлень, а веб-інтерфейс на React дозволяє працювати з будь-якого пристрою без встановлення додаткового ПЗ. Розроблене рішення може бути використане як

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

готовий інструмент для оптимізації управління будівельними проєктами або як основа для подальшого розширення функціоналу.

Структура та обсяг проекту роботи. Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 34 найменувань (3 стор.), 9 додатків (14 стор.), 24 таблиці та 40 ілюстрацій по тексту, з них 6 оформлено на окремій сторінці. Повний обсяг роботи становить 83 сторінки, з них 58 сторінок основного тексту.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

CI/CD – автоматизована інтеграція та розгортання

CQRS – розділення операцій команд і запитів

DTO – об'єкт передачі даних

JWT – токен для аутентифікації

SPA – односторінкова вебпрограма

EF – об'єктно-реляційне відображення

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ

1.1 Постановка задачі

Рівень цифровізації малого та середнього будівельного бізнесу залишається критично низьким [3, 23]. Поки великі гравці експлуатують корпоративні ERP-системи, локальні підрядники продовжують координувати процеси через побутові месенджери на зразок Viber та розрізнені таблиці Excel. Аналіз реальної предметної області виявив серйозну проблему маршрутизації даних [28]. Типовий експлуатаційний сценарій виглядає так: менеджер ставить задачу в загальний чат, підрядник там само звітує фотографіями, після чого інженер марнує час на пошук потрібних медіафайлів серед сотень сторонніх повідомлень для технічного приймання.

Завдання просто губляться. І причина не в людській недобросовісності, а в тому, що груповий чат технічно не здатен виступати надійним носієм проєктної інформації. Інженер може погодити етап за старими фотографіями через втрату актуальних кадрів, а менеджер позбавлений можливості швидко оцінити кількість активних робіт без ручного парсингу всієї переписки.

Ситуація ускладнюється суворою специфікою галузі. Будівництво вимагає жорсткого розмежування відповідальності. Менеджер планує та призначає виконавців. Підрядник виконує роботу і фіксує це на фото. Інженер здійснює контроль і виносить рішення. Ці ролі не є взаємозамінними, тому їх змішування в одному неструктурованому потоці порушує базові принципи цілісності даних.

Головна ідея платформи: вивести робочий процес із неконтрольованих чатів та ізолювати його в спеціалізованому середовищі. Замість того, щоб розробляти громіздкий моноліт із десятками теоретичних, але не затребуваних на практиці модулів, фокус робився на базовому життєздатному функціоналі. Система має закривати виключно реальні «нагальні експлуатаційні проблеми» будівельного майданчика.

Щоб довести цю концепцію до робочого релізу, застосовувався класичний інженерний пайплайн. Усе почалося з аудиту наявного програмного

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

забезпечення, який одразу виявив архітектурну перевантаженість аналогів. Зрозумівши це, на базі конкретних юзкейсів були прописані суворі функціональні вимоги. Далі йшло проєктування схеми бази даних та підбір відповідного технологічного стеку. Найбільше часу пішло на безпосереднє написання коду для клієнта та сервера, паралельно з тестуванням критичної бізнес-логіки. А щоб виключити людський фактор при перевірці коду, завершальним технічним етапом стало налаштування автоматизованого CI/CD конвеєра.

На виході маємо не просто концепт, а розгорнутий інформаційний продукт, здатний підтримувати повний операційний цикл об'єкта. Планування робіт, звітність підрядників, технічний нагляд інженера – усе це тепер консолідовано в єдиний цифровий контур, де фізично немає місця стороннім месенджерам чи екселівським таблицям.

Програма спирається на конкретний набір функцій. Головна вимога стосується поділу прав користувачів. Інженер, менеджер та підрядник отримують повністю ізольовані кабінети. Кожен бачить лише свої інструменти. Застосунок програмно блокує переведення задачі у новий статус без прикріпленої фотографії. Схема бази даних постійно фіксує кожну дію. Такий алгоритм створює прозоре середовище та зупиняє маніпуляції з даними.

Реальний будівельний майданчик має складні умови роботи. Найчастіше там нестабільне покриття мережі. Тому база даних використовує транзакції. При раптовому обриві зв'язку інформація не зберігається частково і не пошкоджується. Інженер перевіряє звіти на ноутбуці в офісі. Підрядник фотографує результат на телефон і одразу завантажує його на об'єкті. Продукт працює безпосередньо у браузері. Хмарний хостинг звільняє компанію від потреби купувати власні сервери.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз аналогів програмного продукту

Перш ніж переходити до проектування власної архітектури, знадобилося проаналізувати наявні ринкові рішення. Головна мета цього етапу цілком прагматична: зрозуміти, який функціонал уже існує, та головне – чому жоден із популярних інструментів не здатен адекватно закрити специфічні потреби будівельного майданчика, описані вище.

Для порівняння відбиралися чотири системи з принципово різними підходами до управління. Наприклад, Trello розглядався як базова Kanban–дошка для елементарної організації задач. Jira, навпаки, виступала як важкий трекер, заточений під IT–команди. Зі спеціалізованого галузевого софту до огляду потрапила платформа Procore, тоді як Microsoft Project аналізувався виключно як еталон класичного календарного планування.

Розглянемо систему управління проектами Trello. Цей інструмент побудований навколо концепції електронної дошки, де картки з описом завдань і вкладеннями переміщуються між колонками відповідно до їх поточного стану [30]. Візуальний інтерфейс програмного продукту, що відображає цю структуру (рис. 1.1). Завдяки мінімальному порогу входу платформа є дуже популярною серед невеликих команд.

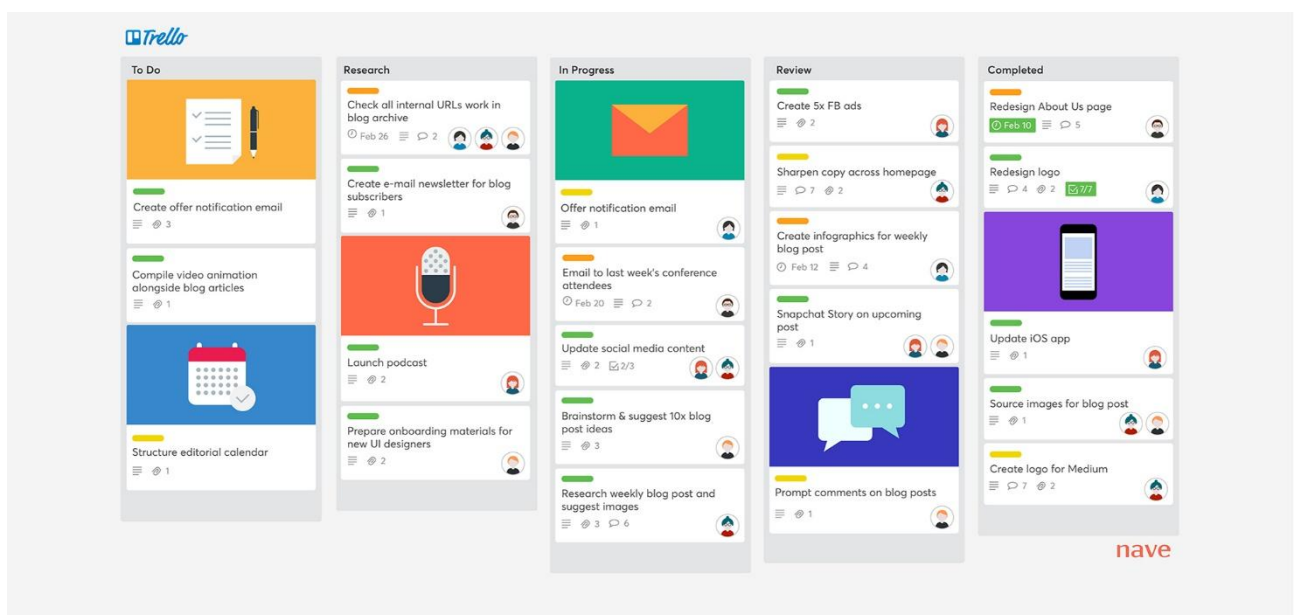


Рис. 1.1 – Інтерфейс Trello, приклад дошки задач

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

На перший погляд здається, що для простого відстеження завдань цього достатньо. Але проблема виявляється одразу, як тільки намагаєшся вписати в цю модель реальний будівельний процес. Головна проблема – повна відсутність ролей. Система не розуміє, хто з нею працює: менеджер, підрядник чи інженер мають абсолютно ідентичний інтерфейс та відкриті права. Більше того, картки завдань дозволяється хаотично перекидати між колонками в будь-який час. На реальному ж майданчику така свобода дій неприпустима, адже кожне завдання має проходити суворо визначений ланцюжок статусів без можливості їх перестрибнути. Фотографії прикріпити технічно можна, але система ніяк не пов'язує їх із конкретним етапом виконання – інженер, переглядаючи картку, просто бачить набір файлів без контексту. Механізму приймання роботи немає взагалі. Зрештою, Trello залишається не більше ніж базовим інструментом візуалізації. Це класична канбан-дошка, повністю позбавлена внутрішньої доменної логіки. Звісно, для гнучкого управління дрібними офісними завданнями така архітектурна простота є безсумнівним плюсом. Проте реальне виробництво вимагає кардинально іншого рівня контролю. Будівельний процес не терпить хаотичності.

Як альтернативу часто розглядають Jira від компанії Atlassian [14]. В індустрії розробки програмного забезпечення цей продукт уже давно перетворився на безумовний галузевий стандарт. Архітектура платформи робить головний акцент на глибокій кастомізації. Користувачам доступні інструменти для моделювання будь-яких робочих процесів, а вбудовані механізми системи безперервно фіксують історію кожної дії у логах. Матриця прав доступу тут багаторівнева і дозволяє точково обмежувати видимість даних. Додатковою перевагою є відкритість системи до інтеграцій із десятками зовнішніх сервісів (рис. 1.2).

Теоретично Jira дозволяє налаштувати будь-який процес – у тому числі будівельний. На практиці це означає, що хтось повинен витратити кілька днів на конфігурацію, а потім пояснити підряднику на майданчику, що таке «запит», «спринт» і «story points».

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

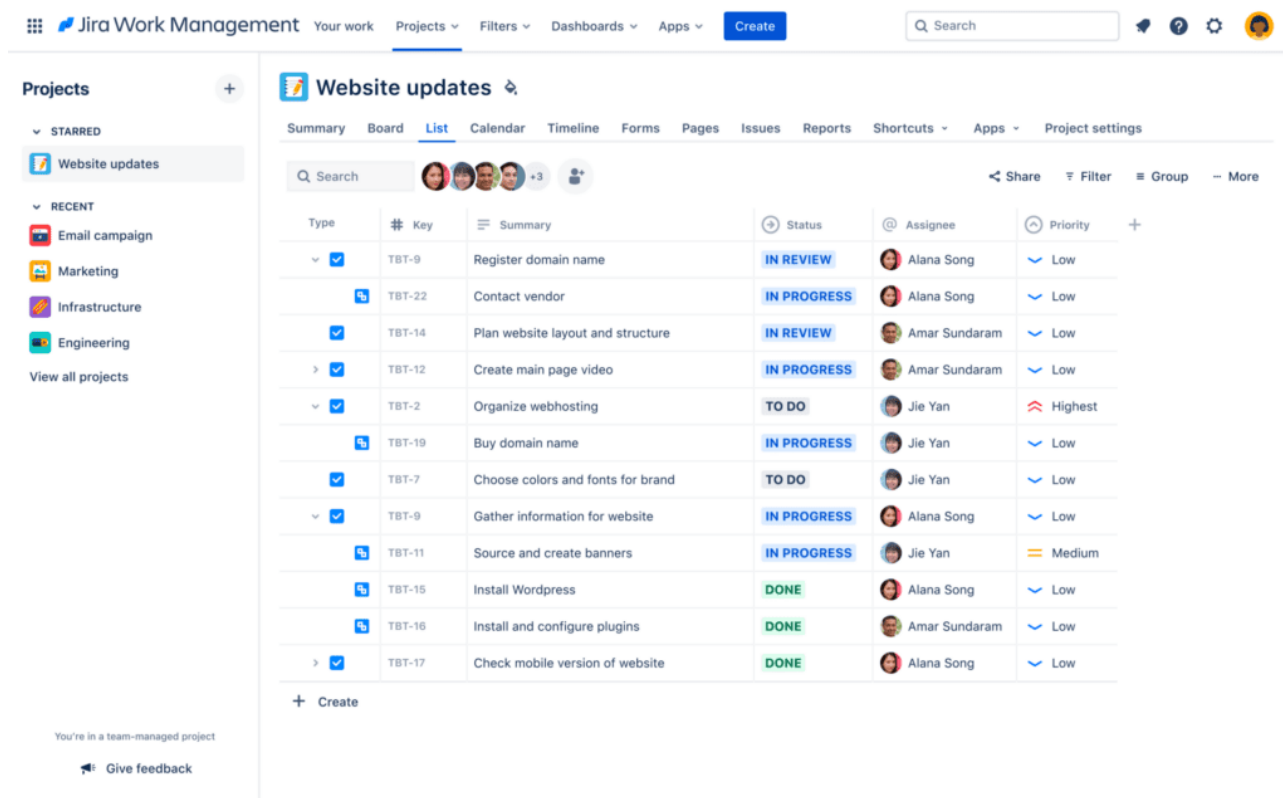


Рис. 1.2 – Інтерфейс Jira, перегляд задачі

Головна проблема полягає в глибокому архітектурному конфлікті на рівні доменної моделі. Базові абстракції платформи не відповідають будівельним реаліям. З інженерної точки зору, система не дозволяє імплементувати «фотодоказ» як повноцінну сутність та валідатор переходу стану. Натомість фото залишається звичайним прикріпленням файлом. Спроба обійти це обмеження вимагає використання антипатернів та сторонніх вебхуків, що стрімко деградує підтримуваність коду. Зважаючи на необхідність розробки таких кастомних «милиць» у поєднанні з базовою вартістю ліцензії від 8 доларів за користувача, використання цієї платформи для малого будівельного бізнесу є фінансово та технічно невиправданим.

Наступною доцільно розглянути систему Procore. На відміну від попередніх універсальних інструментів, ця платформа від початку проєктувалася виключно під домен будівельної галузі [23]. Її архітектура максимально точно відображає реальні бізнес-процеси на об'єкті. Базовий функціонал містить комплекс спеціалізованих модулів. Серед них інструменти для управління

проектною документацією, електронні наряди на виконання робіт, журнали технічних інспекцій, а також підсистеми контролю кошторисів і безпеки на майданчику (рис. 1.3).

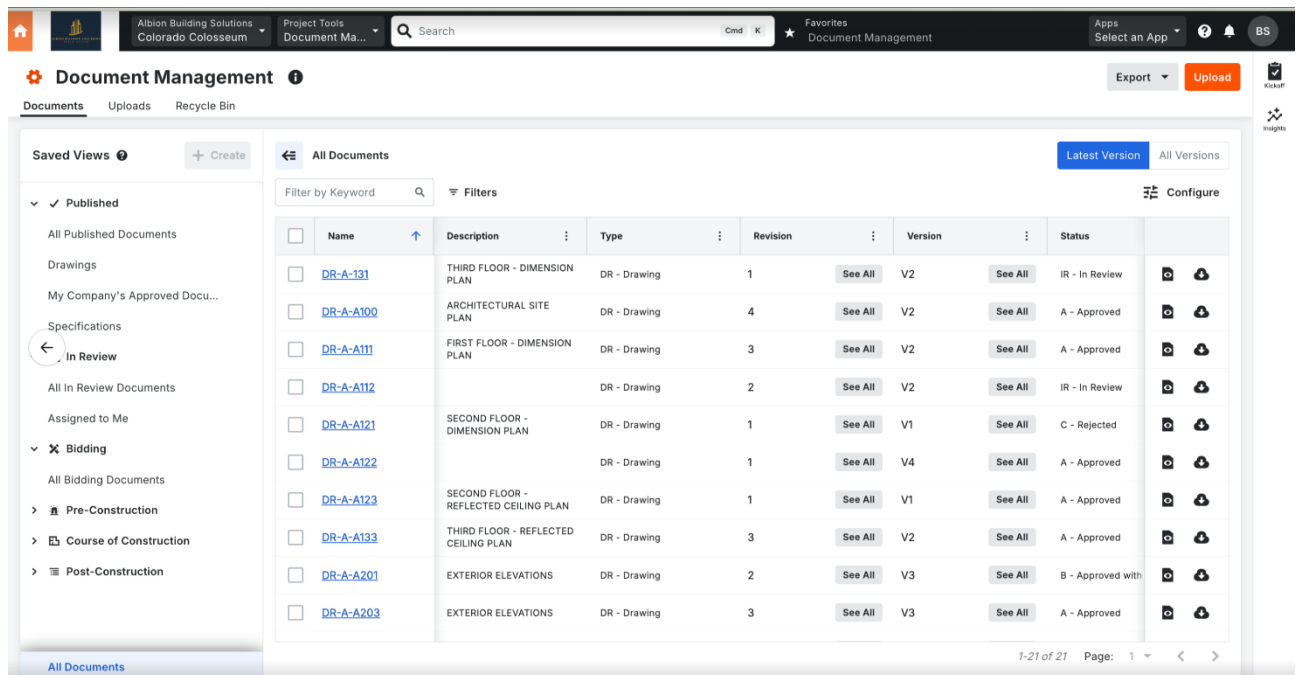


Рис. 1.3 – Інтерфейс Procore, функціональний блок управління завданнями

З усіх розглянутих рішень Procore найближчий до задачі, що вирішується в даній роботі: є прив'язка виконавців до нарядів з урахуванням ролей, завантаження фото з мобільного пристрою, механізм актів приймання, розгалужена звітність. Система справді розуміє предметну область.

Але тут і починається парадокс. Мінімальний тарифний план стартує від 375 доларів на місяць, а реальна вартість із повним набором модулів виходить на тисячі. Procore орієнтований на великі підрядні організації з окремим IT-відділом і бюджетом на впровадження не на команду з десяти осіб, яка досі координується через Viber. Функціональна надлишковість тут не перевага, а проблема: десятки модулів, тривале навчання персоналу, складне впровадження все це робить систему неприйнятною саме там, де вона могла б бути найкориснішою.

Окремої уваги заслуговує класичне рішення на ринку систем управління – програмний комплекс Microsoft Project. Один із найстаріших інструментів [16] у

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

категорії, який існує з 1984 року і традиційно використовується для побудови календарних планів у форматі діаграм Ганта. Орієнтований на керівників проєктів: планування ресурсів, відстеження критичного шляху, бюджетування (рис. 1.4).

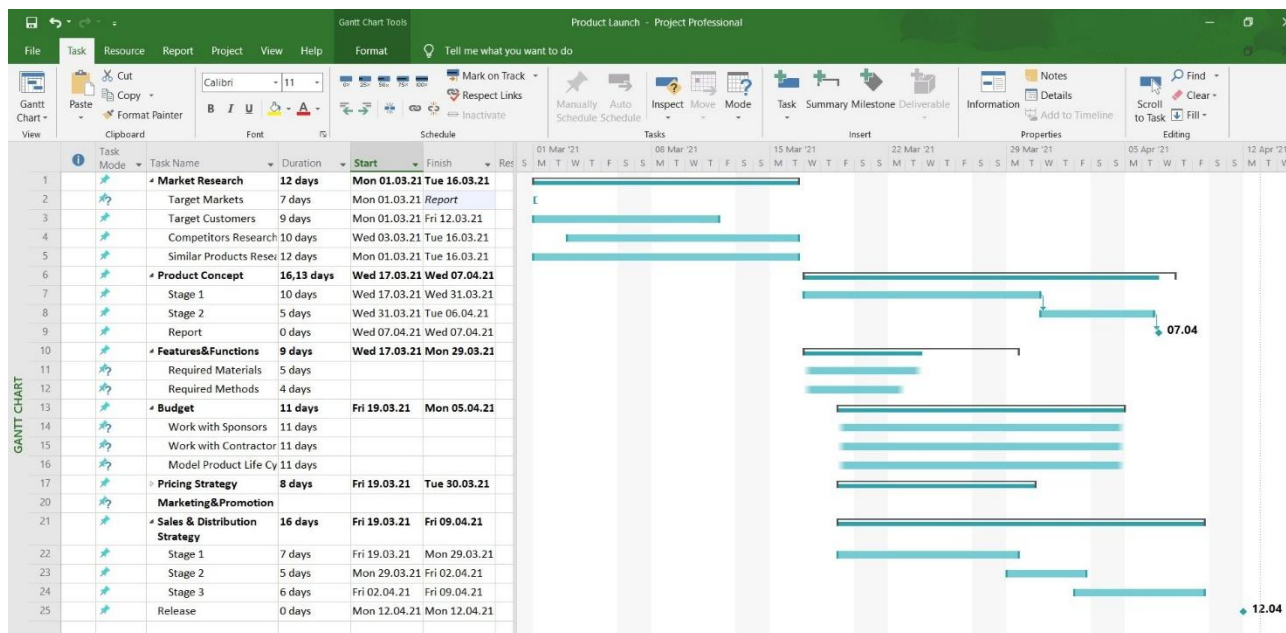


Рис. 1.4 – Інтерфейс Microsoft Project, діаграма Ганта

Проблема тут навіть не в деталях реалізації, а в самому призначенні продукту. Microsoft Project – це інструмент планування, а не виконання. Керівник може детально розписати план робіт, але підрядник не має можливості звітувати про виконання в рамках тієї самої платформи. Поняття фотодоказів, подання звіту та технічного приймання тут відсутні не через недогляд, а тому що система проєктувалась для іншого завдання. Фактично вона закриває лише перший етап описаного процесу, залишаючи всі наступні поза увагою. Настільна версія є платною (від 10 доларів на місяць), а повноцінна робота вимагає встановленого клієнта, що ускладнює використання безпосередньо на майданчику.

За результатами аналізу складено порівняльну таблицю відповідності кожного рішення ключовим вимогам (табл. 1.1).

Порівняльна характеристика розглянутих систем

Критерій оцінювання	Trello	Jira	Procore	MS Project	Власна розробка
Ролі будівельного процесу	Ні	Частково	Так	Частково	Так
Фіксована послідовність станів завдання	Ні	Так	Так	Ні	Так
Подання фотодоказів як окрема сутність	Ні	Ні	Так	Ні	Так
Механізм технічного приймання інженером	Ні	Ні	Частково	Ні	Так
Повернення завдання на доопрацювання	Ні	Так	Частково	Ні	Так
Прив'язка завдань до будівельного об'єкта	Ні	Ні	Так	Так	Так
Доступність для нетехнічного персоналу	Так	Ні	Частково	Ні	Так
Вартість ліцензії (SaaS)	Безкоштовно	Близько 80\$/міс	Близько 375\$/міс	Близько 100\$/міс	—

Проведений аналіз дозволяє зробити кілька важливих спостережень. Загальні інструменти Trello і Jira є достатньо гнучкими, але нічого не знають про будівництво: ні про ролі підрядника та інженера, ні про фотодокази, ні про прив'язку завдань до фізичного об'єкта. Їх адаптація потребувала б такого обсягу ручного налаштування, що простіше написати власне рішення. Procore розуміє предметну область, але коштує як річний бюджет невеликого підряду і несе з собою функціональність, яка в даному контексті є зайвою. Microsoft Project закриває виключно задачу планування і навіть не претендує на решту циклу.

Жоден із розглянутих інструментів не відповідає визначеним вимогам при прийнятному рівні складності та вартості. Це і є передумовою для розробки спеціалізованої системи, орієнтованої на конкретний процес виконання будівельного проєкту.

1.3 Визначення архітектури інформаційної платформи

Визначення архітектурної моделі є першим і найважливішим кроком. Архітектура визначає, наскільки легко ізолювати бізнес-логіку від деталей

реалізації, протестувати окремий сценарій без підняття всієї системи і змінити одну частину, не зачепивши іншу.

Архітектурні рішення інформаційної системи управління будівельними проєктами та робочими процесами «BuildFlow» (далі – система BuildFlow) проєктувався з огляду на жорсткі вимоги предметної області. З продуктом одночасно працюють кілька типів користувачів із кардинально різними правами доступу. До того ж будівельне завдання проходить через складний багатоетапний життєвий цикл. Ці фактори вимагатимуть розмежування операцій читання та запису даних на рівні коду. Далі детально розібрано кожен імплементований архітектурний патерн та аргументовано, чому від класичних альтернатив доведеться відмовитися.

Глобально архітектура буде розділена на повністю ізольований бекенд (HTTP API) та клієнтський SPA–додаток. Від класичного підходу із серверним рендерингом (SSR) довелося відмовитися практично одразу. Причина лежить у площині UX та специфіки ролей: менеджер, підрядник та інженер працюють з кардинально відмінними екранами. Якщо формувати сторінки на сервері, кожне перемикання між робочими просторами викликатиме повне перезавантаження інтерфейсу. Натомість жорстке відокремлення API від клієнта вирішує цю проблему. Більше того, така архітектура дає змогу для майбутнього масштабування: у перспективі до цього ж бекенду можна підключити нативний мобільний додаток, не переписуючи серверну логіку.

Щодо внутрішньої будови бекенду, то тут буде імплементовано патерн Clean Architecture (рис. 1.5). Якщо відкинути теорію та оцінити користь для BuildFlow, головною перевагою цього підходу є надійність ізоляції бізнес логіки від технічних деталей інфраструктури. Це дозволяє зосередитися на роботі сервісу, не переймаючись тим, як саме влаштована база даних чи серверна частина. Це означає, що ядро системи не знає, з якою саме базою даних вона працюватиме чи який фреймворк оброблятиме HTTP–запити. Зовнішні модулі підлаштовуються під ядро, а не навпаки.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

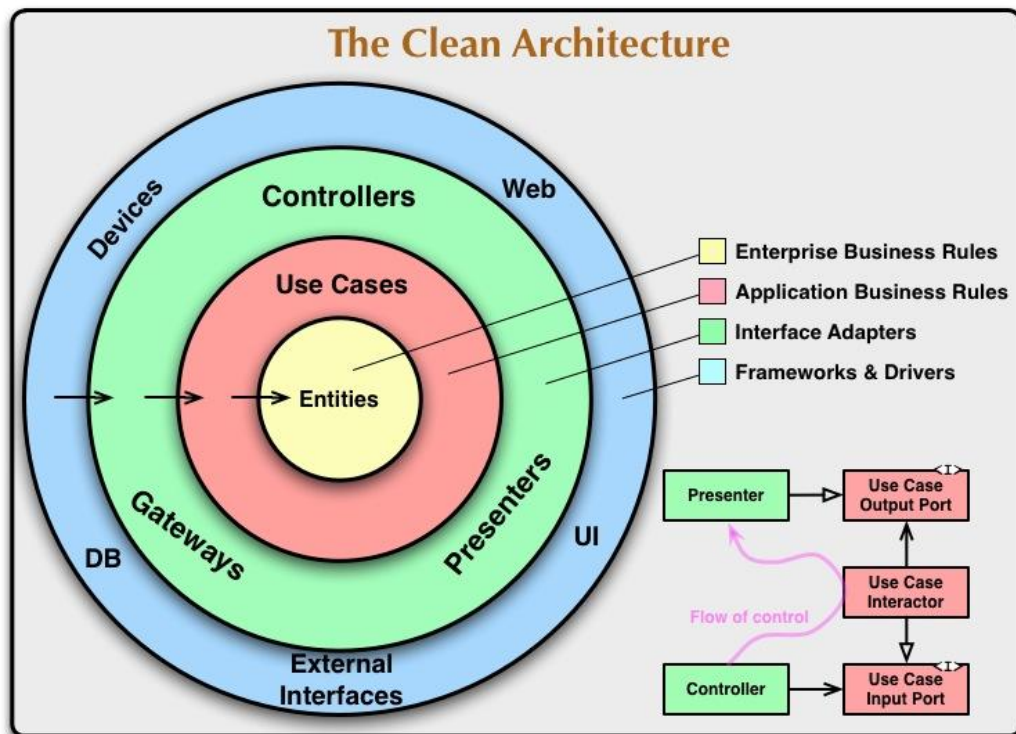


Рис. 1.5 – Схема чистої архітектури за Робертом Мартіном [18]

Традиційна тришарова архітектура розглядалась як простіша альтернатива, але має принциповий недолік для даної системи: шар бізнес-логіки безпосередньо залежить від шару доступу до даних, що унеможливорює тестування правил переходів між станами завдання без реальної бази. Чиста архітектура вирішуватиме це через інверсію залежностей. Шар логіки визначатиме інтерфейс репозиторію, інфраструктурний шар його реалізуватиме.

У BuildFlow це буде реалізовано чотирма шарами. Шар предметної області [7, 9] – найглибший, без жодних зовнішніх залежностей. Тут зосереджені сутності системи та всі бізнес-правила: неможливість переходу між станами завдання в обхід встановленої послідовності, обмеження на подання звіту лише призначеним виконавцем. Шар прикладної логіки координує сутності та зовнішні сервіси через обробники сценаріїв використання і визначає інтерфейси, не знаючи їх реалізацій. Шар інфраструктури реалізує ці інтерфейси: підключається до бази даних, взаємодіє зі сховищем файлів. Шар точки входу приймає HTTP-запити, перевіряє автентифікацію і передає дані без власної логіки.

Варто детальніше розглянути обрану стратегію збереження даних, оскільки вона містить низку нестандартних технічних рішень. Сучасні хмарні платформи пропонують кілька спеціалізованих типів сховищ (рис. 1.6) і вибір між ними не очевидний.



Рис. 1.6 – Типи хмарних сховищ даних [4]

У системі два принципово різні типи даних. Структуровані – завдання, користувачі, об'єкти, звіти, рецензії зберігаються у реляційній базі. Реляційна модель тут не данина традиції: зміна стану завдання потребує транзакційності, аналітичні вибірки по об'єкту потребують JOIN–запитів. Жоден із хмарних типів сховищ цього не забезпечує, а навпаки – вони вирішують інші задачі. Файлові дані, фотодокази та креслення зберігаються у сховищі об'єктів. Зберігати їх у реляційній базі технічно можна, але це збільшує її обсяг і уповільнює табличні запити без жодної продуктивної користі.

Система буде розгорнута на хмарній платформі з автоматизованим пайплайном доставки змін. Кожна зміна у вихідному коді проходить перевірку якості, тести і збірку і лише після цього потрапляє у виробниче середовище. Конкретні інструменти розглядаються у підрозділі 1.4, порядок розгортання – у розділі 3.

У прикладному шарі буде застосовано CQRS – розмежування команд і запитів. Усі операції системи поділяються на дві категорії: команди змінюють стан, запити лише читають дані. Для кожного сценарію існує окремий об'єкт і окремий обробник (рис. 1.7).

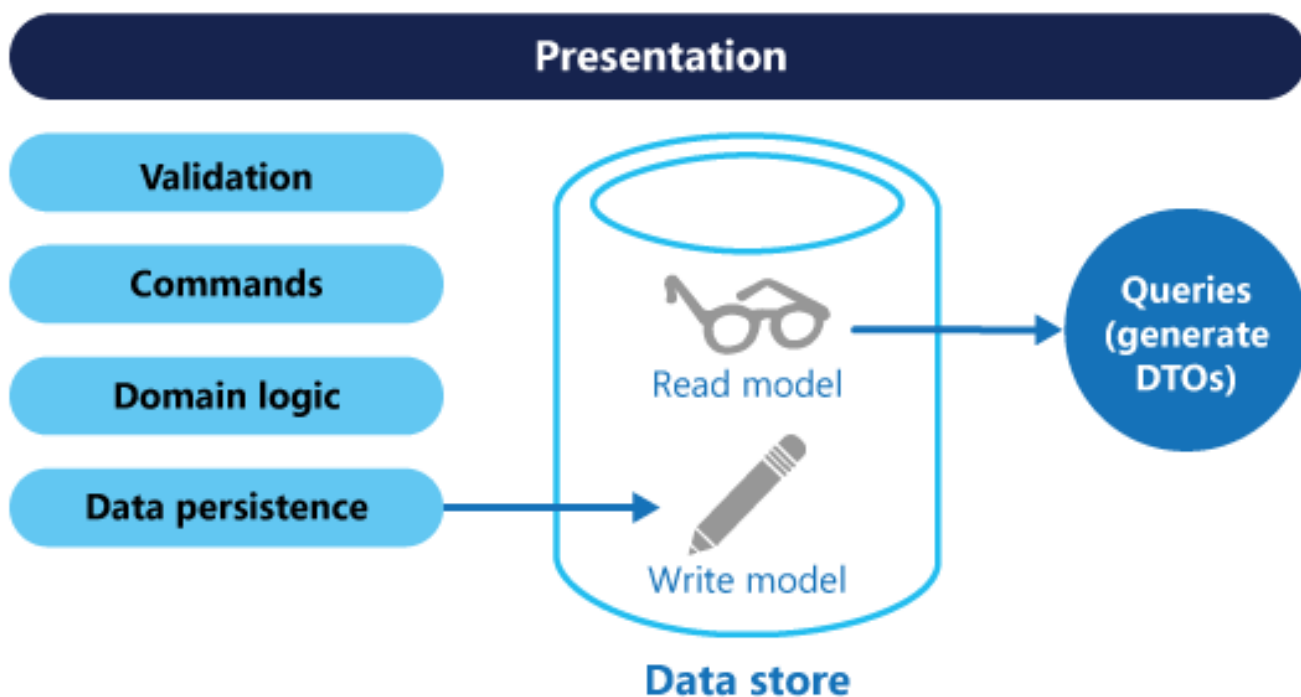


Рис. 1.7 – Схема патерну розмежування команд і запитів [6]

Альтернативою є сервісний шар, де один клас містить усі операції над сутністю. Підхід простий на старті, але з ростом кількості сценаріїв стає громіздким: залежності одного класу розростаються, одна зміна може зачепити непов'язаний сценарій, покриття тестами ускладнюється.

Для BuildFlow вибір CQRS продиктований конкретними обставинами. Різні ролі мають принципово різні патерни роботи з даними – менеджер переглядає аналітику, підрядник подає фотозвіт, інженер виносить рішення. Виокремлення цих операцій у самостійні обробники дозволяє зосередити логіку сценарію в одному місці. Запити й команди мають різні вимоги до оптимізації: запити часто потребують скороченої проєкції для швидкої передачі клієнту, команди – повного завантаження сутності для перевірки правил. Нарешті, кожен обробник тестується автономно, для цього достатньо підмінити репозиторій тестовим замінником. Детальна реалізація прикладного шару розглядається у розділі 2.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір технологічного стеку для системи BuildFlow базуватиметься на критеріях відповідності чистій архітектурі, зрілості інструментів та їхній нативній інтеграції з хмарною інфраструктурою.

Серверну частину буде реалізовано на платформі .NET 10 [17] (мова C# 13). Під час проєктування було відхилено Java Spring Boot (через складнішу інтеграцію з Azure) та Node.js (через відсутність строгої типізації, що є критичним недоліком для реалізації складної доменної логіки). Платформу .NET обрано завдяки статичній типізації та розвиненій екосистемі. Для оркестрації локального середовища розробки застосовано .NET Aspire, що дозволяє запускати всі мікросервіси та залежності єдиною командою без ручного конфігурування портів.

Для доступу до бази даних буде використано Entity Framework Core 10 [10]. Цей ORM–фреймворк забезпечує зручний механізм міграцій та підтримку LINQ без ручного написання SQL–запитів. Альтернативний мікро–ORM Dapper було відхилено: для даної системи надійність версіонування схеми має вищий пріоритет за мінімальний приріст продуктивності (табл. 1.2).

Фізичним сховищем даних буде обрано Microsoft SQL Server. Цей провайдер гарантує стовідсоткову сумісність локальних EF–міграцій із керованим хмарним середовищем Azure SQL Database.

Таблиця 1.2

Порівняння платформ серверної частини

Критерій	.NET 10 (C#)	Java Spring	Node.js
1	2	3	4
Строга статична типізація	Так	Так	Ні
Підтримка асинхронного програмування	Так	Частково	Так
Нативна інтеграція з Azure	Так	Ні	Ні

Продовження таблиці 1.2

1	2	3	4
Підтримка Clean Architecture	Так	Так	Частково
Безкоштовне використання	Так	Так	Так
Обрано	Так	Ні	Ні

Клієнтська частина буде реалізована з використанням React 19 [24] і TypeScript [31] 6. Angular як повноцінний фреймворк із жорсткою структурою виправдовує себе у великих командах, але надлишковий для системи такого масштабу. Vue.js простіший у старті, проте поступається за обсягом екосистеми та інструментами управління серверним станом. React буде обрано за гнучкість компонентної моделі і зрілу екосистему для побудови рольових інтерфейсів. Як збирач буде використовуватися Vite 8, він забезпечуватиме миттєвий старт середовища розробки і швидку збірку завдяки нативній підтримці ES-модулів (табл. 1.3).

Таблиця 1.3

Порівняння інструментів клієнтської частини

Критерій	React + TypeScript	Angular	Vue.js
Строга типізація	Так	Так	Частково
Компонентна архітектура	Так	Так	Так
Обсяг екосистеми	Високий	Середній	Середній
Гнучкість структури проєкту	Висока	Низька	Висока
Поріг входження	Середній	Високий	Низький
Обрано	Так	Ні	Ні

Для зберігання структурованих даних буде обрано PostgreSQL 16. MySQL 8 не підходить через слабшу підтримку складних аналітичних запитів, Microsoft SQL Server – через ліцензійні витрати, що є невиправданими для академічного проєкту. PostgreSQL закриває обидві проблеми: відкрита ліцензія, надійні транзакції, зручна інтеграція з EF Core через провайдер Npgsql (табл. 1.4). Крім того, вбудована підтримка типу даних JSONB дозволить зберігати напівструктуровані метадані проєктів, не перевантажуючи жорстку реляційну схему. Додатковою перевагою є механізм багатоверсійного керування

конкурентним доступом (MVCC). Це гарантуватиме стабільну продуктивність та відсутність блокувань бази даних під час одночасного оновлення статусів завдань різними учасниками процесу.

Таблиця 1.4

Порівняння систем управління базами даних

Критерій	PostgreSQL 16	MySQL 8	MS SQL Server
Відкрите програмне забезпечення	Так	Так	Ні
Підтримка складних запитів	Висока	Середня	Висока
Інтеграція з EF Core	Так	Так	Так
Безкоштовне використання	Так	Так	Ні
Хмарна підтримка Azure	Так	Так	Так
Обрано	Так	Ні	Ні

Інфраструктуру проєкту буде розгорнуто в екосистемі Microsoft Azure (рис. 1.8). Обчислювальні процеси делеговано Azure App Service, файли будуть зберігатися в Azure Blob Storage, а транзакційна інформація – в Azure Database for PostgreSQL. Використання Microsoft Entra ID зніме потребу в самостійній реалізації автентифікації, а фінансових квот підписки Azure for Students достатньо для всіх етапів розгортання.

Керування вихідним кодом та автоматизацію CI/CD буде реалізовано в GitLab [13]. Кожне оновлення ініціює статичний аналіз, модульні тести та інтеграційне тестування за допомогою Testcontainers [29], яке динамічно піднімає ізольовані екземпляри бази даних. Збій будь-якої перевірки автоматично блокує збірку релізних артефактів, унеможливаючи деплой дефектного коду.

Фінальне пакування буде здійснюватися засобами Docker [8]. Контейнеризація остаточно вирішує проблему несумісності середовищ, гарантуючи ідентичну роботу системи на локальній машині та цільових серверах. Зібрані та протестовані образи маркуються унікальними тегами. Після цього вони відправляються до захищеного реєстру, звідки хмарна служба Azure App Service автоматично підтягує актуальну конфігурацію під час кожного релізу.

Це, в майбутньому, дозволить масштабувати бекенд системи простим збільшенням кількості працюючих реплік.

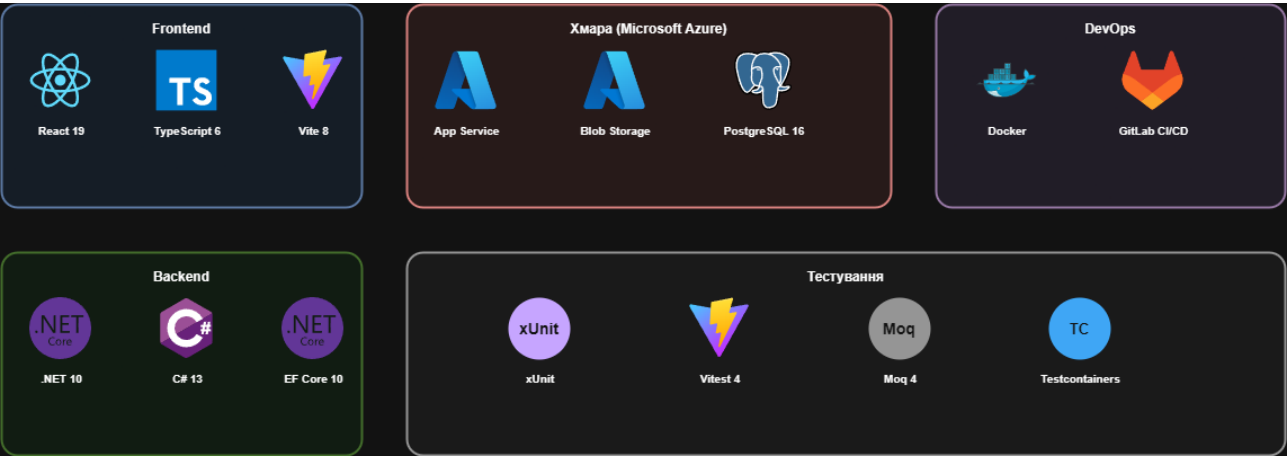


Рис. 1.8 – Технологічний стек системи BuildFlow

Вимоги до ресурсів доцільно розмежувати на дві частини: серверні та клієнтські [15]. Їхні профілі навантаження кардинально різняться. Бекенд безперервно тримає з'єднання з базою даних і сховищем медіафайлів. Паралельно він змушений обробляти потік запитів від усіх активних користувачів. Саме тому виділені під сервер потужності напряму диктують загальну швидкість відгуку застосунку.

Сам хостинг функціонує за моделлю PaaS. Це архітектурне рішення ліквідує рутину системного адміністрування. Хмарний провайдер самостійно оновлює операційну систему, застосовує критичні патчі безпеки та веде цілодобовий моніторинг стану інфраструктури (табл. 1.5). Такий підхід дозволяє повністю сфокусувати зусилля на реалізації доменної логіки, делегуючи управління базовим рівнем платформи.

Таблиця 1.5

Вимоги до ресурсів серверної частини

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Кількість ядер процесора	1	2
Обсяг оперативної пам'яті	1 ГБ	2 ГБ
Обсяг дискового простору	10 ГБ	32 ГБ
Пропускна здатність мережі	10 Мбіт/с	100 Мбіт/с
Операційна система	Linux Ubuntu 22.04+	Linux Ubuntu 22.04+

Мінімальній конфігурації відповідає тарифний план App Service B1 (1 vCPU, 1.75 ГБ RAM), рекомендованій – B2 (2 vCPU, 3.5 ГБ RAM). Для академічного середовища з обмеженою кількістю одночасних сесій план B1 є цілком прийнятним варіантом.

Дисковий простір використовується для логів і тимчасових файлів – самі фотодокази та креслення зберігаються у Azure Blob Storage окремо, тому вимога у 10 ГБ є консервативною оцінкою з достатнім запасом. База даних PostgreSQL [22] розгортається на окремому ресурсі Azure Database for PostgreSQL Flexible Server і масштабується незалежно від App Service.

Клієнтський рівень системи ізольовано у площині веббраузера. Кінцевому користувачу не потрібно завантажувати або встановлювати жодні локальні пакети (табл. 1.6). Фізичний пристрій виконує функцію звичайного термінала. Його апаратні ресурси витрачаються суворо на відмальовування інтерфейсних компонентів та прийом готових відповідей від API [21].

Таблиця 1.6

Вимоги до клієнтського пристрою

Параметр	Вимога
Браузер	Google Chrome 90+, Mozilla Firefox 88+, Microsoft Edge 90+
Обсяг оперативної пам'яті	від 2 ГБ
Швидкість інтернет-з'єднання	від 1 Мбіт/с
Роздільна здатність екрану	від 1280 × 720 пікселів
Підтримка JavaScript	Увімкнена

Перелічені браузери підтримують Fetch API та File API, необхідні для роботи із запитами і завантаження фотодоказів. Safari не включено через відмінності у реалізації ряду вебстандартів без додаткового тестування. Роздільна здатність від 1280 × 720 обумовлена версткою інтерфейсу – таблиця завдань і панель фільтрів при меншій ширині відображаються у скороченому вигляді. Швидкість з'єднання від 1 Мбіт/с достатня для текстових даних; при завантаженні фото розміром 3–5 МБ рекомендоване з'єднання від 10 Мбіт/с забезпечить суттєво кращий досвід. Загалом стабільність інтернет-каналу в цьому контексті є критичнішою за його ширину. Враховуючи специфіку

будівельних майданчиків, у разі раптових обривів мережі клієнтська частина просто призупиняє передачу важких медіафайлів до моменту відновлення повноцінного зв'язку.

Висновки до першого розділу

Аналіз існуючих рішень виявив характерну закономірність: загальні інструменти: Trello, Jira не мають жодного контексту про будівельний процес і потребували б значного ручного налаштування для адаптації до предметної області. Procore розуміє специфіку галузі, але орієнтований на великі організації з відповідним бюджетом і функціональною надлишковістю. Microsoft Project закриває лише задачу планування, залишаючи весь подальший цикл виконання поза увагою. Вбудованого механізму приймання виконаних робіт із фотодоказами та чітким розподілом ролей між менеджером, підрядником та інженером немає в жодному з розглянутих продуктів.

Процес проєктування архітектури базувався виключно на специфіці будівельного майданчика. Інструменти, продиктовані виключно сучасними ІТ-трендами, свідомо ігнорувалися. Фізична ізоляція серверної та клієнтської складових усуває їхню взаємозалежність і гарантує подальшу автономність модулів. Чиста архітектура з інверсією залежностей дозволяє тестувати доменну логіку без підняття бази даних. Патерн розмежування команд і запитів ізолює кожен сценарій використання в окремий обробник.

Технологічний стек – .NET 10, React 19, PostgreSQL 16, Azure – підібраний під цю архітектуру і утворює зв'язну екосистему без надлишкового налаштування. Цей технологічний фундамент остаточно закриває етап високорівневого планування. Безпосередня розробка алгоритмів та внутрішніх схем даних розкривається в наступному підрозділі.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ

2.1 Визначення варіантів використання та об'єктно—орієнтованої структури системи

Проектування системи розпочато з визначення акторів і сценаріїв їх взаємодії, без цього будь—які рішення щодо структури даних є передчасними.

Система обслуговує п'ять категорій учасників із різними наборами прав (табл. 2.1). Адміністратор і менеджер мають схожий доступ до управління об'єктами, відмінність лише в тому, що адміністратор також керує обліковими записами. Підрядник та інженер працюють виключно в контексті конкретного завдання і не мають доступу до функцій планування.

Таблиця 2.1

Актори системи

Актор	Роль у процесі	Основні дії в системі
Адміністратор	Управління системою	Реєстрація користувачів, управління об'єктами, перегляд усіх даних
Менеджер	Планування та контроль	Створення об'єктів, планування робіт, призначення завдань
Підрядник	Виконання завдань	Перегляд завдань, зміна статусу, подання звіту з фотодоказами
Інженер	Технічний контроль	Перегляд звітів, прийом або відхилення виконаної роботи

Функціональні вимоги декомпозовано на дев'ять ізольованих прецедентів (табл. 2.2). Цей масив повністю охоплює логіку взаємодії з платформою. Стартовою операцією є реєстрація профілю, кінцевою – технічний аудит об'єкта. Доступ до ініціалізації тригерів суворо розподілено між ролями. Менеджерський рівень формує план та делегує задачі. Підрядник фіксує фізичний прогрес і завантажує звітність. Інженер володіє ексклюзивним правом на верифікацію етапів. Візуалізація цієї архітектури у форматі UML—діаграми [32] міститься в додатку А.

Варіанти використання системи BuildFlow

№	Варіант використання	Актори	Короткий опис
1	Автентифікація	Усі	Реєстрація та вхід до системи з видачею маркера доступу
2	Управління будівельними об'єктами	Адміністратор, Менеджер	Створення, редагування та перегляд будівельних об'єктів
3	Управління плановими роботами	Адміністратор, Менеджер	Додавання, редагування планових робіт, прикріплення документації
4	Призначення завдання підряднику	Менеджер	Вибір підрядника та переведення планової роботи в активне завдання
5	Виконання завдання	Підрядник	Підтвердження початку роботи, зміна статусу завдання
6	Подання звіту про виконання	Підрядник	Завантаження фотодоказів та текстового коментаря
7	Рецензування поданого звіту	Інженер	Перегляд фотодоказів, прийом або відхилення роботи з коментарем
8	Реєстрація інциденту	Усі	Фіксація позаштатної ситуації на об'єкті
9	Перегляд аналітики та сповіщень	Менеджер, Інженер, Інвестор	Зведені показники по об'єктах і завданнях, стрічка подій

З варіантів використання безпосередньо впливає об'єктно–орієнтована модель предметної області [9] (рис. 2.1). Центральним елементом є будівельне завдання, навколо якого обертається більша частина взаємодій між акторами.

WorkTask – головна сутність і кореневий агрегат. Управляє власним станом через методи, кожен з яких перевіряє допустимість переходу та роль ініціатора. Містить часовий проміжок виконання, посилання на об'єкт і призначеного підрядника, колекцію поданих звітів і прикріплених креслень.

ConstructionObject – фізичний об'єкт будівництва, у межах якого створюються завдання. Містить перелік запланованих робіт і відомості про відповідальних осіб.

PlannedWorkItem – запис про заплановану роботу. Слугує прообразом завдання: коли менеджер призначає планову роботу підряднику, на її основі створюється конкретне WorkTask.

User – обліковий запис учасника з однією або кількома ролями, що визначають доступний функціонал.

Incident – фіксація позаштатної ситуації на об'єкті з власним циклом станів, незалежним від циклу завдання.

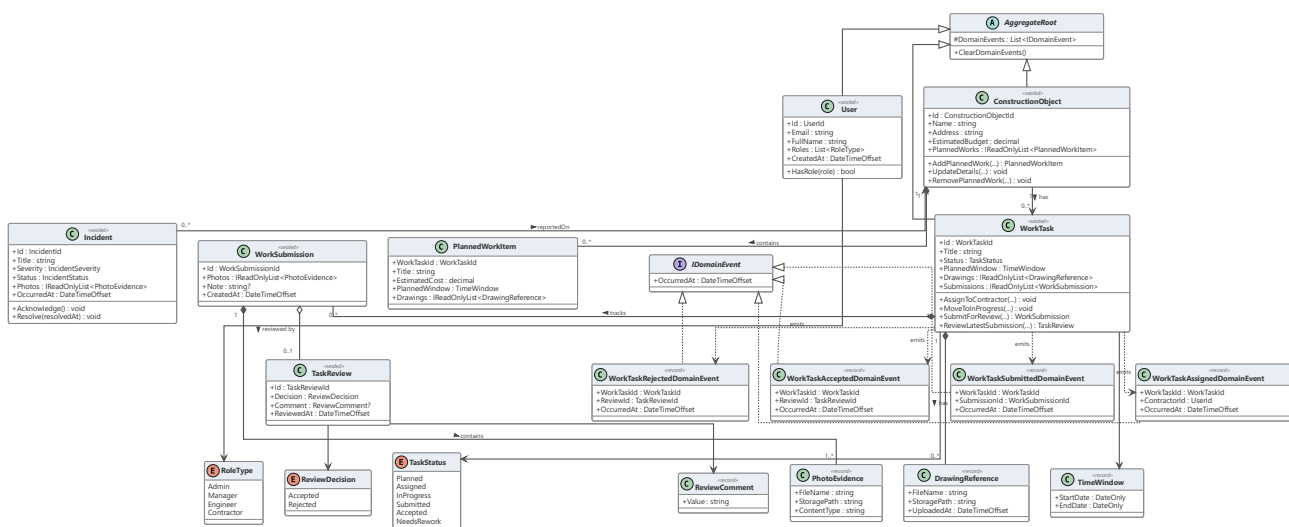


Рис. 2.1 – Діаграма класів предметної області системи BuildFlow

Окрім сутностей, модель містить об'єкти-значення – незмінні структури без власного ідентифікатора (табл. 2.3).

Таблиця 2.3

Об'єкти–значення предметної області

Об'єкт—значення	Призначення
1	2
WorkTaskDefinition	Групує назву, опис, часовий проміжок і перелік креслень завдання
TimeWindow	Незмінний проміжок часу; перевіряє, що дата завершення не передує даті початку
DrawingReference	Метадані прикріпленого файлу проєктної документації
PhotoEvidence	Метадані фотодокументу; перевіряє належність типу вмісту до зображень
ReviewComment	Текст коментаря до рецензії з обмеженням довжини
ReviewDecision	Перелічуваний тип: «Прийнято» або «Відхилено»

1	2
RoleType	Перелічуваний тип ролі: Адмін, Менеджер, Інженер, Підрядник
TaskStatus	Перелічуваний тип стану завдання

Ідентифікатори сутностей реалізовані як строго типізовані структури: WorkTaskId, UserId, ConstructionObjectId. Передати ідентифікатор одного типу туди, де очікується інший, неможливо фізично – компілятор відхилить такий код. На практиці це усуває цілий клас помилок, які в системах із безіменними Guid-ами проявляються лише у runtime.

Описана структура є прямим відображенням реального процесу будівельного підряду і слугує основою для проєктування схеми бази даних у наступному підрозділі.

2.2 Розробка бази даних системи

Реляційна схема бази даних є прямим наслідком доменної моделі, визначеної в попередньому підрозділі. Кожен агрегатний корінь отримав власну таблицю, вкладені об'єкти–значення або зберігаються як стовпці того самого рядка, або серіалізуються у формат JSONB – залежно від того, чи потрібна по них адресна SQL–вибірка.

Концептуальну структуру даних відображає ER–діаграма (рис. 2.2). Вона охоплює п'ять основних таблиць та зв'язки між ними. Таблиця «work_tasks» є центральною: вона посилається на «construction_objects» через зовнішній ключ construction_object_id і на users двічі через assigned_contractor_id і created_by_id, що відображає розподіл ролей між виконавцем і менеджером, який створив завдання. На рівні фізичної схеми для цих відношень налаштовано жорсткі обмеження цілісності, які алгоритмічно блокують видалення об'єктів або користувачів за наявності прив'язаних до них активних завдань. Для прискорення виконання агрегаційних запитів та пошуку за ідентифікаторами виконавців на відповідні колонки накладено некластерні індекси.

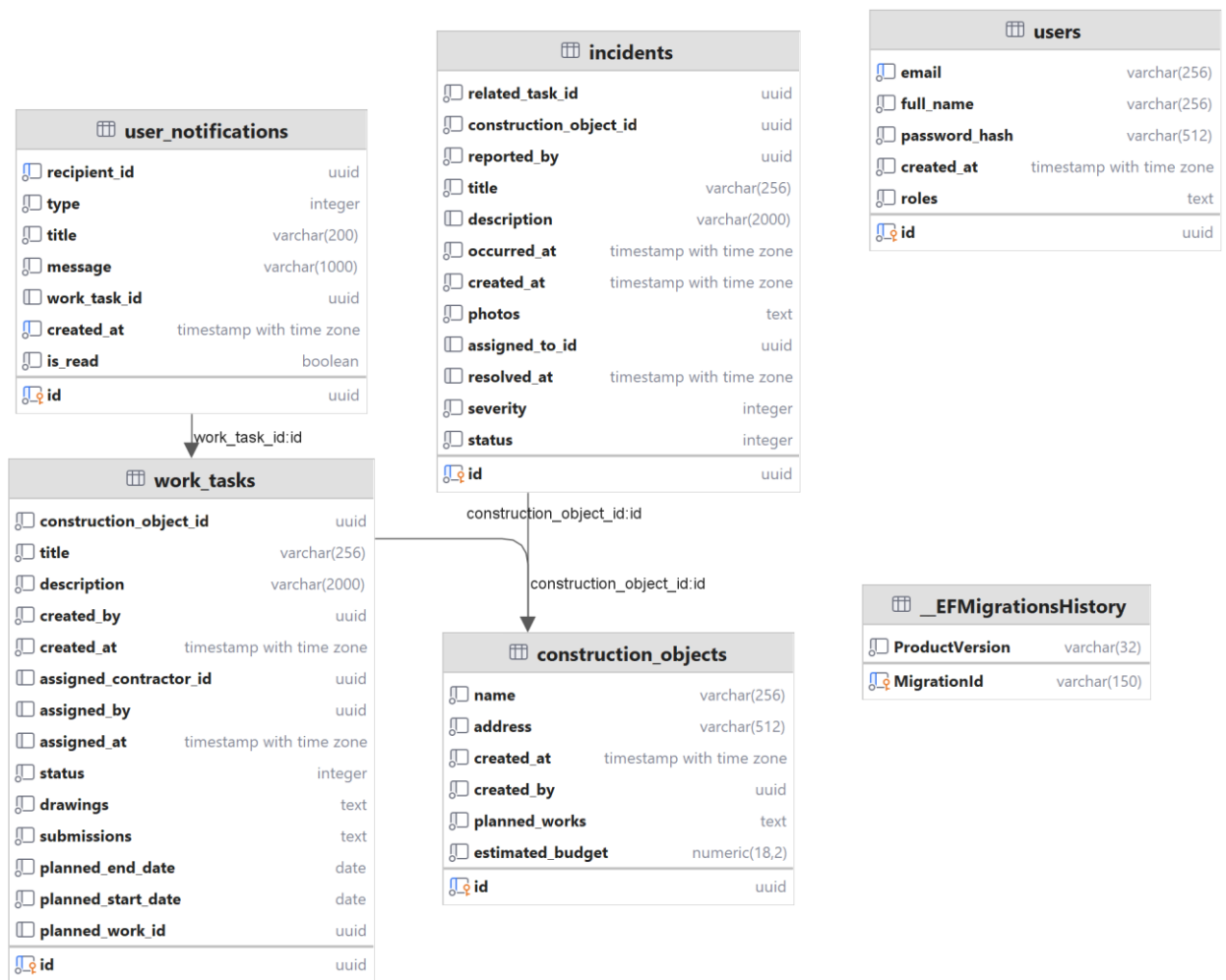


Рис. 2.2 – ER–діаграма бази даних

Нижче наведено структуру кожної таблиці із зазначенням типів даних, обмежень і призначення стовпців.

Для забезпечення рольової моделі доступу та надійної ідентифікації учасників будівельного процесу використовується таблиця «users». Вона інкапсулює облікові записи, криптографічні хеші паролів та відповідні рівні привілеїв у форматі JSON-масиву, що гарантує безпеку та чітке розмежування прав на рівні всієї системи (табл. 2.4).

Таблиця 2.4

Структура таблиці «users»

Стовпець	Тип	Обмеження	Опис
1	2	3	4
id	uuid	PK	Унікальний ідентифікатор користувача

Продовження таблиці 2.4

1	2	3	4
email	varchar(256)	NOT NULL, UNIQUE	Електронна адреса
full_name	varchar(256)	NOT NULL	Повне ім'я
password_hash	varchar(512)	NOT NULL	Хеш паролю
roles	text	NOT NULL	JSON–масив ролей
created_at	timestampz	NOT NULL	Дата реєстрації

Фізичні майданчики та пов'язана з ними базова інфраструктурна інформація агрегуються в таблиці «construction_objects». Ця сутність виступає основним контекстом для фінансових показників та інтегрованого масиву планових робіт, об'єднуючи їх у єдину структуру даних для подальшого управління (табл. 2.5).

Таблиця 2.5

Структура таблиці «construction_objects»

Стовпець	Тип	Обмеження	Опис
id	uuid	PK	Унікальний ідентифікатор об'єкта
name	varchar(256)	NOT NULL	Назва об'єкта
address	varchar(512)	NOT NULL	Адреса
estimated_budget	numeric(18,2)	NOT NULL	Кошторисна вартість
created_by	uuid	NOT NULL, FK → users	Ідентифікатор автора запису
created_at	timestampz	NOT NULL	Дата створення
planned_works	text	NOT NULL	JSON–масив планових робіт

Ядром доменної логіки системи є таблиця «work_tasks». Вона фіксує не лише базові метадані, а й повний життєвий цикл виконання роботи від призначення конкретного підрядника до збереження масиву графічних фотодоказів і звітів у вигляді нереляційних JSON-структур (табл. 2.6).

Таблиця 2.6

Структура таблиці «work_tasks»

Стовпець	Тип	Обмеження	Опис
1	2	3	4
id	uuid	PK	Унікальний ідентифікатор завдання

Продовження таблиці 2.6

1	2	3	4
construction_object_id	uuid	NOT NULL, FK → construction_objects	Належність до об'єкта
planned_work_id	uuid	NULL	Посилання на вихідну планову роботу
title	varchar(256)	NOT NULL	Назва завдання
description	varchar(2000)	NOT NULL	Опис
status	int	NOT NULL	Стан завдання (TaskStatus)
planned_start_date	date	NOT NULL	Планова дата початку
planned_end_date	date	NOT NULL	Планова дата завершення
created_by	uuid	NOT NULL, FK → users	Автор завдання
created_at	timestampz	NOT NULL	Дата створення
assigned_contractor_id	uuid	NULL, FK → users	Призначений підрядник
assigned_by	uuid	NULL, FK → users	Хто призначив
assigned_at	timestampz	NULL	Дата призначення
drawings	text	NOT NULL	JSON–масив креслень
submissions	text	NOT NULL	JSON–масив звітів виконання

Для моніторингу відхилень від нормального технологічного процесу спроектовано таблицю «incidents». Її структура забезпечує оперативну реєстрацію, категоризацію за рівнем критичності та відстеження процесу усунення позаштатних ситуацій, що виникають на об'єктах або під час виконання конкретних завдань (табл. 2.7).

Таблиця 2.7

Структура таблиці «incidents»

Стовпець	Тип	Обмеження	Опис
1	2	3	4
id	uuid	PK	Унікальний ідентифікатор інциденту
construction_object_id	uuid	NOT NULL, FK → construction_objects	Об'єкт, де стався інцидент
related_task_id	uuid	NOT NULL, FK → work_tasks	Пов'язане завдання
reported_by	uuid	NOT NULL, FK → users	Хто зафіксував
assigned_to_id	uuid	NULL, FK → users	Відповідальний за усунення
title	varchar(256)	NOT NULL	Короткий опис

Продовження таблиці 2.7

1	2	3	4
description	varchar(2000)	NULL	Детальний опис
severity	int	NOT NULL	Рівень критичності
status	int	NOT NULL	Стан інциденту
occurred_at	timestampz	NOT NULL	Час виникнення
created_at	timestampz	NOT NULL	Час реєстрації в системі
resolved_at	timestampz	NULL	Час усунення
photos	text	NOT NULL	JSON–масив фотодоказів

Асинхронне інформування учасників про зміну статусів завдань, призначення нових робіт чи появу критичних інцидентів реалізується через таблицю «user_notifications». Вона виконує функцію персоналізованого журналу подій, що забезпечує своєчасну реакцію користувачів на зміни в системі та підтримує безперервність робочого процесу (табл. 2.8).

Таблиця 2.8

Структура таблиці «user_notifications»

Стовпець	Тип	Обмеження	Опис
id	uuid	PK	Унікальний ідентифікатор сповіщення
user_id	uuid	NOT NULL, FK → users	Отримувач
message	text	NOT NULL	Текст сповіщення
is_read	boolean	NOT NULL	Ознака прочитання
created_at	timestampz	NOT NULL	Дата створення

Таблиці «construction_objects» і «work_tasks» містять стовпці типу JSONB із серіалізованими масивами вкладених об'єктів: planned_works і drawing_references у першій, submissions із вкладеними photo_evidences у другій. Такий підхід відповідає принципу агрегатної межі: ці об'єкти не мають власної ідентичності поза агрегатом, читаються завжди разом із батьківським записом і не підлягають адресній вибірці. Виносити їх в окремі таблиці означало б ускладнити схему без жодної практичної вигоди.

Схема відповідає третій нормальній формі в частині основних сутностей. Серіалізовані JSONB–стовпці є свідомим відступом від класичної нормалізації –

виправданим там, де дані є частиною агрегату і ніколи не запитуються окремо. Перелічувані типи станів зберігаються як цілі числа, що усуває потребу в довідникових таблицях для незмінних переліків.

Для прискорення типових вибірок визначено індекси на стовпцях із найвищою селективністю: зовнішні ключі `work_tasks.construction_object_id` і `work_tasks.assigned_contractor_id`, а також `work_tasks.status` для фільтрації завдань за поточним станом.

Схему сформовано за підходом Code First засобами EF Core 10. Кожному класу домену відповідає окремий клас конфігурації, що описує назви таблиць і стовпців, обмеження довжини та перетворення типів, без жодних анотацій безпосередньо на класах домену. Такий поділ зберігає доменну модель незалежною від інфраструктурних деталей. Зміни схеми фіксуються у міграційних файлах і застосовуються автоматично при запуску застосунку – початковий стан бази створюється першою міграцією, кожна наступна додає лише інкрементальні зміни.

2.3 Проєктування та реалізація алгоритмів роботи системи

Схема даних описує що зберігається, але не як система обробляє запити. У цьому підрозділі розглянуто три ключові алгоритми та автомат станів завдання як центральну бізнес–логіку системи.

Ініціалізація сесії відбувається через API–контролер. Вхідна пара облікових даних транслюється до ізольованого сервісу ідентифікації. Алгоритм порівнює хеш–суми безпосередньо з таблицею користувачів. Будь–яка невідповідність миттєво відсікається HTTP–статусом 401 Unauthorized. У разі успішної перевірки бекенд випускає криптографічно підписаний JWT–токен [1]. Його корисне навантаження одразу містить унікальний ідентифікатор та рольовий рівень доступу. Клієнтський додаток кешує цей маркер локально у браузері. Надалі кожна транзакція з сервером автоматично підписується через стандартний заголовок Authorization (рис. 2.3).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

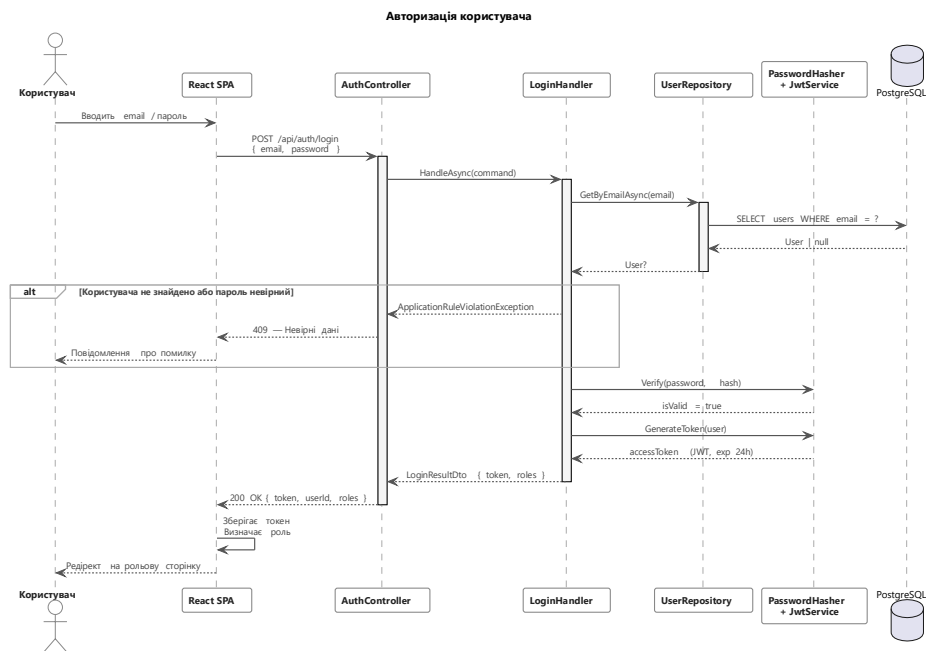


Рис. 2.3 – Діаграма послідовності автентифікації користувача

Центральною бізнес-логікою системи є автомат станів будівельного завдання. Завдання проходить через шість станів (табл. 2.9), і переходи між ними відбуваються виключно через методи сутності WorkTask, кожен перевіряє поточний стан і роль ініціатора перед тим як дозволити перехід. Стан «Прийнято» є кінцевим. Стан «Потребує доопрацювання» цікавий тим, що відкриває цикл повторного подання: підрядник прикріплює новий звіт, завдання знову переходить у стан перевірки.

Таблиця 2.9

Переходи станів будівельного завдання

Поточний стан	Подія	Ініціатор	Новий стан
Заплановано	Призначення підрядника	Менеджер	Призначено
Призначено	Підтвердження початку	Підрядник	Виконується
Виконується	Подання звіту	Підрядник	Подано на перевірку
Подано на перевірку	Прийняття роботи	Інженер	Прийнято
Подано на перевірку	Відхилення роботи	Інженер	Потребує доопрацювання
Потребує доопрацювання	Повторне подання	Підрядник	Подано на перевірку

При поданні звіту підрядник прикріплює фотодокази і необов'язковий коментар. Обробник одразу перевіряє два моменти: щонайменше одне зображення має бути прикріплене, і MIME–тип кожного файлу має відповідати формату зображення. Файли інших форматів відхиляються тут, не доходячи до сховища. Прийняті файли завантажуються, їхні шляхи фіксуються у записі звіту, після чого обробник завантажує завдання з бази і викликає метод подання. Усі зміни запис звіту і зміна стану завдання зберігаються єдиною транзакцією. Повну послідовність наведено в додатку Е.

Рецензування влаштоване простіше: інженер переглядає фотодокази і обирає одне з двох рішень. Відхилення без коментаря валідатор блокує ще до звернення до бази, а порожній коментар при негативному рішенні є помилкою на рівні вхідних даних, а не доменної логіки. Обробник викликає метод рецензування на сутності, що фіксує рішення і змінює стан: прийнято переводить у «Прийнято», відхилено – у «Потребує доопрацювання». В обох випадках підрядник отримує сповіщення. Діаграму послідовності наведено в додатку Є.

Кожен із описаних алгоритмів реалізовано окремим обробником команди в шарі застосунку. Повну структуру шару з переліком усіх обробників наведено в додатку Г, структуру інфраструктурного шару – в додатку Д.

2.4 Реалізація інформаційної системи моніторингу та управління будівельними проектами

Реалізація серверної частини будується навколо кількох принципів, закладених ще на архітектурному рівні: кожен сценарій є самостійною одиницею, вхідні дані перевіряються до того як потрапити до логіки, контролер не знає нічого про бізнес–правила.

Кожна дія користувача реалізована як окрема команда – незмінний запис із даними, необхідними для виконання операції (структуру команди подання звіту наведено в додатку Ж.1). Поведінки команда не містить: її єдина функція це передати дані від контролера до обробника. Обробник має єдиний відкритий

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

метод і виконує рівно один сценарій, не знаючи про існування суміжних операцій.

Відповідний обробник починається з виклику валідатора [12], лише після успішної перевірки відбуваються звернення до бази і зміни стану сутності (фрагмент коду наведено в додатку Ж.2). Якщо команда не пройшла структурну перевірку, обробник завершується на першому рядку і жодних побічних ефектів не виникає. Далі обробник завантажує завдання зі сховища, якщо запис не знайдено, одразу повертається виняток з кодом 409. Знайдене завдання передається доменному методу разом із сформованими об'єктами фотодоказів: перетворення вхідних даних на доменні типи відбувається в обробнику, а не всередині сутності. Усі зміни зберігаються єдиною транзакцією через UnitOfWork [5, 7, 26] або повністю, або не зберігаються взагалі. Повний алгоритм взаємодії з компонентами наведено в додатку Е.

Для кожної команди визначено окремий валідатор, розташований поруч із нею у тій самій папці (приклад наведено в додатку Ж.3). Таке розташування не випадкове: зміна команди одразу вимагає перегляду валідатора, вони еволюціонують разом. Валідатор відповідає за структурну відповідність: чи заповнені поля, чи відповідає формат. Доменна відповідає за логіку чи має підрядник право подавати звіт саме по цьому завданню, яке перевіряється всередині доменного методу. Практичний наслідок: помилка валідації повертає 400 із переліком порушень, порушення доменного правила – 422. Валідатори реєструються автоматично через сканування збірки без ручного перерахування.

Контролер приймає запит, формує команду і передає її обробнику. Більше він нічого не робить (фрагмент коду наведено в додатку Ж.4). Атрибут авторизації обмежує доступ на рівні HTTP, але перевірка ролі всередині доменного методу відбувається незалежно.

Обробку помилок централізовано в єдиному глобальному обробнику винятків, що перетворює всі необроблені помилки на відповіді у форматі RFC 7807 [20]. Жоден контролер не містить try-catch, а помилки вільно поширюються

до єдиної точки, формат відповіді однаковий незалежно від місця виникнення винятку (табл. 2.10).

Таблиця 2.10

Відповідність типів винятків кодам HTTP–відповіді

Тип винятку	Код відповіді	Умова виникнення
Порушення правил валідації	400	Некоректні або неповні вхідні дані
Порушення бізнес–правила застосунку	409	Недопустима операція над сутністю
Порушення інваріанту предметної області	422	Перехід стану, заборонений доменною логікою
Необроблений виняток	500	Внутрішня помилка сервера

Висновки до другого розділу

Другий розділ охопив проєктування і реалізацію системи від визначення акторів до конкретних рішень на рівні коду.

Центральним результатом моделювання стало будівельне завдання з власним автоматом станів. Підрядник не може подати звіт по завданню яке ще не призначено, інженер не може винести рішення без поданого звіту. Ці обмеження живуть у доменному коді, а не в домовленостях між розробниками. Решта сутностей – будівельний об'єкт, звіт виконання, рецензія – спроектована навколо цього центрального елемента і відображає реальні ролі учасників процесу.

Схема бази даних відображає доменну модель безпосередньо: кожен агрегатний корінь отримав власну таблицю. Вкладені колекції серіалізовані у JSONB, це рішення було свідомим відхиленням від нормалізації [27] там де реляційна структура додала б складності без практичної вигоди. Міграції версіонуються і застосовуються автоматично, що усуває розходження між кодом і схемою бази даних.

Серверна реалізація розмежовує відповідальності між рівнями так, що кожен тестується окремо: контролер не знає про бізнес–правила, обробник не знає про HTTP, доменний метод не знає про базу даних. Патерн розмежування команд і запитів ізолює кожен сценарій використання в окремий обробник – зміна одного не зачіпає інші.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІЙНОЮ СИСТЕМОЮ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ

3.1 Порядок встановлення та налаштування параметрів системи

Система BuildFlow розгортається у хмарному середовищі Azure, а процес від коміту до появи оновленої версії у виробництві повністю автоматизований. Розробник не виконує жодних ручних дій для публікації – достатньо внести зміни до основної гілки репозиторію. Решту бере на себе конвеєр [13].

Система використовує п'ять хмарних служб (табл. 3.1). Кожна відповідає за окремий аспект роботи і може масштабуватись незалежно від решти.

Таблиця 3.1

Хмарна інфраструктура системи

Служба	Платформа	Призначення
Azure App Service	Microsoft Azure	Хостинг серверної частини у контейнері
Azure Static Web (Blob Storage)	Microsoft Azure	Хостинг клієнтської частини
Azure Database for PostgreSQL	Microsoft Azure	Реляційна база даних
Azure Blob Storage	Microsoft Azure	Сховище файлів (фотодокази, креслення)
Docker Hub	Docker Inc.	Реєстр образів контейнерів

Клієнтський застосунок функціонує в умовах суворої ізоляції. Будь-які прямі транзакції з браузера до бази даних чи файлового сховища фізично заблоковані на рівні архітектури. Єдиною дозволеною точкою взаємодії виступає серверне API на базі App Service. Саме цей вузол бере на себе всю оркестрацію: він керує структурованими записами у PostgreSQL та маніпулює бінарними об'єктами через Blob Storage. Маршрут доставки готового коду проходить через транзитний хаб – Docker Hub. CI-конвеєр компілює образ і публікує його в реєстр, після чого середовище Azure отримує автоматичну команду на розгортання оновленого контейнера.

Серверна частина пакується у Docker-образ із використанням багатоетапної збірки. На першому етапі виконується компіляція у середовищі з повним .NET SDK. На другому – до фінального образу копіюються лише скомпільовані артефакти, без вихідного коду і засобів збірки. Такий підхід зменшує розмір кінцевого образу і виключає потрапляння зайвих інструментів у виробниче середовище. Базовим образом є `mcr.microsoft.com/dotnet/aspnet` мінімальний образ із середовищем виконання .NET.

Конвеєр безперервної доставки реалізовано засобами GitLab CI/CD [13] (рис. 3.1). Механізм розбито на п'ять лінійних етапів із жорсткою залежністю. Логіка пайплайну не допускає компромісів: провал модульних тестів миттєво обриває процес збірки, а будь-яка помилка при генерації артефактів автоматично скасовує розгортання. Додатково налаштовано сувору фільтрацію за гілками репозиторію. Стадії контейнеризації та фінального деплою ініціюються виключно під час інтеграції змін в основну гілку. Усі інші, робочі гілки проходять виключно фазу аудиту якості та ізольованого тестування, без жодного впливу на виробниче середовище.

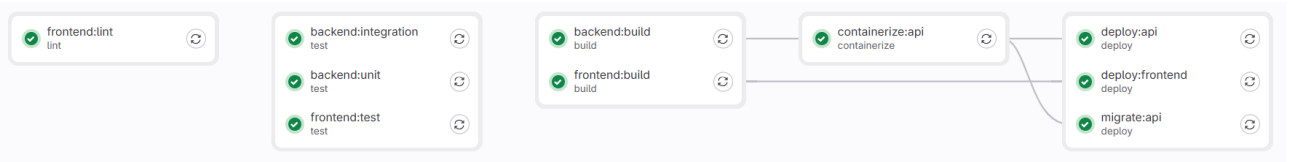


Рис. 3.1 – Діаграма конвеєра CI/CD

Таблиця 3.2

Стадії конвеєра

Стадія	Завдання	Зміст
1	2	3
lint	frontend:lint	Перевірка стилю коду клієнтської частини
test	frontend:test	Модульні тести клієнтської частини
test	backend:unit	Модульні тести серверної частини
test	backend:integration	Інтеграційні тести з реальною базою даних
build	frontend:build	Збірка статичних файлів клієнтської частини
build	backend:build	Компіляція та публікація серверної частини
containerize	containerize:api	Збірка Docker-образу та публікація до реєстру

Продовження таблиці 3.2

1	2	3
deploy	migrate:api	Застосування міграцій до бази даних
deploy	deploy:api	Оновлення образу на Azure App Service
deploy	deploy:frontend	Завантаження файлів до Azure Blob Storage

Стадія `lint` перевіряє відповідність коду встановленим правилам форматування і виявляє потенційно небезпечні конструкції ще до запуску тестів. Модульні тести серверної частини перевіряють доменну логіку і обробники команд без підключення до бази даних. Інтеграційні тести виконуються проти реальної бази у тимчасовому контейнері а не за допомогою моків. Стадія міграцій перед розгортанням гарантує, що нова версія коду завжди отримує актуальну схему бази даних.

Чутливими параметрами конфігурації є рядок підключення до бази, ключ підпису токенів, облікові дані хмарних служб не зберігаються у вихідному коді. Вони передаються через захищені змінні GitLab CI/CD і встановлюються безпосередньо на Azure App Service [2] під час розгортання (табл. 3.3). Захищені змінні не відображаються у логах конвеєру навіть при явному виведенні, тому що платформа автоматично маскує їх значення. Під час розгортання конвеєр передає ці змінні до Azure через CLI-команду оновлення конфігурації, після чого вони стають доступними застосунку як змінні середовища операційної системи.

Таблиця 3.3

Параметри конфігурації виробничого середовища

Параметр	Призначення
ConnectionStrings_BuildFlow	Рядок підключення до хмарної бази даних
Jwt_SecretKey	Ключ підпису токенів доступу
FileStorage_Provider	Режим сховища файлів (AzureBlob або MockLocal)
AzureStorage_ConnectionString	Рядок підключення до сховища файлів
AzureStorage_ContainerName	Назва контейнера у сховищі файлів

Перемикання між локальним і хмарним режимами зберігання відбувається через один параметр – `FileStorage_Provider`. Для локального середовища використовується `MockLocal`, для виробничого – `AzureBlob`. Код серверної

частини при цьому не змінюється, а різна поведінка досягається через підміну реалізації інтерфейсу в контейнері залежностей при старті застосунку.

Для розробки і тестування система запускається локально без підключення до хмарних служб, для цього достатньо Docker і Docker Compose [8].

```
docker compose -f docker-compose.local.yml up --build
```

Команда запускає три контейнери, що утворюють повну локальну копію системи (табл. 3.4).

Таблиця 3.4

Сервіси локального середовища

Контейнер	Образ	Порт	Призначення
buildflow-api	buildflow/api:local	5000	Серверна частина
buildflow-web	buildflow/web:local	3000	Клієнтська частина
buildflow-db	postgres:16	5432	База даних PostgreSQL

Процес локального розгортання ініціює автоматичне виконання міграцій бази даних. Одразу після формування структури таблиць клієнтський інтерфейс активується за адресою <http://localhost:3000>. Взаємодія з бекендом маршрутизується через шлюз на порту 8080. Таке архітектурне рішення перетворює робочу станцію на точну структурну копію бойового сервера. Ризик виникнення розбіжностей між середовищами нівелюється повністю. Обробка медіаконтенту також відбувається автономно. Система примусово відсікає будь-які запити до хмарних провайдерів. Натомість бінарний потік спрямовується безпосередньо на фізичний накопичувач і фіксується у каталозі uploads/. Щодо демонтажу інфраструктури, то повне знищення контейнерів разом із прив'язаними томами ініціюється єдиним консольним запитом:

```
docker compose -f docker-compose.local.yml down -v
```

Після повторного запуску міграції застосовуються заново і система починає роботу з порожньою базою даних.

3.2 Структура інтерфейсу інформаційної системи моніторингу та управління будівельними проєктами

Робота з системою починається зі сторінки входу (рис. 3.2). Екран розділено на дві частини: зліва розташовано короткий опис призначення системи і основних сценаріїв використання, справа – форма з полями електронної пошти і пароля. Новий користувач ще до введення даних отримує уявлення про те, з чим він матиме справу. Перехід до форми реєстрації здійснюється без перезавантаження сторінки.

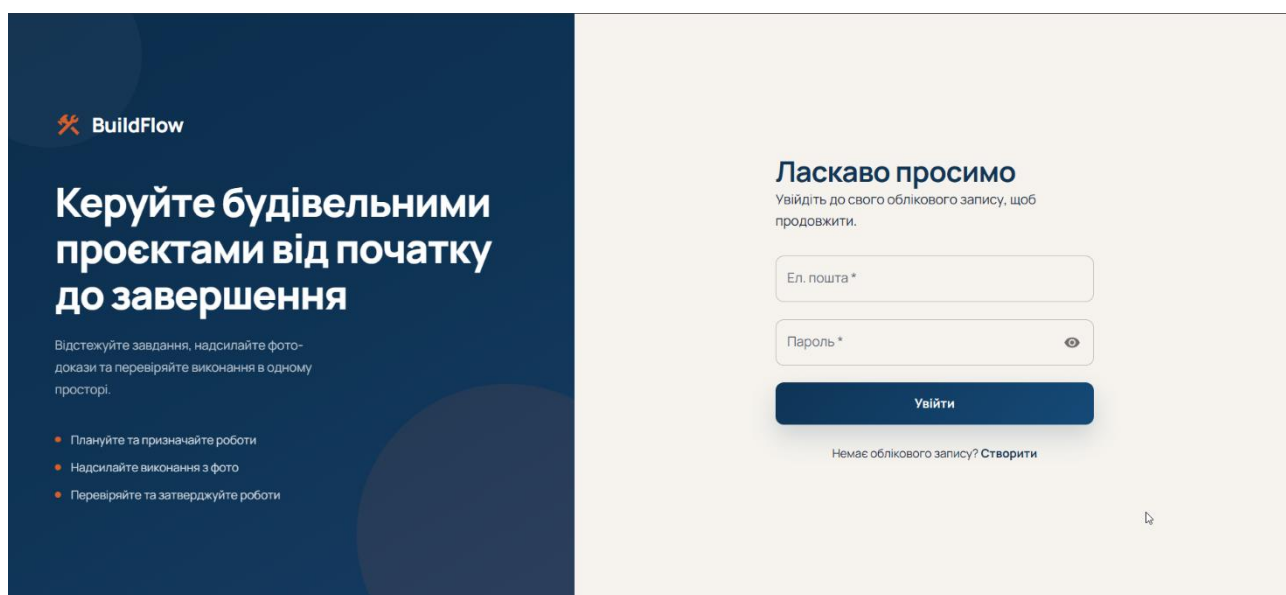


Рис. 3.2 – Сторінка входу до системи

Реєстрація (рис. 3.3) передбачає вибір ролі зі спадного списку – менеджер, підрядник або інженер. Роль адміністратора у списку відсутня: призначення цієї ролі здійснюється поза публічною формою реєстрації. Ліва частина екрану під час заповнення форми відображає короткий опис обраної ролі з переліком доступних дій у системі. Вміст інформаційного блоку змінюється динамічно, реагуючи на подію вибору іншого пункту в списку. Це візуальне рішення додано в інтерфейс навмисно, щоб попередити створення облікових записів із хибними правами. Під час збереження профілю бекенд фіксує ідентифікатор обраної ролі у відповідній таблиці бази даних. Надалі саме цей запис визначатиме рівень доступу до ендпойнтів системи через механізм RBAC.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		



Приєднуйтесь до своєї будівельної команди

Створіть обліковий запис, щоб керувати завданнями, надсилати виконані роботи або перевіряти завершення.

Менеджер
Створюйте об'єкти, плануйте та призначайте завдання

Підрядник
Отримуйте завдання та надсилайте фото-докази

Інженер
Перевіряйте та затверджуйте виконані роботи

Створити обліковий запис

Налаштуйте доступ для своєї ролі в будівельній команді.

Роль *

Підрядник

Оберіть свою роль у будівельному проєкті

Створити обліковий запис

Вже маєте обліковий запис? [Увійти](#)

Рис. 3.3 – Форма реєстрації нового користувача

Після успішного входу система відображає панель керування відповідно до ролі користувача. Менеджеру доступні три картки основних сценаріїв роботи – планування, виконання і перевірка, а також кнопка переходу до переліку об'єктів будівництва (рис. 3.4). Панель підрядника влаштована інакше (рис. 3.5): центральною дією є перехід до власних завдань, функції планування відсутні. Бічне меню теж відрізняється тим, що підрядник бачить лише розділи «Мої завдання» і «Сповіщення», тоді як менеджер має доступ до об'єктів будівництва.



Панель керування

Сповіщення

Об'єкти будівництва

ОК

Ольга Коваль

Менеджер

Менеджер

Панель керування

Оберіть основний сценарій роботи відповідно до вашої ролі.

Плануйте роботи

Надсилайте виконання

Перевіряйте якість

Швидкий доступ

Підказка

Об'єкти будівництва

Об'єкти

Рис. 3.4 – Панель керування менеджера

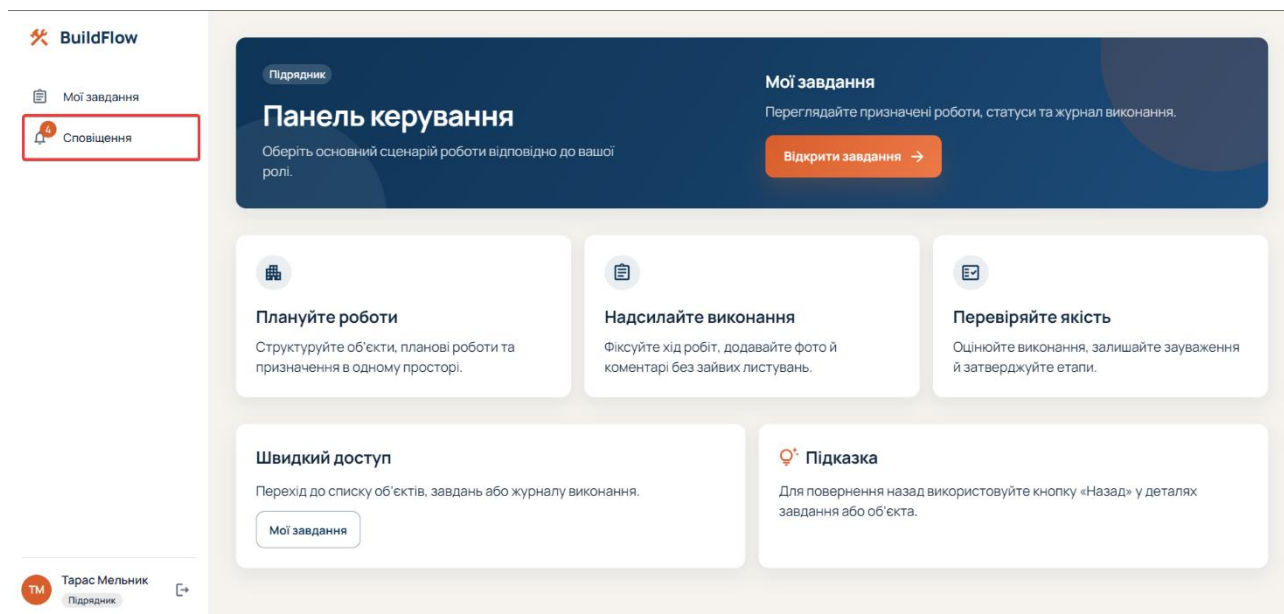


Рис. 3.5 – Панель керування підрядника

Перелік об'єктів будівництва (рис. 3.6) відображається у вигляді таблиці з колонками назви, адреси, орієнтовного бюджету і кількості планових робіт. Пошукове поле над таблицею виконує фільтрацію без додаткових запитів до сервера, а результат оновлюється під час введення. На рисунку 3.7 зображено приклад пошуку за частиною адреси: введення «Соборна» залишає у списку лише відповідний об'єкт. Нижня частина сторінки містить елементи пагінації з вибором кількості рядків.

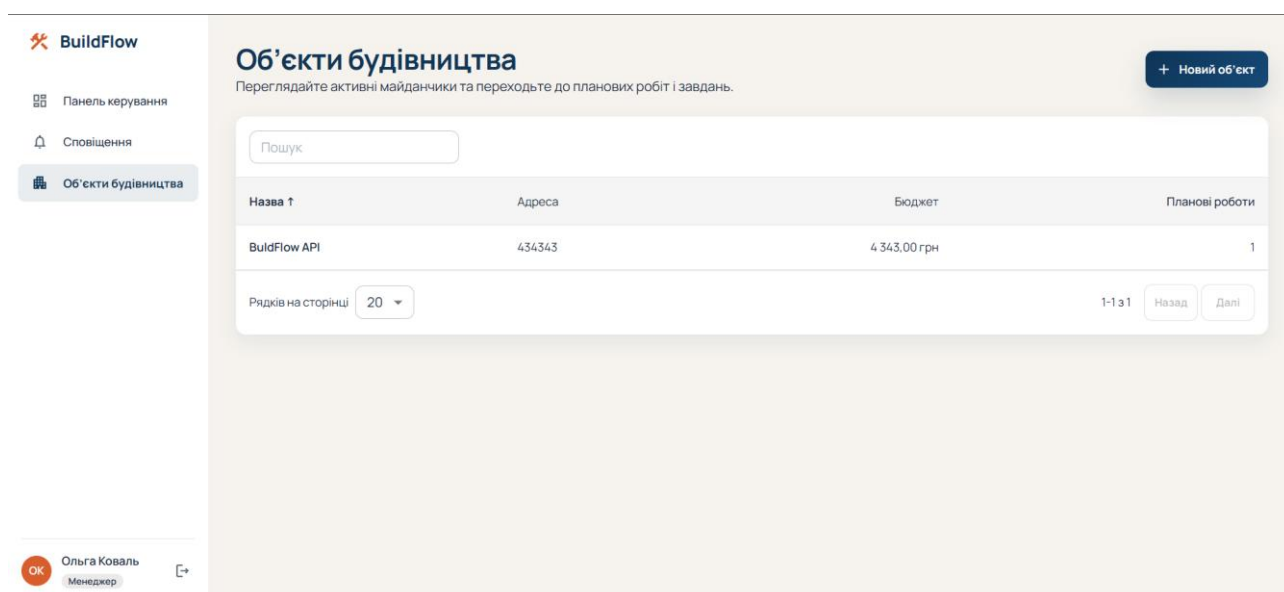


Рис. 3.6 – Список об'єктів будівництва

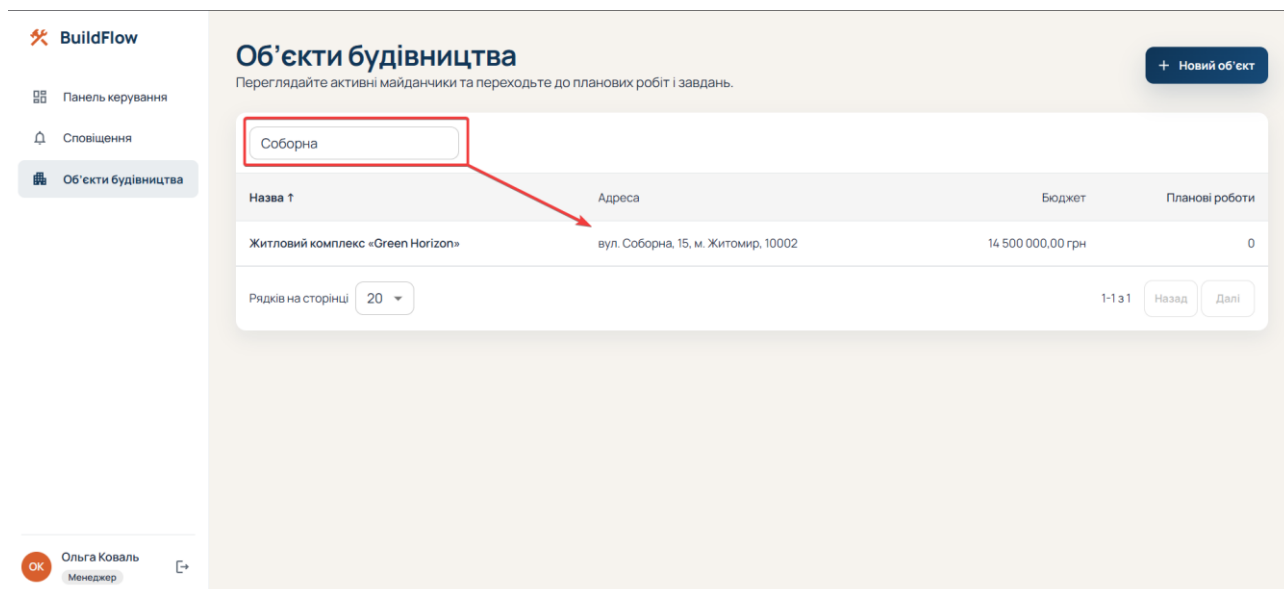


Рис. 3.7 – Пошук об'єктів за адресою

Новий об'єкт створюється через діалогове вікно (рис. 3.8) з полями назви, адреси та орієнтовного бюджету. Усі три поля є обов'язковими. Після підтвердження запис з'являється у таблиці без перезавантаження сторінки.

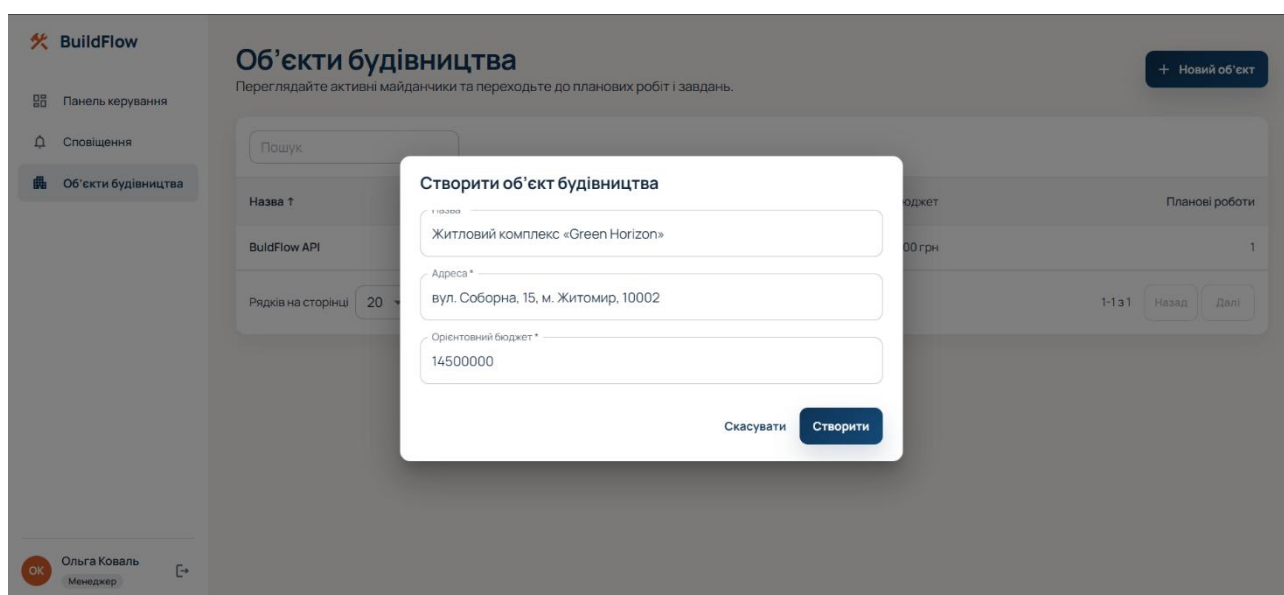


Рис. 3.8 – Діалог створення об'єкта будівництва

Горизонтальний індикатор прогресу у верхній частині сторінки (рис. 3.9) формується не на клієнті, а є результатом окремого аналітичного запиту до серверу. Обробник збирає всі завдання об'єкта, групує їх за статусами і повертає відсоток прийнятих робіт відносно загальної кількості. Окремо рахується

завантаженість бюджету: відношення сумарної планової вартості до затвердженого кошторису. Таким чином менеджер бачить два незалежних показники, а саме виконання і витрати, без потреби власноруч рахувати щось у таблицях. Логіку агрегації наведено в дод. Ж.6

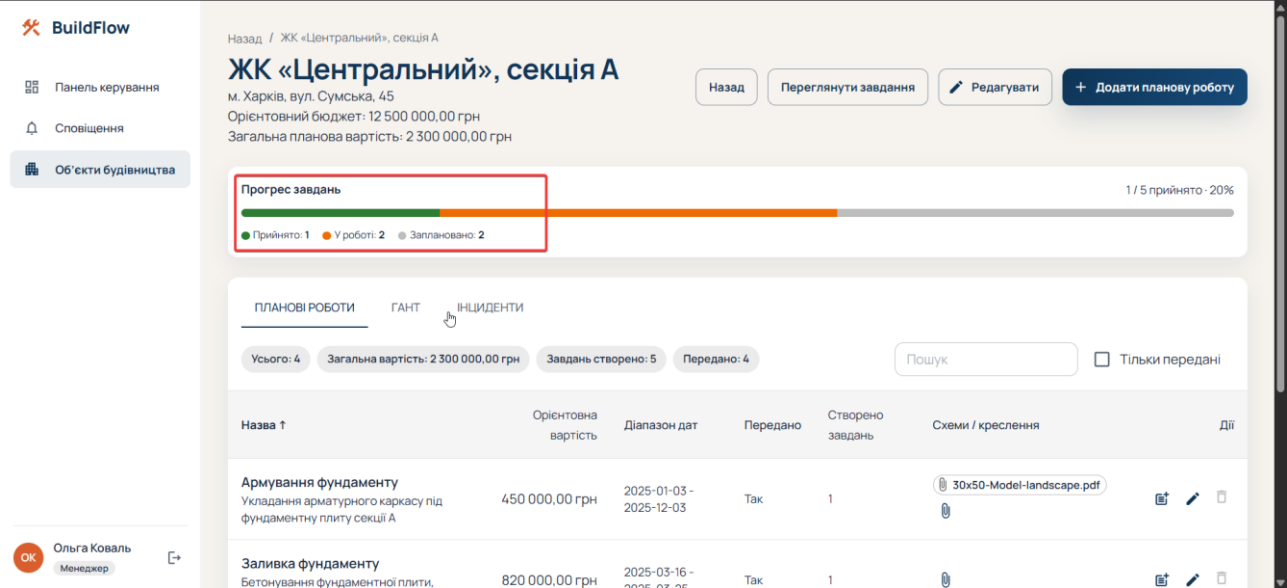


Рис. 3.9 – Деталі об'єкта з прогресом виконання та списком планових робіт

Вкладка «Гант» (рис. 3.10) показує часову шкалу планових робіт по об'єкту. Поточна дата відмічена вертикальною лінією, роботи з простроченим терміном виділені червоним кольором.

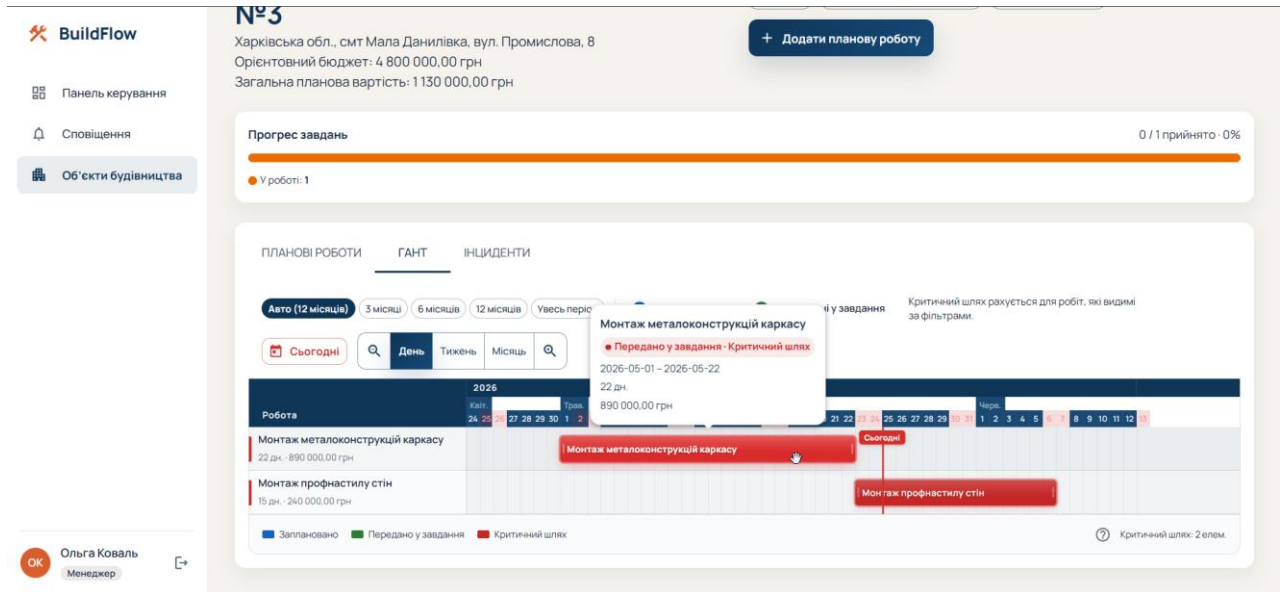


Рис. 3.10 – Діаграма Ганта по об'єкту

Підрядник керує своїми завданнями через сторінку «Мої завдання» (рис. 3.11). Вгорі розміщено шість карток з числовими показниками розподілу завдань за статусами: усього, призначено, у роботі, надіслано, прийнято і прострочено. Таблиця нижче підтримує пошук за назвою, фільтрацію за статусом і сортування за дедлайном. Завдання з простроченим терміном позначаються червоною міткою безпосередньо у рядку таблиці. Перемикач «Лише протерміновані» (рис. 3.12) звужує список до призначених завдань із простроченим терміном – обидва умови застосовуються одночасно.

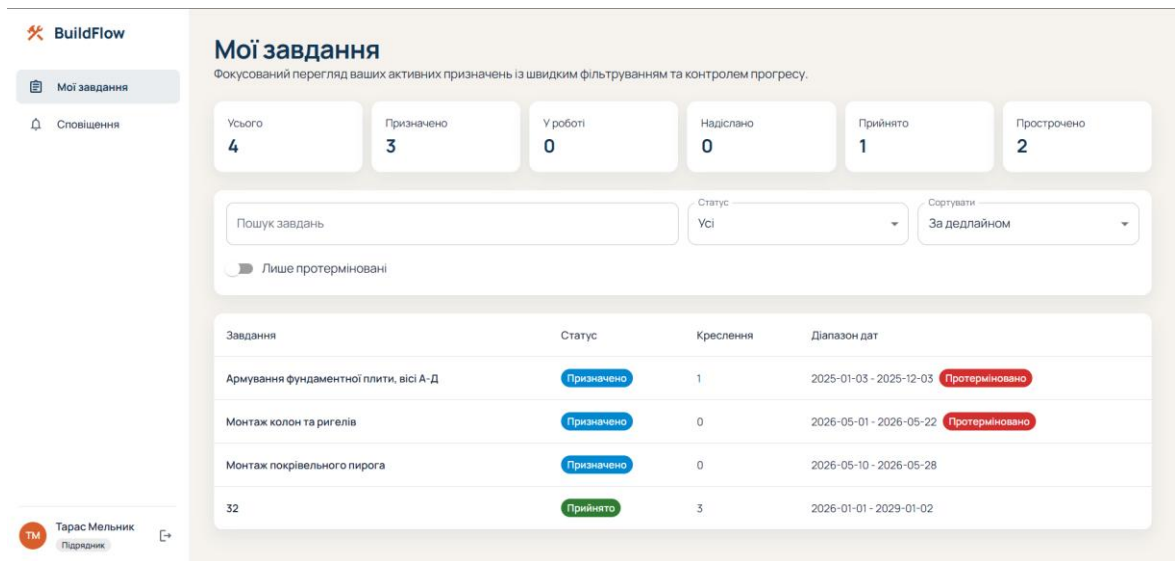


Рис. 3.11 – Список завдань підрядника з лічильниками статусів

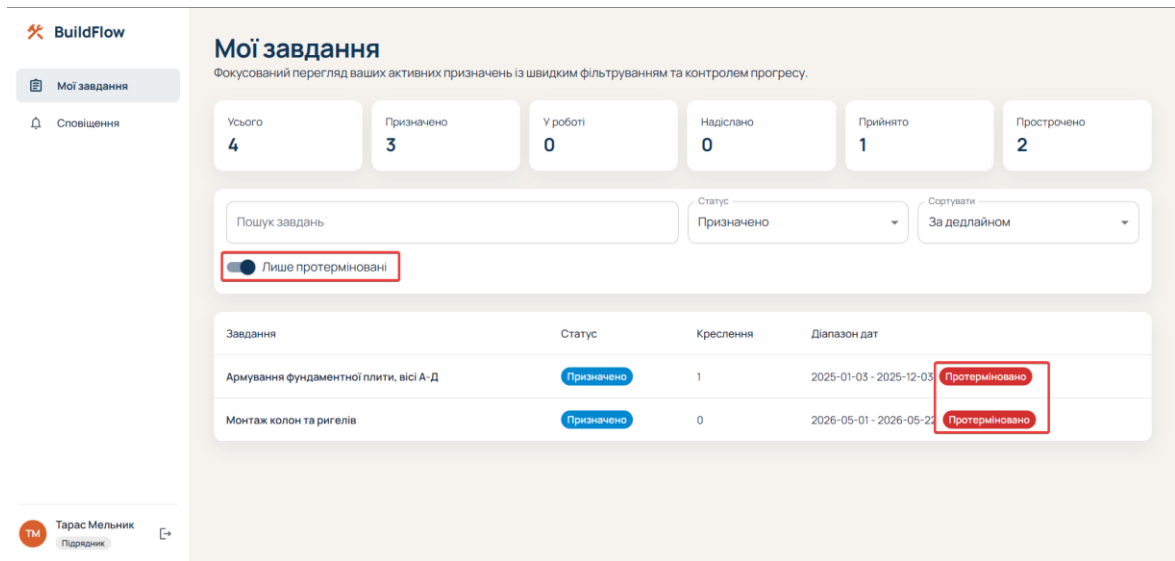


Рис. 3.12 – Фільтрація протермінованих завдань

Сторінка завдання (рис. 3.13) у верхній частині показує чотири картки: дати початку і завершення планового терміну, кількість прикріплених креслень і ім'я призначеного виконавця. Нижче розташований блок «Креслення» з переліком файлів і можливістю їх завантаження, а також блок «Дії» з кнопкою «Розпочати завдання».

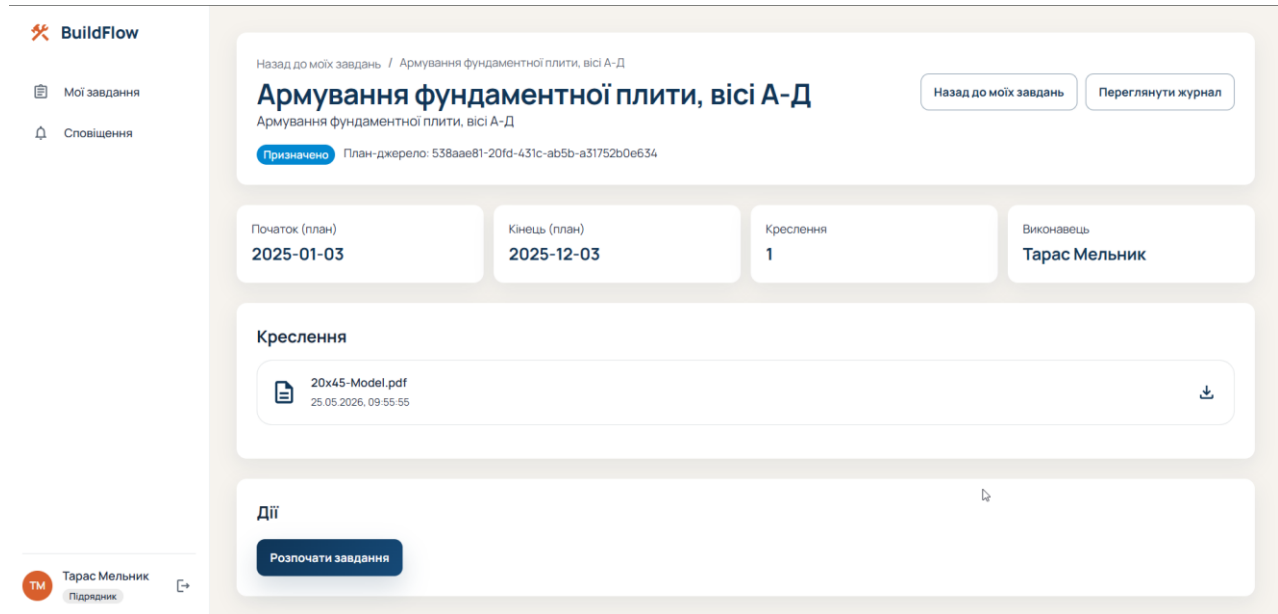


Рис. 3.13 – Деталі завдання з боку підрядника

Після переведення завдання в роботу підрядник подає звіт про виконання. Для завантаження фотодоказів (рис. 3.14) передбачено зону з підтримкою перетягування файлів і стандартний діалог вибору. Підтримку сучасного формату WEBP сюди додали спеціально, адже він добре стискає зображення і економить мобільний трафік, коли підрядник вантажить фото прямо з об'єкта. Саму зону скидання файлів виділено пунктирною рамкою для кращої помітності. Доки користувач не обере хоча б одне фото, головна кнопка відправки залишається заблокованою. Це просте рішення захищає систему від створення порожніх звітів. Система приймає зображення форматів JPG, PNG і WEBP; файли інших типів відхиляються на етапі вибору. Поле примітки є необов'язковим. Після відправлення звіту блок «Дії» показує надіслані фотографії і повідомлення про очікування перевірки (рис. 3.15), подальші дії підрядника на цьому етапі недоступні.

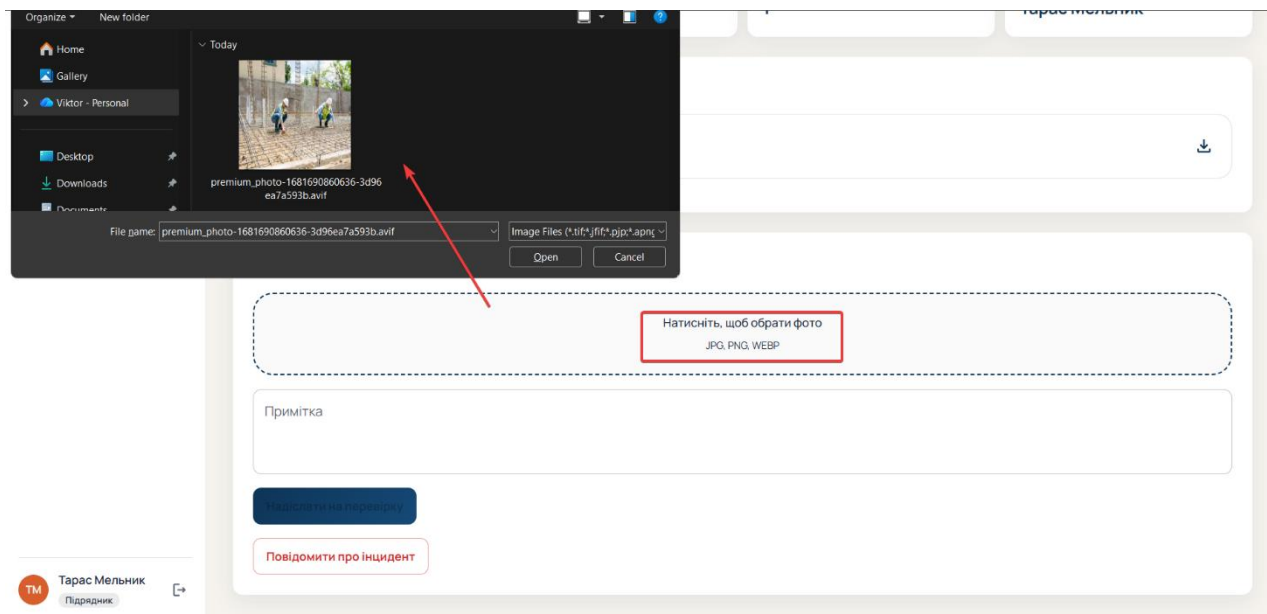


Рис. 3.14 – Завантаження фотодоказів виконання роботи

Завантаження фотодоказів (див. рис. 3.14) влаштоване так: клієнт відправляє файли на сервер через multipart-запит, а сервер самостійно зберігає їх у Azure Blob Storage і повертає тільки публічні URL. Це рішення прийнято свідомо для того, щоб браузер не мав прямого доступу до сховища і не тримав облікові дані. Кожен файл отримує унікальне ім'я на основі GUID, що виключає колізії при паралельному завантаженні від різних підрядників. Реалізацію сервісу зберігання файлів наведено в дод. Ж.6.

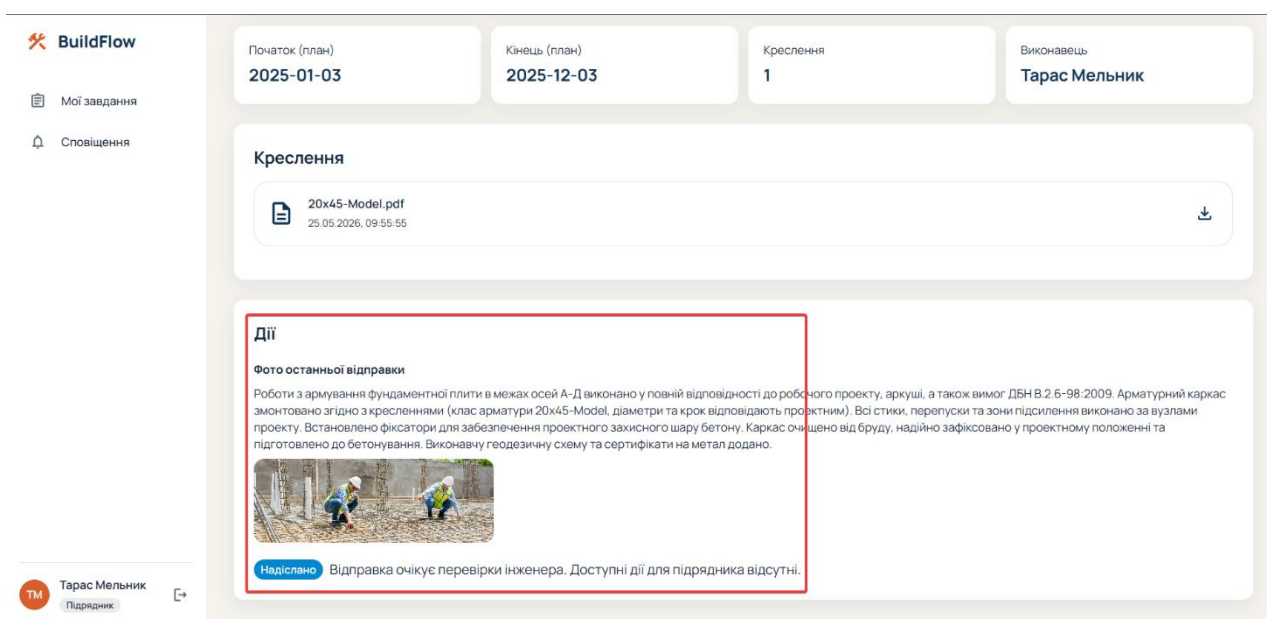


Рис. 3.15 – Завдання у статусі очікування рецензії інженера

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк. 54
Змн.	Арк.	№ докум.	Підпис	Дата		

Результат рецензування відображається підряднику одразу після прийняття рішення інженером (рис. 3.16). Статус завдання змінюється на «Прийнято», у блоці «Дії» з'являється відповідне позначення і фотографія останньої відправки. Лічильник сповіщень у бічному меню фіксує нову подію. Для кращого візуального сприйняття знак статусу забарвлюється у зелений колір, що дозволяє миттєво зчитати позитивний результат.

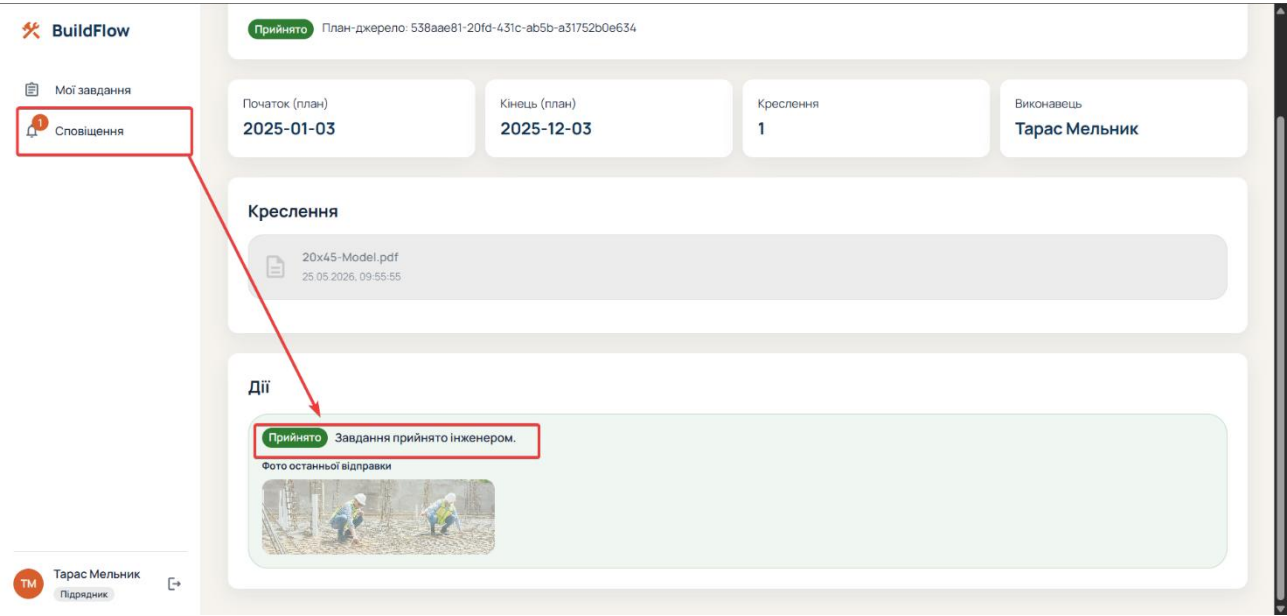


Рис. 3.16 – Завдання після прийняття інженером

Сторінка сповіщень (рис. 3.17) відображає хронологічний перелік системних подій. Кожен запис містить тип події у вигляді кольорового позначення, часову мітку і посилання на відповідне завдання. Призначення завдань позначається синім кольором, підтвердження виконання – зеленим.

Механізм формування сповіщень працює на базі тригерів предметної області. Серверна частина автоматично генерує новий запис у таблиці «user_notifications» під час кожної зміни статусу завдання або призначення нового виконавця. Застосунок отримує цей масив даних та відображає його у вигляді списку. Коли користувач натискає на посилання всередині картки події, система миттєво оновлює прапорець прочитання в базі даних. Такий алгоритм гарантує абсолютну точність лічильника нових повідомлень на бічній навігаційній панелі.

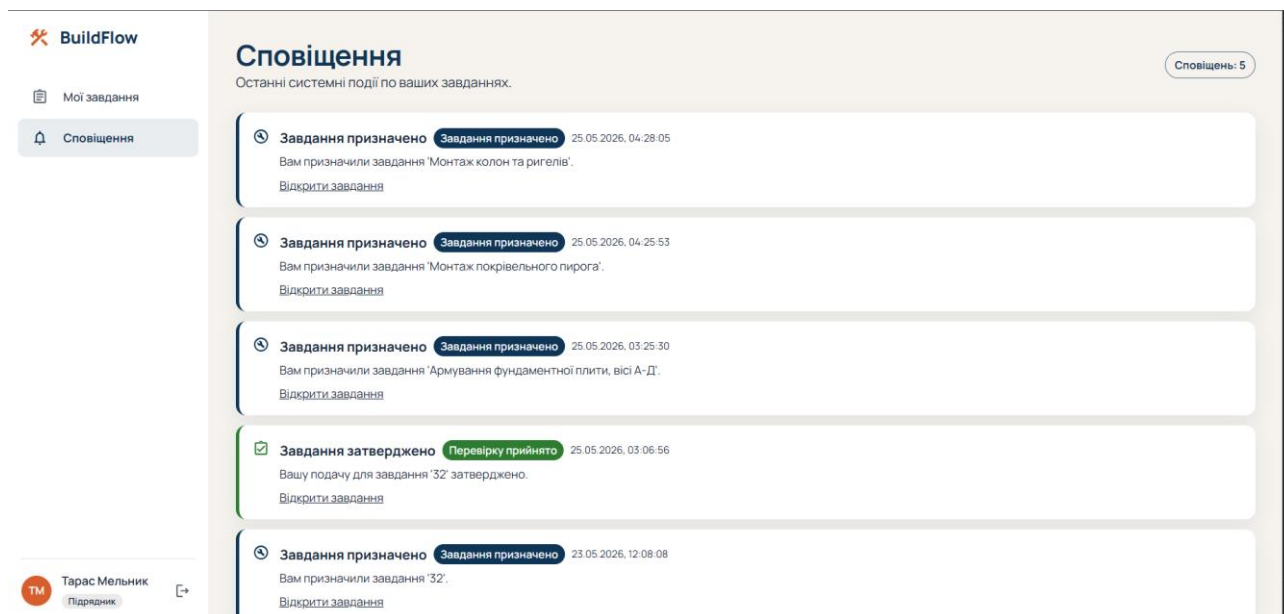


Рис. 3.17 – Сторінка сповіщень підрядника

Вкладка «Інциденти» на сторінці об'єкта (рис. 3.18) містить перелік зафіксованих позаштатних ситуацій із зазначенням рівня критичності і поточного стану. Кожен запис включає опис, дату події, фотоматеріали і поля для визначення серйозності та відповідального виконавця. Після вирішення ситуації статус змінюється на «Вирішено» (рис. 3.19) із фіксацією дати закриття. Система зберігає повну історію проблемних ситуацій для аналізу причин затримок.

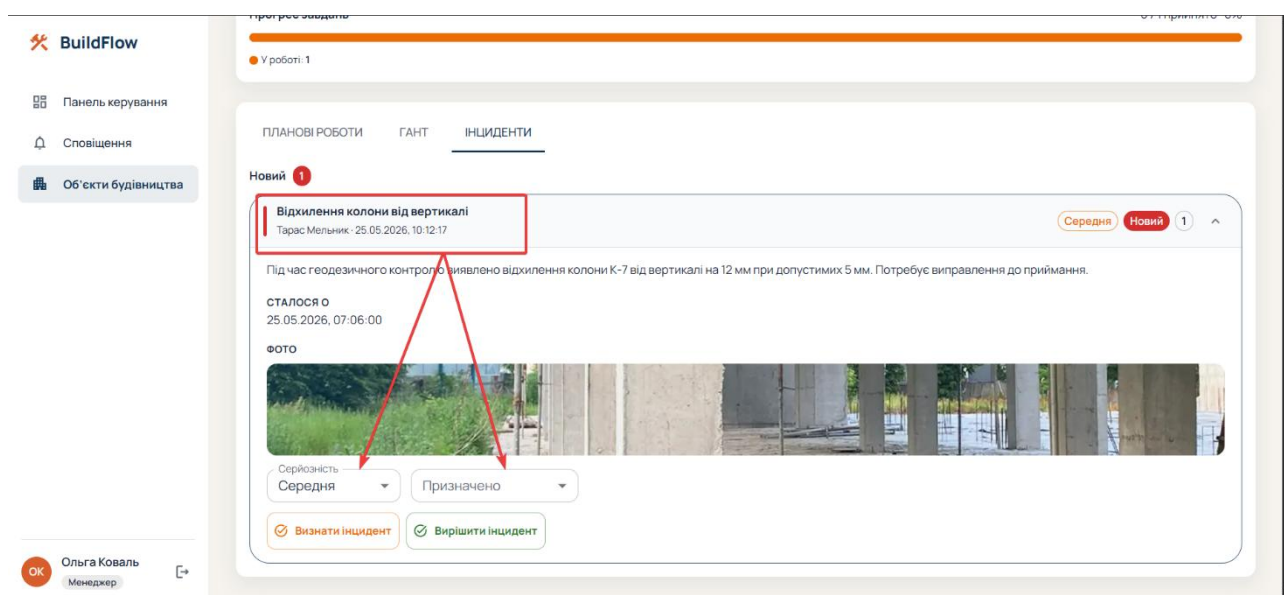


Рис. 3.18 – Список інцидентів по об'єкту

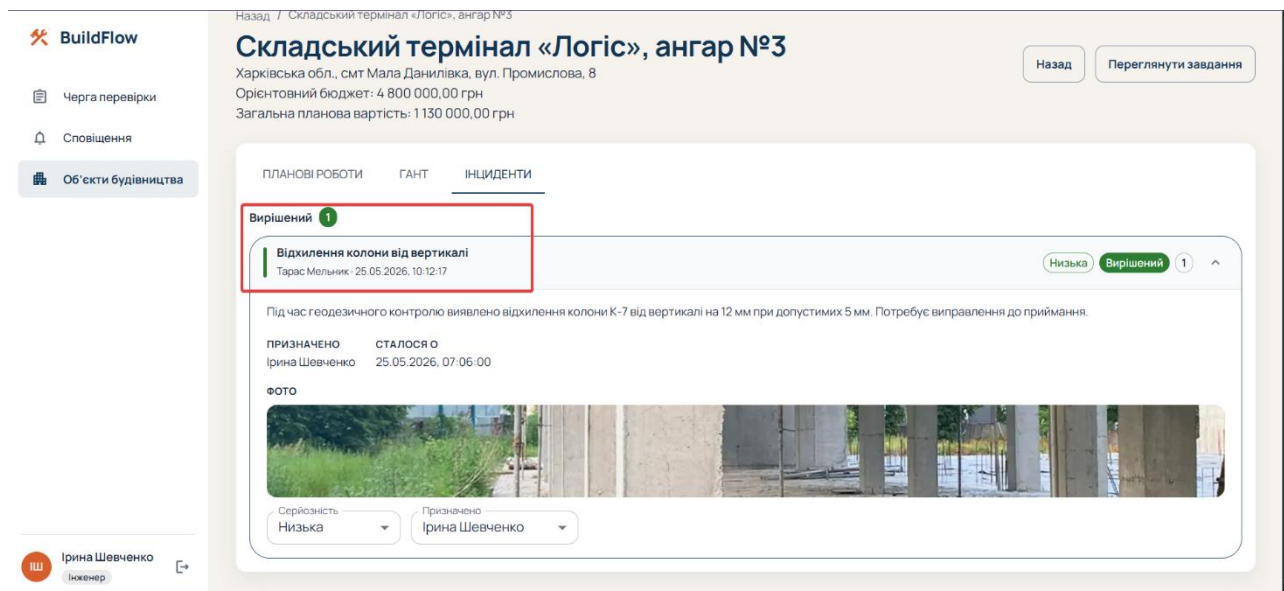


Рис. 3.19 – Інцидент зі статусом «Вирішено»

Підрядник має можливість зафіксувати інцидент безпосередньо зі сторінки свого завдання. Діалогове вікно (рис. 3.20) містить поля назви, опису, дати та часу події і зону завантаження фотографій. Кнопка підтвердження виділена червоним кольором для того, щоб унеможливити її випадкове натискання і підкреслює нештатний характер дії. Система автоматично прив'язує створений запис до поточного етапу будівництва. Алгоритм виключає втрату контексту помилки під час подальшого технічного аналізу.

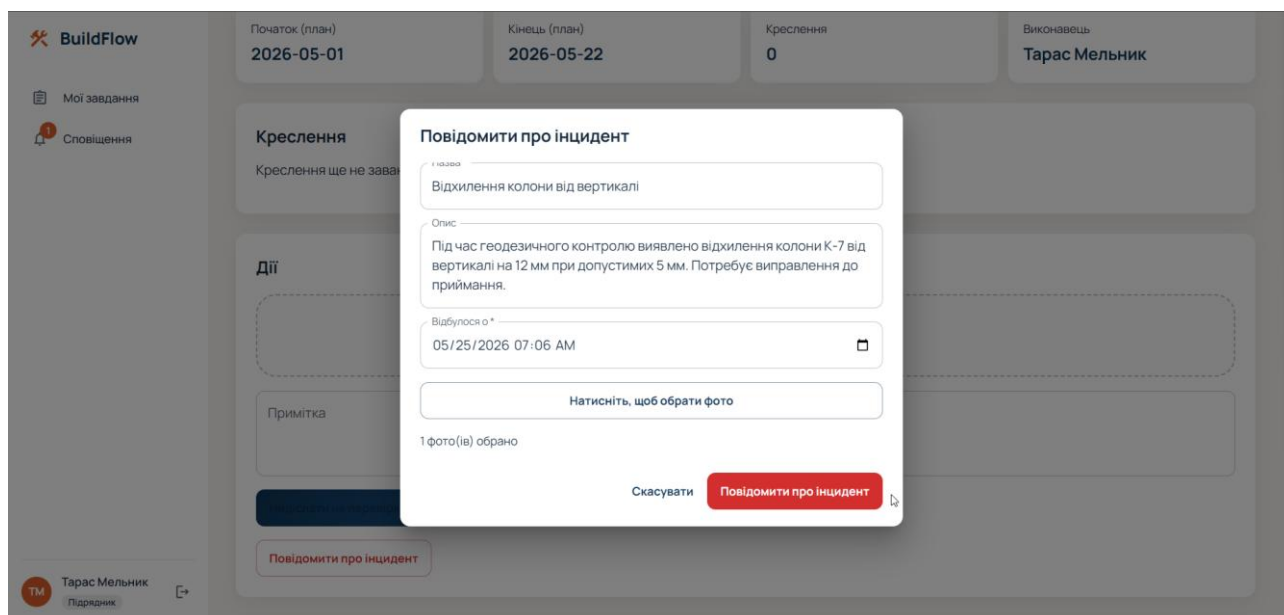


Рис. 3.20 – Форма фіксації інциденту підрядником

3.3 Тестування роботи програмного продукту

Тестування [19] системи охоплює три рівні: модульні тести клієнтської частини на Vitest, модульні тести серверної частини на xUnit і інтеграційні тести репозиторіїв з реальною базою даних через Testcontainers [29]. Кожен рівень вирішує свою задачу і навмисно не дублює інші. Серверна частина має 71 модульний тест, всі проходять успішно (рис. 3.21). Клієнтська частина налічує 11 тестових файлів що охоплюють компоненти, хуки і сторінки.

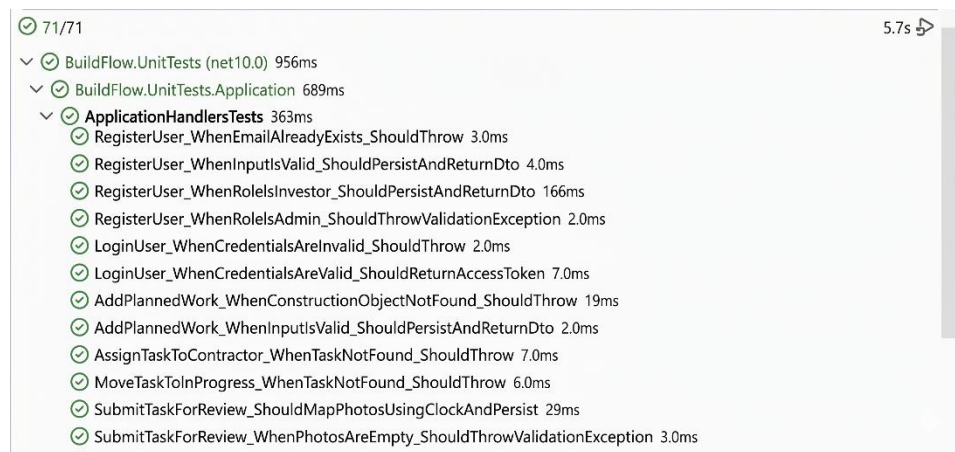


Рис. 3.21 – Результати виконання модульних тестів серверної частини

Доменні тести це найважливіша частина серверного покриття. Вони перевіряють інваріанти сутностей без жодної залежності від бази чи HTTP. Підрядник не може подати звіт без фото, а система піднімає виняток на рівні доменного об'єкта, ще до будь-якого звернення до сховища. Інженер не може відхилити роботу без коментаря. Прийняте завдання не підлягає перепризначенню. Інший підрядник, не той що призначений, не може взяти завдання в роботу навіть якщо має відповідну роль (табл. 3.5).

Таблиця 3.5

Тести доменного рівня

Клас	Тест	Що перевіряється
1	2	3
WorkTaskTests	SubmitForReview_WithoutPhotos_ShouldThrow	Звіт без фото неможливий
WorkTaskTests	ReviewLatestSubmission_RejectedWithoutComment_ShouldThrow	Відхилення вимагає коментар

Продовження таблиці 3.5

1	2	3
WorkTaskTests	ReviewLatestSubmission_ Rejected_ShouldMoveToNeedsRework	Цикл відхилення і повторного подання
WorkTaskTests	MoveToInProgress_ ByAnotherContractor_ShouldThrow	Лише призначений підрядник
WorkTaskTests	ReviewLatestSubmission_ Accepted_ShouldSetTerminalStatus	Прийнято – кінцевий стан, перепризначення заборонене
WorkTaskTests	EnsureCanBeDeleted_ WhenTaskIsNotPlanned_ShouldThrow	Видалення тільки зі статусу Planned
WorkTaskTests	UpdateDetails_ WhenTitleIsEmpty_ShouldThrow	Порожній заголовок завдання недопустимий
ConstructionObjectTests	AddPlannedWork_ ByContractor_ShouldThrow	Підрядник не може додавати планові роботи
ConstructionObjectTests	RegisterTaskCreatedFromPlannedWork_ WithSameTaskIdTwice_ShouldThrow	Повторне перенесення планової роботи заборонено
DomainInvariantsTests	TimeWindow_ WhenEndBeforeStart_ShouldThrow	Дата кінця не раніше початку
DomainInvariantsTests	PhotoEvidence_ WhenContentTypeIsNotImage_ShouldThrow	MIME-тип лише image/*
DomainInvariantsTests	WorkSubmission_ WhenCreatedWithoutPhotos_ShouldThrow	Звіт без фото неможливий
DomainInvariantsTests	WorkSubmission_ AttachReviewTwice_ShouldThrow	Подвійна рецензія заборонена
DomainInvariantsTests	TaskReview_WhenRejected_ShouldThrow	Рецензія відхилення без коментаря
DomainInvariantsTests	PlannedWorkItem_ MarkTransferredTwice_ShouldThrow	Повторне перенесення планової роботи
UserTests	RemoveRole_LastRole_ShouldThrow	Користувач без жодної ролі неможливий
UserTests	SetPassword_ WhenHashIsEmpty_ShouldThrow	Порожній хеш пароля недопустимий
NotificationTests	NewNotification_IsRead_ShouldBeFalse	Нове сповіщення непрочитане
NotificationTests	MarkAsRead_ShouldSetIsReadTrue	Позначення прочитаним

Проведене тестування доменного рівня підтвердило надійність логіки ядра. Кожен із передбачених сценаріїв виконався без помилок. Ці результати доводять наявність контролю життєвого циклу сутностей безпосередньо в доменній моделі. Якщо користувач спробує порушити правила або подасть звіт без прикріплених фотографій, система одразу згенерує виняток. Відповідно ядро самостійно блокує збереження некоректних даних. Інтерфейсні помилки чи зовнішні фактори жодним чином на цей процес не впливають. Тестові класи доменного рівня наведено в додатку К.

Перевірка обробників команд довела правильну роботу патерну CQRS на прикладному рівні. Використання бібліотеки Moq дозволило повністю ізолювати код від бази даних. Завдяки цьому кожен тест виконується за лічені мілісекунди. Усі тридцять сім сценаріїв завершилися успішно. Це засвідчує коректну маршрутизацію запитів та надійну валідацію вхідних параметрів. Додатково тести підтвердили коректну роботу рольової моделі доступу. Програмний продукт успішно відхиляє неавторизовані запити та захищає дані від витоку ще до початку їхньої обробки. Два окремих класи, які наведено в додатку Л, охоплюють різні групи сценаріїв (табл. 3.6).

Таблиця 3.6

Тестові класи обробників команд

Клас	Кількість тестів	Що охоплює
ApplicationHandlersTests	31	Реєстрація, логін, об'єкти, планові роботи, завдання, подання звіту, рецензування, аналітика
EngineerHandlersTests	6	Отримання списку інженерів з перевіркою ролей

Тести клієнтської частини написані на Vitest [33] з React Testing Library [25]. Ключовим є ProtectedRoute: неавтентифікований користувач перенаправляється на логін, автентифікований без обмежень ролей бачить захищений маршрут, користувач з невірною роллю – на дашборд. Помилка в цьому компоненті означає що будь-хто може побачити чужий інтерфейс. Сторінка RoleRedirectPage перевіряє рольову маршрутизацію при вході:

підрядник бачить кнопку «Відкрити завдання», менеджер – «Перейти до об'єктів», інвестор – «Відкрити аналітику» (рис. 3.22, табл. 3.7).

```

✓ src/components/AppShellBadge.test.tsx (6 tests) 6ms
✓ src/features/notifications/api/notifications.test.ts (4 tests) 10ms
✓ src/components/ProtectedRoute.test.tsx (3 tests) 63ms
✓ src/features/construction-objects/api/constructionObjects.test.ts (7 tests) 10ms
✓ src/hooks/useBackOrFallback.test.tsx (2 tests) 330ms
✓ src/features/tasks/pages/TaskJournalNavigation.test.tsx (4 tests) 352ms
  ✓ returns to task detail when back is clicked from journal 323ms
✓ src/components/PageHeaderWithBreadcrumbs.test.tsx (2 tests) 254ms
✓ src/features/notifications/pages/NotificationsPage.test.tsx (5 tests) 323ms
✓ src/features/tasks/pages/EngineerReviewQueuePage.test.tsx (6 tests) 785ms
✓ src/features/dashboard/pages/RoleRedirectPage.test.tsx (3 tests) 816ms
  ✓ renders contractor dashboard landing actions 384ms
✓ src/App.test.tsx (1 test) 381ms
  ✓ renders login page by default when unauthenticated 379ms

Test Files 11 passed (11)
Tests 43 passed (43)
Start at 09:41:40
Duration 5.05s (transform 1.59s, setup 1.80s, import 12.33s, tests 3.33s, environment 15.54s)

```

Рис. 3.22 – Результати виконання тестів клієнтської частини

Таблиця 3.7

Тестові файли клієнтської частини

Файл	Що перевіряється
ProtectedRoute.test.tsx	Захист маршрутів: анонім, автентифікований, невірна роль
RoleRedirectPage.test.tsx	Рольова маршрутизація при вході в систему
EngineerReviewQueuePage.test.tsx	Відображення черги рецензій для інженера
TaskJournalNavigation.test.tsx	Навігація журналу завдань
NotificationsPage.test.tsx	Відображення сторінки сповіщень
AppShellBadge.test.tsx	Лічильник непрочитаних сповіщень
useBackOrFallback.test.tsx	Хук навігації назад з fallback–маршрутом
constructionObjects.test.ts	API–функції будівельних об'єктів
notifications.test.ts	API–функції сповіщень
Файл	Що перевіряється
ProtectedRoute.test.tsx	Захист маршрутів: анонім, автентифікований, невірна роль
RoleRedirectPage.test.tsx	Рольова маршрутизація при вході в систему
EngineerReviewQueuePage.test.tsx	Відображення черги рецензій для інженера

Інтеграційні тести є останнім рівнем. Testcontainers при кожному запуску піднімає чистий PostgreSQL і застосовує всі міграції. Кожен тест отримує порожню базу, а залишки попереднього не впливають на результат. Перевіряється збереження і читання агрегатів, включно з JSONB–колонками де лежать вкладені

колекції планових робіт і фотодоказів. Кілька разів траплялась ситуація де mock-репозиторій відпрацював правильно, а реальна база – ні, через специфіку серіалізації. Інтеграційні тести саме це і виявляють (рис. 3.23).

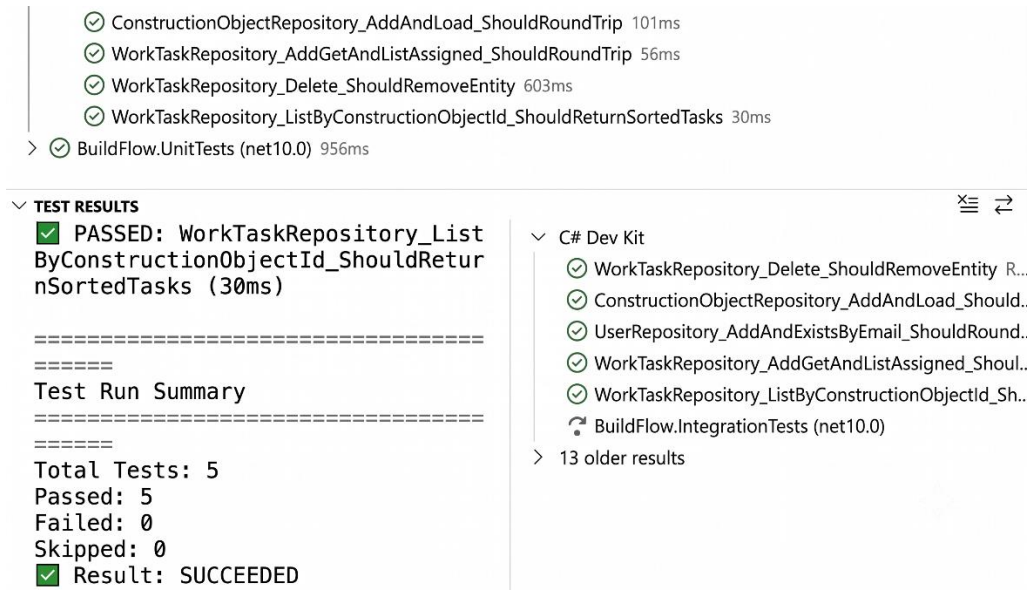


Рис. 3.23 – Результати виконання інтеграційних тестів

Зведені дані щодо обраних інструментів та результатів покриття коду зібрано нижче (табл. 3.8).

Таблиця 3.8

Загальне покриття тестами

Рівень	Інструмент	Кількість тестів	Результат
Доменна логіка	xUnit + FluentAssertions	29	Пройдено
Обробники команд	xUnit + Moq	42	Пройдено
Компоненти клієнта	Vitest + Testing Library	43	Пройдено
Репозиторії	xUnit + Testcontainers	—	Пройдено

Загалом для перевірки працездатності проєкту написано 114 тестів. Такий комбінований підхід дозволив протестувати систему з різних боків, не покладаючись на один інструмент. Серверна частина перевірялася фреймворком xUnit [34]. Щоб тестувати обробники команд ізольовано від бази даних, зовнішні залежності підмінялися за допомогою бібліотеки Moq, а результати звірялися через FluentAssertions [11]. На клієнтській стороні застосовано зв'язку Vitest та Testing Library. Цей набір перевірів правильність відмальовування компонентів інтерфейсу та надійність захисту приватних роутів. Окрему увагу було приділено

інтеграційному рівню репозиторіїв. Завдяки контейнеризації через Testcontainers вдалося перевірити SQL-запити на реальній базі даних, повністю оминаючи програмні заглушки. Такі показники доводять загальну надійність архітектури та безпомилкову роботу рольової моделі. Застосунок коректно обробляє та зберігає всі інформаційні агрегати. Отже, розроблена система працює стабільно. Продукт фактично готовий до практичного впровадження у реальні процеси будівельного майданчика.

Висновки до третього розділу

Програмний продукт розгортається в хмарі Azure. Архітектуру було розділено на п'ять незалежних вузлів. До них ввійшли бекенд, фронтенд, база даних, файлове сховище та реєстр контейнерів. Щоб відмовитися від ручного деплою, налаштовано пайплайн через GitLab CI/CD. Кожен коміт у головну гілку автоматично ініціює перевірку коду, ініціалізацію тестів та збірку. Після цього нова версія самостійно розгортається на сервері. Для розробки на локальній машині створено файл Docker Compose. Він піднімає точну копію всієї системи буквально однією командою.

На стороні клієнта реалізовано суворе розмежування доступу. Після авторизації інтерфейс повністю адаптується під посаду. Завдяки цьому весь процес будівництва вдалося замкнути в одному застосунку: від планування до завантаження фотозвітів та фінального приймання робіт.

Якість коду підтверджено на трьох рівнях. Серверна частина має 71 модульний тест для ізольованої перевірки бізнес-правил без підключення до бази. На фронтенді 43 тести контролюють захист маршрутів та правильність відображення сторінок. Найважливішим етапом є інтеграційні тести. За допомогою бібліотеки Testcontainers вони піднімають справжній екземпляр PostgreSQL. Це дозволяє перевірити реальну роботу репозиторіїв із базою даних. Увесь тестовий набір виконується успішно.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Практичним підсумком роботи є готова система BuildFlow. Ринок програмного забезпечення зараз не має доступних інструментів для малих підприємств. Більшість популярних програм коштують занадто дорого і містять багато зайвих функцій. Вони також погано адаптовані до реальних умов будівельного майданчика. Саме тому виникла гостра потреба написати власний продукт. Готовий застосунок цифровізує управління об'єктами та скасовує паперовий документообіг.

Базою для написання коду стала концепція Clean Architecture. Архітектурний шаблон CQRS чітко розмежував операції збереження та читання бази даних. Це рішення помітно прискорило роботу платформи під час завантаження великих обсягів інформації. Основу продукту становить доменна модель. Вона функціонує як кінцевий автомат. Ця модель керує зміною статусів завдань і диктує загальну структуру бази даних. Жорсткі правила ядра повністю унеможливають збереження некоректних даних.

Серверний застосунок написаний під платформу .NET. Клієнтську частину забезпечує бібліотека React. Оновлення коду автоматизує налаштований конвеєр безперервного розгортання. Це гарантує швидку та безпечну доставку нових функцій користувачам. Розроблена рольова модель суворо розмежовує права доступу. Інженери, підрядники та менеджери отримали ізольований функціонал. Система жорстко блокує здачу робіт без прикріплених фотографій. Такий механізм формує абсолютну прозорість контролю якості на майданчику.

Працездатність логіки підтверджують 114 успішних тестів. Перевірка охопила доменний, прикладний та інтеграційний рівні коду. Жоден тестовий сценарій не виявив інфраструктурних вразливостей. Продукт остаточно вирішує проблему хаотичної комунікації учасників будівництва в месенджерах. Платформа успішно пройшла комплексну перевірку стабільності. Розроблене програмне забезпечення цілком готове до експлуатації на реальних об'єктах.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Core Security. URL: <https://learn.microsoft.com/en-us/aspnet/core/security> (дата звернення: 28.04.2026).
2. Azure App Service. URL: <https://learn.microsoft.com/en-us/azure/app-service> (дата звернення: 21.03.2026).
3. Azure DevOps Boards. URL: <https://learn.microsoft.com/en-us/azure/devops/boards> (дата звернення: 12.03.2026).
4. Azure Storage. URL: <https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction> (дата звернення: 21.03.2026).
5. Cloud Design Patterns. URL: <https://learn.microsoft.com/en-us/azure/architecture/patterns> (дата звернення: 12.04.2026).
6. CQRS pattern. URL: <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs> (дата звернення: 10.04.2026).
7. DDD and CQRS patterns. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns> (дата звернення: 04.04.2026).
8. Docker. URL: <https://docs.docker.com> (дата звернення: 19.03.2026).
9. Domain analysis for microservices. URL: <https://learn.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis> (дата звернення: 07.04.2026).
10. Entity Framework Core. URL: <https://learn.microsoft.com/en-us/ef/core> (дата звернення: 23.04.2026).
11. FluentAssertions. URL: <https://fluentassertions.com/introduction> (дата звернення: 08.05.2026).
12. FluentValidation. URL: <https://docs.fluentvalidation.net> (дата звернення: 08.05.2026).
13. GitLab CI/CD. URL: <https://docs.gitlab.com/ee/ci> (дата звернення: 05.05.2026).
14. Jira Software. URL: <https://support.atlassian.com/jira-software-cloud> (дата звернення: 06.03.2026).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Microservices architecture style. URL: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices> (дата звернення: 18.04.2026).
16. Microsoft Project. URL: <https://support.microsoft.com/en-us/project> (дата звернення: 09.03.2026).
17. .NET. URL: <https://learn.microsoft.com/en-us/dotnet> (дата звернення: 14.03.2026).
18. .NET Application Architecture Guides. URL: <https://dotnet.microsoft.com/en-us/learn/dotnet/architecture-guides> (дата звернення: 20.04.2026).
19. .NET – тестування застосунків. URL: <https://learn.microsoft.com/en-us/dotnet/core/testing> (дата звернення: 11.05.2026).
20. Nottingham M., Wilde E., Dalal S. Problem Details for HTTP APIs : RFC 9457. Internet Engineering Task Force, 2023. URL: <https://www.rfc-editor.org/rfc/rfc9457> (дата звернення: 01.04.2026).
21. OpenAPI Specification. URL: <https://spec.openapis.org/oas/latest.html> (дата звернення: 26.04.2026).
22. PostgreSQL 16. URL: <https://www.postgresql.org/docs/16> (дата звернення: 24.03.2026).
23. Procore. URL: <https://www.procore.com> (дата звернення: 01.03.2026).
24. React. URL: <https://react.dev> (дата звернення: 16.03.2026).
25. React Testing Library. URL: <https://testing-library.com/docs/react-testing-library/intro> (дата звернення: 17.05.2026).
26. Refactoring.Guru – Design Patterns. URL: <https://refactoring.guru/design-patterns> (дата звернення: 15.04.2026).
27. Relational data. URL: <https://learn.microsoft.com/en-us/azure/architecture/data-guide/relational-data> (дата звернення: 01.04.2026).
28. Talaver O. V., Vakaliuk T. A. Reliable distributed systems: review of modern approaches. Journal of Edge Computing. 2023. Vol. 2, No. 1. P. 84–101. DOI: <https://doi.org/10.55056/jec.586> .

29. Testcontainers. URL: <https://testcontainers.com/guides> (дата звернення: 14.05.2026).
30. Trello. URL: <https://trello.com/guide> (дата звернення: 03.03.2026).
31. TypeScript. URL: <https://www.typescriptlang.org/docs> (дата звернення: 16.03.2026).
32. Unified Modeling Language (UML). URL: <https://www.omg.org/spec/UML> (дата звернення: 30.04.2026).
33. Vitest. URL: <https://vitest.dev/guide> (дата звернення: 20.05.2026).
34. xUnit.net. URL: <https://xunit.net/?tabs=cs> (дата звернення: 23.05.2026).

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

Діаграма варіантів використання

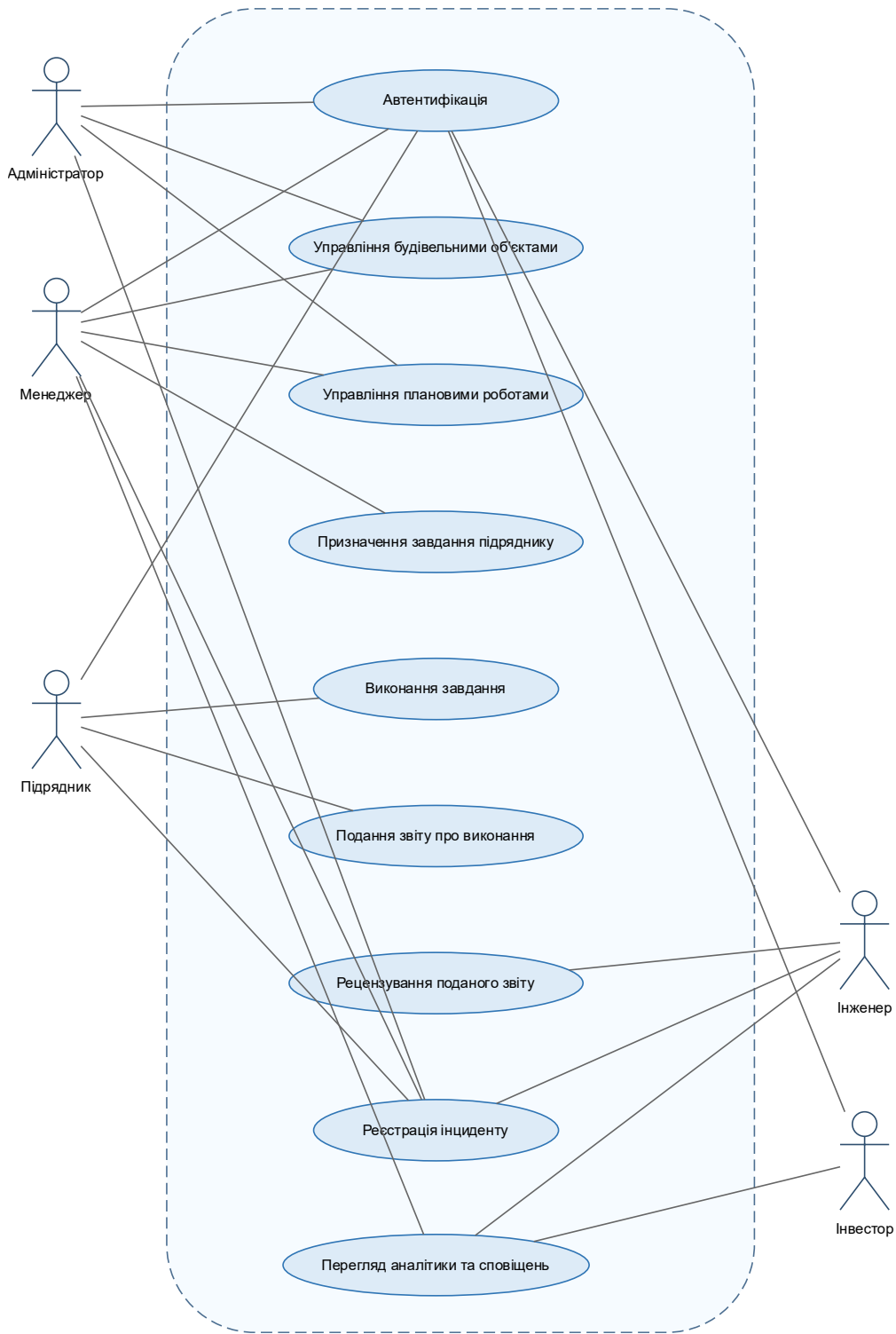


Рис. А.1 – Діаграма варіантів використання

Діаграма компонентів

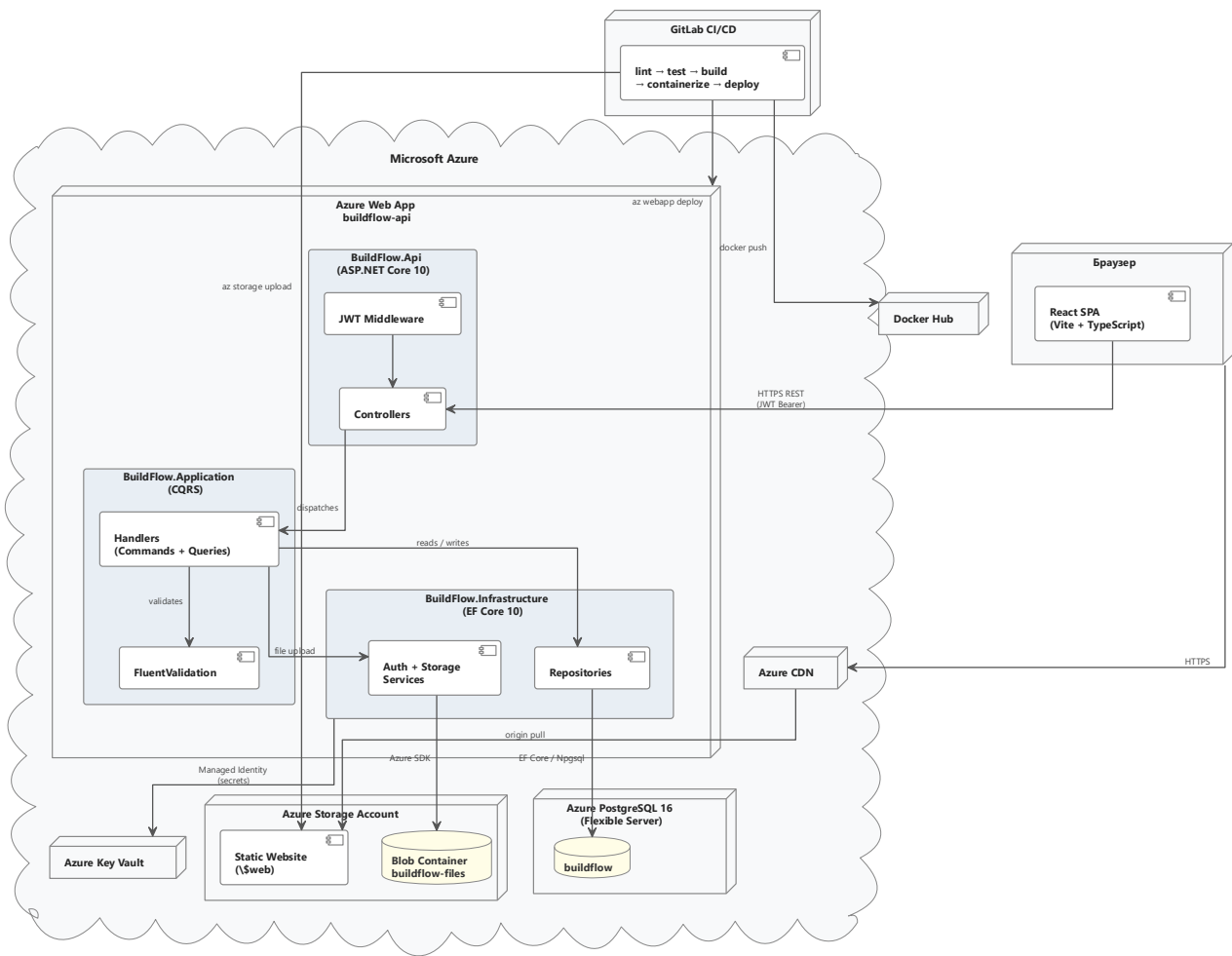


Рис. Б.1 – Діаграма компонентів

Діаграма шару застосунку (Application Layer)

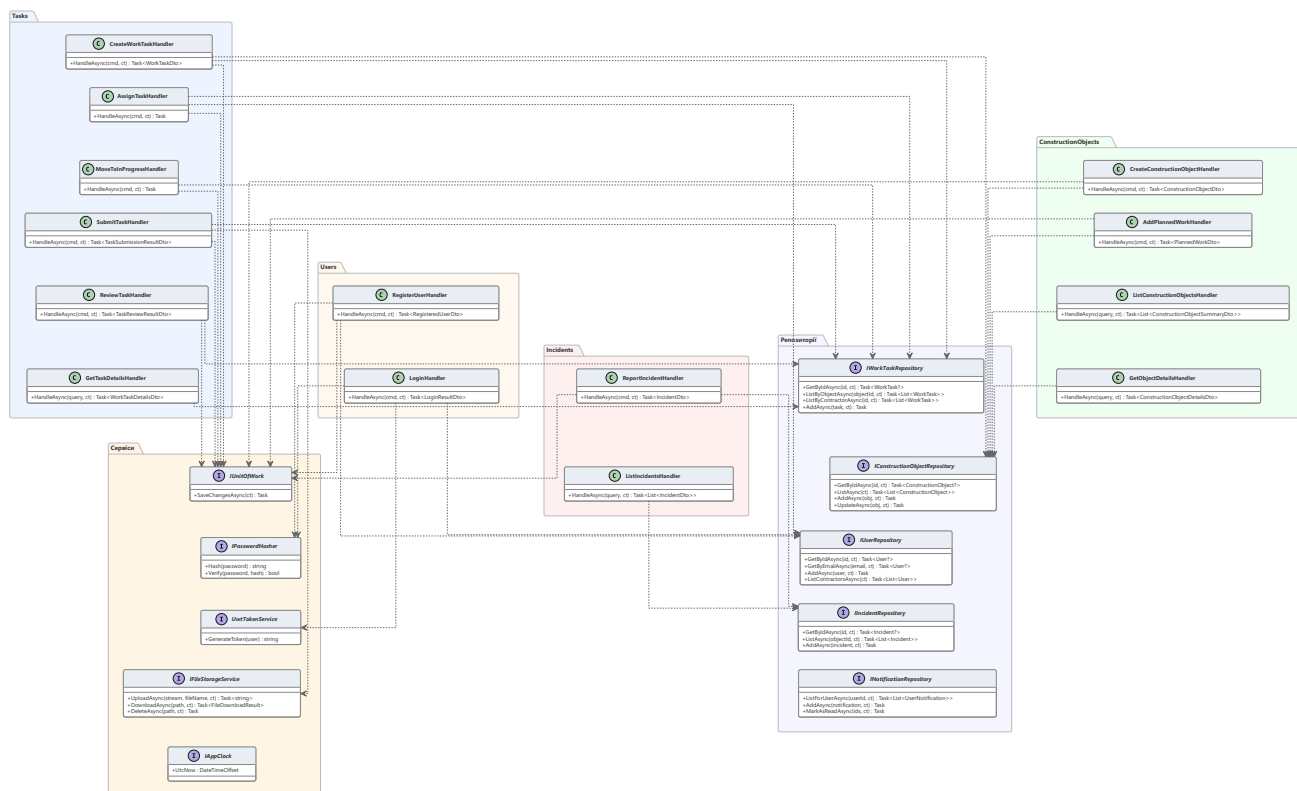


Рис. В.1 – Діаграма шару застосунку

Діаграма шару інфраструктури (Infrastructure Layer)

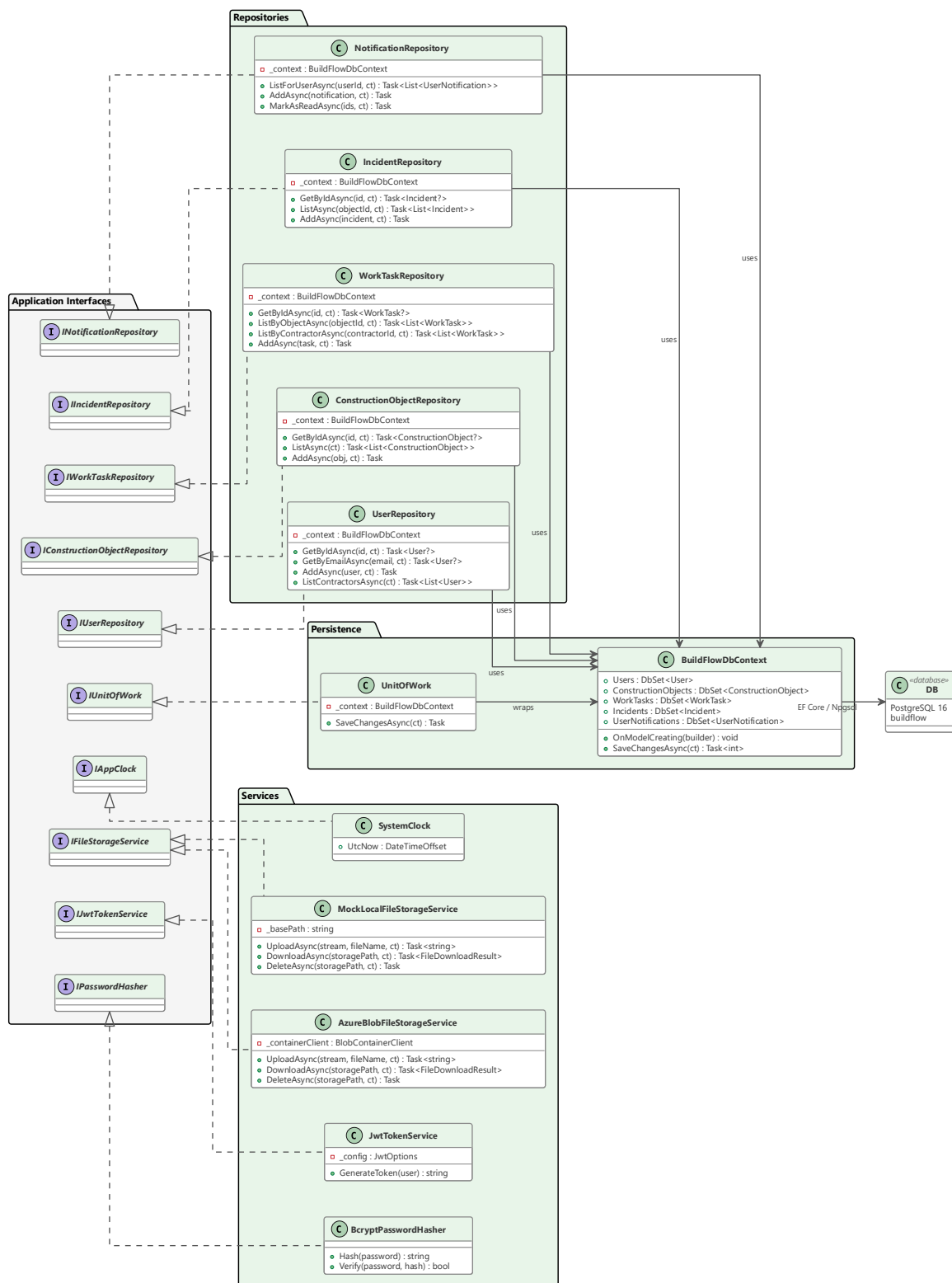


Рис. Г.1 – Діаграма шару інфраструктури

Діаграма послідовності подання звіту

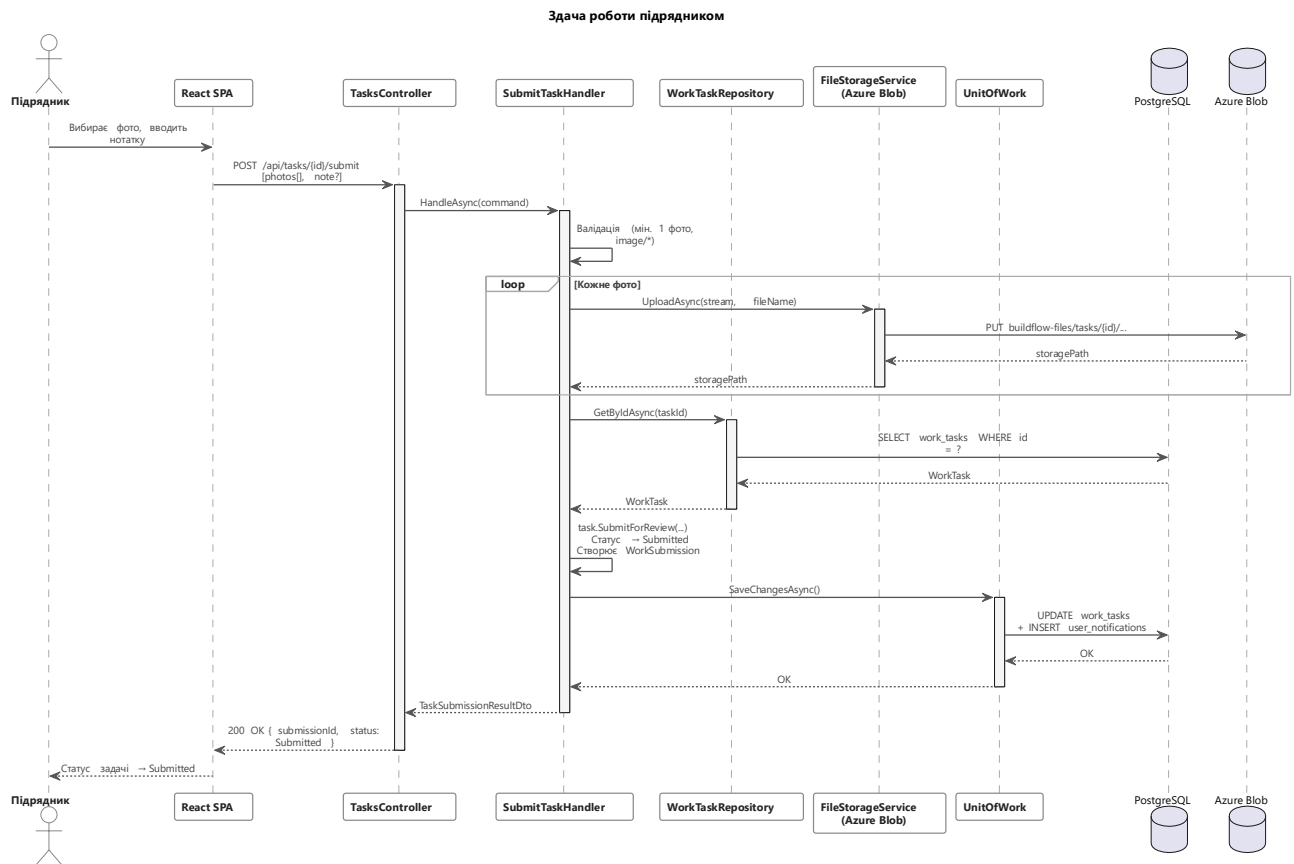


Рис. Д.1 – Діаграма послідовності подання звіту

Діаграма послідовності рецензування звіту

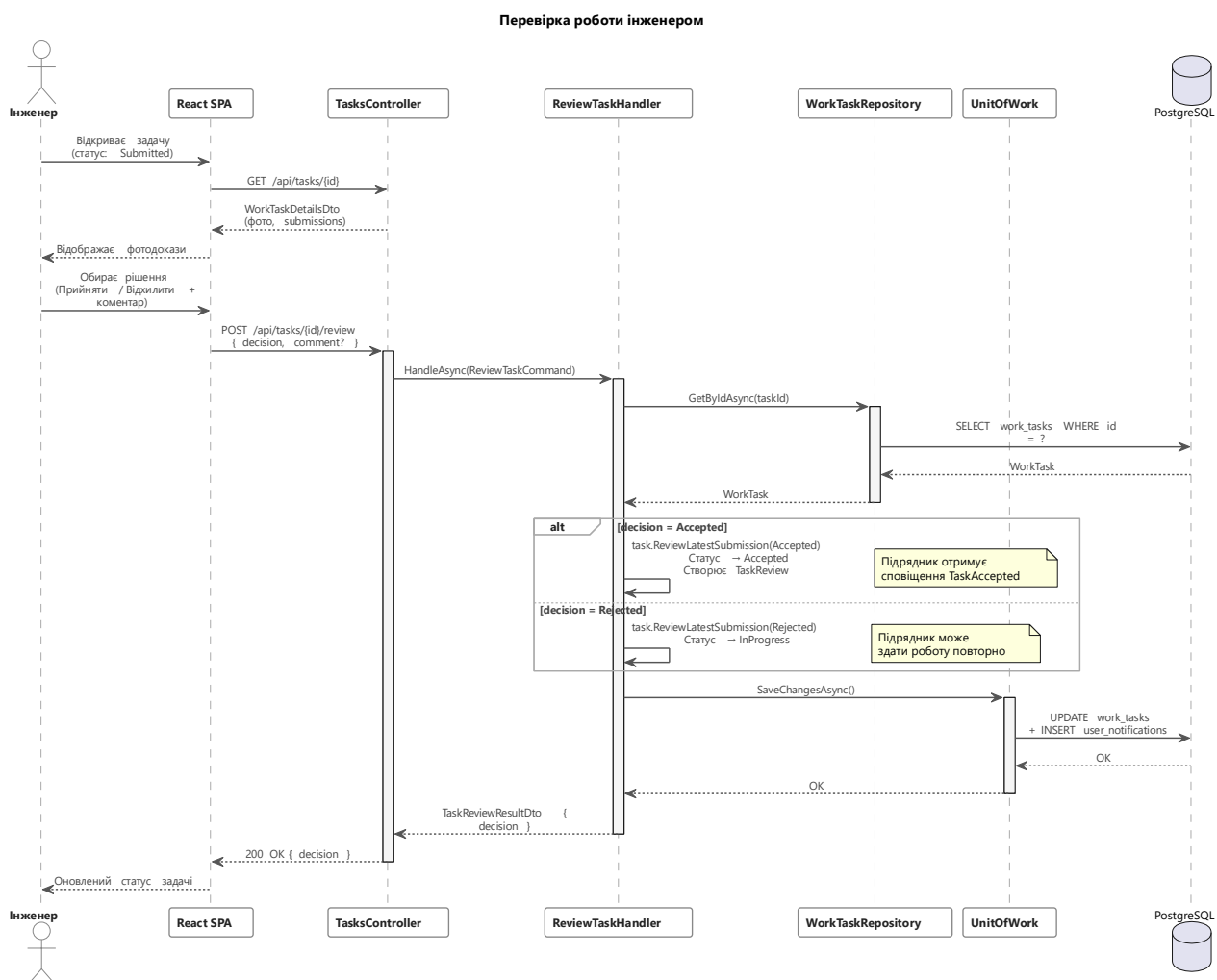


Рис. Е.1 – Діаграма послідовності рецензування звіту

Фрагменти коду серверної частини

Ж.1 Команда подання звіту про виконання завдання

```
public sealed record SubmitTaskForReviewCommand(
    WorkTaskId WorkTaskId,
    UserId ActorId,
    IReadOnlyCollection<RoleType> ActorRoles,
    IReadOnlyCollection<PhotoEvidenceInput> Photos,
    string? Note);
```

Ж.2 Обробник команди подання звіту

```
public async Task<TaskSubmissionResultDto> HandleAsync(
    SubmitTaskForReviewCommand command, CancellationToken ct)
{
    await _validator.ValidateAndThrowAsync(command, ct);

    var task = await _workTaskRepository.GetByIdAsync(command.WorkTaskId, ct)
        ?? throw new ApplicationRuleViolationException("Task was not found.");

    var photos = command.Photos
        .Select(p => new PhotoEvidence(
            p.FileName, p.StoragePath, p.ContentType, _clock.UtcNow))
        .ToArray();

    task.SubmitForReview(
        command.ActorId, command.ActorRoles, photos, command.Note,
        _clock.UtcNow);

    await _unitOfWork.SaveChangesAsync(ct);
    return new TaskSubmissionResultDto(task.Id, task.Status);
}
```

Ж.3 Валідатор команди призначення підрядника

```
public sealed class AssignTaskToContractorCommandValidator
    : AbstractValidator<AssignTaskToContractorCommand>
{
    public AssignTaskToContractorCommandValidator()
    {
        RuleFor(x => x.WorkTaskId.Value).NotEmpty();
        RuleFor(x => x.ContractorId.Value).NotEmpty();
        RuleFor(x => x.ActorRoles)
            .NotNull()
            .Must(roles => roles.Count > 0)
            .WithMessage("At least one actor role is required.");
    }
}
```

Ж.4 Метод контролера призначення підрядника

```
[HttpPut("tasks/{taskId:guid}/assignment")]
[Authorize(Roles = "Admin,Manager")]
public async Task<ActionResult> AssignContractor(
    Guid taskId,
    [FromBody] AssignTaskRequest request,
    CancellationToken cancellationToken)
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    var command = new AssignTaskToContractorCommand(
        new WorkTaskId(taskId),
        new UserId(request.ContractorId),
        ActorId,
        ActorRoles);

    await assignTaskToContractorHandler.HandleAsync(command, cancellationToken);
    return NoContent();
}

```

Ж.5 Метод обробника аналітичного запиту

```

using BuildFlow.Application.Abstractions.Repositories;
using BuildFlow.Application.Contracts.Dtos;
using BuildFlow.Application.Exceptions;
using TaskLifecycleStatus = BuildFlow.Domain.ValueObjects.TaskStatus;

namespace
BuildFlow.Application.Features.ConstructionObjects.GetConstructionObjectAnalytics;

public sealed class GetConstructionObjectAnalyticsHandler(
    IConstructionObjectRepository constructionObjectRepository,
    IWorkTaskRepository workTaskRepository)
{
    public async Task<ConstructionObjectAnalyticsDto> HandleAsync(
        GetConstructionObjectAnalyticsQuery query,
        CancellationToken cancellationToken)
    {
        var constructionObject = await
constructionObjectRepository.GetByIdAsync(query.ConstructionObjectId,
cancellationToken)
            ?? throw new ApplicationRuleViolationException("Construction object
was not found.");

        var tasks = await workTaskRepository
            .ListByConstructionObjectIdAsync(query.ConstructionObjectId,
cancellationToken);

        var totalTasks = tasks.Count;
        var plannedCount = tasks.Count(task => task.Status ==
TaskLifecycleStatus.Planned);
        var assignedCount = tasks.Count(task => task.Status ==
TaskLifecycleStatus.Assigned);
        var inProgressCount = tasks.Count(task => task.Status ==
TaskLifecycleStatus.InProgress);
        var submittedCount = tasks.Count(task => task.Status ==
TaskLifecycleStatus.Submitted);
        var acceptedCount = tasks.Count(task => task.Status ==
TaskLifecycleStatus.Accepted);

        var acceptedPercentage = totalTasks == 0
            ? 0m
            : Math.Round(acceptedCount * 100m / totalTasks, 2);

        var totalEstimatedCost = constructionObject.PlannedWorks.Sum(plannedWork
=> plannedWork.EstimatedCost);
        var budgetUsagePercentage = constructionObject.EstimatedBudget == 0m
            ? 0m
            : Math.Round(totalEstimatedCost * 100m /
constructionObject.EstimatedBudget, 2);

        return new ConstructionObjectAnalyticsDto(

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        query.ConstructionObjectId,
        totalTasks,
        plannedCount,
        assignedCount,
        inProgressCount,
        submittedCount,
        acceptedCount,
        acceptedPercentage,
        constructionObject.EstimatedBudget,
        totalEstimatedCost,
        budgetUsagePercentage);
    }
}

```

Ж.6 Сервіс зберігання файлів

```

using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using BuildFlow.Application.Abstractions;
using Microsoft.Extensions.Configuration;

namespace BuildFlow.Infrastructure.Storage;

public sealed class AzureBlobFileStorageService : IFileStorageService
{
    private readonly BlobContainerClient _containerClient;
    private readonly SemaphoreSlim _containerInitializationLock = new(1, 1);
    private bool _containerInitialized;

    public AzureBlobFileStorageService(IConfiguration configuration)
    {
        var connectionString = configuration["AzureStorage:ConnectionString"]
            ?? throw new
InvalidOperationException("AzureStorage:ConnectionString was not found.");

        var containerName = configuration["AzureStorage:ContainerName"]
            ?? throw new InvalidOperationException("AzureStorage:ContainerName
was not found.");

        var blobServiceClient = new BlobServiceClient(connectionString);
        _containerClient =
blobServiceClient.GetBlobContainerClient(containerName);
    }

    public async Task<string> UploadAsync(
        Stream content,
        string originalFileName,
        string contentType,
        string folder,
        CancellationToken cancellationToken)
    {
        await EnsureContainerInitializedAsync(cancellationToken);

        var safeFolder = string.IsNullOrEmpty(folder)
            ? "files"
            : folder.Trim().Trim('/');

        var extension = Path.GetExtension(originalFileName);
        var blobName = $"{safeFolder}/{Guid.NewGuid():N}{extension}";

        var blobClient = _containerClient.GetBlobClient(blobName);

        var uploadOptions = new BlobUploadOptions

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            HttpHeaders = new BlobHttpHeaders
            {
                ContentType = string.IsNullOrEmpty(contentType) ?
"application/octet-stream" : contentType
            }
        };

        await blobClient.UploadAsync(content, uploadOptions, cancellationTokens);

        return blobClient.Uri.ToString();
    }

    private async Task EnsureContainerInitializedAsync(CancellationTokens
cancellationTokens)
    {
        if (_containerInitialized)
        {
            return;
        }

        await _containerInitializationLock.WaitAsync(cancellationTokens);
        try
        {
            if (_containerInitialized)
            {
                return;
            }

            try
            {
                // Do not request public access level; many storage accounts
disable public access.
                await
_containerClient.CreateIfNotExistsAsync(PublicAccessType.None,
cancellationTokens: cancellationTokens);
                _containerInitialized = true;
            }
            catch (Azure.RequestFailedException ex)
            {
                // If creation fails due to account-level public access
restrictions or permission issues,
                // surface a clearer message but allow the application to
continue where appropriate.
                throw new InvalidOperationException("Failed to initialize blob
container. Check Azure Storage account settings and credentials.", ex);
            }
        }
        finally
        {
            _containerInitializationLock.Release();
        }
    }
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестовий клас для доменного рівня сутності WorkTask

```

using BuildFlow.Domain.Entities;
using BuildFlow.Domain.Exceptions;
using BuildFlow.Domain.Identifiers;
using BuildFlow.Domain.ValueObjects;
using FluentAssertions;
using TaskLifecycleStatus = BuildFlow.Domain.ValueObjects.TaskStatus;

namespace BuildFlow.UnitTests.Domain;

public sealed class WorkTaskTests
{
    [Fact]
    public void SubmitForReview_WithoutPhotos_ShouldThrow()
    {
        var contractorId = UserId.New();
        var task = CreateAssignedTask(contractorId);
        task.MoveToInProgress(contractorId, [RoleType.Contractor],
            DateTimeOffset.UtcNow);

        var act = () => task.SubmitForReview(
            contractorId,
            [RoleType.Contractor],
            Array.Empty<PhotoEvidence>(),
            null,
            DateTimeOffset.UtcNow);

        act.Should().Throw<DomainRuleViolationException>()
            .WithMessage("*At least one photo*");
    }

    [Fact]
    public void ReviewLatestSubmission_RejectedWithoutComment_ShouldThrow()
    {
        var contractorId = UserId.New();
        var engineerId = UserId.New();
        var task = CreateAssignedTask(contractorId);
        task.MoveToInProgress(contractorId, [RoleType.Contractor],
            DateTimeOffset.UtcNow);
        task.SubmitForReview(contractorId, [RoleType.Contractor],
            [CreatePhoto()], null, DateTimeOffset.UtcNow);

        var act = () => task.ReviewLatestSubmission(
            engineerId,
            [RoleType.Engineer],
            ReviewDecision.Rejected,
            null,
            DateTimeOffset.UtcNow);

        act.Should().Throw<DomainRuleViolationException>()
            .WithMessage("*Review comment is required*");
    }

    [Fact]
    public void ReviewLatestSubmission_Rejected_ShouldMoveToNeedsRework_AndAllowResubmit()
    {
        var contractorId = UserId.New();

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        var engineerId = UserId.New();
        var task = CreateAssignedTask(contractorId);
        task.MoveInProgress(contractorId, [RoleType.Contractor],
DateTimeOffset.UtcNow);
        task.SubmitForReview(contractorId, [RoleType.Contractor],
[CreatePhoto()], "first submit", DateTimeOffset.UtcNow);

        task.ReviewLatestSubmission(
            engineerId,
            [RoleType.Engineer],
            ReviewDecision.Rejected,
            "Need better finishing",
            DateTimeOffset.UtcNow);

        task.Status.Should().Be(TaskLifecycleStatus.NeedsRework);

        task.SubmitForReview(contractorId, [RoleType.Contractor],
[CreatePhoto()], "second submit", DateTimeOffset.UtcNow);

        task.Status.Should().Be(TaskLifecycleStatus.Submitted);
        task.Submissions.Should().HaveCount(2);
    }

    [Fact]
    public void MoveInProgress_ByAnotherContractor_ShouldThrow()
    {
        var contractorId = UserId.New();
        var anotherContractor = UserId.New();
        var task = CreateAssignedTask(contractorId);

        var act = () => task.MoveInProgress(anotherContractor,
[RoleType.Contractor], DateTimeOffset.UtcNow);

        act.Should().Throw<DomainRuleViolationException>()
            .WithMessage("*Only assigned contractor*");
    }

    [Fact]
    public void ReviewLatestSubmission_Accepted_ShouldSetTerminalStatus()
    {
        var contractorId = UserId.New();
        var engineerId = UserId.New();
        var managerId = UserId.New();
        var task = CreateAssignedTask(contractorId);
        task.MoveInProgress(contractorId, [RoleType.Contractor],
DateTimeOffset.UtcNow);
        task.SubmitForReview(contractorId, [RoleType.Contractor],
[CreatePhoto()], null, DateTimeOffset.UtcNow);

        task.ReviewLatestSubmission(
            engineerId,
            [RoleType.Engineer],
            ReviewDecision.Accepted,
            null,
            DateTimeOffset.UtcNow);

        task.Status.Should().Be(TaskLifecycleStatus.Accepted);

        var act = () => task.AssignToContractor(contractorId, managerId,
[RoleType.Manager], DateTimeOffset.UtcNow);
        act.Should().Throw<DomainRuleViolationException>()
            .WithMessage("*cannot be reassigned*");
    }

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

[Fact]
public void UpdateDetails_WhenValidInput_ShouldChangeTitleAndDescription()
{
    var task = CreatePlannedTask();

    task.UpdateDetails("Updated title", "Updated description");

    task.Title.Should().Be("Updated title");
    task.Description.Should().Be("Updated description");
}

[Fact]
public void UpdateDetails_WhenTitleIsEmpty_ShouldThrow()
{
    var task = CreatePlannedTask();

    var act = () => task.UpdateDetails(string.Empty, "Updated description");

    act.Should().Throw<DomainRuleViolationException>()
        .WithMessage("*Task title is required.*");
}

[Fact]
public void EnsureCanBeDeleted_WhenTaskIsNotPlanned_ShouldThrow()
{
    var contractorId = UserId.New();
    var task = CreateAssignedTask(contractorId);

    var act = () => task.EnsureCanBeDeleted();

    act.Should().Throw<DomainRuleViolationException>()
        .WithMessage("*Only planned task can be deleted.*");
}

private static WorkTask CreateAssignedTask(UserId contractorId)
{
    var managerId = UserId.New();
    var task = new WorkTask(
        WorkTaskId.New(),
        ConstructionObjectId.New(),
        new WorkTaskDefinition(
            "Install windows",
            "Floor 2 living room",
            new TimeWindow(DateOnly.FromDateTime(DateTime.UtcNow),
                DateOnly.FromDateTime(DateTime.UtcNow.AddDays(5)))),
        managerId,
        DateTimeOffset.UtcNow);

    task.AssignToContractor(contractorId, managerId, [RoleType.Manager],
        DateTimeOffset.UtcNow);
    return task;
}

```

Тестові класи обробників

```

using BuildFlow.Application.Abstractions.Repositories;
using BuildFlow.Application.Exceptions;
using BuildFlow.Application.Features.Users.GetEngineerList;
using BuildFlow.Domain.Entities;
using BuildFlow.Domain.Identifiers;
using BuildFlow.Domain.ValueObjects;
using FluentAssertions;
using Moq;

namespace BuildFlow.UnitTests.Application;

public sealed class EngineerHandlersTests
{
    [Theory]
    [InlineData(RoleType.Admin)]
    [InlineData(RoleType.Manager)]
    [InlineData(RoleType.Engineer)]
    public async Task GetEngineerList_ByPrivilegedRole_ShouldReturnList(RoleType role)
    {
        var users = new[]
        {
            CreateUser("alice@example.com", "Alice", RoleType.Engineer),
            CreateUser("bob@example.com", "Bob", RoleType.Engineer),
        };

        var repo = new Mock<IUserRepository>();
        repo.Setup(r => r.ListByRoleAsync(RoleType.Engineer, default)).ReturnsAsync(users);

        var handler = new GetEngineerListHandler(repo.Object);
        var result = await handler.HandleAsync(
            new GetEngineerListQuery(UserId.New(), [role]), default);

        result.Should().HaveCount(2);
    }
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        result.Select(u => u.Email).Should().BeEquivalentTo("alice@example.com",
"bob@example.com");
    }

    [Fact]
    public async Task GetEngineerList_ByContractor_ShouldThrow()
    {
        var repo = new Mock<IUserRepository>();
        var handler = new GetEngineerListHandler(repo.Object);

        var act = async () => await handler.HandleAsync(
            new GetEngineerListQuery(UserId.New(), [RoleType.Contractor]),
            default);

        await act.Should().ThrowAsync<ApplicationRuleViolationException>()
            .WithMessage("*admin, manager or engineer*");
    }

    [Fact]
    public async Task GetEngineerList_WhenNoEngineers_ShouldReturnEmpty()
    {
        var repo = new Mock<IUserRepository>();
        repo.Setup(r => r.ListByRoleAsync(RoleType.Engineer, default))
            .ReturnsAsync(Array.Empty<User>());

        var handler = new GetEngineerListHandler(repo.Object);
        var result = await handler.HandleAsync(
            new GetEngineerListQuery(UserId.New(), [RoleType.Admin]), default);

        result.Should().BeEmpty();
    }

    [Fact]
    public async Task GetEngineerList_ShouldMapEmailAndFullName()
    {
        var user = CreateUser("eng@site.com", "Site Engineer",
            RoleType.Engineer);

        var repo = new Mock<IUserRepository>();

```

```

        repo.Setup(r => r.ListByRoleAsync(RoleType.Engineer,
default)).ReturnsAsync([user]);

        var handler = new GetEngineerListHandler(repo.Object);
        var result = await handler.HandleAsync(
            new GetEngineerListQuery(UserId.New(), [RoleType.Manager]),
default);

        var dto = result.Single();
        dto.Email.Should().Be("eng@site.com");
        dto.FullName.Should().Be("Site Engineer");
    }

    private static User CreateUser(string email, string name, RoleType role) =>
        new(UserId.New(), email, name, [role], DateTimeOffset.UtcNow);
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		84