МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: **«Проектування та розробка сервісу реєстрації проблем міста Острога »**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Жарчинський Маріан Сергійович

Керівник:
викладач кафедри інформаційних технологій та аналітики даних
Місай Володимир Віталійович

Рецензент:
кандидат технічних наук, доцент,
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «20» травня 2026 р.

Острог, 2026

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проектування та розробка сервісу реєстрації проблем міста Острога

Автор: Жарчинський Маріан Сергійович

Науковий керівник: викладач кафедри ІТАД, Місай Володимир Віталійович

Захищена «……»…… 2026 року.

Пояснювальна записка до кваліфікаційної роботи: *54с., 27 рис., 1 табл., 0 додатків, 25 джерел.*

Ключові слова: *ВЕБ-ПЛАТФОРМА, .NET, REACT, ШТУЧНИЙ ІНТЕЛЕКТ, AWS, CI/CD, CLEAN ARCHITECTURE, РЕЄСТРАЦІЯ ПРОБЛЕМ, ОСТРОГ.*

Короткий зміст праці:

Кваліфікаційна робота присвячена створенню веб-платформи "Острог Разом", яка забезпечує взаємодію між мешканцями та комунальними службами міста Острога. Реалізовано клієнт-серверну архітектуру з використанням .NET 10 (Backend) та React 19 (Frontend). Увагу приділено інтеграції штучного інтелекту для обробки даних та розгортанню хмарної інфраструктури на базі AWS. Система дозволяє користувачам повідомляти про проблеми, відстежувати їхній статус на карті та коментувати. Сервіс є сучасним рішенням для залучення громадян до вирішення локальних проблем.

_____

ANNOTATION

of qualification paper

for bachelor's degree

Theme: Design and development of a problem registration service for the city of Ostroh

Author: Marian Zharchynskyi

Scientific supervisor: Lecturer of the Department of ITAD, Misai Volodymyr Vitalievich

Defended: «……»…….…… 2026 year.

Explanatory note to the qualification work: 54 pages, 27 figures, 1 table, 0 appendices, 25 sources.

Keywords: *WEB PLATFORM, .NET, REACT, ARTIFICIAL INTELLIGENCE, AWS, CI/CD, CLEAN ARCHITECTURE, PROBLEM REGISTRATION, OSTROH, BACKEND, FRONTEND.*

Abstract:

The qualification thesis is devoted to the creation of the "Острог Разом" web platform, which provides interaction between residents and municipal services of the city of Ostroh. A client-server architecture was implemented using .NET 10 (Backend) and React 19 (Frontend). Special attention was paid to the integration of artificial intelligence for data processing and the deployment of cloud infrastructure based on AWS. The system allows users to report problems, track their status on the map, and leave comments. The service is a modern solution for engaging citizens in solving local problems.

_____

ЗМІСТ

ВСТУП

У сучасному світі цифрова трансформація охоплює всі сфери суспільного життя, включаючи взаємодію громадян з органами місцевого самоврядування. Концепція Smart City стає частиною стратегій розвитку багатьох міст. Ця концепція передбачає використання IT-технологій для підвищення якості життя та ефективності управління міськими ресурсами. Ефективне управління неможливе без прозорого механізму зворотного зв'язку між жителями та владою. Своєчасне виявлення та усунення інфраструктурних проблем є критично важливим для забезпечення безпеки та комфорту громадян.

Актуальність теми зумовлена тим, що в місті Острозі відсутня єдина централізована інформаційна система. Така система дозволяла б мешканцям, студентам і туристам ефективно повідомляти про виявлені міські проблеми. Традиційні підходи до обробки звернень, як-от паперові заяви чи телефонні дзвінки, є застарілими. Вони характеризуються бюрократизацією, низькою швидкістю та відсутністю прозорості у відстеженні статусу заявок. Такі методи не дозволяють здійснювати просторовий аналіз проблем та акумулювати статистику для прийняття управлінських рішень. Впровадження веб-платформи з використанням штучного інтелекту та хмарних обчислень здатне змінити підхід до управління міським середовищем.

Метою кваліфікаційної роботи є проектування та розробка програмного продукту - веб-сервісу «Острог Разом», який забезпечить ефективний моніторинг та управління процесом вирішення міських проблем за допомогою інтелектуальних алгоритмів та хмарної інфраструктури.

Завдання дослідження охоплюють ґрунтовний аналіз предметного середовища, проєктування архітектури та інформаційної модели сервісу, а також обґрунтований вибір сучасного інструментарію, методів програмної реалізації та тестування веб-платформи для забезпечення ефективного вирішення поставленої мети. Для досягнення цієї мети необхідно реалізувати модулі авторизації, інтерактивної

картографії, черги реального часу для сповіщень та інтелектуальний модуль первинної обробки контенту.

Об’єктом дослідження є процес реєстрації, обробки та управління вирішенням міських проблем у контексті взаємодії громадян та органів самоврядування. Предметом дослідження виступають методи, інструментальні засоби та інформаційні технології, які використовуються для проектування веб-орієнтованих систем з інтеграцією штучного інтелекту.

Практичне значення полягає у створенні готового до впровадження програмного продукту, який дозволить автоматизувати обробку звернень громадян в Острозі. Використання платформи комунальними підприємствами сприятиме підвищенню ефективності роботи та зменшенню часу реакції на критичні інциденти. Спроектована система має високий потенціал до масштабування та може бути адаптована для впровадження в інших малих і середніх громадах України з метою покращення їхньої цифрової стійкості.

Під час виконання кваліфікаційної роботи та підготовки пояснювальної записки інструменти на базі великих мовних моделей (зокрема, Google Gemini та ChatGPT) використовувалися як допоміжний інженерний та дослідницький асистент. ШІ-моделі були задіяні для формалізації та генерації складних технічних описів, обґрунтування багаторівневих архітектурних рішень, написання базових шаблонів конфігураційних файлів інфраструктури (Terraform, пайплайни GitHub Actions), а також для створення розгорнутої технічної документації. Такий підхід дозволив автоматизувати рутинні інженерні процеси документування та зосередити основну увагу на реалізації складної бізнес-логіки системи. Усі згенеровані фрагменти коду та тексту пройшли ретельну експертну перевірку, адаптацію та верифікацію автором відповідно до вимог предметної області дослідження.

РОЗДІЛ 1

ЗАГАЛЬНІ ПОЛОЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметного середовища та постановка проблеми

Предметним середовищем системи є процес взаємодії між мешканцями, комунальними підприємствами та владою щодо виявлення, моніторингу та усунення локальних проблем. Основним завданням цього процесу є оперативне отримання достовірної інформації про стан інфраструктури: пошкодження доріг, несправності освітлення, проблеми з вивезенням відходів, порушення благоустрою та інші ситуації. Управління сучасним містом неможливе без постійного моніторингу цих аспектів, оскільки вони впливають на безпеку громадян.

На сьогодні процес обробки звернень є здебільшого непрозорим. Мешканець, який виявив проблему, часто не знає, до якої служби звертатися. Процес подачі заявки перетворюється на квест із дзвінками чи візитами до установ. Після реєстрації звернення громадянин фактично втрачає можливість відстежувати його виконання. З боку адміністрації міста також існують складнощі: звернення надходять різними каналами, інформація часто неповна. Найбільшою проблемою є те, що заявкам бракує географічної прив'язки та візуального підтвердження. Це ускладнює класифікацію та призначення виконавців. Відсутність єдиної бази даних унеможливлює аналітику: неможливо швидко визначити проблемні райони чи проконтролювати ефективність роботи бригад.

Для вирішення цих проблем необхідно створити єдину цифрову платформу. У межах її функціонування виділяються декілька ключових ролей. Основною роллю є користувач, який виступає ініціатором процесу. Його функція полягає у реєстрації проблем шляхом надання опису, фотографій та вказування місця на карті. Користувачі повинні мати можливість переглядати заявки інших, залишати коментарі та отримувати сповіщення про зміну статусу звернень.

Другою роллю є координатор, який представляє комунальні служби. Він відповідає за первинну та подальшу обробку проблем. Координатор перевіряє заявку, визначає категорію, призначає виконавців та змінює статус проблеми (наприклад, з «Нова» на «В роботі»). Після усунення проблеми він переводить заявку в статус «Виконано» та додає фотозвіт. Третя роль - адміністратор системи, який володіє повним спектром прав для управління платформою. Він керує обліковими записами, моніторить стабільність роботи та управляє інтеграціями. Адміністратор відповідає за вирішення непередбачених ситуацій та забезпечення безпеки даних.

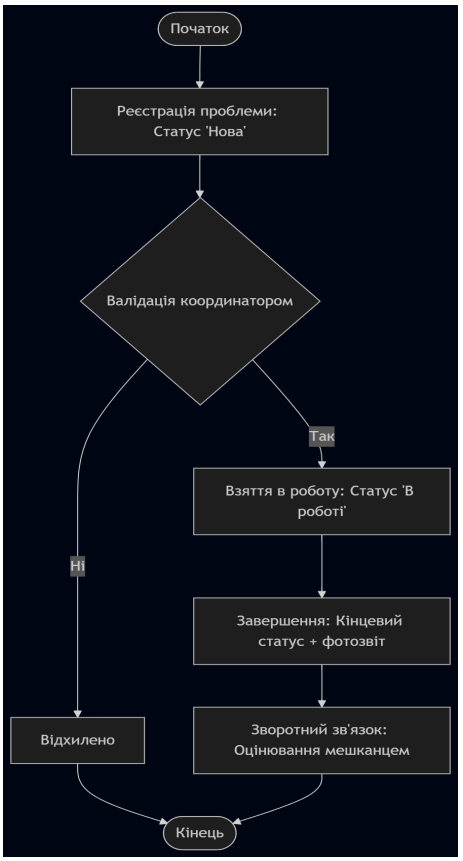

Рис. 1.1. Діаграма діяльності процесу обробки звернення в систем

Джерело: розроблене автором

Процес обробки звернення в системі «Острог Разом» побудований за принципом замкнутого циклу, що візуалізовано на діаграмі діяльності (рис. 1.1). На етапі реєстрації мешканцем проблема автоматично отримує статус «Нова». Після перевірки та валідації даних координатором, звернення переходить у статус «В

роботі» та передається на виконання комунальним службам. По завершенню робіт статус змінюється на кінцевий, що супроводжується завантаженням підтверджуючих фотоматеріалів. Заключним етапом є отримання зворотного зв’язку від мешканця у формі оцінки якості виконаної роботи. Такий алгоритм дозволяє реалізувати прозорий та контрольований цикл обробки даних.

1.2. Огляд та аналіз наявних аналогів

Серед міжнародних рішень вагомим аналогом є британська краудсорсингова платформа «FixMyStreet», яка успішно масштабована у багатьох країнах Європи. Основними перевагами цієї системи є простота інтерфейсу користувача та чітка прив'язка інцидентів до географічної карти, що мінімізує час на заповнення текстових форм. Платформа автоматично перенаправляє повідомлення до відповідного муніципального органу на основі координат. Проте, «FixMyStreet» функціонує переважно як посередник і не має глибоко інтегрованих внутрішніх інструментів для координаторів комунальних служб, що ускладнює повний життєвий цикл контролю за виконанням робіт. Окрім цього, система обмежена класичними методами фільтрації та потребує ручної перевірки у випадках, коли опис проблеми є двозначним або містить мультимедійні дані без чіткого текстового контексту.

Відомим прикладом в Україні є муніципальна система «Контактний центр міста Києва» (1551). Вона забезпечує подачу заявок через веб-портал та мобільний додаток. Перевагою платформи є її інтеграція з комунальними службами та контроль за виконанням робіт. Користувачі відстежують статус заявок та переглядають статистику. Однак проект має недоліки: інтерфейс застарілий і перевантажений елементами, що знижує конверсію подачі заявок. Критичним недоліком є відсутність механізмів обробки даних на основі штучного інтелекту: модерація та класифікація заявок потребують залучення операторів, що призводить до затримок.

Через високе навантаження на диспетчерську службу в періоди пікових інфраструктурних збоїв час первинного розгляду повідомлень суттєво зростає, що знижує загальну ефективність реагування. Крім того, система практично не пристосована до автоматичного аналізу медіафайлів (фото- та відеодоказів) та неструктурованих аудіоповідомлень, які надходять від громадян. Це унеможливлює швидке дублювання та групування схожих інцидентів за географічним або тематичним принципом без прямої участі людини. Як наслідок, виникає потреба у створенні більш гнучкої, інтелектуальної архітектури, здатної самостійно інтерпретувати вхідний потік різнорідних муніципальних даних.

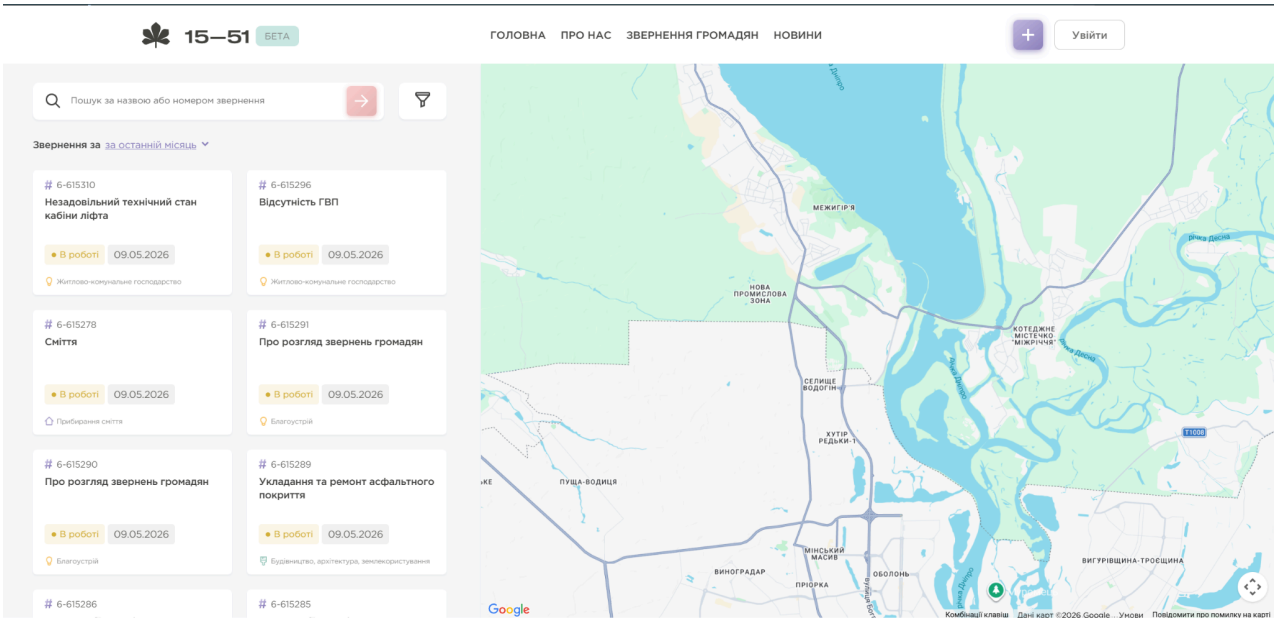

Рис. 1.2. Приклад платформи «Контактний центр міста Києва» (1551)

Джерело: https://1551.gov.ua/main

Іншим типом рішень є комерційні платформи на базі концепції Smart City, впроваджені в містах обласного значення. Вони пропонують базовий функціонал для подачі звернень. Їхня технічна реалізація часто базується на застарілих монолітних архітектурах та CMS типу WordPress. Вони не призначені для високонавантажених сервісів з геоданими. Функціонал зворотного зв'язку обмежений: вони не підтримують обмін повідомленнями в реальному часі, що ускладнює комунікацію.

На міжнародному ринку існують розвинені платформи, такі як SeeClickFix, що успішно функціонують у містах США та Канади. Вони відрізняються сучасним дизайном, високою швидкістю та ефективним використанням векторних картографічних даних. Вони надають широкі можливості для соціальної взаємодії та формують аналітику. Проте, вартість ліцензування та розгортання таких рішень є занадто високою для бюджетів невеликих міст України, таких як Острог.

Таблиця 1.1

Порівняльна характеристика існуючих аналогів та розроблюваної системи

Критерій порівняння	**Контактний центр 1551 (Київ)**	**Комерційні Smart City платформи**	**SeeClickFix (США/Канада)**
Сучасний UI	Базовий сучасний	Базовий (на базі CMS)	Високий рівень UX/UI
Інтеграція ШІ	Відсутня	Відсутня	Обмежена
Робота у Реальному часі	Відсутній	Обмеження архітектури	Часткова підтримка
Робота з геоданими	Базова	Статичні карти	Розвинена

Як демонструє таблиця, існуючі аналоги не повною мірою задовольняють вимоги до інтерактивності та автоматизації процесів. Більшість використовує застарілі підходи та ігнорує можливості машинного навчання. Тому розробка системи «Острог Разом», яка поєднує сучасні архітектурні рішення, реактивні інтерфейси та алгоритми штучного інтелекту, є актуальним завданням.

1.3. Постановка задачі та вимоги до системи

З огляду на результати аналізу предметної області та виявлені недоліки існуючих аналогів, було сформульовано вимоги до інформаційної системи «Острог Разом». Ці вимоги визначають вектор розробки та слугують критеріями для оцінки ефективності готового продукту. Функціональні вимоги охоплюють основні можливості, які повинні бути надані користувачам. Система має забезпечувати швидку автентифікацію. Для цього інтегрується хмарний сервіс Clerk, що дозволить вхід за допомогою облікових записів соціальних мереж та забезпечить захист персональних даних відповідно до протоколів безпеки.

Ключовим функціоналом є можливість створення повідомлення про виявлену проблему. Користувач повинен заповнити форму, де вказуватиметься назва проблеми, детальний опис, категорія та точна географічна локація на інтерактивній карті міста. Система повинна підтримувати завантаження фотографій для візуального підтвердження проблеми. На головній карті повинні відображатися маркери зареєстрованих проблем. Для стимулювання соціальної взаємодії система має реалізувати функціонал обговорення: авторизовані користувачі зможуть залишати коментарі, а система повинна гарантувати їх оновлення в режимі реального часу (завдяки веб-сокетам).

Окремою вимогою є інтеграція модуля штучного інтелекту. Цей модуль має виступати в ролі інтелектуального помічника, який прийматиме голосові звернення і за допомогою алгоритмів автоматично перетворюватиме їх у готову заявку із виокремленими атрибутами (адреса, категорія, пріоритетність).

Нефункціональні вимоги визначають технічні характеристики системи. Серверна частина повинна бути спроектована за принципами Clean Architecture, що вимагає розділення бізнес-логіки, доступу до даних та презентаційного шару. Клієнтська частина повинна бути реалізована як односторінковий додаток (SPA) на базі реактивних фреймворків.

Для забезпечення надійності та швидкої масштабованості система повинна бути контейнеризована (Docker). Розгортання компонентів має здійснюватися в хмарному середовищі (AWS). Вимогою є повна автоматизація процесів розгортання через використання підходу «інфраструктура як код» (IaC) та пайплайнів CI/CD, що виключає людський фактор при оновленні системи.

Висновки до розділу 1

У першому розділі було проведено дослідження теоретичних аспектів предметної області управління міськими інфраструктурними проблемами. Проаналізовано існуючий процес взаємодії між мешканцями та місцевими службами. Виділено ключові ролі користувачів та визначено життєвий цикл типової проблеми в рамках інформаційної системи. Виявлено, що традиційні методи комунікації є недостатньо ефективними та непрозорими, що обґрунтовує необхідність переходу до цифрових рішень.

Здійснений огляд вітчизняних та зарубіжних аналогів показав, що наявні продукти мають архітектурні та функціональні недоліки, такі як: монолітні технологічні підходи, відсутність обміну даними в реальному часі, проблеми з масштабуванням та ігнорування можливостей штучного інтелекту.

Базуючись на виявлених недоліках, було сформульовано постановку задачі для проектування нової веб-платформи «Острог Разом». Визначено перелік функціональних вимог, що включають роботу з інтерактивною картою, підтримку реального часу та використання алгоритмів штучного інтелекту.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Інформаційна модель та структура даних

Для створення надійної інформаційної системи першочерговим завданням є побудова чіткої інформаційної моделі. Ця модель повинна відображати всі сутності предметної області та реляційні зв'язки між ними. Інформаційне забезпечення формується на основі аналізу інформаційних потоків у процесі взаємодії громадян із службами. Якісно спроектована модель даних є фундаментом, який забезпечує цілісність та швидкодію платформи.

Основними інформаційними об'єктами (сутностями), які зберігаються в базі даних, є користувачі, проблеми, текстові коментарі, медіафайли та оцінки. Кожна сутність має унікальний набір атрибутів та перебуває у визначених відношеннях з іншими сутностями.

Сутність «Користувач» є базовою для забезпечення авторизованого доступу. Архітектурне рішення платформи передбачає використання гібридної моделі зберігання даних. Оскільки система інтегрована з хмарним провайдером ідентифікації (Clerk), у локальній базі зберігається лише мінімум публічної інформації. Кожен користувач має ідентифікатор, який синхронізується з сервісом ідентифікації за допомогою веб-хуків. Основні атрибути включають: ім'я, прізвище, пошту, URL-адресу аватара та системну роль (Користувач, Координатор, Адміністратор). Ролева модель є ключовим елементом безпеки системи, визначаючи права доступу користувача. Центральною сутністю системи виступає «Проблема». Вона акумулює інформацію щодо виявленого недоліку інфраструктури. Її атрибутивний склад включає: унікальний ідентифікатор, заголовок, детальний опис, географічні координати (широта та довгота), поточний статус вирішення, зовнішній ключ на ідентифікатор автора, категорію проблеми та системні часові мітки створення та оновлення. Статус проблеми є динамічним атрибутом: «Нова», «В роботі», «Виконано» або «Відхилено».

Для забезпечення соціальної взаємодії реалізовано сутність «Коментар». Вона пов'язується з конкретною проблемою відношенням «один до багатьох» і містить текст повідомлення, ідентифікатор автора та час публікації. Аналогічно спроектовано сутність «Зображення», яка зберігає текстові посилання на фізичні фотоматеріали, завантажені у зовнішнє хмарне сховище. Сутність «Рейтинг» дозволяє реалізувати механізм визначення задоволеності якістю вирішення проблеми шляхом підрахунку оцінок від 1 до 5 балів.

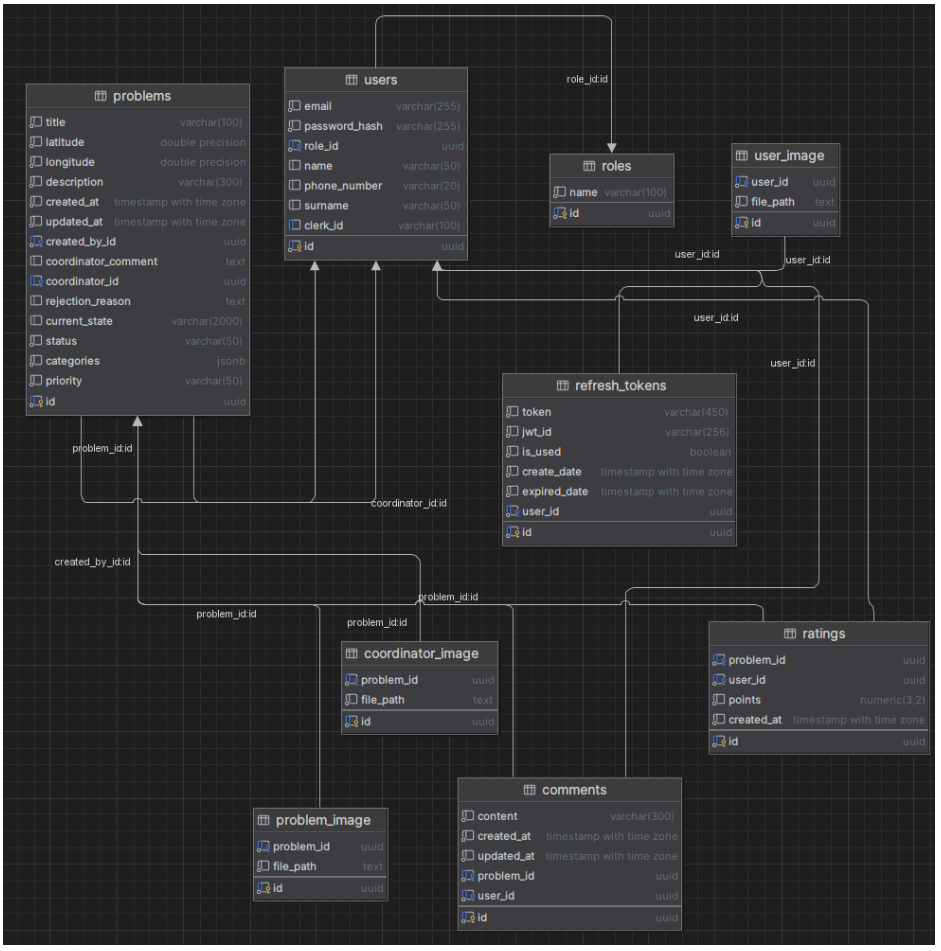

Рис. 2.1. ER-діаграма реляційної бази даних системи

Джерело: розроблене автором

Завдяки відображеній на рисунку структурі бази даних забезпечується висока продуктивність запитів та надійна цілісність даних. Вхідні дані безперервно генеруються кінцевими користувачами через клієнтський веб-інтерфейс: реєстраційні дані, текстова інформація, фотографії та геопросторові координати. Вихідні дані формуються сервером і включають JSON-відповіді з інформацією про

проблеми, результати роботи алгоритмів штучного інтелекту, а також події реального часу, що транслюються через веб-сокети.

2.2. Проектування архітектури системи

Основним архітектурним патерном, обраним для розробки серверної частини, є чиста архітектура (Clean Architecture). Цей підхід спрямований на створення надійних систем, які є незалежними від фреймворків, баз даних та інтерфейсів. Парадигма чистої архітектури полягає у строгому розділенні програмного забезпечення на ізольовані шари, де вектори залежностей спрямовані виключно всередину, до бізнес-логіки. Архітектура серверного додатку платформи логічно розділена на чотири ключові проекти, що відповідають окремим шарам:

1. **Доменний шар (Domain Layer).** Це ядро системи, яке містить сутності предметної області, об'єкти значень та перелічення. Цей шар не має залежностей від інших проектів. Тут інкапсульовані найбільш загальні правила бізнесу, які не змінюються при зміні бази даних.

2. **Шар застосунку (Application Layer).** Він відповідає за бізнес-логіку програми та оркеструє взаємодію між доменними об'єктами. Він містить інтерфейси репозиторіїв, інтерфейси для інтеграційних сервісів та обробники команд і запитів. Взаємодія в цьому шарі побудована на базі архітектурного патерну CQRS. Цей підхід дозволяє архітектурно розділити операції швидкого читання та складні операції зміни стану, що спрощує підтримку коду та масштабування.

3. **Інфраструктурний шар (Infrastructure Layer).** Тут знаходяться конкретні технічні реалізації абстрактних інтерфейсів. Цей шар взаємодіє з операційною системою, базою даних, файловими сховищами, сторонніми веб-API. Для роботи з реляційною базою даних PostgreSQL використовується ORM. Тут реалізована інтеграція з S3-сховищем, інтеграція з API штучного інтелекту Google Gemini та логіка криптографічної перевірки підписів від сервісу авторизації.

4. **Презентаційний шар (API / Presentation Layer).** Це відкрита точка входу в серверну частину програми. Даний шар містить RESTful контролери, які приймають HTTP-запити, здійснюють їх валідацію, формують об'єкти команд або запитів і передають їх у шар Application. Також у цьому шарі розміщені хаби SignalR для забезпечення двостороннього зв'язку між сервером та клієнтами в режимі реального часу за протоколом WebSockets.

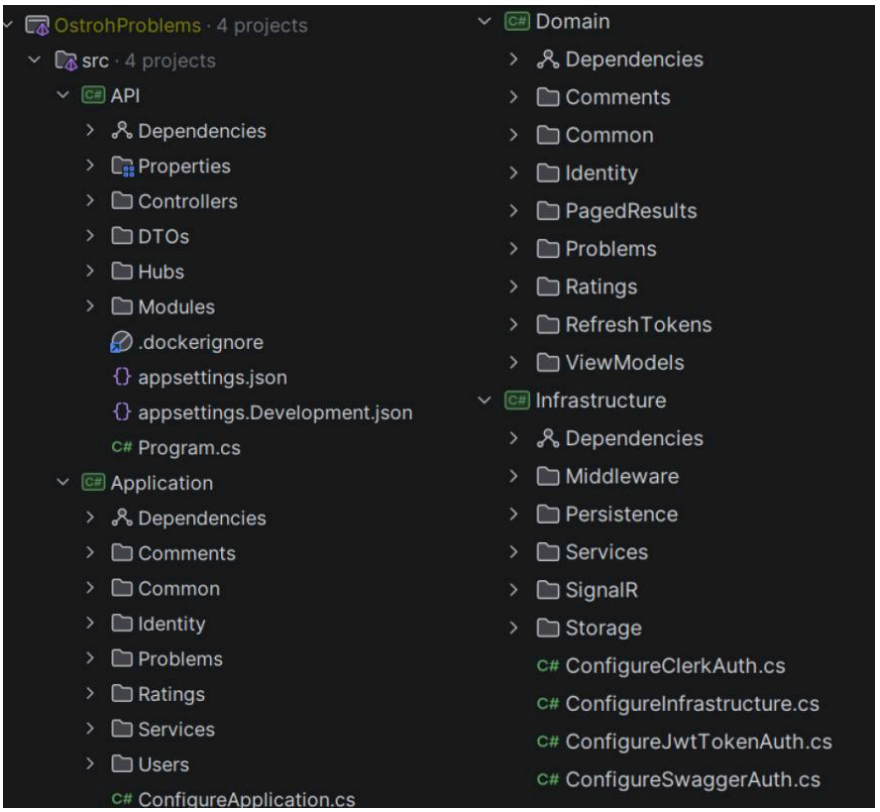

Рис. 2.2. Архітектура системи (Clean Architecture та CQRS)

Джерело: розроблене автором

Як видно з архітектурної схеми, модель авторизації в системі побудована на базі безсерверного хмарного рішення. Взаємодія між сервісом ідентифікації та бекенд-системою відбувається асинхронно. Коли новий користувач реєструється, сервіс миттєво відправляє підписаний HTTP-запит (Webhook) на захищений ендпоінт бекенду. Сервер перевіряє цифровий підпис та безпечно оновлює інформацію в локальній базі даних. Це знімає з розробників тягар зберігання паролів та гарантує найвищий рівень захисту від витоків баз даних.

2.3. Проектування API та механізмів взаємодії

Після моделювання структури бази даних та визначення загальних архітектурних шарів, критично важливим етапом є деталізація інтерфейсу прикладного програмування (API). API виступає сполучною ланкою між клієнтською частиною (Frontend) та внутрішньою логікою сервера (Backend). В системі «Острог Разом» проектування API базується на принципах REST та використанні архітектурного патерну CQRS (Command Query Responsibility Segregation).

Використання CQRS дозволяє розмежувати логіку зміни стану системи та логіку читання даних. Це особливо важливо для високонавантажених картографічних сервісів, де кількість запитів на перегляд маркерів значно перевищує кількість операцій зі створення нових заявок. Для реалізації цього підходу на рівні коду застосовується бібліотека MediatR, яка дозволяє реалізувати слабку зв'язність між контролерами та бізнес-логікою. Кожен HTTP-запит трансформується в об'єкт команди або запиту, що передається відповідному обробнику (Handler). Це забезпечує високу масштабованість та полегшує процес тестування окремих функцій системи.

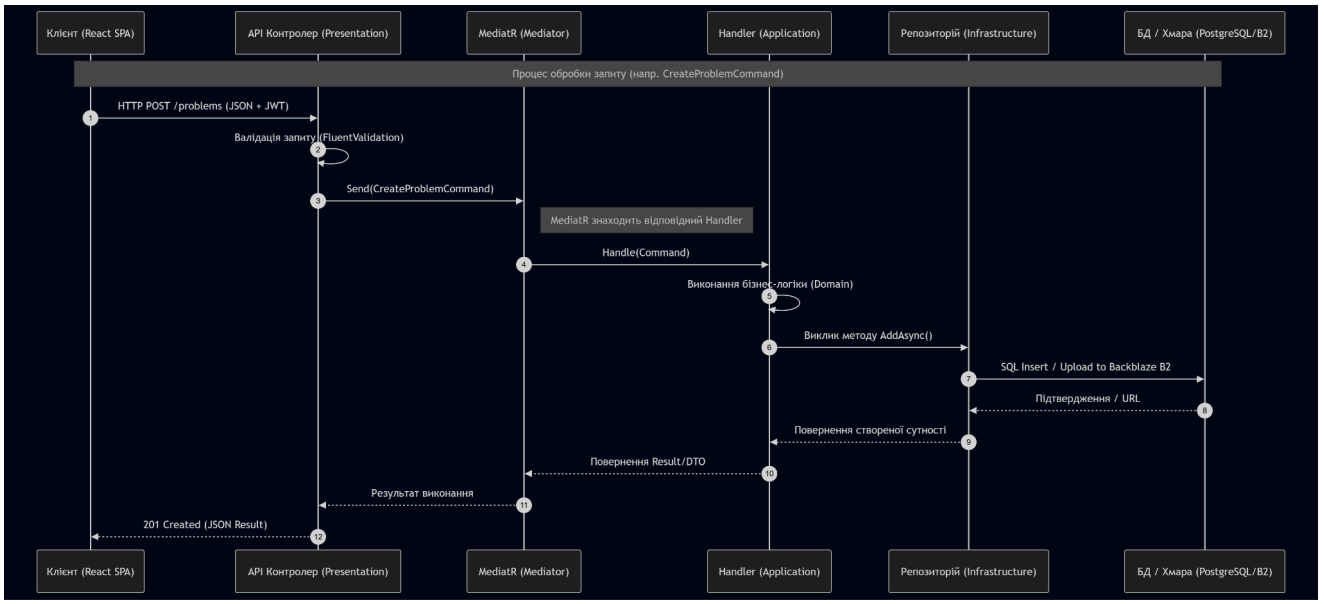

Рис. 2.3. Схема проходження запиту через MediatR та шари архітектури

Джерело: розроблене автором

Окрім стандартних REST-ендпоінтів, система вимагає підтримки двостороннього зв'язку в реальному часі. Для цього спроектовано інтеграцію з бібліотекою SignalR. Це дозволяє реалізувати механізм «живих» сповіщень: коли координатор змінює статус проблеми або додає офіційний коментар, автор проблеми миттєво бачить оновлення без перезавантаження сторінки.

У даному проекті реалізація SignalR підпорядкована загальним принципам Clean Architecture та патерну CQRS, що дозволило уникнути прямої залежності між шаром представлення (API) та механізмами передачі WebSocket-пакетів. Архітектурне рішення базується на використанні «тонких» контролерів (Thin Controllers), обов’язок яких обмежується прийомом вхідних HTTP-запитів та їх передачею в бізнес-шар через медіатор.

Процес реалізації real-time взаємодії (на прикладі модуля коментарів) розділений на три етапи:

1. **Шар представлення (API)**: Контролер CommentsController лише десеріалізує вхідні дані у об'єкт команди CreateCommentCommand та делегує її виконання обробнику через інтерфейс IMediator. Такий підхід гарантує, що контролер залишається незалежним від бізнес-логіки та способів сповіщення клієнтів.

2. **Шар застосунку (Application Layer)**: Саме на цьому рівні, в обробнику команди CreateCommentCommandHandler, зосереджена координація дій. Після успішної валідації та збереження нового коментаря в репозиторії, обробник звертається до абстракції ISignalRService. Використання інтерфейсу замість конкретної реалізації дозволяє дотримуватися принципу інверсії залежностей (DIP). Обробник ініціює одночасне розсилання даних у групу конкретної проблеми для всіх активних слухачів, а також персональне push-сповіщення для автора звернення.

3. **Інфраструктурний шар (Infrastructure Layer)**: Конкретна реалізація сервісу SignalRService розміщена в зовнішньому шарі системи. Вона має доступ до IHubContext, що є частиною фреймворку ASP.NET Core, і виконує

безпосередню фізичну доставку повідомлень за протоколом WebSocket.

Таке архітектурне розмежування має критичне значення для якості програмного коду. Пряме використання SignalR у контролерах призвело б до «забруднення» API-шару логікою сповіщень, підвищуючи рівень зв'язності (Coupling) та ускладнюючи підтримку. Поточна реалізація дозволяє легко ізолювати компоненти під час написання Unit-тестів: оскільки ISignalRService є абстракцією, його можна замінити на заглушку (Mock), перевіряючи логіку збереження коментарів незалежно від мережевих протоколів. Це забезпечує високу гнучкість системи та її готовність до подальшої модифікації без ризику порушення цілісності існуючого функціоналу.

Лістинг 2.1 – Приклад опису контракту API для створення проблеми

```
[Authorize(Roles = $"{RoleNames.Admin}, {RoleNames.User},
{RoleNames.Coordinator}")]

[HttpGet("get-all")]

public async Task<ActionResult<IReadOnlyList<ProblemSummaryDto>>>
GetAll(CancellationToken cancellationToken)

{

    var entities = await problemQueries.GetAll(cancellationToken);

    return entities.Select(e => ProblemSummaryDto.FromDomainModel(e,
imageService.GetImageUrl)).ToList();

}
```

У наведеному лістингу представлено програмну реалізацію ендпоінту GetAll, який відповідає за отримання повного списку зареєстрованих міських проблем. Завдяки атрибуту [Authorize], метод реалізує рольову модель доступу (RBAC), дозволяючи перегляд списку всім авторизованим категоріям користувачів: адміністраторам, координаторам та мешканцям.

Отримані з бази даних доменні моделі конвертуються у полегшений формат

ProblemSummaryDto. Це дозволяє передавати на фронтенд лише необхідну інформацію, оптимізуючи мережевий трафік. Обробка зображень: Під час мапінгу залучається сервіс imageService, який динамічно формує повні URL-адреси до фотографій проблем, що зберігаються у хмарному сховищі. Метод є повністю асинхронним та підтримує CancellationToken для коректного завершення запиту при розриві з'єднання з боку клієнта.

2.4. Інтеграція хмарних технологій та сервісів штучного інтелекту

Сучасна архітектура системи «Острог Разом» передбачає активне використання хмарних сервісів для забезпечення надійності та інтелектуальної обробки даних. Проектування цих інтеграцій вимагає врахування аспектів безпеки та швидкодії.

Для роботи з медіафайлами (фотографіями міських проблеми) обрано хмарне сховище Backblaze B2. Вибір обґрунтований його повною сумісністю з протоколом S3 та високою економічною ефективністю для проектів із великим обсягом неструктурованих даних. Процес взаємодії побудований таким чином, що серверна частина виступає лише посередником, який генерує безпечні посилання для доступу до файлів. Це дозволяє уникнути перевантаження каналів зв’язку сервера при масових зверненнях користувачів до фотогалереї.

Технічно цей механізм реалізовано за допомогою генерації тимчасових підписаних URL-адрес на рівні інфраструктурного сервісу бекенду. Коли користувач завантажує фотографію доказу інциденту, клієнтський додаток React отримує від API зашифроване одноразове посилання, за яким здійснюється пряме завантаження файлу в бакет Backblaze B2, минаючи обчислювальні потужності сервера. Для подальшого відображення контенту в інтерфейсі мапи та сторінок проблем сховище інтегроване з мережею доставки CloudFront (CDN). Це забезпечує “агресивне” кешування статичних зображень на edge-вузлах, мінімізує мережеві затримки (latency) для мешканців Острога та повністю усуває витрати на вихідний трафік

(Egress traffic) між хмарним сховищем та клієнтом, що робить архітектуру зберігання медіафайлів максимально надійною та масштабованою.

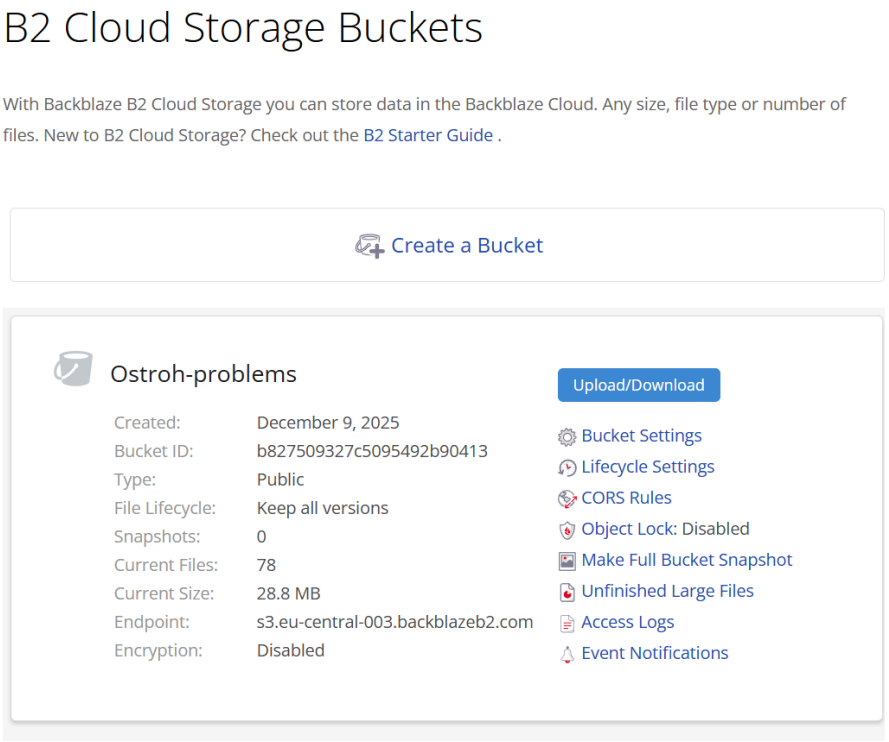

Рис. 2.4. Створене сховище даних на сервісі Backblaze B2

Джерело: розроблене автором

Окремим науково-практичним аспектом проектування є впровадження штучного інтелекту на базі моделі Gemini 3 Flash (версія gemini-3-flash-preview). Вибір даної моделі обґрунтований її високою продуктивністю, низькою затримкою (latency) та передовими можливостями мультимодальної обробки даних, що є критичним для сервісів, які працюють у режимі реального часу. На відміну від класичних інтелектуальних систем, що базуються на каскадних методах (де спочатку голос перетворюється в текст за допомогою сторонніх STT-сервісів, а потім аналізується лінгвістично), у даному проекті спроектовано нативну мультимодальну модель взаємодії.

Використання моделі gemini-3-flash-preview дозволяє системі підтримувати два основні канали подачі звернень: традиційне введення текстового опису через веб-інтерфейс та надсилання «сирих» аудіоповідомлень. У випадку роботи з

голосом, дані передаються до API Gemini у вигляді аудіо-токенів без попередньої транскрибації. Це дозволяє моделі враховувати не лише зміст сказаного, а й інтонаційні акценти, що підвищує точність визначення терміновості проблеми.

Для забезпечення стабільної роботи архітектури, всі ключові інтелектуальні завдання проекту делеговані єдиному обчислювальному ядру Gemini 3 Flash. Це охоплює такі функціональні напрямки:

1. Інтелектуальна обробка аудіо та транскрипція: Пряме перетворення звукового сигналу у структуровані дані.
2. Zero-Shot Classification (Класифікація без вчителя): Завдяки широким знанням моделі про світ, вона здатна самостійно відносити звернення до потрібної категорії (наприклад, «ЖКГ», «Транспорт», «Екологія») без необхідності додаткового донавчання на специфічних наборах даних Острога.
3. Structured JSON Generation: Найважливіший етап для програмної інтеграції. Система використовує механізм суворої генерації JSON, змушуючи модель повертати результат у форматі, що відповідає заздалегідь визначеній схемі даних (Schema Enforcement). Це гарантує, що бекенд-сервер на базі .NET зможе коректно десеріалізувати відповідь ШІ у об’єкт класу Problem із заповненими полями адреси, категорії та рівня пріоритетності.
4. Консультативний чат-бот: Реалізація асистента, який допомагає мешканцям з навігацією по сервісу та надає роз’яснення щодо правил подачі звернень.

Процес взаємодії з API Gemini 3 Flash супроводжується передачею спеціально розробленого системного промпту (System Instructions). Цей інструмент діє як жорстке архітектурне обмеження, яке визначає роль моделі як професійного модератора міських проблем. У промпті зафіксовано перелік існуючих локацій міста Острог, правила класифікації та вимоги до безпеки контенту (Safety Settings). В результаті, нейромережа не просто «спілкується» з користувачем, а виконує математичну задачу мінімізації похибки при виокремленні іменованих сутностей.

Таким чином, поєднання архітектури .NET та інтелектуальних можливостей gemini-3-flash-preview дозволяє створити інноваційний інструмент, де неструктуроване людське звернення перетворюється на чітко структурований цифровий запис за лічені секунди, що радикально підвищує ефективність роботи міських служб.

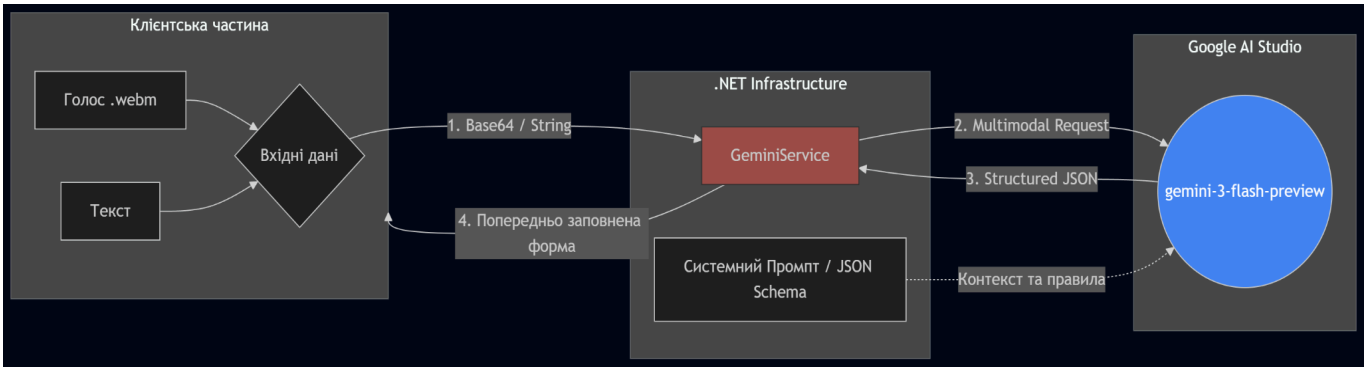

Рис. 2.5. Схема мультимодальної взаємодії API сервісу з моделлю Google Gemini AI

Джерело: розроблене автором

На рисунку 2.5 представлено архітектурну логіку інтелектуального модуля. Процес розпочинається з передачі вхідного сигналу (голосу або тексту) через захищений шлюз бекенду. На відміну від застарілих каскадних моделей (Speech-to-Text -> Text Analysis), архітектура проекту використовує пряму обробку вхідного контексту. Система формує складний запит, що містить інструкції щодо класифікації та обмеження вихідного формату. Це дозволяє миттєво перетворити неструктуроване звернення громадянина у валідний об’єкт бази даних, готовий до обробки координатором без ручного копіювання інформації.

2.5. Архітектура та організація клієнтської частини

Клієнтська частина системи реалізується як сучасний односторінковий додаток (SPA) на базі React 19. Проектування фронтенду в проекті «Острог Разом» зосереджено на забезпеченні швидкодії картографічних компонентів та реактивності інтерфейсу.

Організація коду базується на методології Feature-Sliced Design (FSD). Це дозволяє розділити інтерфейс на незалежні модулі, такі як «Карта проблем», «Форма реєстрації», «Профіль мешканця». Кожна фіча містить власну бізнес-логіку, типи та UI-компоненти. Такий підхід запобігає появі некерованих залежностей у коді, що особливо важливо при розширенні функціоналу, наприклад, при додаванні нових аналітичних дашбордів для адміністрації міста.

Відповідно до архітектурних принципів FSD, клієнтська частина системи структурується за ієрархічними рівнями (layers), де кожен рівень має свою зону відповідальності: від найбільш абстрактних елементів у шарі Shared (універсальні UI-компоненти, стилі та допоміжні функції) до бізнес-суттєвостей у шарі Entities та функціональних сценаріїв у шарі Features. Ключовою перевагою даного підходу є суворе правило спрямованості залежностей: модулі можуть посилатися лише на елементи з нижчих рівнів, що повністю виключає появу циклічних залежностей. Крім того, кожен модуль має чітко визначений «Public API» (через файл index.ts), що приховує внутрішню реалізацію і дозволяє виконувати рефакторинг окремих частин інтерфейсу без ризику порушити цілісність усього додатку. Це робить фронтенд-частину на базі React 19 надзвичайно стійкою до змін та зручною для паралельної розробки декількома програмістами.

Для управління серверним станом (даними, що приходять з API) у проекті спроектовано використання бібліотеки TanStack Query. Вона дозволяє реалізувати механізм кешування даних про проблеми на стороні клієнта. Це означає, що при повторному відкритті карти користувач бачить попередньо завантажені маркери миттєво, доки система виконує запит на перевірку оновлень у фоновому режимі. Безпека клієнтської частини забезпечується інтеграцією з Clerk React SDK, що дозволяє інкапсулювати складні процеси авторизації та керування сесіями, надаючи розробнику готові інструменти для захисту приватних маршрутів.

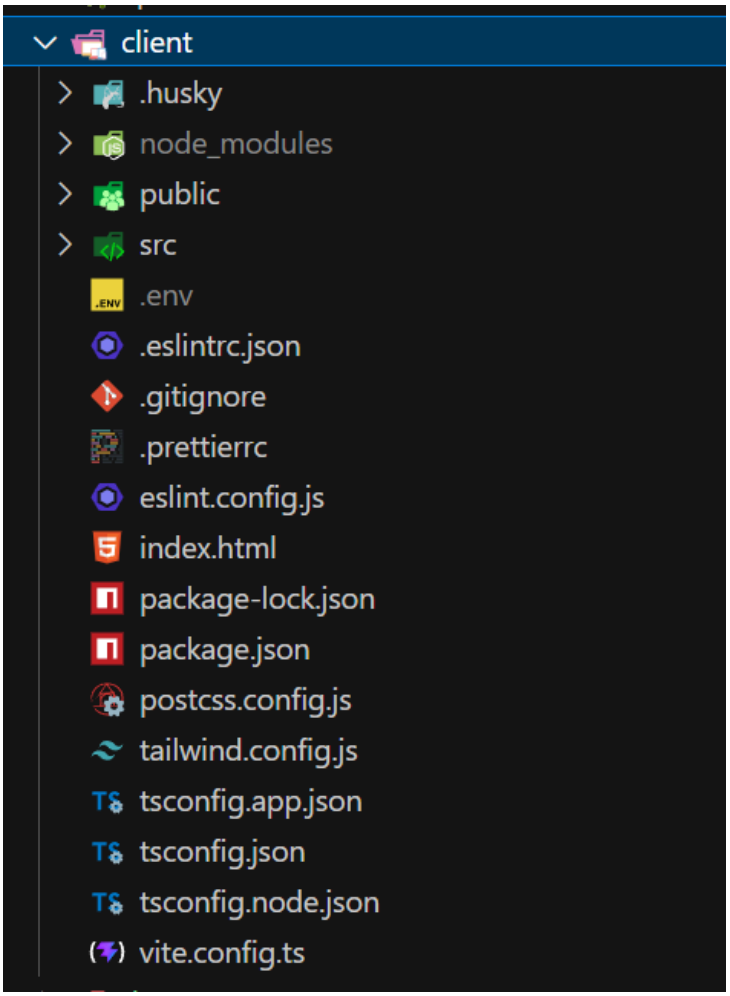

Рис. 2.6. Структура проектних папок клієнтської частини за методологією FSD

Джерело: розроблене автором

На вищих ієрархічних рівнях архітектури FSD розташовуються шари Widgets та Pages, які відповідають за композицію та безпосереднє відображення інтерфейсу. Шар Widgets об'єднує самодостатні бізнес-блоки, виступаючи зв'язуючою ланкою між базовими сутностями та функціональними діями користувача. У свою чергу, шар Pages формує кінцеві екранні форми веб-платформи, де збираються готові віджети для конкретних маршрутів додатку. Такий підхід дозволяє ізолювати логіку управління станом (state management) у межах конкретних сегментів. Завдяки цьому, асинхронні запити до API за допомогою TanStack Query та локальні стани форм інкапсулюються всередині своїх слайсів, не створюючи глобальних побічних ефектів і забезпечуючи прогнозовану поведінку інтерфейсу React 19 навіть при інтенсивному динамічному оновленні муніципальних даних.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено комплексне проектування інформаційної, алгоритмічної та програмної структури системи «Острог Разом». На основі детального аналізу предметної області було розроблено нормалізовану модель бази даних, яка забезпечує цілісність інформації та ефективний реляційний зв'язок між користувачами, зверненнями мешканців, коментарями та мультимедійними файлами. Використання сучасних підходів до проектування даних дозволило мінімізувати дублювання інформації та закласти фундамент для високої швидкодії системи при виконанні складних аналітичних запитів.

Завдяки впровадженню архітектурного шаблону Clean Architecture у поєднанні з патерном CQRS та бібліотекою MediatR, було досягнуто високого рівня модульності серверної частини. Таке розділення на ізольовані шари (Domain, Application, Infrastructure, Presentation) дозволяє розвивати бізнес-логіку незалежно від зовнішніх сервісів, баз даних або фреймворків. Окрему увагу було приділено проектуванню реалізації SignalR, яка інтегрована в загальну архітектуру через abstraction-шари, що забезпечує реактивність інтерфейсу та миттєве сповіщення користувачів про зміну статусів звернень без створення зайвої зв'язності між компонентами.

Важливим науково-практичним результатом розділу є проектування інноваційної моделі взаємодії мешканців із системою через мультимодальний штучний інтелект Google Gemini 3 Flash. Використання технік Zero-Shot Classification та Structured JSON Generation дозволило автоматизувати процес виокремлення сутностей із неструктурованих голосових та текстових повідомлень, що значно знижує поріг входу для користувачів та мінімізує рутинну роботу координаторів міських служб. Гібридна модель авторизації на базі сервісу Clerk та використання хмарного сховища Backblaze B2 для медіаконтенту гарантують високий рівень безпеки та оптимізацію витрат на інфраструктуру.

Окремим вагомим інженерним результатом стало детальне структурування

клієнтської частини веб-платформи за методологією **Feature-Sliced Design (FSD)**. Це дозволило декомпозувати складний користувацький інтерфейс на ізольовані, концептуально замінні шари та сегменти (Slices), повністю усуваючи ризики появи циклічних залежностей між модулями інтерактивної карти, форми подачі інцидентів та профілю користувача. Проектування архітектури фронтенду на базі React 19 та TypeScript із суворим дотриманням правил декларації Public API для кожної фічі забезпечує високу підтримуваність коду та гнучкість системи при подальшому масштабуванні інструментарію.

Окрім цього, у розділі формалізовано логіку поведінки системи відповідно до рольової моделі доступу (**RBAC**). Спроектовані алгоритми та конвеєри обробки даних закладають надійну основу для створення стійкої до навантажень екосистеми, здатної функціонувати в умовах інтенсивного муніципального трафіку та підтримувати стабільний безперервний зв'язок між мешканцями і муніципалітетом.

Обґрунтування вибору сучасного технологічного стеку (.NET 10, React 19) та побудовані детальні схеми взаємодії компонентів формують надійну теоретичну та методологічну основу для подальшого етапу роботи. Спроектована архітектура відповідає принципам високої доступності та масштабованості, що дозволяє перейти до безпосередньої програмної реалізації та розгортання системи в хмарному середовищі, які будуть детально описані та продемонстровані у третьому розділі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1. Засоби розробки та технологічний стек

Для успішної реалізації вебплатформи «Острог Разом» застосовано сучасний технологічний стек, що гарантує високу обчислювальну продуктивність, масштабованість та максимальну кібербезпеку програмного забезпечення.

Основним інструментом створення серверної частини виступає кросплатформна платформа .NET 10 та об'єктно-орієнтована мова C# 14. Вибір .NET 10 обґрунтований її високою швидкістю виконання коду, потужною строгою типізацією та наявністю безпечної екосистеми пакетів. Для створення програмного інтерфейсу (REST API) використовується вбудований фреймворк ASP.NET Core. Взаємодія з базою даних реалізована за допомогою об'єктно-реляційного фреймворку Entity Framework Core (підхід Code-First).

Ключовою особливістю проєкту є глибока інтеграція з сучасною мовною моделлю Google Gemini (gemini-3-flash-preview), яка виконує роль високоінтелектуального помічника. Модель використовується для мультимодальної обробки даних, забезпечуючи розпізнавання голосу, автоматичну класифікацію звернень та глибоку аналітику великих масивів даних.

Клієнтську частину розроблено як SPA-додаток на базі бібліотеки React 19 та мови TypeScript. TypeScript забезпечує строгу статичну типізацію, що знижує кількість помилок та полегшує рефакторинг. Збирачем модулів виступає Vite. Для розробки адаптивного інтерфейсу обрано Tailwind CSS 4 у поєднанні з ShadCN UI.

Для структурованих даних обрано PostgreSQL, а для неструктурованих медіафайлів - S3-сумісне сховище Backblaze B2. Вся інфраструктура розгорнута в хмарі Amazon Web Services (AWS) із використанням підходу Infrastructure as Code (IaC) через Terraform.

3.2. Реалізація серверної частини та архітектурні рішення

Внутрішня структура програмного коду проєкту базується на принципах високої модульності, низької зв'язності та розділення відповідальності відповідно до принципів Clean Architecture. Серверна частина побудована навколо патерну CQRS із використанням бібліотеки MediatR. Класи-контролери є максимально «тонкими» (Thin Controllers): вони лише приймають HTTP-запити, десеріалізують їх у об'єкти команд і передають у бізнес-шар (Application Layer). Вся бізнес-логіка зосереджена в обробниках (Handlers), що дозволяє ізолювати правила системи від технічних деталей. Для гарантування правильності даних використовується FluentValidation, яка відсікає невірні запити через Behavior Pipelines.

Для реалізації ШІ-функціоналу розроблено сервіс GeminiService.cs. Цей модуль інкапсулює логіку формування складних HTTP-запитів до Google API. Важливою технічною особливістю є використання просунутого Prompt Engineering: для кожного завдання (чат-бота, екстракції даних чи аналітики) розроблено унікальні інструкції. На рівні сервера впроваджено механізм «самовідновлення» (Self-healing) JSON-відповідей, який автоматично виправляє синтаксичні помилки моделі перед парсингом.

Для забезпечення реального “живого” з’єднання між клієнтом та сервером - на сервері функціонує вузол SignalR Hub. Коли в системі відбувається зміна (новий коментар або зміна статусу), Hub транслює подію відповідним клієнтським групам через WebSockets. Архітектурно SignalR винесено в інфраструктурний шар, що дозволяє бізнес-логіці ініціювати сповіщення через абстракції, не знаючи про технічні деталі з'єднання.

3.3. Реалізація клієнтської частини та інтерфейсу

Клієнтський вебдодаток організований за методологією Feature-Sliced Design (FSD). Код розбито на ізольовані шари (layers) та сегменти (slices), що дозволяє

масштабувати систему без зростання зв'язності коду. В директорії src/features розміщені автономні модулі для кожної сутності: Проблеми, Коментарі, Мапа, Профіль та AI-модулі.

Однією з найбільш складних частин фронтенду є реалізація «Розумної форми» (create-issue-ai-page). Вона використовує браузерні MediaRecorder API для захоплення аудіо з мікрофона. Відправка важких медіафайлів (аудіо або фото) відбувається через об'єкт FormData. Після отримання структурованого JSON від сервера, форма автоматично оновлює свій стан, заповнюючи поля заголовка, опису та категорії, одночасно встановлюючи маркер на мапі через react-leaflet. Для адміністратора реалізовано окрему аналітичну панель. Вона поєднує в собі візуальні графіки (метрики по місяцях, час вирішення проблем) та інтерактивний AI-чат. Запити до API Gemini супроводжуються агрегованою статистикою бази даних, що дозволяє моделі аналізувати тренди та пропонувати динамічні фільтри (SuggestedFilter) для керування відображенням списку проблем у реальному часі.

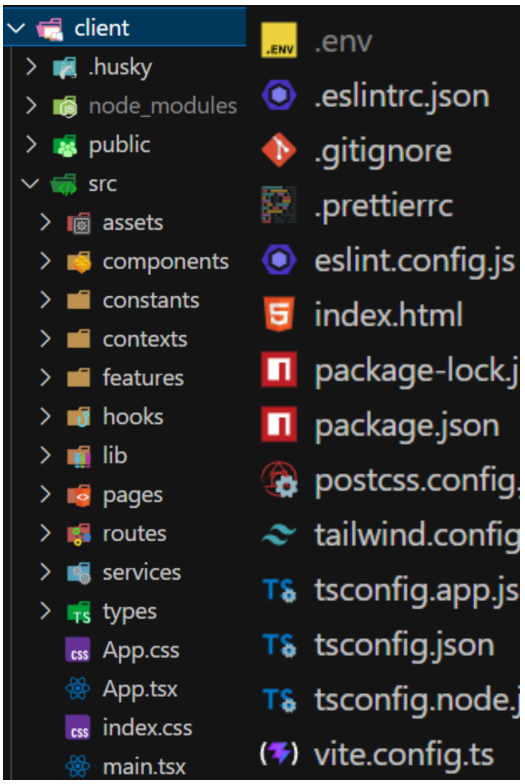

Рис. 3.1. Файлова структура проєкту за методологією Feature-Sliced Design

Джерело: розроблене автором

Окрему увагу під час розробки інтерфейсу було приділено доступності та швидкості рендерингу графічних елементів мапи та аналітичних діаграм. Шляхом інтеграції CSS-фреймворку Tailwind CSS 4.0 та компонентної бази Shadcn/ui, веб платформа отримала повністю адаптивний дизайн, що коректно відображається на будь-яких типах пристроїв - від мобільних телефонів до широкоформатних моніторів комунальних координаторів. Для оптимізації рендерингу важких списків інцидентів та маркерів на карті Leaflet застосовано методи віртуалізації та мемоізації компонентів за допомогою стандартних хуків React 19 (useMemo, useCallback). Це дозволило уникнути надлишкових циклів перерисовування DOM-дерева, забезпечило стабільну частоту кадрів інтерфейсу навіть за умови одночасного виведення кількох сотень активних муніципальних проблем на екран, та заклало надійний технологічний фундамент для подальшого масштабування сервісу «Острог Разом».

3.4. Демонстрація результатів роботи системи

Взаємодія кінцевого користувача з веб-платформою «Острог Разом» спроектована максимально інтуїтивно зрозумілою. Шлях користувача (User Journey) був оптимізований для досягнення цілі з найменшою кількістю кліків.

Для отримання доступу до інтерактивних функцій (створення проблем, коментування) користувач повинен натиснути кнопку «Увійти». Безпечний інтерфейс авторизації (Clerk) запропонує вхід за допомогою облікових записів Google чи Facebook, або реєстрацію через електронну пошту з підтвердженням кодом (OTP). Після підтвердження особи користувач перенаправляється в систему під своєю роллю.

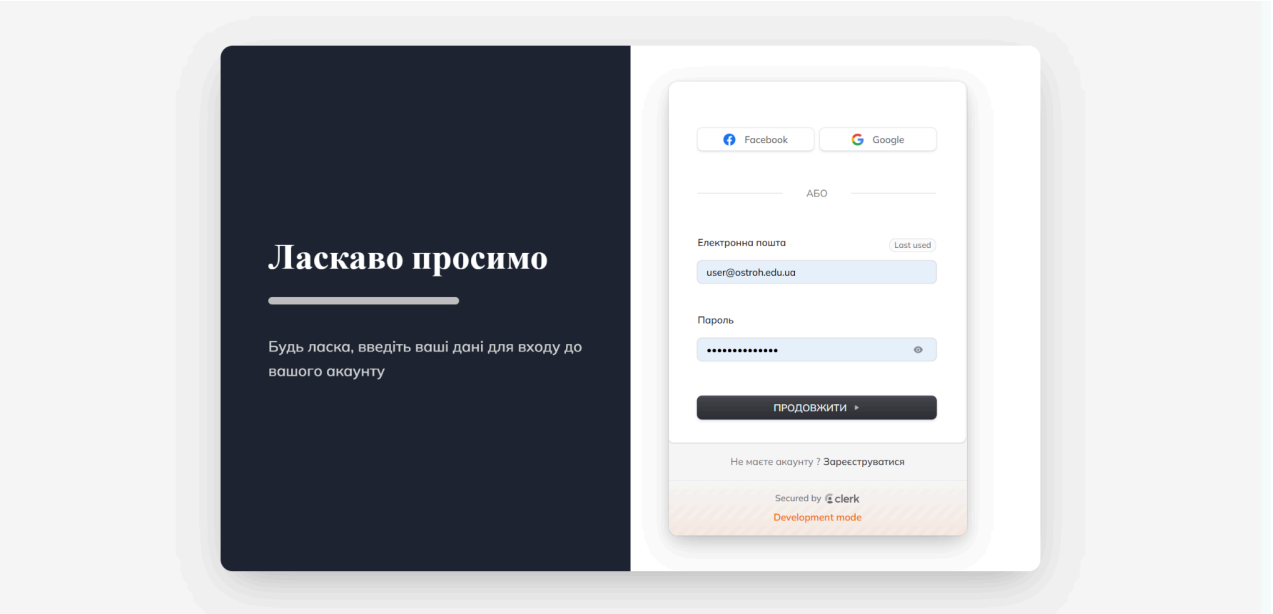

Рис. 3.2. Форма для логіну

Джерело: розроблене автором

Гостьовий режим дозволяє мешканцям вільно переглядати інформацію про стан міста без необхідності реєстрації, що забезпечує відкритість платформи. При вході на головну сторінку користувач одразу отримує доступ до актуальної мапи міста Острог. Система автоматично завантажує всі наявні проблеми та відображає їх у вигляді маркерів. Для зручності сприйняття, якщо в одній локації зосереджено велику кількість заявок, система автоматично об'єднує їх у кластери -кругові індикатори з числом, що показує кількість проблем у цій зоні. Це дозволяє уникнути візуального перевантаження інтерфейсу, а при наближенні мапи кластери динамічно розпадаються на окремі маркери.

Для швидкого знаходження потрібної інформації реалізовано механізм фільтрації за категоріями та статусом виконання, а також доступний текстовий пошук, який миттєво оновлює список релевантних заявок. Водночас система чітко розмежовує права доступу: якщо неавторизований користувач намагається виконати активну дію (наприклад, повідомити про проблему або залишити коментар), платформа автоматично перехоплює цей запит і перенаправляє його на сторінку входу, гарантуючи, що всі обговорення ведуться лише верифікованими учасниками.

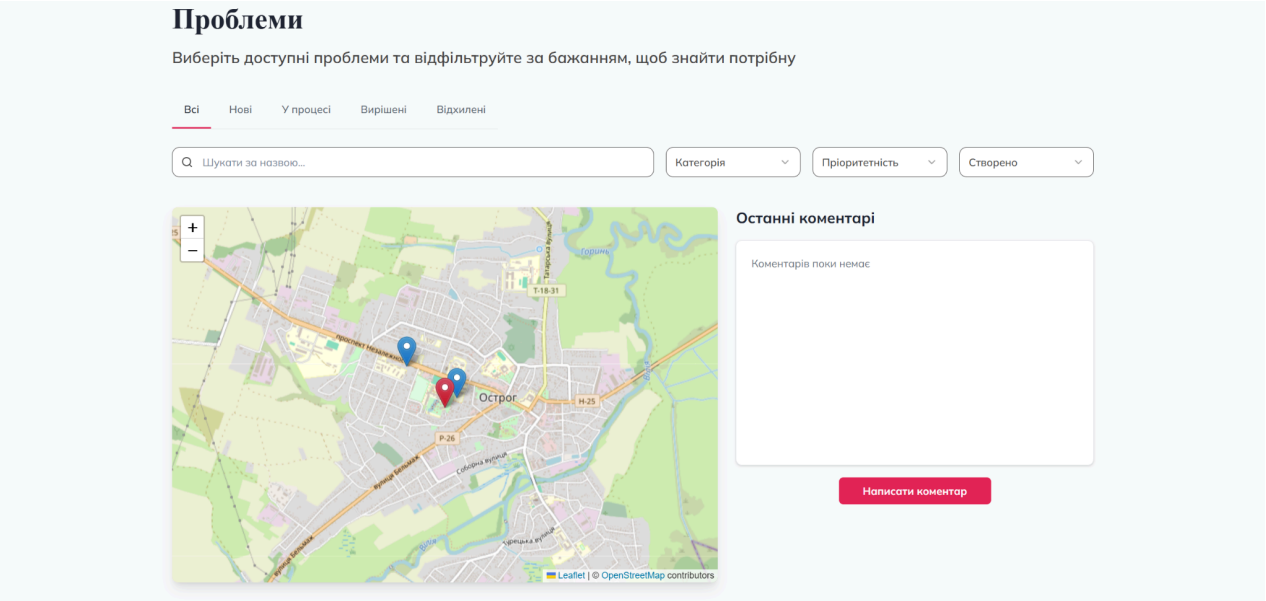

Рис. 3.3. Сторінка для незалогованого користувача

Джерело: розроблено автором

Отже, реалізований гостьовий режим виконує стратегічно важливу функцію забезпечення прозорості та відкритості муніципальних процесів. Він дозволяє будь-якому мешканцю без бар'єрів моніторити поточний стан міської інфраструктури, одночасно гарантуючи захист системи від спаму та стимулюючи громадян до реєстрації для переходу до активної участі в житті громади.

Сценарій роботи Зареєстрованого Користувача демонструє повний цикл взаємодії мешканця з платформою - від подачі заявки до обговорення. Процес створення нової проблеми розроблено максимально інтуїтивним: користувач заповнює коротку форму з детальним описом, обирає категорію та встановлює точну точку інциденту простим кліком по карті. Додатково система дозволяє завантажити фотографії-докази, після чого проблема миттєво реєструється в базі даних, а на загальній карті з'являється новий маркер. Окрім подання заявок, платформа підтримує живі обговорення в реальному часі. Коли користувач залишає коментар, повідомлення миттєво з'являється у стрічці всіх інших користувачів, які в цей момент переглядають цю ж сторінку, без необхідності перезавантажувати вікно браузера. Крім того, користувач має повний контроль над своїми персональними даними в налаштуваннях профілю. Будь-які зміни (нове ім'я або аватар) миттєво

оновлюються по всій системі та автоматично відображаються біля всіх попередніх заявок та коментарів користувача.

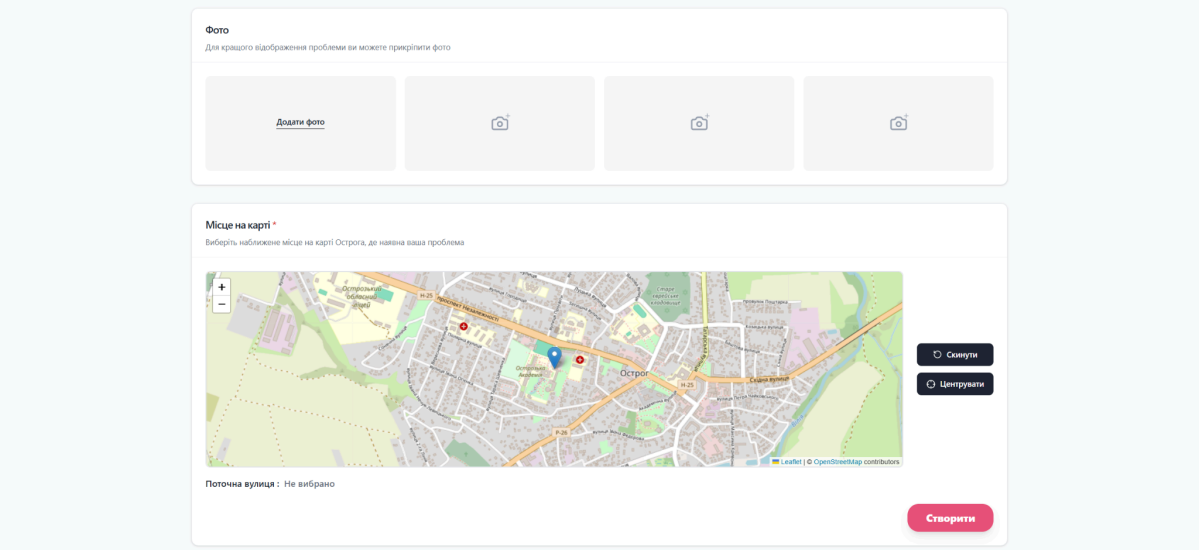

Рис. 3.4. Сторінка створення проблеми

Джерело: розроблено автором

Користувач може переглянути свої подані проблеми на окремій сторінці, відредагувати певні дані та залишити коментар.

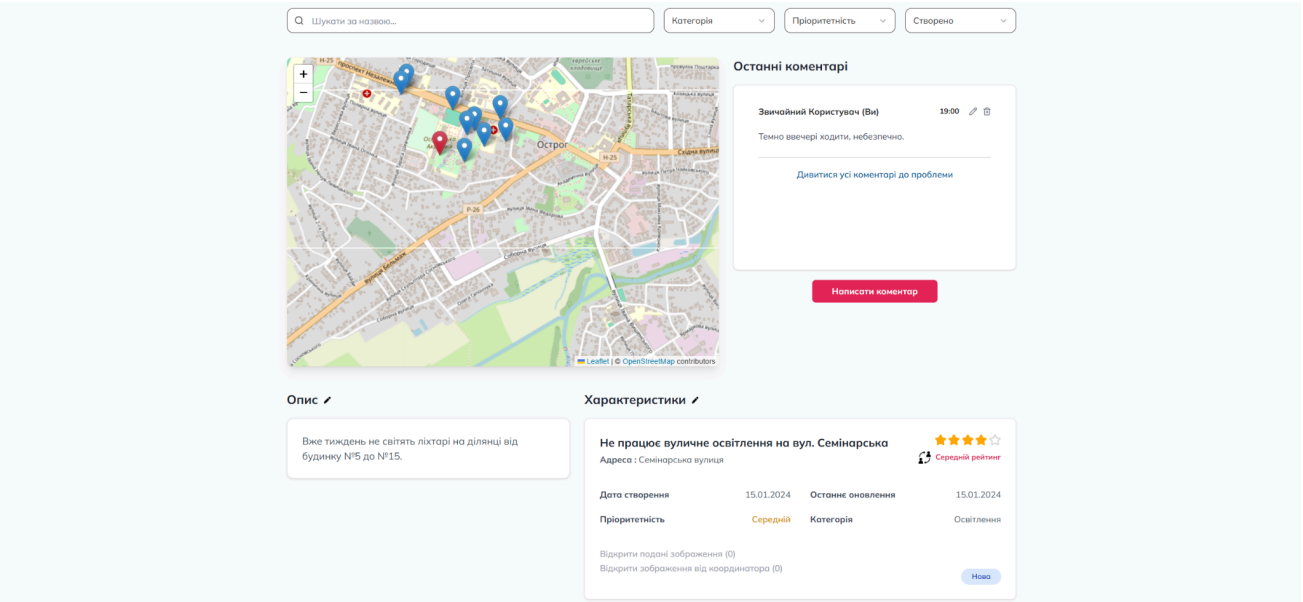

Рис. 3.5. Сторінка перегляду поданої проблеми

Джерело: розроблено автором

Таким чином, розроблений функціонал для авторизованих користувачів створює потужний інструмент електронної демократії. Завдяки максимально спрощеному процесу подачі заявок з точною геоприв'язкою та інтерактивній комунікації в реальному часі, платформа успішно трансформує мешканців міста з пасивних спостерігачів на активних співтворців комфортного міського середовища.

Робота координатора демонструє процес обробки та вирішення заявок представниками комунальних служб. Взаємодія розпочинається з модерації через спеціальну панель управління, де відображаються нові надходження. Аналізуючи зміст, фото та локацію заявки, координатор може прийняти її в роботу (перевівши у статус "У процесі") або відхилити, обов'язково вказавши причину відмови для забезпечення прозорості.

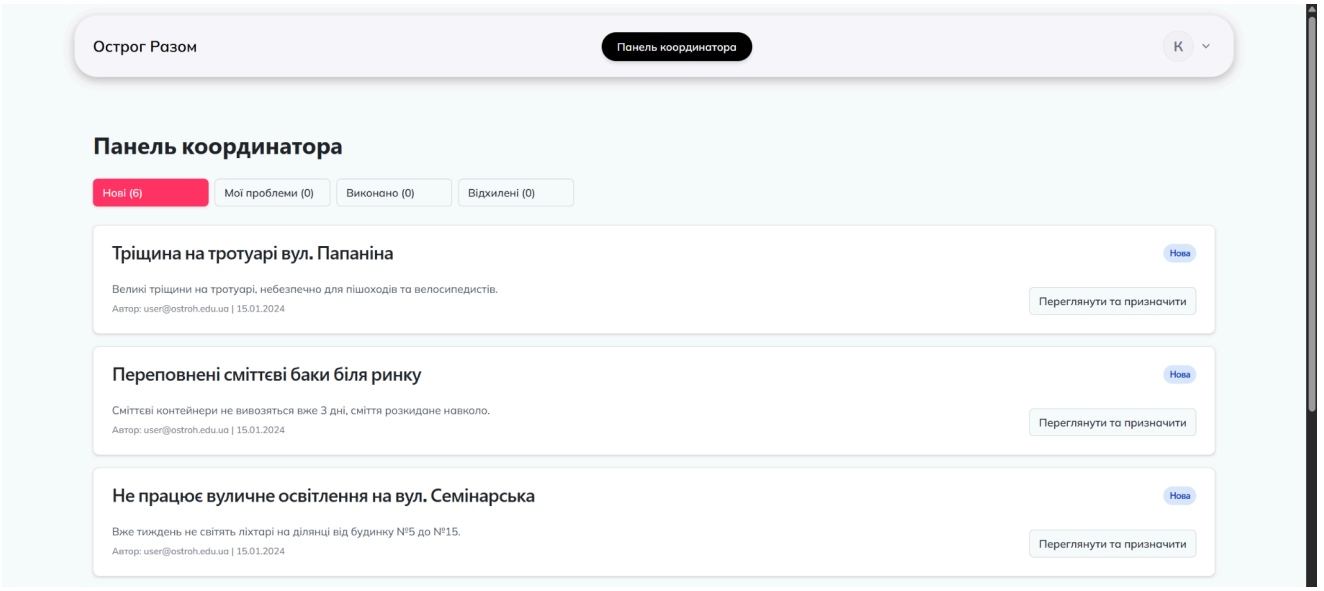

Рис 3.6. Панель з усіма проблемами доступними для координатора

Джерело: розроблено автором

Під час роботи над проблемою координатор використовує інструменти для фіксації прогресу, оновлюючи поточний стан виконання та залишаючи візуально виділені "офіційні коментарі", щоб інформувати громаду про проміжні етапи. Фінальним етапом є закриття заявки: після усунення проблеми координатор завантажує фотозвіт у форматі "було/стало" та переводить проблему в статус

"Вирішена". В результаті маркер на карті змінює колір на зелений, а користувач отримує можливість оцінити якість та швидкість роботи служби через систему рейтингу.

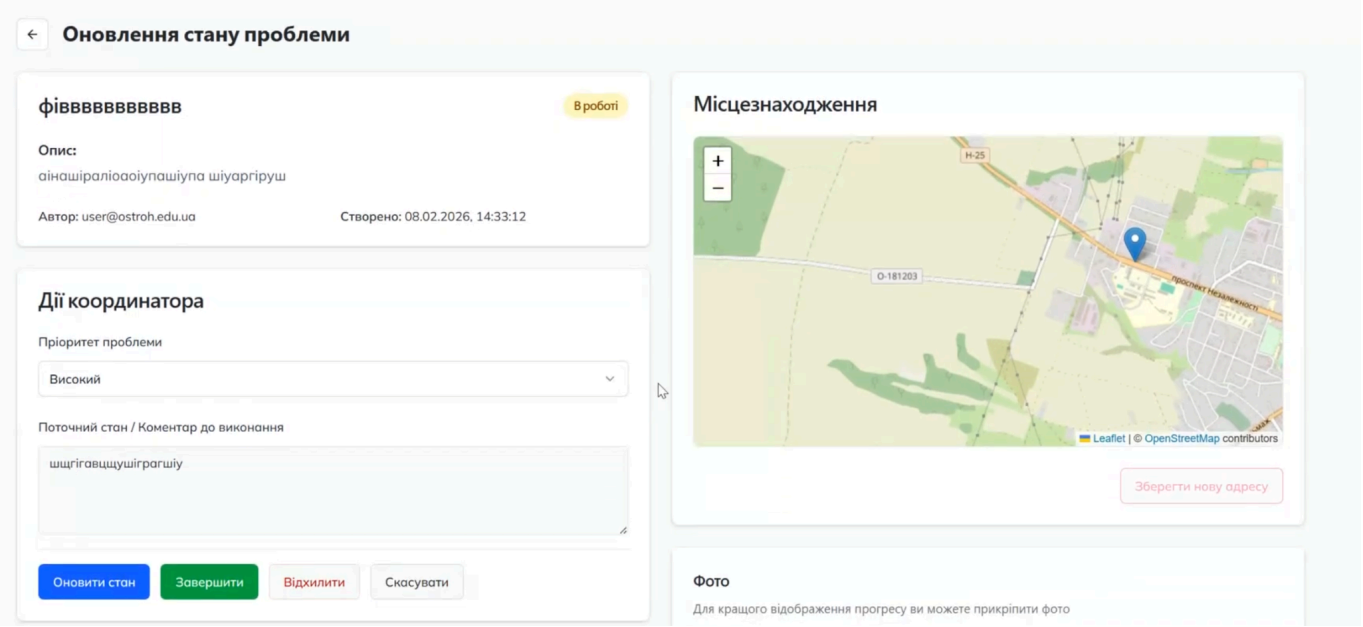

Рис. 3.7. Сторінка для оновлення даних проблеми

Джерело: розроблено автором

Підсумовуючи, робоче середовище координатора повністю цифровізує та оптимізує операційні процеси комунальних служб. Впровадження чіткого алгоритму зміни статусів, інструментів для офіційної комунікації та механізму обов'язкової фотофіксації виконаних робіт суттєво підвищує рівень відповідальності виконавців і формує довіру населення до ефективності роботи місцевих органів.

Сценарій роботи адміністратора передбачає повний контроль над системою та гнучке управління її компонентами. Адміністратор має доступ до загального реєстру користувачів і може за декілька кліків через панель управління надати будь-кому права координатора; завдяки рольовій моделі доступу (RBAC) ці зміни застосовуються миттєво. Для підвищення ефективності аналізу поточних проблем адміністратору доступний зручний перегляд існуючих рейтингів та коментарів, що дозволяє швидше визначати найбільш критичні питання на основі зворотного зв'язку

громадян.

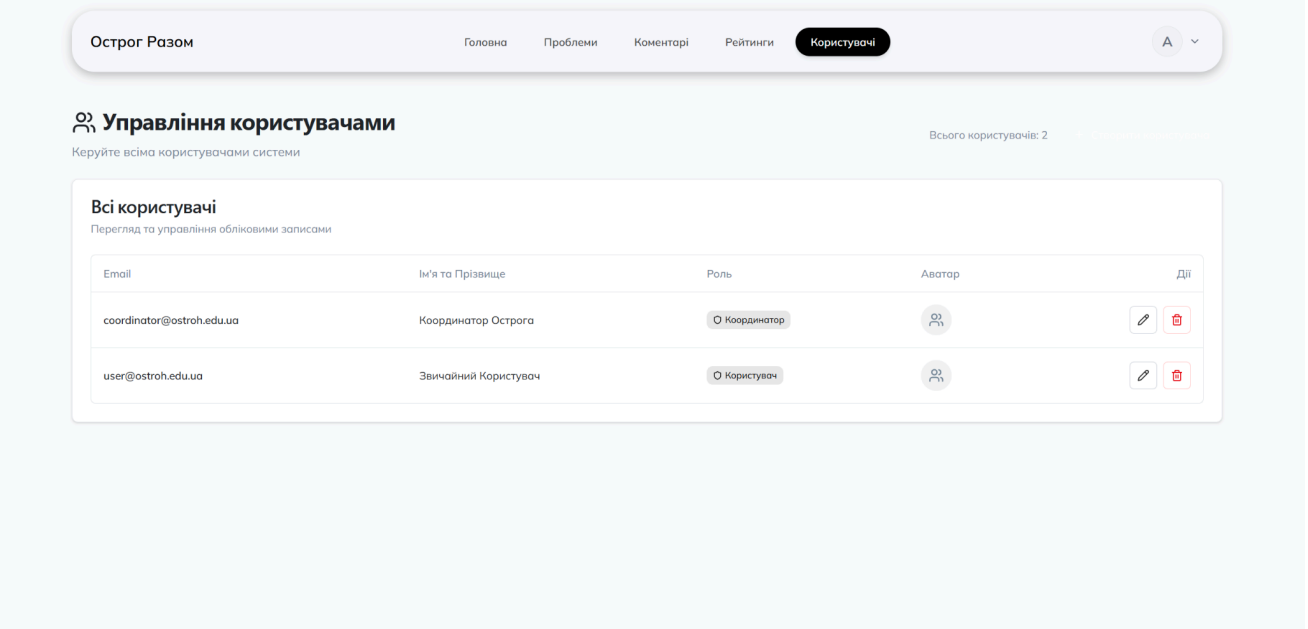

Рис. 3.8. Сторінка адміністратора для управління користувачами

Джерело: розроблено автором

Крім того, система надає інструменти для керування довідником категорій. Адміністратор може в будь-який момент додати нову категорію (наприклад, "Укриття") або відредагувати існуючі, після чого ці зміни автоматично з'являються у фільтрах та формах заявок для всіх користувачів без необхідності втручання в програмний код сайту.

Як вже було зазначено, адміністратор може переглядати та керувати існуючими проблемами. Як висновок, адміністративний модуль гарантує високу життєздатність, безпеку та адаптивність усієї платформи. Завдяки централізованому управлінню рольовою моделлю доступу (RBAC) та можливості динамічного налаштування категорій, адміністратор здатний оперативно масштабувати систему та реагувати на нові інфраструктурні виклики міста без додаткового залучення розробників.

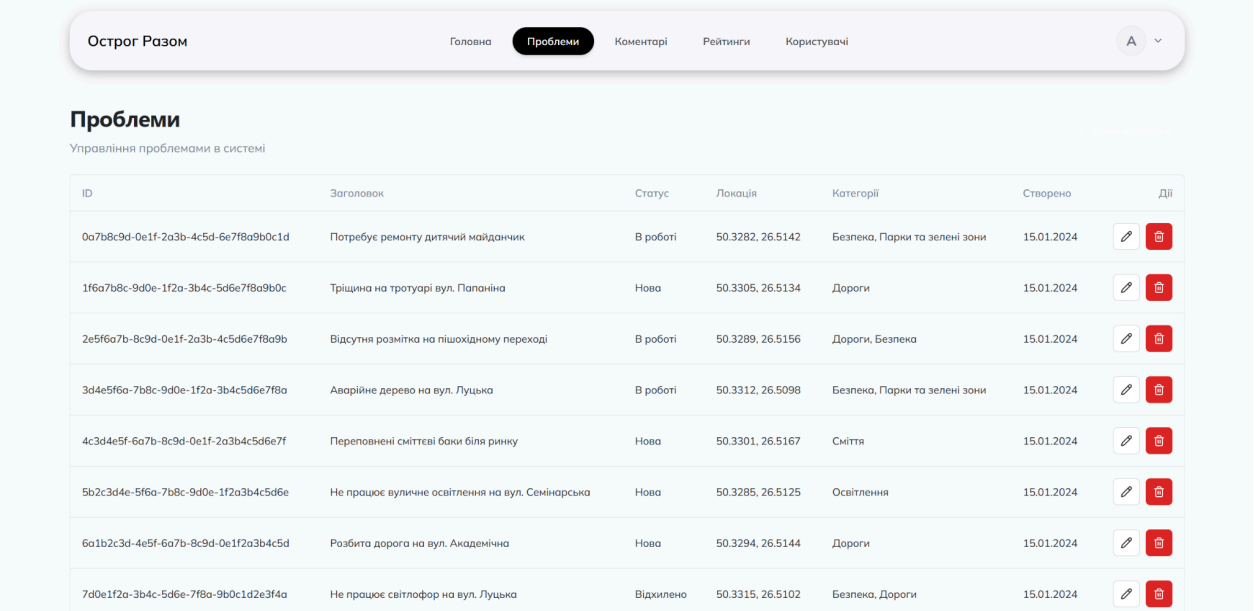

Рис. 3.9. Сторінка адміністратора для управління проблемами

Джерело: розроблено автором

Окремим інноваційним елементом інтерфейсу є Універсальний AI-помічник, доступний у вигляді віджета на всіх сторінках. Він дозволяє користувачу спілкуватися з платформою природною мовою, знаходячи проблеми за семантичним наміром. Наприклад, на запит «покажи проблеми з дорогами біля академії» бот не просто знаходить текст, а фільтрує базу даних за локацією та категорією.

Інженерна реалізація цього модуля базується на інтеграції сервісів бекенду з моделлю Google Gemini 3 Flash за допомогою офіційного SDK. Під час надходження текстового запиту від користувача, система виконує конвеєрну обробку повідомлення за допомогою техніки системного промптингу та режиму генерації структурованого JSON. Модель штучного інтелекту динамічно парсить природну мову і перетворює її на строго типізований об'єкт фільтрації, що містить ідентифікатори категорій інцидентів, географічні радіуси пошуку та часові межі. Отримані аналітичні параметри передаються через API на клієнтську частину React, де глобальний менеджер стану ініціює миттєве перемальовування маркерів на мапі React Leaflet та звужує поточний список відображуваних звернень. Такий підхід

дозволив повністю абстрагувати складні SQL-запити до PostgreSQL за допомогою Entity Framework Core, надавши користувачам інтуїтивний, людиноцентричний інтерфейс для навігації міським середовищем Острога.

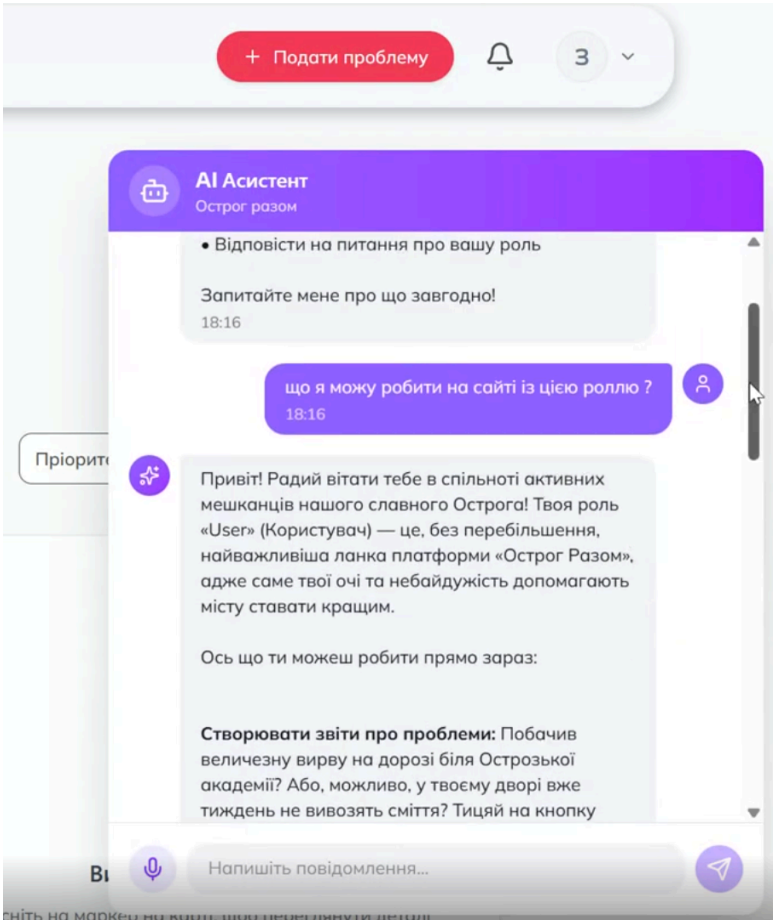

Рис. 3.10. Вікно універсального AI-помічника

Джерело: розроблене автором

Для мешканців міста передбачена сторінка «Розумної форми», де процес створення заявки автоматизовано. ШІ розпізнає локальні назви (наприклад, «біля старого корпусу») та автоматично заповнює географічні координати. Цей функціонал реалізовано завдяки поєднанню можливостей семантичного аналізу тексту моделлю Google Gemini та інтеграції з базою локальних топонімів Острога. Оскільки звичайні картографічні сервіси часто не містять специфічних народних чи історичних назв локацій, які використовують містяни та студенти (як-от «Дамба», «Новий корпус», «Старомонастирська»), розроблений інтелектуальний конвеєр зіставляє ці згадки з конкретними просторовими орієнтирами.

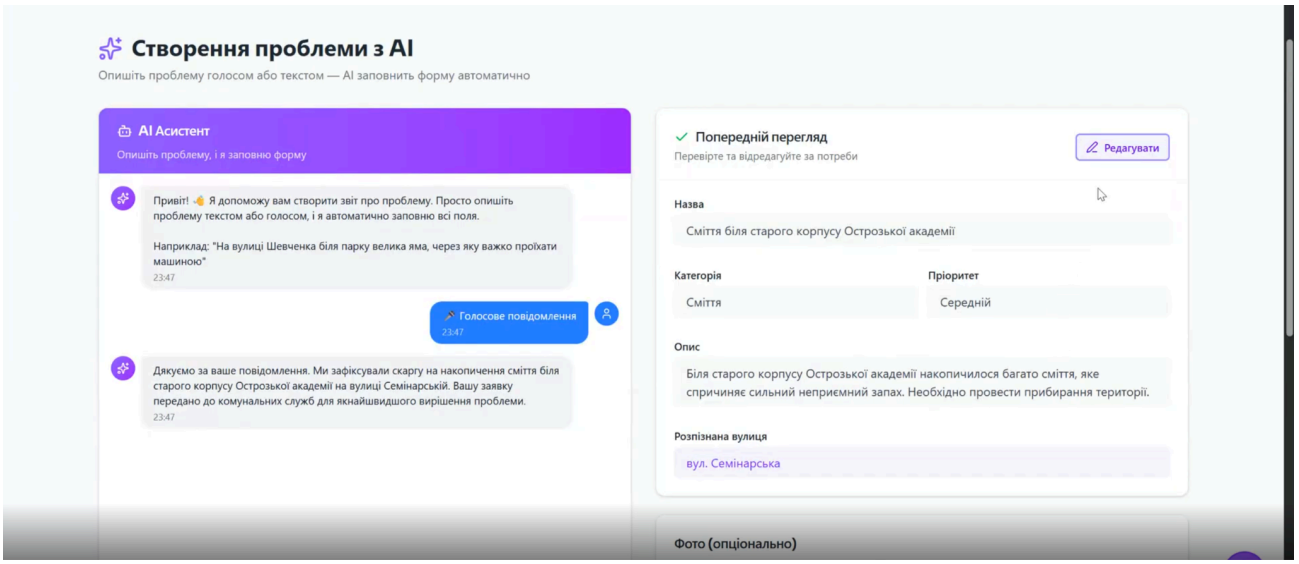

Рис. 3.11. Сторінка створення проблеми через ШІ-асистента

Джерело: розроблене автором

Адміністративний модуль доповнено Аналітичним Дашбордом. На ньому відображаються динамічні діаграми актуальних метрик платформи. Окремий AI-чат для адміністратора дозволяє проводити глибокий аналіз даних, наприклад, виявляти найбільш проблемні райони міста за певний період.

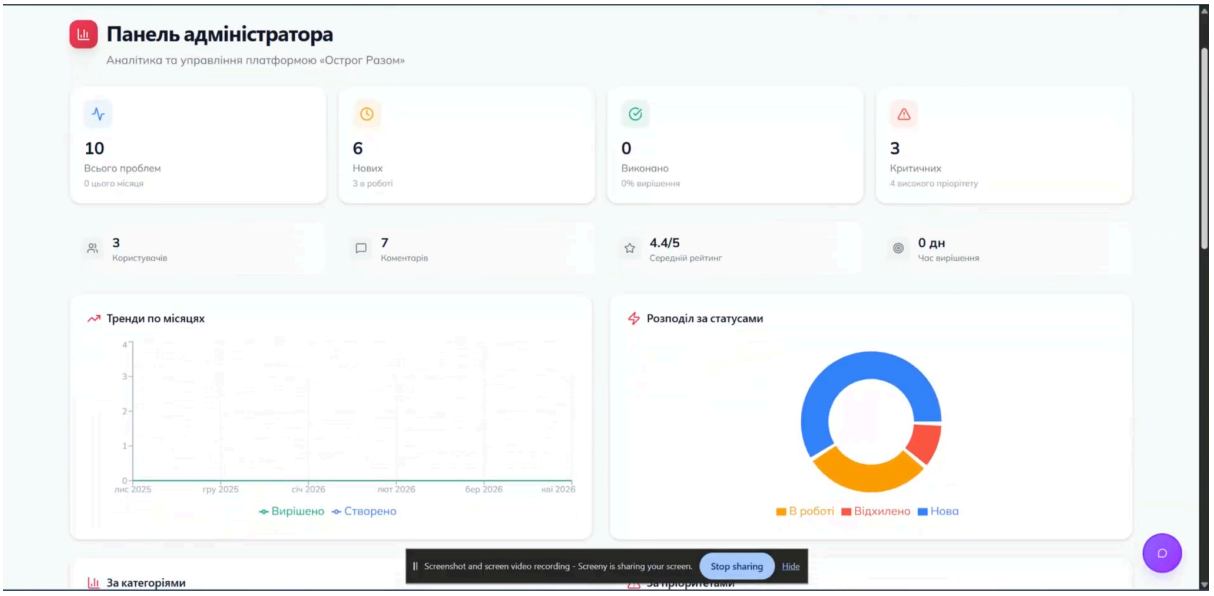

Рис. 3.12. Панель глобальної статистики адміністратора частина 1

Джерело: розроблене автором

Практичне візуальне втілення цього інтерфейсу демонструє поточний стан муніципальних звернень у реальному часі за допомогою карток ключових KPI-метрик та інтегрованих графіків аналітики. Основну частину екранної форми займають лінійна діаграма «Тренди по місяцях», яка відображає часову динаміку створення й вирішення інфраструктурних проблем певного періоду, а також кругова діаграма «Розподіл за статусами», що наочно ілюструє відсоткове співвідношення заявок у станах «В роботі», «Відхилено» та «Нова».

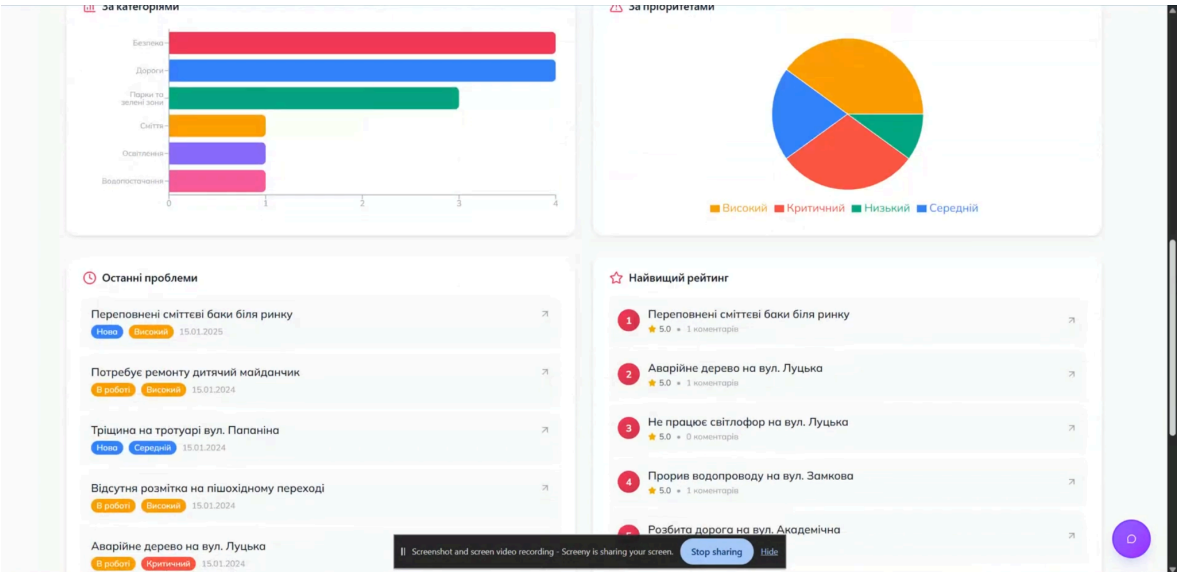

Рис. 3.13. Панель глобальної статистики адміністратора частина 2

Джерело: розроблене автором

Представлена панель глобальної статистики виступає центральним вузлом моніторингу для адміністрації міста. Завдяки наочній візуалізації ключових показників ефективності (KPI), керівництво може в реальному часі оцінювати загальне навантаження на комунальні служби, виявляти найбільш проблемні сфери життєдіяльності міста та приймати обґрунтовані управлінські рішення на основі об’єктивних цифрових даних. Такий підхід забезпечує високу прозорість муніципальних процесів та дозволяє оперативно реагувати на критичні зміни в інфраструктурному стані громади

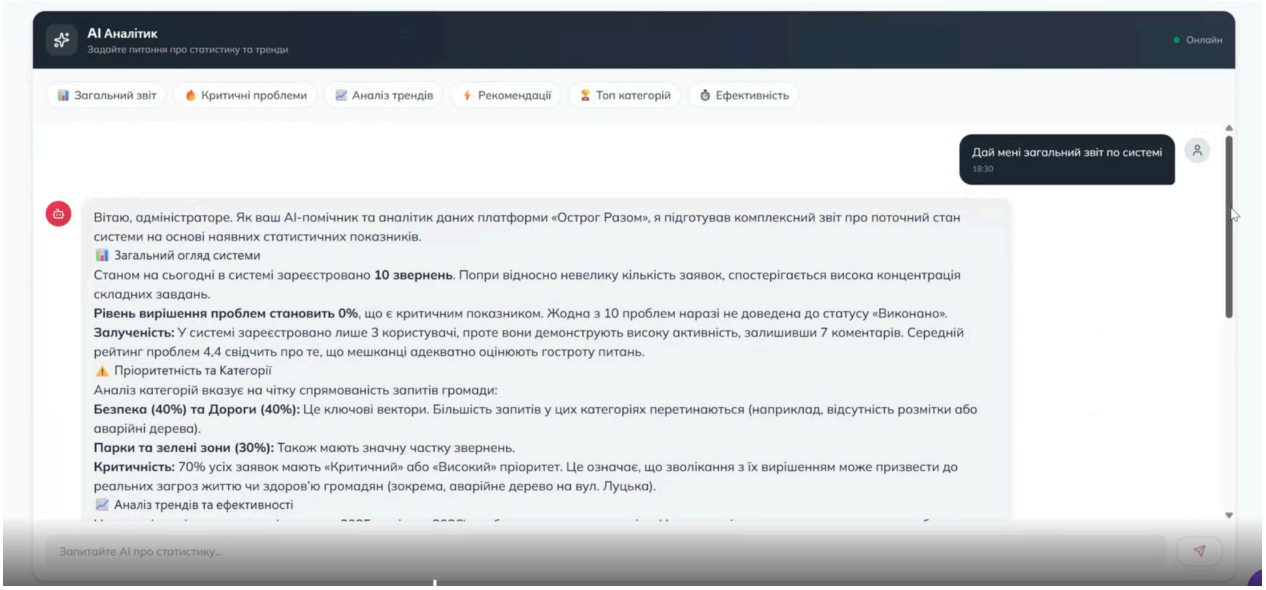

Рис. 3.14. Секція використання ШІ для адміністратора з прикладом запиту

Джерело: розроблене автором

Застосування інтелектуального аналітичного чату на базі моделі Gemini 3 Flash радикально змінює спосіб роботи з великими масивами муніципальних даних. Замість ручного налаштування складних багаторівневих фільтрів, адміністратор отримує можливість взаємодіяти з базою даних через природну мову. Функція автоматичного формування та застосування рекомендованих параметрів дозволяє миттєво локалізувати специфічні інциденти за семантичним описом, що значно прискорює процес аудиту звернень та стратегічного планування ремонтних робіт у місті.

3.5. Автоматизація розгортання та хмарна інфраструктура

Важливим елементом культури розробки проєкту є впровадження процесів безперервної інтеграції та доставки (CI/CD) на базі GitHub Actions. Конфігурація пайплайну розбита на два паралельні потоки (jobs), що дозволяє одночасно оновлювати клієнтську та серверну частини системи, значно скорочуючи час релізу.

Для забезпечення високої якості програмного продукту, попри відсутність розгалуженої системи модульних тестів (Unit Tests) на поточному етапі розробки, було впроваджено альтернативні механізми контролю. На етапі Frontend (Check & Deploy) обов'язковим кроком є виконання скрипту npm run check. Ця процедура включає статичний аналіз коду за допомогою лінтерів та перевірку форматування, що дозволяє виявляти потенційні помилки типізації TypeScript та синтаксичні огріхи ще до початку збірки. Таким чином, строга типізація виступає превентивним засобом забезпечення надійності інтерфейсу.

На серверній стороні у потоці Backend (Check & Deploy) роль «контрольних воріт» виконує етап dotnet build. У середовищі, ізольованому від локальних налаштувань розробника, система виконує повну компіляцію проєкту в режимі Release. Будь-яка архітектурна помилка або невідповідність контрактів даних призведе до негайної зупинки пайплайну, що гарантує потрапляння в хмару лише синтаксично цілісного та працездатного коду. Такий підхід фокусується на «здоров’ї» інфраструктури та збірки, що є критично важливим для швидкого розгортання MVP-версії платформи.

Після проходження етапів перевірки, пайплайн автоматизує хмарні операції. Для фронтенду здійснюється синхронізація з AWS S3 та інвалідація кешу CloudFront. Для бекенду виконується автоматична збірка Docker-образу, його тегування та публікація в Amazon ECR, після чого подається команда на оновлення сервісу в кластері ECS Fargate. Весь процес - від пушу коду в репозиторій до оновлення веб-сайту - відбувається повністю автоматично без втручання людини.

Усі конфіденційні дані, необхідні для безперешкодної взаємодії автоматизованого пайплайну з хмарними сервісами (такі як ключі доступу AWS, ідентифікатори CloudFront та секретні токени Clerk), надійно ізольовані та зашифровані в межах репозиторію за допомогою інструменту GitHub Secrets. Використання змінних оточення дозволило повністю виключити присутність секретних ключів у відкритому коді, що відповідає сучасним стандартам безпеки

DevSecOps. Створений CI/CD конвеєр не лише забезпечує високу швидкість ітерацій та доставки нових функцій, але й мінімізує вплив людського фактора

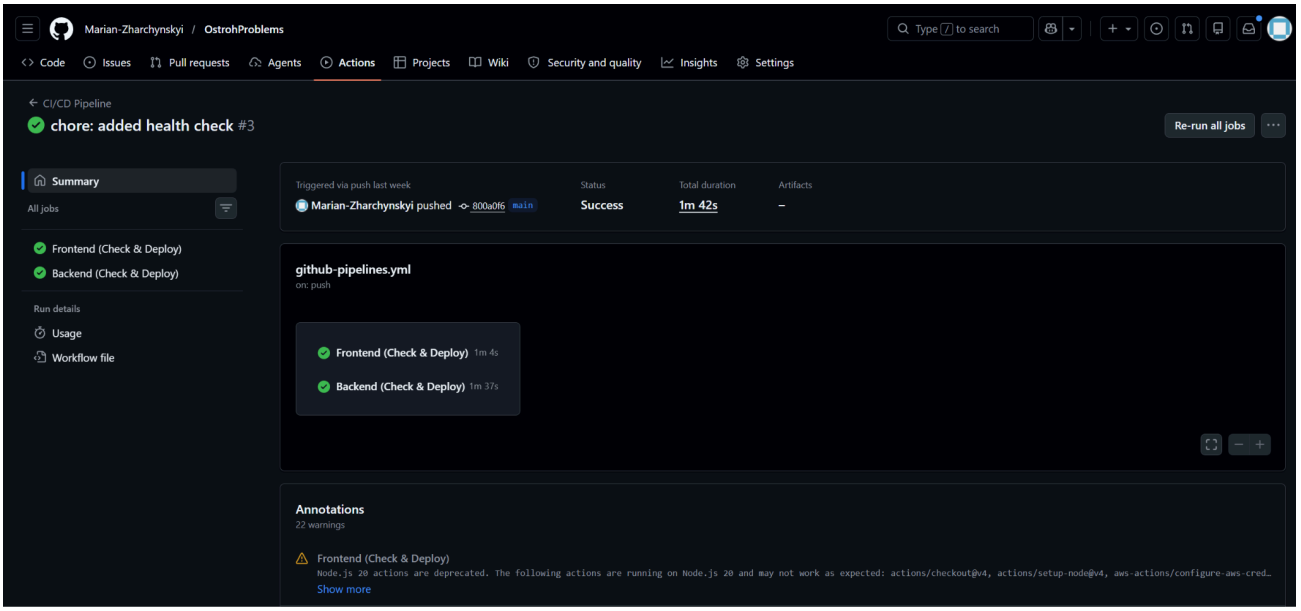

Рис. 3.15. Приклад успішного виконання пайплайну

Джерело: розроблене автором

Наведений приклад демонструє надійність автоматизованої системи розгортання. Використання збірки та лінтингу як альтернативи повноцінному тестуванню дозволяє підтримувати високу якість коду та стабільність платформи «Острог Разом» на етапі її активного функціонального розширення.

Використання декларативних файлів конфігурації дозволило розбити опис ресурсів на логічні модулі, що забезпечує високу швидкість розгортання та повну відтворюваність хмарного середовища. Такий підхід повністю нівелює ризики «людського фактору» при налаштуванні складних мережевих правил та сервісів, а вибір регіону у Франкфурті гарантує мінімальну мережеву затримку (latency) для мешканців міста Острог та стабільну доступність усіх необхідних компонентів AWS-інфраструктури. Завдяки фіксації архітектурних параметрів у коді, будь-яка зміна в топології мережі VPC або конфігурації контейнерів Fargate проходить попередній аудит і автоматично синхронізується з хмарою.

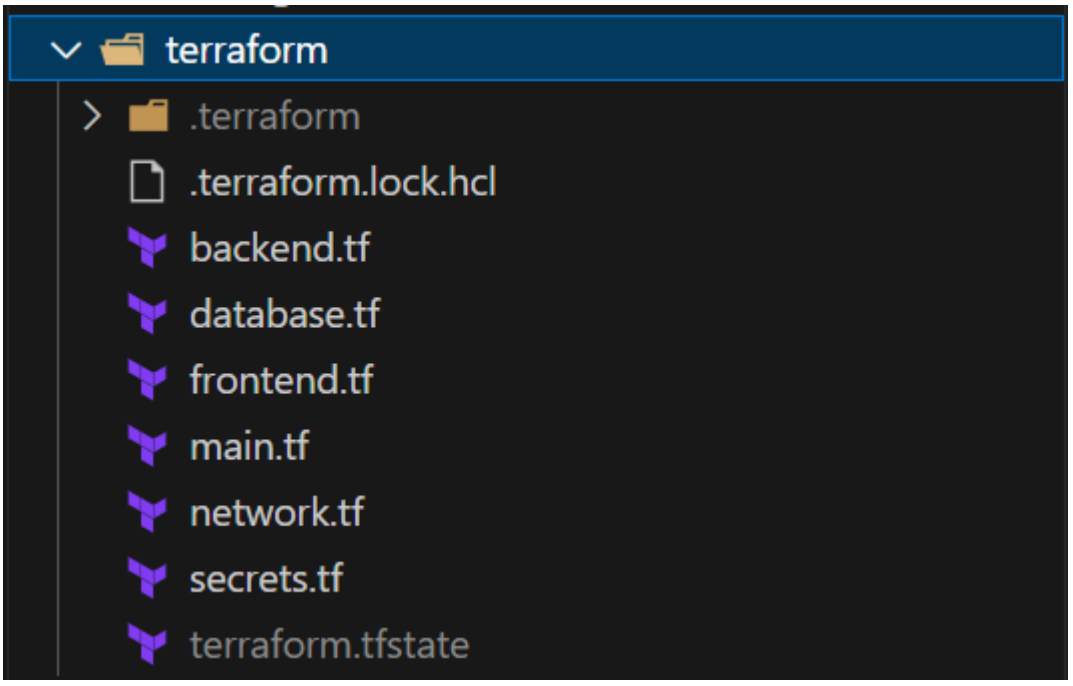

Рис. 3.16. Структура Terraform-конфігурації проєкту

Джерело: розроблене автором

Створено ізольовану VPC (10.0.0.0/16) із розділенням на публічні та приватні підмережі. Задля економії коштів обчислювальні ресурси ECS Fargate розміщено в публічній підмережі для прямого доступу до інтернету без NAT Gateway, тоді як база даних RDS PostgreSQL розміщена в ізольованому приватному сегменті.

Використовується Amazon RDS (PostgreSQL 16) на інстансі db.t4g.micro (ARM-архітектура). Доступ до бази жорстко обмежений групами безпеки лише для внутрішніх контейнерів API.

Docker-образи зберігаються в Amazon ECR. Виконання коду відбувається в безсерверному режимі AWS Fargate. Трафік розподіляється через Application Load Balancer (ALB). Секрети (API ключі Gemini, B2, Clerk) надійно зберігаються в AWS Secrets Manager.

Статичні файли React-додатка розміщено в Amazon S3. Доступ до них забезпечується через мережу Amazon CloudFront (CDN) із використанням механізму Origin Access Control (OAC) для безпеки. Це гарантує швидку доставку контенту та коректну роботу SPA-роутингу.

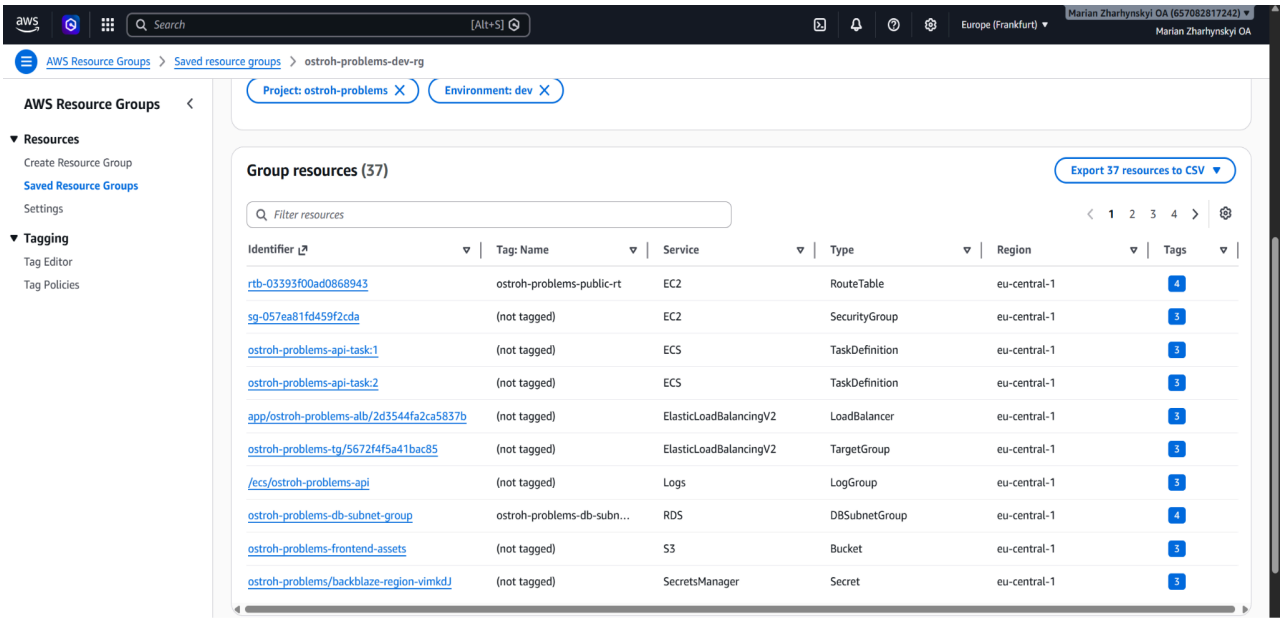

Рис. 3.17. Ресурс група хмарної інфраструктури проєкту в консолі AWS

Джерело: розроблене автором

Останнім етапом інфраструктурних налаштувань стала конфігурація власного доменного імені проєкту ostroh-problems.pp.ua. У Terraform-конфігурації чітко визначено політики CORS, що дозволяють запити до API лише з довірених адрес (основного домену та технічних адрес розробки). Для забезпечення конфіденційності весь HTTP-трафік примусово перенаправляється на зашифрований протокол HTTPS. Криптографічні SSL-сертифікати автоматично випускаються та оновлюються через AWS Certificate Manager (ACM), а DNS-записи надійно маршрутизуються через керований сервіс Amazon Route53.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи проведено вичерпний аналіз та практичну апробацію програмного й апаратного забезпечення платформи «Острог Разом». Обґрунтовано вибір сучасного технологічного стеку, що базується на платформі .NET 10 та бібліотеці React 19, що разом із використанням мови TypeScript та інструменту збірки Vite дозволило досягти максимальної швидкодії

клієнтської та серверної частин. Використання реляційної СУБД PostgreSQL забезпечило надійне збереження структурованих даних та високу швидкість виконання складних SQL-запитів, що є критичним для систем із великою кількістю зв'язків між мешканцями, заявками та службами міста.

Детально описано архітектуру реалізації, яка базується на суворому дотриманні принципів Clean Architecture та патерну CQRS. Впровадження медіатора MediatR та конвеєрів валідації FluentValidation дозволило створити стійку до помилок систему з низькою зв'язністю компонентів. Особливу увагу приділено реалізації двостороннього зв'язку в реальному часі за допомогою бібліотеки SignalR, що забезпечило миттєву доставку сповіщень та оновлення статусів проблем без перезавантаження сторінок. Клієнтська частина, організована за методологією Feature-Sliced Design (FSD), продемонструвала високу модульність, дозволяючи ізольовано розвивати фічі карти, профілю та інтелектуальних асистентів.

Важливим практичним досягненням стало впровадження інтелектуальних сервісів на базі мультимодальної моделі Google Gemini 3 Flash. Розроблена система нативно обробляє голосові повідомлення, автоматично заповнюючи складні електронні форми та виконуючи точну геолокацію об’єктів у межах міста Острог за непрямими описами. Реалізація Admin AI Dashboard із функцією семантичного аналізу та генерації динамічних фільтрів дозволила автоматизувати аналітичну роботу адміністратора, перетворюючи неструктуровані запити природною мовою у валідні вибірки з бази даних, що значно прискорює прийняття управлінських рішень.

Питання безпеки та оптимізації ресурсів вирішено через інтеграцію спеціалізованих хмарних сервісів. Використання Clerk забезпечило надійну автентифікацію користувачів із підтримкою багатофакторних методів, а підключення S3-сумісного сховища Backblaze B2 дозволило ефективно розділити потоки обробки текстової інформації та важкого медіаконтенту. Це гарантує високу доступність системи та захист персональних даних мешканців, одночасно знижуючи навантаження на основний обчислювальний сервер.

Вся інфраструктура проекту була повністю автоматизована за допомогою парадигми Infrastructure as Code (IaC) та інструменту Terraform, що дозволило розгорнути відмовостійке середовище в регіоні AWS eu-central-1. Використання контейнеризації Docker та безсерверних обчислень AWS Fargate у поєднанні з налаштованими пайплайнами CI/CD в GitHub Actions забезпечило безперервну доставку оновлень та стабільну роботу системи під навантаженням. Налаштування мережі доставки контенту CloudFront та підключення власного домену із SSL-сертифікатами підтверджують готовність платформи до реальної промислової експлуатації.

Розроблене керівництво користувача та фінальна демонстрація результатів підтверджують, що створена платформа є інтуїтивно зрозумілою для мешканців та ефективною для представників муніципальної влади. Проект успішно вирішує завдання цифровізації міського управління, пропонуючи прозорий та сучасний інструмент взаємодії громади та комунальних служб міста Острог, що відповідає найвищим стандартам розробки програмного забезпечення.

ВИСНОВКИ

У результаті виконання завдань кваліфікаційної роботи було успішно спроектовано та професійно розроблено інформаційну систему «Острог Разом». Цей програмний продукт призначений для цілодобового моніторингу, інтелектуальної реєстрації та прозорого адміністративного управління процесом вирішення інфраструктурних проблем міста Острога. Основна мета роботи була повністю досягнута, а всі поставлені теоретичні та інженерні завдання виконані у повному обсязі.

Під час дослідження було проведено ґрунтовний аналіз предметної області управління міською інфраструктурою та процесів цифровізації муніципальних послуг. На основі систематизації недоліків існуючих рішень, таких як система 1551 та зарубіжні аналоги, обґрунтовано необхідність впровадження мультимодальних методів взаємодії для підвищення інклюзивності сервісу. Це дозволило сформувати вичерпний перелік функціональних вимог, що стали фундаментом для подальшого проектування.

Наступним важливим етапом стала розробка високонадійної серверної архітектури на базі .NET 10, що базується на принципах **Clean Architecture** та патерні **CQRS**. Завдяки впровадженню бібліотеки **MediatR**, вдалося досягти низької зв'язності компонентів, а інтеграція протоколу **SignalR** забезпечила високу реактивність системи. Це дозволило реалізувати миттєве оновлення статусів проблем та відображення коментарів у реальному часі без необхідності перезавантаження сторінок користувачами.

Клієнтська частина системи представлена сучасним SPA-додатком на базі React 19 та TypeScript, організованим за методологією **Feature-Sliced Design (FSD)**. Такий підхід забезпечив створення масштабованого інтерфейсу, де картографічні сервіси Leaflet, безсерверна модель авторизації Clerk та системи управління станом TanStack Query функціонують як незалежні модулі. Адаптивний дизайн (Mobile-First) гарантує безперебійну роботу платформи на будь-яких типах пристроїв.

Особливу наукову новизну проекту становить впровадження інноваційного AI-модуля на базі моделі **Google Gemini 3 Flash**. У роботі практично реалізовано мультимодальну обробку голосових повідомлень, що дозволяє автоматично виокремлювати іменовані сутності, такі як адреси та категорії проблем, та генерувати на їх основі структуровані JSON-дані. Додатково створено інтелектуальний аналітичний інструментарій для адміністратора, який забезпечує глибокий семантичний аналіз бази даних за допомогою запитів природною мовою.

Питання розгортання та життєвого циклу програмного забезпечення вирішено через повну автоматизацію хмарної інфраструктури в середовищі **Amazon Web Services (AWS)**. За допомогою **Terraform** було реалізовано підхід «Інфраструктура як код», що дозволило розгорнути відмовостійку систему з використанням безсерверних обчислень AWS Fargate та мережі доставки контенту CloudFront. Налаштовані CI/CD пайплайни через GitHub Actions забезпечують безпечну та безперервну доставку оновлень до кінцевого користувача.

Створений програмний веб-сервіс «Острог Разом» має високу практичну та суспільну значущість, оскільки його впровадження дозволить фундаментально автоматизувати застарілі процеси обробки звернень громадян. Використання такої платформи радикально підвищить прозорість роботи комунальних служб та створить зручне цифрове середовище для комунікації між громадою та владою міста. Результати цієї кваліфікаційної роботи відповідають сучасним індустріальним стандартам і можуть бути використані як готове технологічне рішення для муніципального сектору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amazon Web Services. AWS Certificate Manager: автоматизація випуску та оновлення SSL/TLS сертифікатів. URL: https://docs.aws.amazon.com/acm/latest/userguide/acm-overview.html (дата звернення: 12.03.2026).
2. Amazon Web Services. AWS Fargate: безсерверні обчислення для контейнерів ECS. Посібник користувача. URL: https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html (дата звернення: 18.02.2026).
3. Amazon Web Services. AWS SDK for .NET Documentation [Електронний ресурс]. URL: https://docs.aws.amazon.com/sdk-for-net/ (дата звернення: 22.03.2026).
4. Amazon Web Services. AWS Secrets Manager: безпечне управління конфіденційними даними та ключами доступу. URL: https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html (дата звернення: 05.04.2026).
5. Amazon Web Services. Amazon Elastic Container Registry (ECR): посібник зі збереження Docker-образів. URL: https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html (дата звернення: 10.04.2026).
6. Axios. Документація проміс-орієнтованого HTTP-клієнта для браузера та node.js. URL: https://axios-http.com/docs/intro (дата звернення: 25.02.2026).
7. Backblaze B2. Керівництво розробника з інтеграції S3-сумісного хмарного сховища. URL: https://www.backblaze.com/cloud-storage/docs/s3-compatible-api (дата звернення: 07.03.2026).
8. Clerk. Документація сервісу автентифікації та управління користувачами в React-додатках. URL: https://clerk.com/docs/quickstarts/react (дата звернення: 14.02.2026).
9. Cloudfront Origin Access Control (OAC). Технічна документація з безпечного доступу до S3 бакетів. URL:

https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-restricting-access-to-s3.html (дата звернення: 11.04.2026).

10. Docker. Контейнеризація додатків та управління образами: офіційна документація. URL: https://docs.docker.com/get-started/ (дата звернення: 20.03.2026).
11. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p. (дата звернення: 12.02.2026).
12. Feature-Sliced Design. Архітектурна методологія для фронтенд-проєктів: документація та концепції. URL: https://feature-sliced.design/docs/get-started/tutorial (дата звернення: 01.03.2026).
13. FluentValidation. Посібник зі створення строго типізованих правил валідації для .NET об'єктів. URL: https://docs.fluentvalidation.net/en/latest/ (дата звернення: 09.03.2026).
14. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p. [Електронний ресурс]. URL: https://martinfowler.com/eaaCatalog/ (дата звернення: 06.02.2026).
15. GitHub Actions. Автоматизація робочих процесів, CI/CD та інтеграція з хмарними сервісами. URL: https://docs.github.com/en/actions (дата звернення: 08.04.2026).
16. Google AI Studio. Документація моделі Gemini 3 Flash та інтеграція мультимодальних запитів через API. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 28.02.2026).
17. HashiCorp. Terraform: інфраструктура як код. Офіційний посібник з налаштування провайдерів та ресурсів. URL: https://developer.hashicorp.com/terraform/intro (дата звернення: 15.03.2026).
18. Leaflet. Офіційна документація JavaScript-бібліотеки для інтерактивних карт. URL: https://leafletjs.com/reference.html (дата звернення: 10.03.2026).
19. Lucide. Колекція відкритих іконок для веб-інтерфейсів: інтеграція з React-компонентами. URL: https://lucide.dev/docs/guide/packages/lucide-react (дата звернення: 24.03.2026).

20. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p. [Електронна копія]. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 08.02.2026).
21. MediatR. Офіційна вікі-документація бібліотеки для реалізації патерну Mediator в .NET додатках. URL: https://github.com/jbogard/MediatR/wiki (дата звернення: 04.03.2026).
22. Meta Platforms Inc. Документація бібліотеки React 19 для розробки користувацьких інтерфейсів. React Docs. URL: https://react.dev/ (дата звернення: 14.02.2026).
23. React Hook Form. Ефективне управління формами та валідація в React-додатках: офіційна документація. URL: https://react-hook-form.com/get-started (дата звернення: 20.02.2026).
24. React Leaflet. Інтеграція мап Leaflet у React-додатки: документація та приклади використання. URL: https://react-leaflet.js.org/docs/start-introduction/ (дата звернення: 12.03.2026).
25. Security Groups в Amazon VPC. Налаштування віртуальних брандмауерів для захисту хмарних ресурсів. URL: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html (дата звернення: 03.04.2026).